Subversion 权威指南

针对 Subversion 1.5 (根据 r3301 编译)

Ben Collins-Sussman
Brian W. Fitzpatrick
C. Michael Pilato

目录

插图清单

表格清单

范例清单

前言

Karl Fogel
Chicago, March 14, 2004.

一个差劲的常见问题列表(FAQ)总是充斥着作者渴望被问到的问题，而不是人们真正想要了解的问题。也许你曾经见过下面这样的问题：

> Q：怎样使用Glorbosoft XYZ最大程度的提高团队生产率？
>
> A：许多客户希望知道怎样利用我们革命性的专利办公套件最大程度的提高生产率。答案非常简单：首先，点击“文件” 菜单，找到“提高生产率”菜单项，然后…

The problem with such FAQs is that they are not, in a literal sense, FAQs at all. No one ever called the tech support line and asked, “How can we maximize productivity?” Rather, people asked highly specific questions, such as “How can we change the calendaring system to send reminders two days in advance instead of one?” and so on. But it's a lot easier to make up imaginary Frequently Asked Questions than it is to discover the real ones. Compiling a true FAQ sheet requires a sustained, organized effort: over the lifetime of the software, incoming questions must be tracked, responses monitored, and all gathered into a coherent, searchable whole that reflects the collective experience of users in the wild. It calls for the patient, observant attitude of a field naturalist. No grand hypothesizing, no visionary pronouncements here—open eyes and accurate note-taking are what's needed most.

What I love about this book is that it grew out of just such a process, and shows it on every page. It is the direct result of the authors' encounters with users. It began with Ben Collins-Sussman's observation that people were asking the same basic questions over and over on the Subversion mailing lists: what are the standard workflows to use with Subversion? Do branches and tags work the same way as in other version control systems? How can I find out who made a particular change?

Frustrated at seeing the same questions day after day, Ben worked intensely over a month in the summer of 2002 to write *The Subversion Handbook*, a 60-page manual that covered all the basics of using Subversion. The manual made no pretense of being complete, but it was distributed with Subversion and got users over that initial hump in the learning curve. When O'Reilly decided to publish a full-length Subversion book, the path of least resistance was obvious: just expand the Subversion handbook.

The three coauthors of the new book were thus presented with an unusual opportunity. Officially, their task was to write a book top-down, starting from a table of contents and an initial draft. But they also had access to a steady stream—indeed, an uncontrollable geyser—of bottom-up source material. Subversion was already in the hands of thousands of early adopters, and those users were giving tons of feedback, not only about Subversion, but also about its existing documentation.

During the entire time they wrote this book, Ben, Mike, and Brian haunted the Subversion mailing lists and chat rooms incessantly, carefully noting the problems users were having in real-life situations. Monitoring such feedback was part of their job descriptions at CollabNet anyway, and it gave them a huge advantage when they set out to document Subversion. The book they produced is grounded firmly in the bedrock of experience, not in the shifting sands of wishful thinking; it combines the best aspects of user manual and FAQ sheet. This duality might not be noticeable on a first reading. Taken in order, front to back, the book is simply a straightforward description of a piece of software. There's the overview, the obligatory guided tour, the chapter on administrative configuration, some advanced

topics, and of course, a command reference and troubleshooting guide. Only when you come back to it later, seeking the solution to some specific problem, does its authenticity shine out: the telling details that can only result from encounters with the unexpected, the examples honed from genuine use cases, and most of all the sensitivity to the user's needs and the user's point of view.

Of course, no one can promise that this book will answer every question you have about Subversion. Sometimes the precision with which it anticipates your questions will seem eerily telepathic; yet occasionally, you will stumble into a hole in the community's knowledge and come away empty-handed. When this happens, the best thing you can do is email `<users@subversion.tigris.org>` and present your problem. The authors are still there and still watching, and the authors include not just the three listed on the cover, but many others who contributed corrections and original material. From the community's point of view, solving your problem is merely a pleasant side effect of a much larger project—namely, slowly adjusting this book, and ultimately Subversion itself, to more closely match the way people actually use it. They are eager to hear from you, not only because they can help you, but because you can help them. With Subversion, as with all active free software projects, *you are not alone*.

让这本书将成为你的第一个伙伴。

序言

> “即使你能确认什么是完美，也不要让完美成为好的敌人，更何况你不能确认。因为落入过去陷阱的不悦，你会在设计时因为担心自己的缺陷而无所作为。”
>
> —Greg Hudson，Subversion开发者

In the world of open source software, the Concurrent Versions System (CVS) was the tool of choice for version control for many years. And rightly so. CVS was open source software itself, and its nonrestrictive modus operandi and support for networked operation allowed dozens of geographically dispersed programmers to share their work. It fit the collaborative nature of the open source world very well. CVS and its semi-chaotic development model have since become cornerstones of open source culture.

But CVS was not without its flaws, and simply fixing those flaws promised to be an enormous effort. Enter Subversion. Subversion was designed to be a successor to CVS, and its originators set out to win the hearts of CVS users in two ways—by creating an open source system with a design (and “look and feel”) similar to CVS, and by attempting to avoid most of CVS's noticeable flaws. While the result isn't necessarily the next great evolution in version control design, Subversion *is* very powerful, very usable, and very flexible. And for the most part, almost all newly started open source projects now choose Subversion instead of CVS.

This book is written to document the 1.5 series of the Subversion version control system. We have made every attempt to be thorough in our coverage. However, Subversion has a thriving and energetic development community, so already a number of features and improvements are planned for future versions that may change some of the commands and specific notes in this book.

1. 读者

本书是为了那些在计算机领域有丰富知识，并且希望使用Subversion管理数据的人士准备的。尽管Subversion可以在多种不同的操作系统上运行，但其基本用户操作界面是基于命令行的，也就是我们将要在本书中讲述和使用的命令行工具(**svn**)。

出于一致性的考虑，本书的例子假定读者使用的是类Unix的操作系统，并且熟悉Unix和命令行界面。当然，**svn**程序也可以在人Microsoft Windows这样的非Unix平台上运行，除了一些微小的不同，如使用反斜线(\)代替正斜线(/)作为路径分隔符，在Windows上运行svn程序的输入和输出与在Unix平台上运行完全一致。

大多数读者可能是那些需要跟踪代码变化的程序员或者系统管理员，这是Subversion最普遍的用途，因此这个场景贯穿于整本书的例子中。但是Subversion可以用来管理任何类型的数据：图像、音乐、数据库、文档等等。对于Subversion，数据就是数据而已。

本书假定读者从来没有使用过任何版本控制工具，同时，我们也努力使CVS(或其他系统)用户能够轻松的投入到Subversion使用当中，在侧栏不时会出现一些涉及CVS的内容，此外，在附录的一个章节中总结了Subversion和CVS的区别。

Note also that the source code examples used throughout the book are only examples. While they will compile with the proper compiler incantations, they are intended to illustrate a particular scenario and not necessarily to serve as examples of good programming style or practices.

2. 怎样阅读本书

Technical books always face a certain dilemma: whether to cater to *top-down* or to *bottom-up* learners. A top-down learner prefers to read or skim documentation, getting a large overview of how the system works; only then does she actually start using the software. A bottom-up learner is a “learn by doing” person—someone who just wants to dive into the software and figure it out as she goes, referring to book sections when necessary. Most books tend to be written for one type of person or the other, and this book is undoubtedly biased toward top-down learners. (And if you're actually reading this section, you're probably already a top-down learner yourself!) However, if you're a bottom-up person, don't despair. While the book may be laid out as a broad survey of Subversion topics, the content of each section tends to be heavy with specific examples that you can try-by-doing. For the impatient folks who just want to get going, you can jump right to 附录 A, *Subversion* 快速入门指南.

本书适用于具有不同背景知识的各个层次的读者—从未使用过版本控制的新手到经验丰富的系统管理员都能够从本书中获益。根据基础的不同，某些的章节可能对某些读者更有价值。下面的内容可以看作是为不同类型的读者提供的“推荐阅读清单”：

Experienced system administrators
: 假定你从前使用过版本控制，并且迫切需要建立起Subversion服务器并尽快运行起来，第 5 章 版本库管理和第 6 章 服务配置将会告诉你如何建立起一个版本库，并将其在网络上发布。此后，依靠你的CVS使用经验，第 2 章 基本使用和附录 B, *CVS* 用户的 *Subversion* 指南将向你展示怎样使用Subversion客户端软件。

新用户
: 如果管理员已经为你准备好了Subversion服务，你所需要的是学习如何使用客户端。如果你没有使用版本控制系统(像CVS)的经验，那么第 1 章 基本概念和第 2 章 基本使用是重要的入门教程，其中介绍了版本控制的重要思想。

高级用户
: 无论是用户还是管理员，项目终将会壮大起来。那时，就需要学习更多Subversion的高级功能，像如何使用分支和执行合并(第 4 章 分支与合并)、怎样使用Subversion的属性(第 3 章 高级主题)、怎样配制运行参数(第 7 章 定制你的 *Subversion* 体验)等等。这两章在学习的初期并不重要，但熟悉了基本操作之后还是非常有必要了解一下。

开发者
: 你应该已经很熟悉Subversion了，并且想扩展它或使用它的API开发新软件。第 8 章 嵌入 *Subversion*将最适合你。

The book ends with reference material—第 9 章 *Subversion* 完全参考 is a reference guide for all Subversion commands, and the appendixes cover a number of useful topics. These are the chapters you're mostly likely to come back to after you've finished the book.

3. 本书约定

本节描述了本书中使用的各种约定。

等宽字体
: Used for literal user input, command output, and command-line options

斜体
Used for program and Subversion tool subcommand names, file and directory names, and new terms

等宽字体
用于代码和文本中的可替换部分

Also, we sprinkled especially helpful or important bits of information throughout the book (in contextually relevant locations), set off visually so they're easy to find. Look for the following icons as you read:

注意

This icon designates a special point of interest.

提示

This icon designates a helpful tip or recommended best practice.

警告

This icon designates a warning. Pay close attention to these to avoid running into problems.

4. 本书的结构

以下是各个章节的内容介绍：

第 1 章 基本概念
介绍了版本控制的基础知识及不同的版本模型，同时讲述了Subversion版本库，工作拷贝和修订版本的概念。

第 2 章 基本使用
引领你开始一个Subversion用户的工作。示范怎样使用Subversion获得、修改和提交数据。

第 3 章 高级主题
覆盖了许多普通用户最终要面对的复杂特性，例如版本化的元数据、文件锁定和peg修订版本。

第 4 章 分支与合并
讨论分支、合并与标签，包括最佳实践的介绍，常见用例的描述，怎样取消修改，以及怎样从一个分支转到另一个分支。

第 5 章 版本库管理
讲述Subversion版本库的基本概念，怎样建立、配置和维护版本库，以及哪些工具可以完成上述的工作。

第 6 章 服务配置
描述了如何配置Subversion服务器，以及三种访问版本库的方式，`HTTP`、`svn`协议和本地磁盘访问。这里也介绍了认证，授权与匿名访问的细节。

第 7 章 定制你的 *Subversion* 体验
研究了Subversion的客户端配置文件，对国际化字符的处理，以及Subversion如何与外置工具交互。

第 8 章 嵌入 *Subversion*
Describes the internals of Subversion, the Subversion filesystem, and the working copy administrative areas from a programmer's point of view. It also demonstrates how to use the public APIs to write a program that uses Subversion.

第 9 章 *Subversion* 完全参考
以大量的实例，详细描述了**svn**、**svnadmin**和**svnlook**的所有子命令。

附录 A, *Subversion* 快速入门指南
因为缺乏耐心，我们会立刻解释如何安装和使用Subversion，我们已经告诉你了。

附录 B, *CVS* 用户的 *Subversion* 指南
详细比较了Subversion与CVS的异同，并针对如何消除多年使用CVS养成的坏习惯提出建议。内容包括Subversion修订版本号、版本化的目录、离线操作、**update**与**status**的对比、分支、标签、元数据、冲突处理和认证。

附录 C, *WebDAV* 和自动版本
描述了WebDAV与DeltaV的细节，并介绍了如何将Subversion版本库作为可读/写的DAV共享装载。

附录 D, 版权
A copy of the Creative Commons Attribution License, under which this book is licensed.

5. 本书是免费的

This book started out as bits of documentation written by Subversion project developers, which were then coalesced into a single work and rewritten. As such, it has always been under a free license (see 附录 D, 版权). In fact, the book was written in the public eye, originally as part of the Subversion project itself. This means two things:

- 总可以在Subversion的版本库里找到本书的最新版本。

- You can make changes to this book and redistribute it however you wish—it's under a free license. Your only obligation is to maintain proper attribution to the original authors. Of course, we'd much rather you send feedback and patches to the Subversion developer community, instead of distributing your private version of this book.

The online home of this book's development and most of the volunteer-driven translation efforts regarding it is `http://svnbook.red-bean.com`. There you can find links to the latest releases and tagged versions of the book in various formats, as well as instructions for accessing the book's Subversion repository (where its DocBook XML source code lives). Feedback is welcomed—encouraged, even. Please submit all comments, complaints, and patches against the book sources to `<svnbook-dev@red-bean.com>`.

6. 致谢

没有Subversion就不可能有(即使有也没什么价值)这本书。所以作者衷心感谢Brian Behlendorf和CollabNet，他们独到的眼光开创了这个充满冒险但雄心勃勃的开源项目；Jim Blandy贡献了Subversion最初的名字和设计—我们爱你，Jim。还有Karl Fogel，一个好朋友和伟大的社区领袖。[1]

Thanks to O'Reilly and our various editors: Chuck Toporek, Linda Mui, Tatiana Apandi, Mary Brady, and Mary Treseler. Their patience and support has been tremendous.

Finally, we thank the countless people who contributed to this book with informal reviews, suggestions, and patches. While this is undoubtedly not a complete list, this book would be incomplete and incorrect without their help: Bhuvaneswaran A, David Alber, C. Scott Ananian, David Anderson, Ariel Arjona, Seth Arnold, Jani Averbach, Charles Bailey, Ryan Barrett, Francois Beausoleil, Brian R. Becker, Yves Bergeron, Karl Berry, Jennifer Bevan, Matt Blais, Jim Blandy, Phil Bordelon, Sietse Brouwer, Tom Brown, Zack Brown, Martin Buchholz, Paul Burba, Sean Callan-Hinsvark, Branko Cibej, Archie Cobbs, Jason Cohen, Ryan Cresawn, John R. Daily, Peter Davis, Olivier Davy, Robert P. J. Day, Mo DeJong, Brian Denny, Joe Drew, Markus Dreyer, Nick Duffek, Boris Dusek, Ben Elliston, Justin Erenkrantz, Jens M. Felderhoff, Kyle Ferrio, Shlomi Fish, Julian Foad, Chris Foote, Martin Furter, Vlad Georgescu, Peter Gervai, Dave Gilbert, Eric Gillespie, David Glasser, Marcel Gosselin, Lieven Govaerts, Steve Greenland, Matthew Gregan, Tom Gregory, Maverick Grey, Art Haas, Mark E. Hamilton, Eric Hanchrow, Liam Healy, Malte Helmert, Michael Henderson, Øyvind A. Holm, Greg Hudson, Alexis Huxley, Auke Jilderda, Toby Johnson, Jens B. Jorgensen, Tez Kamihira, David Kimdon, Mark Benedetto King, Robert Kleemann, Erik Kline, Josh Knowles, Andreas J. Koenig, Axel Kollmorgen, Nuutti Kotivuori, Kalin Kozhuharov, Matt Kraai, Regis Kuckaertz, Stefan Kueng, Steve Kunkee, Scott Lamb, Wesley J. Landaker, Benjamin Landsteiner, Vincent Lefevre, Morten Ludvigsen, Dennis Lundberg, Paul Lussier, Bruce A. Mah, Jonathon Mah, Karl Heinz Marbaise, Philip Martin, Feliciano Matias, Neil Mayhew, Patrick Mayweg, Gareth McCaughan, Craig McElroy, Simon McKenna, Christophe Meresse, Jonathan Metillon, Jean-Francois Michaud, Jon Middleton, Robert Moerland, Marcel Molina Jr., Tim Moloney, Alexander Mueller, Tabish Mustufa, Christopher Ness, Roman Neuhauser, Mats Nilsson, Greg Noel, Joe Orton, Eric Paire, Dimitri Papadopoulos-Orfanos, Jerry Peek, Chris Pepper, Amy Lyn Pilato, Kevin Pilch-Bisson, Hans Polak, Dmitriy Popkov, Michael Price, Mark Proctor, Steffen Prohaska, Daniel Rall, Srinivasa Ramanujan, Jack Repenning, Tobias Ringstrom, Jason Robbins, Garrett Rooney, Joel Rosdahl, Christian Sauer, Ryan Schmidt, Jochem Schulenklopper, Jens Seidel, Daniel Shahaf, Larry Shatzer, Danil Shopyrin, Erik Sjoelund, Joey Smith, W. Snyder, Stefan Sperling, Robert Spier, M. S. Sriram, Russell Steicke, David Steinbrunner, Sander Striker, David Summers, Johan Sundstroem, Ed Swierk, John Szakmeister, Arfrever Frehtes Taifersar Arahesis, Robert Tasarz, Michael W. Thelen, Mason Thomas, Erik van der Kolk, Joshua Varner, Eric Wadsworth, Chris Wagner, Colin Watson, Alex Waugh, Chad Whitacre, Andy Whitcroft, Josef Wolf, Luke Worth, Hyrum Wright, Blair Zajac, Florian Zumbiehl, and the entire Subversion community.

6.1. 来自 Ben Collins-Sussman

Thanks to my wife Frances, who, for many months, got to hear “But honey, I'm still working on the book,” rather than the usual “But honey, I'm still doing email.” I don't know where she gets all that patience! She's my perfect counterbalance.

[1]噢，还要感谢Karl为了本书所付出的辛勤工作。

Thanks to my extended family and friends for their sincere encouragement, despite having no actual interest in the subject. (You know, the ones who say, “Ooh, you wrote a book?” and then when you tell them it's a computer book, sort of glaze over.)

感谢我身边让我富有的朋友，不要那样看我—你们知道你们是谁。

感谢父母对我的低级格式化，和难以置信的角色典范 ，感谢儿子给我机会传承这些东西。

6.2. 来自 Brian W. Fitzpatrick

非常非常感谢我的妻子Marie的理解，支持和最重要的耐心。感谢引导我学会UNIX编程的兄弟Eric，感谢我的母亲和祖母的支持，对我在圣诞夜里埋头工作的理解。

To Mike and Ben: it was a pleasure working with you on the book. Heck, it's a pleasure working with you at work!

感谢所有在Subversion和Apache软件基金会的人们给我机会与你们在一起，没有一天我不从你们那里学到知识。

Lastly, thanks to my grandfather, who always told me that “freedom equals responsibility.” I couldn't agree more.

6.3. 来自 C. Michael Pilato

Special thanks to Amy, my best friend and wife of more than ten incredible years, for her love and patient support, for putting up with the late nights, and for graciously enduring the version control processes I've imposed on her. Don't worry, sweetheart—you'll be a TortoiseSVN wizard in no time!

Gavin, you're able to read half of the words in this book yourself now; sadly, it's the other half that provide the key concepts. And sorry, Aidan — I couldn't find a way to work Disney/Pixar characters into the text. But Daddy loves you both, and can't wait to teach you about programming.

妈妈和爸爸，感谢你们的支持和热情，岳父岳母，以同样的理由感谢你们，还要感谢你们难以置信的女儿。

Hats off to Shep Kendall, through whom the world of computers was first opened to me; Ben Collins-Sussman, my tour guide through the open source world; Karl Fogel, you *are* my `.emacs`; Greg Stein, for oozing practical programming know-how; and Brian Fitzpatrick, for sharing this writing experience with me. To the many folks from whom I am constantly picking up new knowledge—keep dropping it!

最后，对所有为我展现完美卓越创造力的人们—感谢。

7. Subversion 是什么?

Subversion是一个自由/开源的版本控制系统。也就是说，在Subversion管理下，文件和目录可以超越时空。也就是Subversion允许你数据恢复到早期版本，或者是检查数据修改的历史。正因为如此，许多人将版本控制系统当作一种神奇的“时间机器” 。

Subversion的版本库可以通过网络访问，从而使用户可以在不同的电脑上进行操作。从某种程度上来说，允许用户在各自的空间里修改和管理同一组数据可以促进团队协作。因为修改不再是单线进行，开发速度会更快。此外，由于所有的工作都已版本化，也就不必担心由于错误的更改而影响软件质量—如果出现不正确的更改，只要撤销那一次更改操作即可。

某些版本控制系统本身也是软件配置管理(SCM)系统，这种系统经过精巧的设计，专门用来管理源代码树，并且具备许多与软件开发有关的特性—比如，对编程语言的支持，或者提供程序构建工具。不过Subversion并不是这样的系统。它是一个通用系统，可以管理任何类型的文件集。对你来说，这些文件这可能是源程序—而对别人，则可能是一个货物清单或者是数字电影。

7.1. Subversion 是正确的工具吗?

如果你是一个考虑如何使用Subversion的用户或管理员，你要问自己的第一件事就是： "这是这项工作的正确工具吗？"，Subversion是一个梦幻般的锤子，但要小心不要把任何问题当作钉子。

If you need to archive old versions of files and directories, possibly resurrect them, or examine logs of how they've changed over time, then Subversion is exactly the right tool for you. If you need to collaborate with people on documents (usually over a network) and keep track of who made which changes, then Subversion is also appropriate. This is why Subversion is so often used in software development environments— working on a development team is an inherently social activity, and Subversion makes it easy to collaborate with other programmers. Of course, there's a cost to using Subversion as well: administrative overhead. You'll need to manage a data repository to store the information and all its history, and be diligent about backing it up. When working with the data on a daily basis, you won't be able to copy, move, rename, or delete files the way you usually do. Instead, you'll have to do all of those things through Subversion.

假定你能够接受额外的工作流程，你一定要确定不要使用Subversion来解决其他工具能够完成的很好的工作。例如，因为Subversion会复制所有的数据到参与者，一个常见的误用是将其作为普通的分布式系统。问题是此类数据通常很少修改，在这种情况下，使用Subversion有点“过了”。[2]有一些可以复制数据更简单的工具，没有必要过度的跟踪变更，例如**rsync**或**unison**。

7.2. Subversion 的历史

早在2000年，CollabNet，Inc. (`http://www.collab.net`)就开始寻找CVS替代产品的开发人员。CollabNet提供了一个名为CollabNet企业版(CEE)的协作软件套件。这个软件套件的一个组成部分就是版本控制系统。尽管CEE在最初采用了CVS作为其版本控制系统，但是CVS的局限性从一开始就很明显，CollabNet知道，迟早要找到一个更好的替代品。遗憾的是，CVS已经成为开源世界事实上的标准，很大程度上是因为没有更好的替代品，至少是没有可以自由使用的替代品。所以CollabNet决定从头编写一个新的版本控制系统，这个系统保留CVS的基本思想，但是要修正其中错误和不合理的特性。

In February 2000, they contacted Karl Fogel, the author of *Open Source Development with CVS* (Coriolis, 1999), and asked if he'd like to work on this new project. Coincidentally, at the time Karl

[2]或者像一个朋友说的，“用别克拍苍蝇。”

was already discussing a design for a new version control system with his friend Jim Blandy. In 1995, the two had started Cyclic Software, a company providing CVS support contracts, and although they later sold the business, they still used CVS every day at their jobs. Their frustration with CVS had led Jim to think carefully about better ways to manage versioned data, and he'd already come up with not only the name “Subversion,” but also the basic design of the Subversion data store. When CollabNet called, Karl immediately agreed to work on the project, and Jim got his employer, Red Hat Software, to essentially donate him to the project for an indefinite period of time. CollabNet hired Karl and Ben Collins-Sussman, and detailed design work began in May 2000. With the help of some well-placed prods from Brian Behlendorf and Jason Robbins of CollabNet, and from Greg Stein (at the time an independent developer active in the WebDAV/DeltaV specification process), Subversion quickly attracted a community of active developers. It turned out that many people had encountered the same frustrating experiences with CVS and welcomed the chance to finally do something about it.

最初的设计小组设定了简单的开发目标。他们不想在版本控制方法学中开垦处女地，他们只是希望修正CVS。他们决定Subversion应符合CVS的特性，并保留相同的开发模型，但不再重复CVS的一些显著缺陷。尽管Subversion并不需要成为CVS的完全替代品，但它应该与CVS保持足够的相似性，以使CVS用户可以轻松的转移到Subversion上。

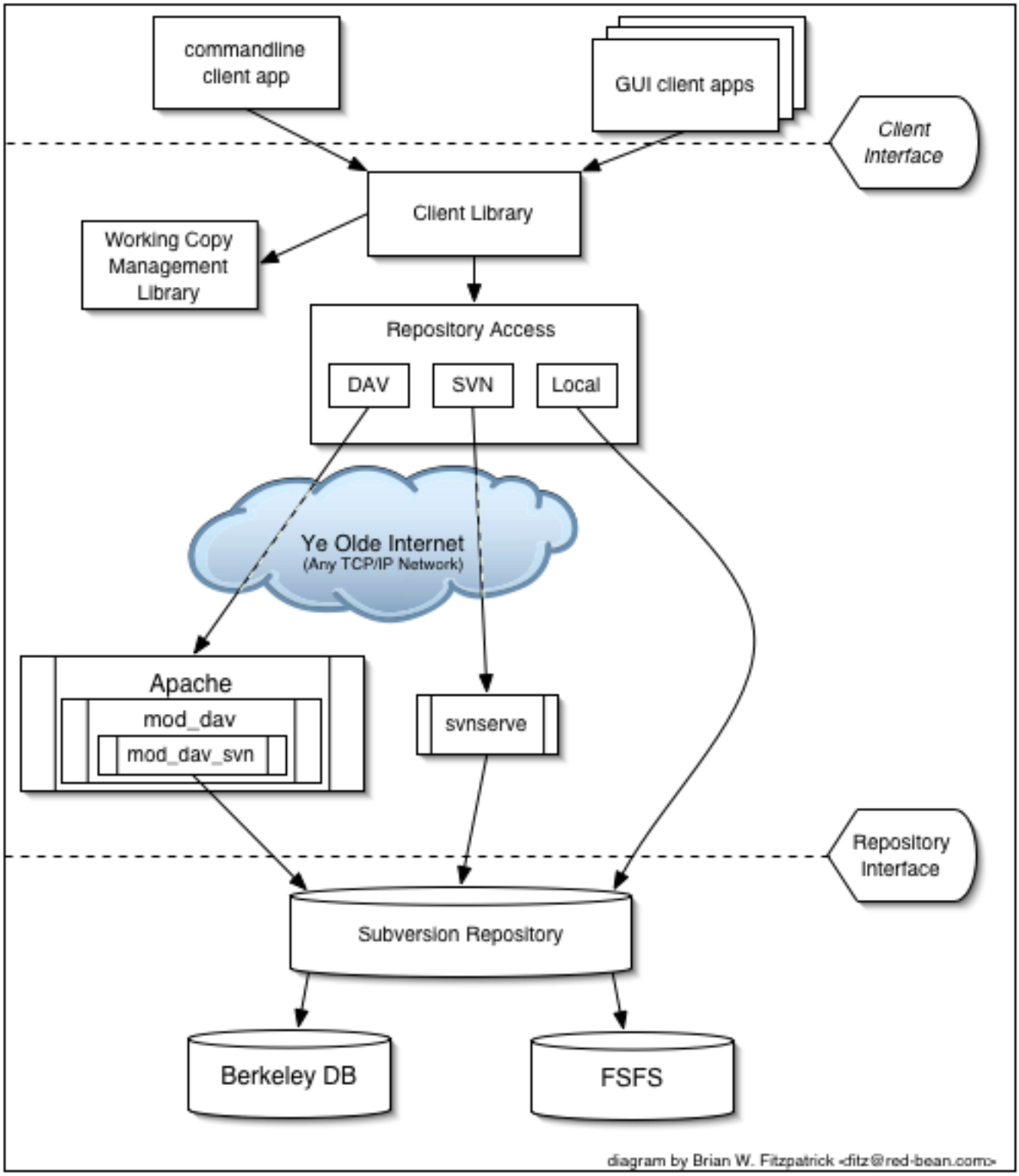
经过14个月的编码，2001年8月31日，Subversion能够“自己管理自己”了，开发者停止使用CVS保存Subversion的代码，而使用Subversion本身。

虽然CollabNet启动了这个项目，并且一直提供了大量的工作支持(它为一些全职的Subversion开发者提供薪水)，但Subversion像其它许多开源项目一样，被松散的、透明的规则管理着，这样的规则激励着知识界的精英们。CollabNet的版权许可证完全符合Debian的自由软件方针。也就是说，任何人都可以根据自己的意愿自由的下载、修改和重新发布Subversion，不需要CollabNet或其他人的授权。

7.3. Subversion 的架构

图 1 “Subversion 的架构”给出了Subversion设计总体上的“俯视图”。

图 **1. Subversion** 的架构



图中的一端是保存所有版本数据的Subversion版本库，另一端是Subvesion的客户程序，管理着所有版本数据的本地影射(称为“工作拷贝”)，在这两极之间是各种各样的版本库访问(RA)层，某些使用电脑网络通过网络服务器访问版本库，某些则绕过网络服务器直接访问版本库。

7.4. Subversion 的组件

Subversion, once installed, has a number of different pieces. The following is a quick overview of what you get. Don't be alarmed if the brief descriptions leave you scratching your head—*plenty* more pages in this book are devoted to alleviating that confusion.

svn
: The command-line client program

svnversion
: A program for reporting the state (in terms of revisions of the items present) of a working copy

svnlook
: A tool for directly inspecting a Subversion repository

svnadmin
: A tool for creating, tweaking, or repairing a Subversion repository

mod_dav_svn
: A plug-in module for the Apache HTTP Server, used to make your repository available to others over a network

svnserve
: 一个单独运行的服务器程序，可以作为守护进程或由SSH调用。这是另一种使版本库可以通过网络访问的方式。

svndumpfilter
: A program for filtering Subversion repository dump streams

svnsync
: A program for incrementally mirroring one repository to another over a network

7.5. Subversion 有什么新东西?

此书的第一版在2004年发布，也就是Subversion1.5发布后不久。经过4年，Subversion发布了5个主要的新版本，修正了bug，增加了主要的新特性。我们一直保持本书在线版本的更新，我们现在很兴奋的是O'Reilly将会发布包含Subversion的1.5版本的第二版，1.5版本是项目的一个主要里程碑。下面是从Subversion 1.0以来主要变更的总结，注意这不是完整的列表；详细信息可以访问Subversion的网站`http://subversion.tigris.org`。

Subversion 1.1 (2004年9月)
: Release 1.1 introduced FSFS, a flat-file repository storage option for the repository. While the Berkeley DB backend is still widely used and supported, FSFS has since become the default choice for newly created repositories due to its low barrier to entry and minimal maintenance requirements. Also in this release came the ability to put symbolic links under version control, auto-escaping of URLs, and a localized user interface.

Subversion 1.2 (2005年5月)
: Release 1.2 introduced the ability to create server-side locks on files, thus serializing commit access to certain resources. While Subversion is still a fundamentally concurrent version control system,

certain types of binary files (e.g. art assets) cannot be merged together. The locking feature fulfills the need to version and protect such resources. With locking also came a complete WebDAV auto-versioning implementation, allowing Subversion repositories to be mounted as network folders. Finally, Subversion 1.2 began using a new, faster binary-differencing algorithm to compress and retrieve old versions of files.

Subversion 1.3 (2005年12月)

1.3版本为**svnserve**服务器引入路径为基础的授权控制，与Apache服务器对应的特性匹配。Apache服务器自己也获得了新的日志特性，Subversion其他语言的API绑定也取得了巨大的进步。

针对Subversion1.4(2006年9月)

Release 1.4 introduced a whole new tool—**svnsync**—for doing one-way repository replication over a network. Major parts of the working copy metadata were revamped to no longer use XML (resulting in client-side speed gains), while the Berkeley DB repository backend gained the ability to automatically recover itself after a server crash.

Subversion 1.5 (2008年6月)

Release 1.5 took much longer to finish than prior releases, but the headliner feature was gigantic: semi-automated tracking of branching and merging. This was a huge boon for users, and pushed Subversion far beyond the abilities of CVS and into the ranks of commercial competitors such as Perforce and ClearCase. Subversion 1.5 also introduced a bevy of other user-focused features, such as interactive resolution of file conflicts, partial checkouts, client-side management of changelists, powerful new syntax for externals definitions, and SASL authentication support for the **svnserve** server.

第 1 章 基本概念

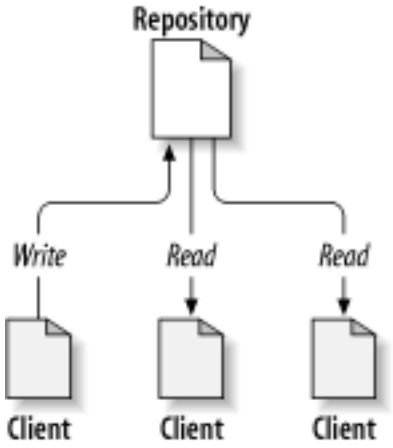
本章主要为那些不熟悉版本控制技术的入门者提供一个简单扼要的、非系统的介绍。我们将从版本控制的基本概念开始，随后阐述Subversion的独特理念，并演示一些使用Subversion的例子。

虽然我们在本章中以分享程序源代码作为例子，但是记住Subversion可以管理任何类型的文件集—它并非是程序员专用的。

1.1. 版本库

Subversion是一个“集中式”的信息共享系统。版本库是Subversion的核心部分，是数据的中央仓库。版本库以典型的文件和目录结构形式文件系统树来保存信息。任意数量的客户端连接到Subversion版本库，读取、修改这些文件。客户端通过写数据将信息分享给其他人，通过读取数据获取别人共享的信息。图 1.1 “一个典型的客户/服务器系统”展示了这种系统：

图 **1.1.** 一个典型的客户/服务器系统



这有什么意义吗？说了这么多，Subversion听起来和一般的文件服务器没什么不同。事实上，Subversion的版本库的确是一种文件服务器，但不是“一般”的文件服务器。Subversion版本库的特别之处在于，它会记录每一次改变：每个文件的改变，甚至是目录树本身的改变，例如文件和目录的添加、删除和重新组织。

When a client reads data from the repository, it normally sees only the latest version of the filesystem tree. But the client also has the ability to view *previous* states of the filesystem. For example, a client can ask historical questions such as “What did this directory contain last Wednesday?” and “Who was the last person to change this file, and what changes did he make?” These are the sorts of questions that are at the heart of any *version control system*: systems that are designed to track changes to data over time.

1.2. 版本模型

The core mission of a version control system is to enable collaborative editing and sharing of data. But different systems use different strategies to achieve this. It's important to understand these different strategies, for a couple of reasons. First, it will help you compare and contrast existing version control

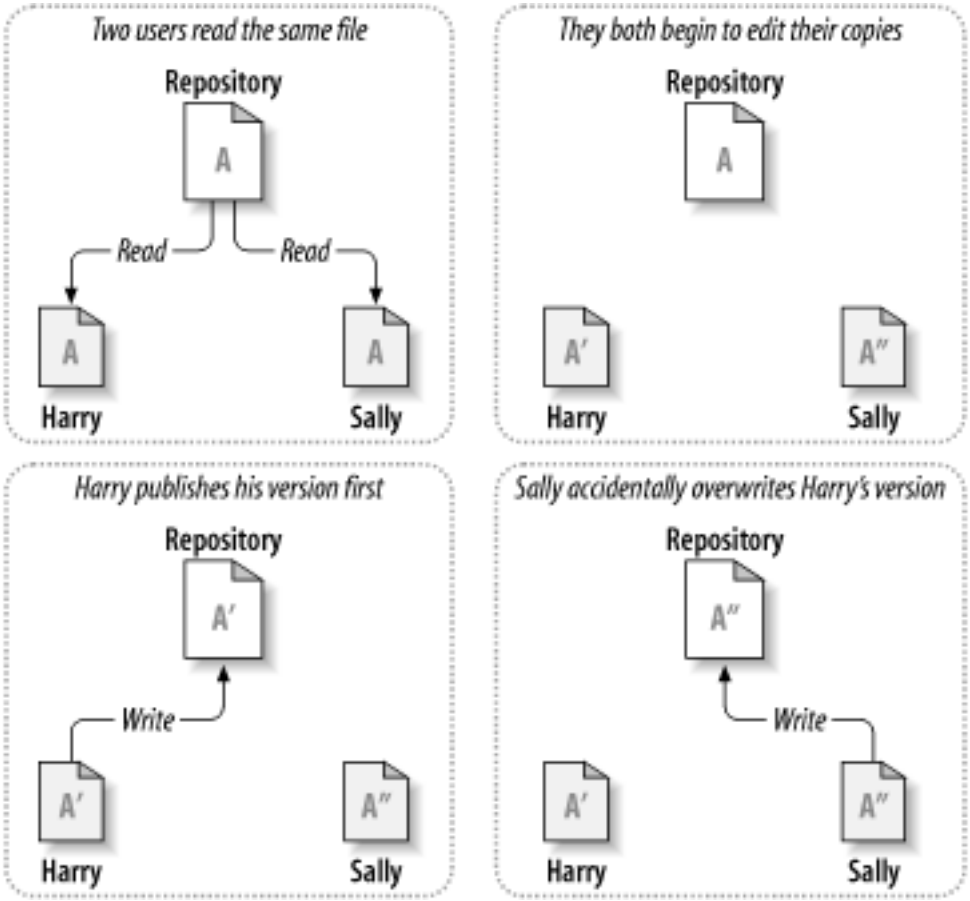
systems, in case you encounter other systems similar to Subversion. Beyond that, it will also help you make more effective use of Subversion, since Subversion itself supports a couple of different ways of working.

1.2.1. 文件共享的问题

所有的版本控制系统都需要解决这样一个基础问题：怎样让系统允许用户共享信息，而不会让他们因意外而互相干扰？版本库里意外覆盖别人的更改非常的容易。

Consider the scenario shown in 图 1.2 “需要避免的问题”. Suppose we have two coworkers, Harry and Sally. They each decide to edit the same repository file at the same time. If Harry saves his changes to the repository first, it's possible that (a few moments later) Sally could accidentally overwrite them with her own new version of the file. While Harry's version of the file won't be lost forever (because the system remembers every change), any changes Harry made *won't* be present in Sally's newer version of the file, because she never saw Harry's changes to begin with. Harry's work is still effectively lost—or at least missing from the latest version of the file—and probably by accident. This is definitely a situation we want to avoid!

图 **1.2.** 需要避免的问题

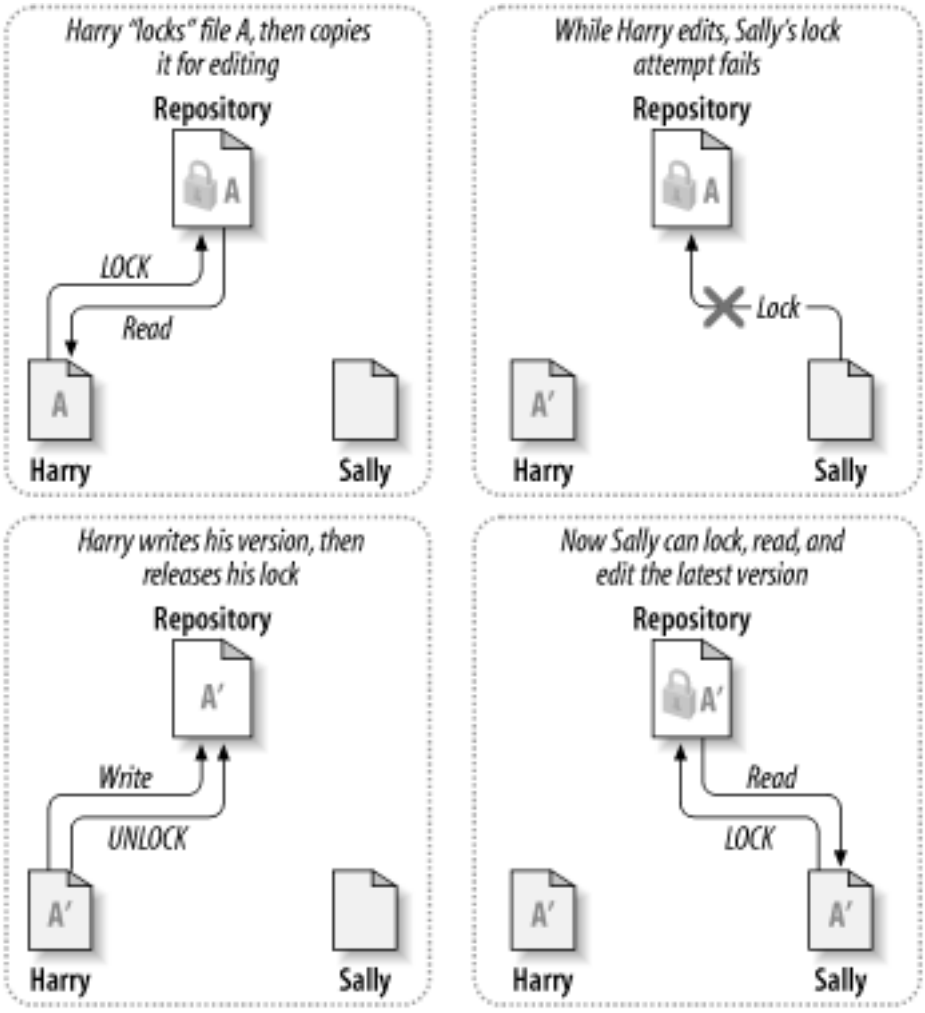


1.2.2. “锁定-修改-解锁”方案

Many version control systems use a *lock-modify-unlock* model to address the problem of many authors clobbering each other's work. In this model, the repository allows only one person to change a file at

a time. This exclusivity policy is managed using locks. Harry must “lock” a file before he can begin making changes to it. If Harry has locked a file, Sally cannot also lock it, and therefore cannot make any changes to that file. All she can do is read the file and wait for Harry to finish his changes and release his lock. After Harry unlocks the file, Sally can take her turn by locking and editing the file. 图 1.3 “ “锁定–修改–解锁”方案” demonstrates this simple solution.

图 **1.3.** “锁定–修改–解锁”方案



锁定–修改–解锁模型有一点问题就是限制太多，经常会成为用户的障碍：

- 锁定可能导致管理问题。有时候Harry会锁住文件然后忘了此事，这就是说Sally一直等待解锁来编辑这些文件，她在这里僵住了。然后Harry去旅行了，现在Sally只好去找管理员放开锁，这种情况会导致不必要的耽搁和时间浪费。

- 锁定可能导致不必要的线性化开发。如果Harry编辑一个文件的开始，Sally想编辑同一个文件的结尾，这种修改不会冲突，设想修改可以正确的合并到一起，他们可以轻松的并行工作而没有太多的坏处，没有必要让他们轮流工作。

- 锁定可能导致错误的安全状态。假设Harry锁定和编辑一个文件A，同时Sally锁定并编辑文件B，如果A和B互相依赖，这种变化是必须同时作的，这样A和B不能正确的工作了，锁定机制对防止此类问题将无能为力—从而产生了一种处于安全状态的假相。很容易想象Harry

和Sally都以为自己锁住了文件，而且从一个安全，孤立的情况开始工作，因而没有尽早发现他们不匹配的修改。锁定经常成为真正交流的替代品

1.2.3. “拷贝–修改–合并”方案

Subversion，CVS和一些版本控制系统使用拷贝–修改–合并模型，在这种模型里，每一个客户联系项目版本库建立一个个人工作拷贝—版本库中文件和目录的本地映射。用户并行工作，修改各自的工作拷贝，最终，各个私有的拷贝合并在一起，成为最终的版本，这种系统通常可以辅助合并操作，但是最终要靠人工去确定正误。

Here's an example. Say that Harry and Sally each create working copies of the same project, copied from the repository. They work concurrently and make changes to the same file A within their copies. Sally saves her changes to the repository first. When Harry attempts to save his changes later, the repository informs him that his file A is *out of date*. In other words, file A in the repository has somehow changed since he last copied it. So Harry asks his client to *merge* any new changes from the repository into his working copy of file A. Chances are that Sally's changes don't overlap with his own; once he has both sets of changes integrated, he saves his working copy back to the repository. 图 1.4 “拷贝–修改–合并”方案 and 图 1.5 “拷贝–修改–合并”方案(续) show this process.

图 **1.4.** “拷贝–修改–合并”方案

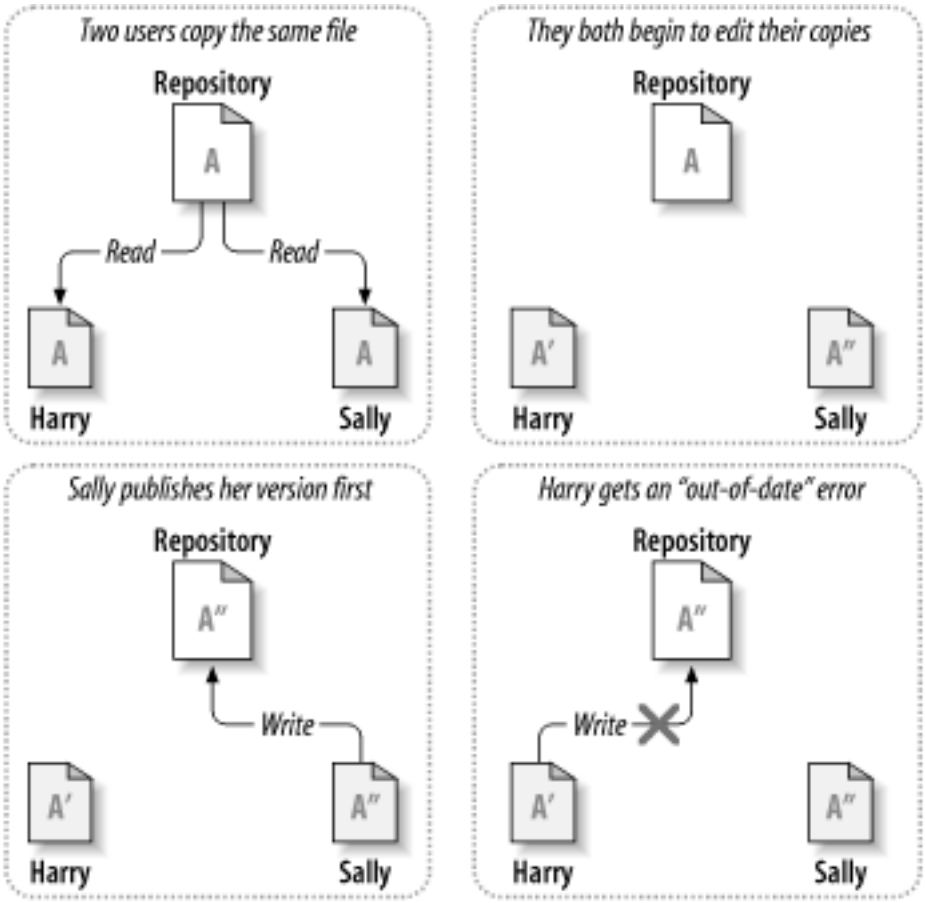
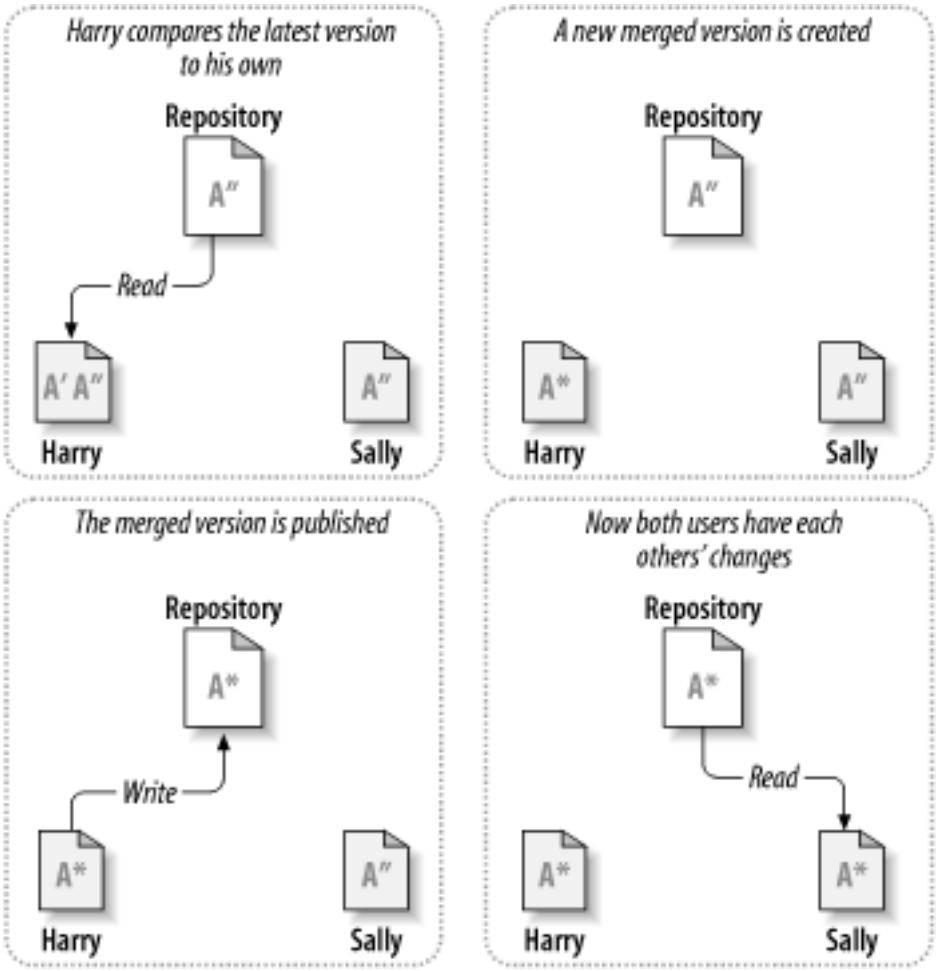


图 **1.5.** “拷贝–修改–合并”方案(续)



但是如果Sally和Harry的修改交迭了该怎么办？这种情况叫做冲突，这通常不是个大问题，当Harry告诉他的客户端去合并版本库的最新修改到自己的工作拷贝时，他的文件A就会处于冲突状态：他可以看到一对冲突的修改集，并手工的选择保留一组修改。需要注意的是软件不能自动的解决冲突，只有人可以理解并作出智能的选择，一旦Harry手工的解决了冲突—也许需要与Sally讨论—它可以安全的把合并的文件保存到版本库。

拷贝–修改–合并模型感觉有一点混乱，但在实践中，通常运行的很平稳，用户可以并行的工作，不必等待别人，当工作在同一个文件上时，也很少会有交迭发生，冲突并不频繁，处理冲突的时间远比等待解锁花费的时间少。

最后，一切都要归结到一条重要的因素：用户交流。当用户交流贫乏，语法和语义的冲突就会增加，没有系统可以强制用户完美的交流，没有系统可以检测语义上的冲突，所以没有任何证据能够承诺锁定系统可以防止冲突，实践中，锁定除了约束了生产力，并没有做什么事。

什么时候锁定是必需的

While the lock-modify-unlock model is considered generally harmful to collaboration, sometimes locking is appropriate.

The copy-modify-merge model is based on the assumption that files are contextually mergeable—that is, that the majority of the files in the repository are line-based text files (such as program source code). But for files with binary formats, such as artwork or sound, it's often impossible to merge conflicting changes. In these situations, it really is necessary for users to take strict turns when changing the file. Without serialized access, somebody ends up wasting time on changes that are ultimately discarded.

While Subversion is primarily a copy-modify-merge system, it still recognizes the need to lock an occasional file, and thus provides mechanisms for this. We discuss this feature in 第 3.7 节 “锁定”.

1.3. Subversion 实践

是时候从抽象转到具体了，在本小节，我们会展示一个Subversion真实使用的例子。

1.3.1. Subversion 版本库的 URL

正如我们在整本书里描述的，Subversion使用URL来识别Subversion版本库中的版本化资源，通常情况下，这些URL使用标准的语法，允许服务器名称和端口作为URL的一部分：

```
$ svn checkout http://svn.example.com:9834/repos
…
```

但是Subversion处理URL的一些细微的不同之处需要注意，例如，使用`file:`访问方法的URL(用来访问本地版本库)必须与习惯一致，可以包括一个`localhost`服务器名或者没有服务器名：

```
$ svn checkout file:///var/svn/repos
…
$ svn checkout file://localhost/var/svn/repos
…
```

同样，在Windows平台下使用`file://`模式时需要使用一个非正式的“标准”语法来访问本机上不在同一个磁盘分区中的版本库。下面的任意一个URL路径语法都可以工作，其中的X表示版本库所在的磁盘分区：

```
C:\> svn checkout file:///X:/var/svn/repos
…
C:\> svn checkout "file:///X|/var/svn/repos"
…
```

在第二个语法里，你需要使用引号包含整个URL，这样竖线字符才不会被解释为管道。当然，也要注意URL使用普通的斜线而不是Windows本地(不是URL)的反斜线。

注意

You cannot use Subversion's `file://` URLs in a regular web browser the way typical `file://` URLs can. When you attempt to view a `file://` URL in a regular web browser, it reads and displays the contents of the file at that location by examining the filesystem directly. However, Subversion's resources exist in a virtual filesystem (see 第 8.1.1 节 “版本库层”), and your browser will not understand how to interact with that filesystem.

最后，必须注意Subversion的客户端会根据需要自动编码URL，这一点和一般的web浏览器一样，举个例子，如果一个URL包含了空格或是一个字符编码大于128的ASCII字符：

```
$ svn checkout "http://host/path with space/project/españa"
```

then Subversion will escape the unsafe characters and behave as though you had typed:

```
$ svn checkout http://host/path%20with%20space/project/espa%C3%B1a
```

If the URL contains spaces, be sure to place it within quotation marks so that your shell treats the whole thing as a single argument to the **svn** program.

版本库的 URL

You can access Subversion repositories through many different methods—on local disk or through various network protocols, depending on how your administrator has set things up for you. A repository location, however, is always a URL. 表 1.1 “版本库访问 URL” describes how different URL schemes map to the available access methods.

表 **1.1.** 版本库访问 **URL**

模式	访问方法
`file:///`	直接版本库访问(本地磁盘)
`http://`	通过配置Subversion的Apache服务器的WebDAV协议
`https://`	与`http://`相似，但是包括SSL加密。
`svn://`	通过`svnserve`服务自定义的协议
`svn+ssh://`	与`svn://`相似，但通过SSH封装。

关于Subversion解析URL的更多信息，见第 1.3.1 节 “Subversion 版本库的 URL”。关于不同的网络服务器类型，见第 6 章 服务配置。

1.3.2. 工作副本

你已经阅读过了关于工作拷贝的内容；现在我们要讲一讲客户端怎样建立和使用它。

一个Subversion工作拷贝是你本地机器上的一个普通目录，保存着一些文件，你可以任意的编辑文件，而且如果是源代码文件，你可以像平常一样编译，你的工作拷贝是你的私有工作区，在你明确的做了特定操作之前，Subversion不会把你的修改与其他人的合并，也不会把你的修改展示给别人，你甚至可以拥有同一个项目的多个工作拷贝。

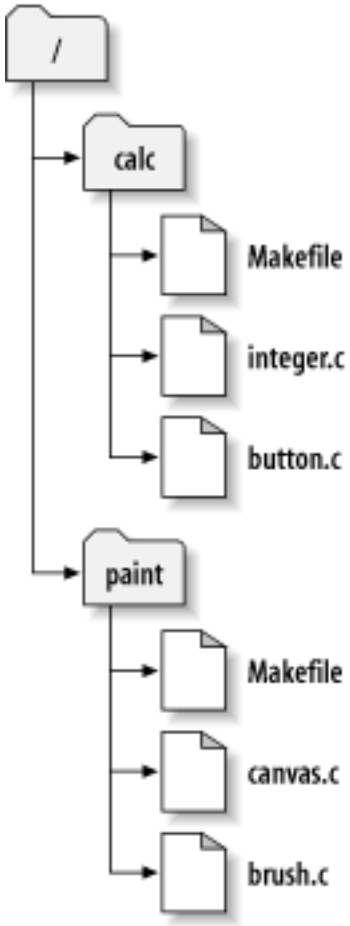
当你在工作拷贝作了一些修改并且确认它们工作正常之后，Subversion提供了一个命令可以“发布”你的修改给项目中的其他人(通过写到版本库)，如果别人发布了各自的修改，Subversion提供了手段可以把这些修改与你的工作目录进行合并(通过读取版本库)。

工作副本也包括一些由 Subversion 创建并维护的额外文件，用来协助执行命令。通常情况下，你的工作副本的每个文件夹都有一个以 `.svn` 为名的文件夹，也被叫做工作副本的管理目录，这个目录里的文件能够帮助 Subversion 识别哪些文件做过修改，哪些文件相对于别人的工作已经过期。

一个典型的Subversion的版本库经常包含许多项目的文件(或者说源代码)，通常每一个项目都是版本库的子目录，在这种布局下，一个用户的工作拷贝往往对应版本库的的一个子目录。

举一个例子，你的版本库包含两个软件项目，`paint`和`calc`。每个项目在它们各自的顶级子目录下，见图 1.6 “版本库的文件系统”。

图 **1.6.** 版本库的文件系统



为了得到一个工作拷贝，你必须检出(*check out*)版本库的一个子树，(术语“check out”听起来像是锁定或者保留资源，实际上不是，只是简单的得到一个项目的私有拷贝)，举个例子，你检出 `/calc`，你可以得到这样的工作拷贝：

```
$ svn checkout http://svn.example.com/repos/calc
A    calc/Makefile
A    calc/integer.c
A    calc/button.c
Checked out revision 56.

$ ls -A calc
Makefile  button.c integer.c .svn/
```

列表中的A表示Subversion增加了一些条目到工作拷贝，你现在有了一个/calc的个人拷贝，有一个附加的目录—.svn—保存着前面提及的Subversion需要的额外信息。

假定你修改了button.c，因为.svn目录记录着文件的修改日期和原始内容，Subversion可以告诉你已经修改了文件，然而，在你明确告诉它之前，Subversion不会将你的改变公开，将改变公开的操作被叫做提交(*committing*，或者是*checking in*)修改到版本库。

To publish your changes to others, you can use Subversion's **svn commit** command:

```
$ svn commit button.c -m "Fixed a typo in button.c."
Sending        button.c
Transmitting file data .
Committed revision 57.
```

Now your changes to button.c have been committed to the repository, with a note describing your change (namely, that you fixed a typo). If another user checks out a working copy of /calc, she will see your changes in the latest version of the file.

假设你有个合作者，Sally，她和你同时取出了/calc的一个工作拷贝，你提交了你对button.c的修改，Sally的工作拷贝并没有改变，Subversion只在用户要求的时候才改变工作拷贝。

To bring her project up to date, Sally can ask Subversion to *update* her working copy, by using the **svn update** command. This will incorporate your changes into her working copy, as well as any others that have been committed since she checked it out.

```
$ pwd
/home/sally/calc

$ ls -A
.svn/ Makefile integer.c button.c

$ svn update
U    button.c
Updated to revision 57.
```

svn update命令的输出表明Subversion更新了button.c的内容，注意，Sally不必指定要更新的文件，subversion利用.svn以及版本库的进一步信息决定哪些文件需要更新。

1.3.3. 修订版本

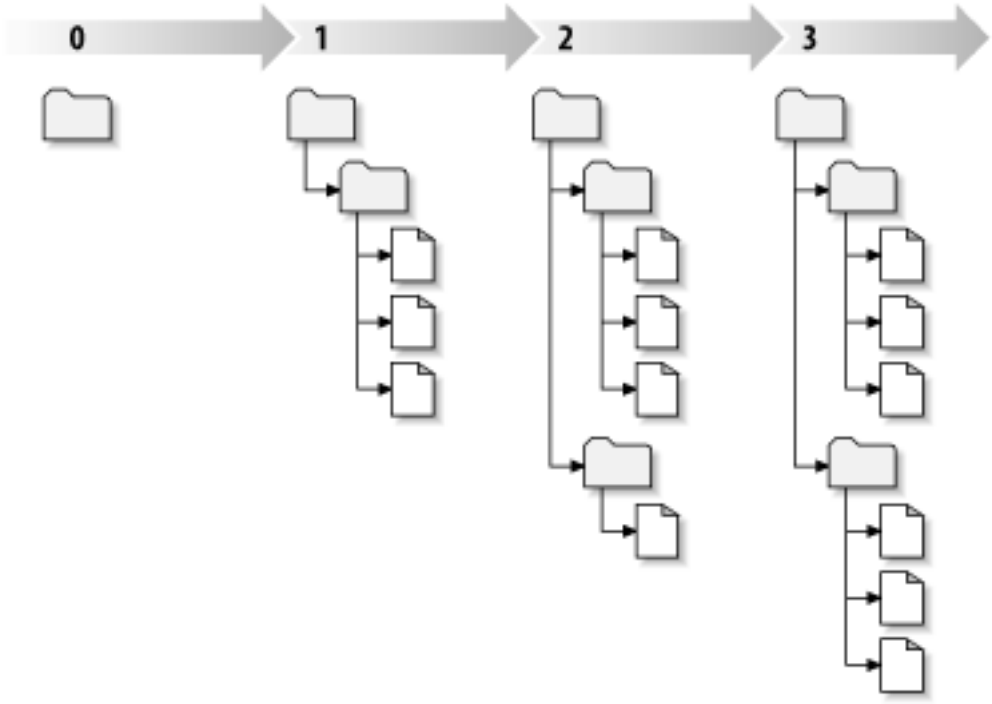
An **svn commit** operation publishes changes to any number of files and directories as a single atomic transaction. In your working copy, you can change files' contents; create, delete, rename, and copy files and directories; and then commit a complete set of changes as an atomic transaction.

By atomic transaction, we mean simply this: either all of the changes happen in the repository, or none of them happens. Subversion tries to retain this atomicity in the face of program crashes, system crashes, network problems, and other users' actions.

每当版本库接受了一个提交，文件系统进入了一个新的状态，叫做一次修订(*revision*)，每一个修订版本被赋予一个独一无二的自然数，一个比一个大，初始修订号是0，只创建了一个空目录，没有任何内容。

图 1.7 “版本库”可以更形象的描述版本库，想象有一组修订号，从0开始，从左到右，每一个修订号有一个目录树挂在它下面，每一个树好像是一次提交后的版本库“快照”。

图 **1.7.** 版本库



全局版本号

Unlike most version control systems, Subversion's revision numbers apply to *entire trees*, not individual files. Each revision number selects an entire tree, a particular state of the repository after some committed change. Another way to think about it is that revision N represents the state of the repository filesystem after the Nth commit. When Subversion users talk about “revision 5 of `foo.c`,” they really mean “`foo.c` as it appears in revision 5.” Notice that in general, revisions N and M of a file do *not* necessarily differ! Many other version control systems use per-file revision numbers, so this concept may seem unusual at first. (Former CVS users might want to see 附录 B, *CVS* 用户的 *Subversion* 指南 for more details.)

需要特别注意的是，工作拷贝并不一定对应版本库中的单个修订版本，他们可能包含多个修订版本的文件。举个例子，你从版本库检出一个工作拷贝，最近的修订号是4：

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

此刻，工作目录与版本库的修订版本4完全对应，然而，你修改了`button.c`并且提交之后，假设没有别的提交出现，你的提交会在版本库建立修订版本5，你的工作拷贝会是这个样子的：

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

Suppose that, at this point, Sally commits a change to `integer.c`, creating revision 6. If you use **svn update** to bring your working copy up to date, it will look like this:

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

Sally对`integer.c`的改变会出现在你的工作拷贝，你对`button.c`的改变还在，在这个例子里，`Makefile`在4、5、6修订版本都是一样的，但是Subversion会把他的`Makefile`的修订号设为6来表明它是最新的，所以你在工作拷贝顶级目录作一次干净的更新，会使得所有内容对应版本库的同一修订版本。

1.3.4. 工作副本怎样跟踪版本库

对于工作拷贝的每一个文件，Subversion在管理区域`.svn/`记录两项关键的信息：

- What revision your working file is based on (this is called the file's *working revision*)
- A timestamp recording when the local copy was last updated by the repository

给定这些信息，通过与版本库通讯，Subversion可以告诉我们工作文件是处于如下四种状态的那一种：

未修改且是当前的
: 文件在工作目录里没有修改，在工作修订版本之后没有修改提交到版本库。**svn commit**操作不做任何事情，**svn update**不做任何事情。

本地已修改且是当前的
: The file has been changed in the working directory, and no changes to that file have been committed to the repository since you last updated. There are local changes that have not been committed to the repository; thus an **svn commit** of the file will succeed in publishing your changes, and an **svn update** of the file will do nothing.

Unchanged, and out of date
: 这个文件在工作目录没有修改，但在版本库中已经修改了。这个文件最终将更新到最新版本，成为当时的公共修订版本。**svn commit**不做任何事情，**svn update**将会取得最新的版本到工作拷贝。

Locally changed, and out of date
: 这个文件在工作目录和版本库都得到修改。一个**svn commit**将会失败，这个文件必须首先更新，**svn update**命令会合并公共和本地修改，如果Subversion不可以自动完成，将会让用户解决冲突。

这看起来需要记录很多事情，但是**svn status**命令可以告诉你工作拷贝中文件的状态，关于此命令更多的信息，请看第 2.4.3.1 节 “查看你的修改概况”。

1.3.5. 混合修订版本的工作副本

As a general principle, Subversion tries to be as flexible as possible. One special kind of flexibility is the ability to have a working copy containing files and directories with a mix of different working revision numbers. Unfortunately, this flexibility tends to confuse a number of new users. If the earlier example showing mixed revisions perplexed you, here's a primer on why the feature exists and how to make use of it.

1.3.5.1. 更新和提交是分开的

One of the fundamental rules of Subversion is that a “push” action does not cause a “pull,” nor vice versa. Just because you're ready to submit new changes to the repository doesn't mean you're ready to receive changes from other people. And if you have new changes still in progress, **svn update** should gracefully merge repository changes into your own, rather than forcing you to publish them.

这个规则的主要副作用就是工作拷贝需要记录额外的信息来追踪混合修订版本，并且也需要能容忍这种混合，当目录本身也是版本化的时候情况更加复杂。

For example, suppose you have a working copy entirely at revision 10. You edit the file `foo.html` and then perform an **svn commit**, which creates revision 15 in the repository. After the commit succeeds, many new users would expect the working copy to be entirely at revision 15, but that's not the case! Any number of changes might have happened in the repository between revisions 10 and 15. The client knows nothing of those changes in the repository, since you haven't yet run **svn update**, and **svn commit** doesn't pull down new changes. If, on the other hand, **svn commit** were to automatically download the newest changes, it would be possible to set the entire working copy to revision 15—but then we'd be breaking the fundamental rule of “push” and “pull” remaining separate actions. Therefore, the only safe thing the Subversion client can do is mark the one file—`foo.html`—as being at revision 15. The rest of the working copy remains at revision 10. Only by running **svn update** can the latest changes be downloaded and the whole working copy be marked as revision 15.

1.3.5.2. 混合修订版本很常见

The fact is, *every time* you run **svn commit** your working copy ends up with some mixture of revisions. The things you just committed are marked as having larger working revisions than everything else. After several commits (with no updates in between), your working copy will contain a whole mixture of revisions. Even if you're the only person using the repository, you will still see this phenomenon.

To examine your mixture of working revisions, use the **svn status** command with the `--verbose` option (see 第 2.4.3.1 节 “查看你的修改概况” for more information).

Often, new users are completely unaware that their working copy contains mixed revisions. This can be confusing, because many client commands are sensitive to the working revision of the item they're examining. For example, the **svn log** command is used to display the history of changes to a file or directory (see 第 2.5.1 节 “产生历史修改列表”). When the user invokes this command on a working copy object, he expects to see the entire history of the object. But if the object's working revision is quite old (often because **svn update** hasn't been run in a long time), the history of the *older* version of the object is shown.

1.3.5.3. 混合版本很有用

If your project is sufficiently complex, you'll discover that it's sometimes nice to forcibly *backdate* (or update to a revision older than the one you already have) portions of your working copy to an earlier revision; you'll learn how to do that in 第 2 章 基本使用. Perhaps you'd like to test an earlier version of a submodule contained in a subdirectory, or perhaps you'd like to figure out when a bug first came into existence in a specific file. This is the “time machine” aspect of a version control system—the feature that allows you to move any portion of your working copy forward and backward in history.

1.3.5.4. 混合版本有限制

无论你如何在工作拷贝中利用混合修订版本，这种灵活性还是有限制的。

First, you cannot commit the deletion of a file or directory that isn't fully up to date. If a newer version of the item exists in the repository, your attempt to delete will be rejected to prevent you from accidentally destroying changes you've not yet seen.

第二，如果目录已经不是最新的了，你不能提交一个目录的元数据更改。你将会在第 3 章 高级主题学习附加“属性”，一个目录的工作修订版本定义了许多条目和属性，因而对一个过期的版本提交属性会破坏一些你没有见到的属性。

1.4. 总结

We covered a number of fundamental Subversion concepts in this chapter:

- We introduced the notions of the central repository, the client working copy, and the array of repository revision trees.
- We saw some simple examples of how two collaborators can use Subversion to publish and receive changes from one another, using the “copy-modify-merge” model.
- We talked a bit about the way Subversion tracks and manages information in a working copy.

现在，你一定对Subversion在多数情形下的工作方式有了很好的认识，有了这些知识的武装，你一定已经准备好跳到下一章去了，一个关于Subversion命令与特性的详细教程。

第 2 章 基本使用

现在，我们将要深入到Subversion的使用细节当中，完成本章时，你将学会所有Subversion日常使用的命令，你将从把数据导入到Subversion开始，接着是初始化的检出(check out)，然后是做出修改并检查，你也将会学到如何在工作拷贝中获取别人的修改，检查他们，并解决所有可能发生的冲突。

Note that this chapter is not meant to be an exhaustive list of all of Subversion's commands—rather, it's a conversational introduction to the most common Subversion tasks that you'll encounter. This chapter assumes that you've read and understood 第 1 章 基本概念 and are familiar with the general model of Subversion. For a complete reference of all commands, see 第 9 章 *Subversion* 完全参考.

2.1. 求助!

Before reading on, here is the most important command you'll ever need when using Subversion: **svn help**. The Subversion command-line client is self-documenting—at any time, a quick **`svn help`** ***`subcommand`*** will describe the syntax, options, and behavior of the subcommand.

```
$ svn help import
import: Commit an unversioned file or tree into the repository.
usage: import [PATH] URL

  Recursively commit a copy of PATH to URL.
  If PATH is omitted '.' is assumed.
  Parent directories are created as necessary in the repository.
  If PATH is a directory, the contents of the directory are added
  directly under URL.
  Unversionable items such as device files and pipes are ignored
  if --force is specified.

Valid options:
  -q [--quiet]             : print nothing, or only summary information
  -N [--non-recursive]     : obsolete; try --depth=files or --depth=immed
  --depth ARG              : limit operation by depth ARG ('empty', 'files
                             'immediates', or 'infinity')
…
```

选项(Options)、开关(Switches)和标志(Flags)，天呐！

The Subversion command-line client has numerous command modifiers (which we call options), but there are two distinct kinds of options: short options are a single hyphen followed by a single letter, and long options consist of two hyphens followed by a number of letters (e.g., `-s` and `--this-is-a-long-option`, respectively). Every option has a long format, but only certain options have an additional short format (these are typically options that are frequently used). To maintain clarity, we *usually* use the long form in code examples, but when describing options, if there's a short form, we'll provide the long form (to improve clarity) and the short form (to make it easier to remember). You should use whichever one you're more comfortable with, but don't try to use both.

2.2. 导入数据到你的版本库

You can get new files into your Subversion repository in two ways: **svn import** and **svn add**. We'll discuss **svn import** now and will discuss **svn add** later in this chapter when we review a typical day with Subversion.

2.2.1. svn import

The **svn import** command is a quick way to copy an unversioned tree of files into a repository, creating intermediate directories as necessary. **svn import** doesn't require a working copy, and your files are immediately committed to the repository. You typically use this when you have an existing tree of files that you want to begin tracking in your Subversion repository. For example:

```
$ svnadmin create /var/svn/newrepos
$ svn import mytree file:///var/svn/newrepos/some/project \
             -m "Initial import"
Adding         mytree/foo.c
Adding         mytree/bar.c
Adding         mytree/subdir
Adding         mytree/subdir/quux.h

Committed revision 1.
```

在上一个例子里，将会拷贝目录`mytree`到版本库的`some/project`下：

```
$ svn list file:///var/svn/newrepos/some/project
bar.c
foo.c
subdir/
```

注意，在导入之后，原来的目录树并没有转化成工作拷贝，为了开始工作，你还是需要运行**svn checkout**导出一个工作拷贝。

2.2.2. 推荐的版本库布局

While Subversion's flexibility allows you to lay out your repository in any way that you choose, we recommend that you create a `trunk` directory to hold the "main line" of development, a `branches` directory to contain branch copies, and a `tags` directory to contain tag copies. For example:

```
$ svn list file:///var/svn/repos
/trunk
/branches
/tags
```

你将会在第 4 章 分支与合并看到标签和分支的详细内容，关于设置多个项目的信息，可以看第 4.7.1 节 "版本库布局"和第 5.2.1 节 "规划你的版本库结构"中关于"项目根目录"的内容。

2.3. 初始化检出

大多数时候，你会使用*checkout*从版本库取出一个新拷贝开始使用Subversion，这样会在本机创建一个项目的"本地拷贝"，这个拷贝包括了命令行指定版本库中的HEAD(最新的)版本：

```
$ svn checkout http://svn.collab.net/repos/svn/trunk
A    trunk/Makefile.in
A    trunk/ac-helpers
A    trunk/ac-helpers/install.sh
A    trunk/ac-helpers/install-sh
A    trunk/build.conf
…
Checked out revision 8810.
```

名称中有什么?

Subversion努力控制版本控制下数据的类型，文件的内容和属性值都是按照二进制数据存储和传递，并且第 3.3.1 节 “文件内容类型”给Subversion提示以说明对于特定文件“文本化的”操作是没有意义的，也有一些地方，Subversion对存放的信息有限制。

Subversion internally handles certain bits of data—for example, property names, pathnames, and log messages—as UTF-8-encoded Unicode. This is not to say that all your interactions with Subversion must involve UTF-8, though. As a general rule, Subversion clients will gracefully and transparently handle conversions between UTF-8 and the encoding system in use on your computer, if such a conversion can meaningfully be done (which is the case for most common encodings in use today).

In WebDAV exchanges and older versions of some of Subversion's administrative files, paths are used as XML attribute values, and property names in XML tag names. This means that pathnames can contain only legal XML (1.0) characters, and properties are further limited to ASCII characters. Subversion also prohibits `TAB`, `CR`, and `LF` characters in path names to prevent paths from being broken up in diffs or in the output of commands such as **svn log** or **svn status**.

While it may seem like a lot to remember, in practice these limitations are rarely a problem. As long as your locale settings are compatible with UTF-8 and you don't use control characters in path names, you should have no trouble communicating with Subversion. The command-line client adds an extra bit of help—to create “legally correct” versions for internal use it will automatically escape illegal path characters as needed in URLs that you type.

Although the preceding example checks out the trunk directory, you can just as easily check out any deep subdirectory of a repository by specifying the subdirectory in the checkout URL:

```
$ svn checkout \
      http://svn.collab.net/repos/svn/trunk/subversion/tests/cmdline/
A    cmdline/revert_tests.py
A    cmdline/diff_tests.py
A    cmdline/autoprop_tests.py
A    cmdline/xmltests
A    cmdline/xmltests/svn-test.sh
…
Checked out revision 8810.
```

Since Subversion uses a copy-modify-merge model instead of lock-modify-unlock (see 第 1.2 节 “版本模型”), you can immediately make changes to the files and directories in your working copy. Your working copy is just like any other collection of files and directories on your system. You can edit and change it, move it around, even delete the entire working copy and forget about it.

警告

因为你的工作拷贝“同你系统上的文件和目录没有任何区别”，你可以随意修改文件，但是你必须告诉Subversion你做的其他任何事。例如，你希望拷贝或移动工作

拷贝的一个文件，你应该使用**svn copy**或者 **svn move**而不要使用操作系统的拷贝移动命令，我们会在本章后面详细介绍。

除非你准备好了提交一个新文件或目录，或改变了已存在的，否则没有必要通知Subversion你做了什么。

目录 .svn 中有什么?

工作拷贝中的任何一个目录包括一个名为`.svn`管理区域，通常列表操作不显示这个目录，但它仍然是一个非常重要的目录，无论你做什么？不要删除或是更改这个管理区域的任何东西，Subversion使用它来管理工作拷贝。

If you accidentally remove the `.svn` subdirectory, the easiest way to fix the problem is to remove the entire containing directory (a normal system deletion, not **svn delete**), then run **`svn update`** from a parent directory. The Subversion client will download the directory you've deleted, with a new `.svn` area as well.

因为你可以使用版本库的URL作为唯一参数取出一个工作拷贝，你也可以在版本库URL之后指定一个目录，这样会将你的工作目录放到你的新目录，举个例子：

```
$  svn checkout http://svn.collab.net/repos/svn/trunk subv
A    subv/Makefile.in
A    subv/ac-helpers
A    subv/ac-helpers/install.sh
A    subv/ac-helpers/install-sh
A    subv/build.conf
…
Checked out revision 8810.
```

这样将把你的工作拷贝放到`subv`而不是和前面那样放到`trunk`，如果`subv`不存在，将会自动创建。

2.3.1. 禁用密码缓存

When you perform a Subversion operation that requires you to authenticate, by default Subversion caches your authentication credentials on disk. This is done for convenience so that you don't have to continually reenter your password for future operations. If you're concerned about caching your Subversion passwords, [1] you can disable caching either permanently or on a case-by-case basis.

To disable password caching for a particular one-time command, pass the `--no-auth-cache` option on the command line. To permanently disable caching, you can add the line `store-passwords = no` to your local machine's Subversion configuration file. See 第 3.11.2 节 “客户端凭证缓存” for details.

[1] Of course, you're not terribly worried—first because you know that you can't *really* delete anything from Subversion, and second because your Subversion password isn't the same as any of the other 3 million passwords you have, right? Right?

2.3.2. 认证为其它用户

Since Subversion caches auth credentials by default (both username and password), it conveniently remembers who you were acting as the last time you modified your working copy. But sometimes that's not helpful—particularly if you're working in a shared working copy such as a system configuration directory or a web server document root. In this case, just pass the `--username` option on the command line, and Subversion will attempt to authenticate as that user, prompting you for a password if necessary.

2.4. 基本的工作循环

Subversion has numerous features, options, bells, and whistles, but on a day-to-day basis, odds are that you will use only a few of them. In this section, we'll run through the most common things that you might find yourself doing with Subversion in the course of a day's work.

典型的工作周期是这样的：

1. 更新你的工作拷贝。
   - **svn update**
2. 做出修改
   - **svn add**
   - **svn delete**
   - **svn copy**
   - **svn move**
3. 检验修改
   - **svn status**
   - **svn diff**
4. 可能会取消一些修改
   - **svn revert**
5. 解决冲突(合并别人的修改)
   - **svn update**
   - **svn resolved**
6. 提交你的修改
   - **svn commit**

2.4.1. 更新你的工作副本

When working on a project with a team, you'll want to update your working copy to receive any changes other developers on the project have made since your last update. Use **svn update** to bring your working copy into sync with the latest revision in the repository:

```
$ svn update
U  foo.c
U  bar.c
Updated to revision 2.
```

这种情况下，其他人在你上次更新之后提交了对`foo.c`和`bar.c`的修改，因此Subversion更新你的工作拷贝来引入这些更改。

When the server sends changes to your working copy via **svn update**, a letter code is displayed next to each item to let you know what actions Subversion performed to bring your working copy up to date. To find out what these letters mean, run **`svn help update`**.

2.4.2. 修改你的工作副本

Now you can get to work and make changes in your working copy. It's usually most convenient to decide on a discrete change (or set of changes) to make, such as writing a new feature, fixing a bug, and so on. The Subversion commands that you will use here are **svn add**, **svn delete**, **svn copy**, **svn move**, and **svn mkdir**. However, if you are merely editing files that are already in Subversion, you may not need to use any of these commands until you commit.

You can make two kinds of changes to your working copy: *file changes* and *tree changes*. You don't need to tell Subversion that you intend to change a file; just make your changes using your text editor, word processor, graphics program, or whatever tool you would normally use. Subversion automatically detects which files have been changed, and in addition, it handles binary files just as easily as it handles text files—and just as efficiently, too. For tree changes, you can ask Subversion to “mark” files and directories for scheduled removal, addition, copying, or moving. These changes may take place immediately in your working copy, but no additions or removals will happen in the repository until you commit them.

版本控制符号连接

在非Windows平台，Subversion可以将特殊类型符号链接(或是“symlink”)版本化，一个符号链接是对文件系统中其他对象的透明引用，可以通过对符合链接操作实现对引用对象的读写操作。

当符号链提交到Subversion版本库，Subversion会记住这个文件实际上是一个符号链，也会知道这个符号链指向的“对象”。当这个符号链检出到另一个支持符号链的操作系统上时，Subversion会重新构建文件系统级的符号链接。当然这样不会影响在Windows这类不支持符号链的系统上，在此类系统上，Subversion只会创建一个包含指向对象路径的文本文件，因为这个文件不能在Windows系统上作为符号链使用，所以它也会防止Windows用户作其他Subversion相关的操作。

Here is an overview of the five Subversion subcommands that you'll use most often to make tree changes:

svn add foo
预定将文件、目录或者符号链`foo`添加到版本库，当你下次提交后，`foo`会成为其父目录的一个子对象。注意，如果`foo`是目录，所有foo中的内容也会预定添加进去，如果你只想添加`foo`本身，请使用`--non-recursive (-N)`参数。

svn delete foo
预定将文件、目录或者符号链`foo`从版本库中删除，如果foo是文件，它马上从工作拷贝中删除，如果是目录，不会被删除，但是Subversion准备好删除了，当你提交你的修改，`foo`就会在你的工作拷贝和版本库中被删除。[2]

svn copy foo bar
Create a new item `bar` as a duplicate of `foo` and automatically schedule `bar` for addition. When `bar` is added to the repository on the next commit, its copy history is recorded (as having originally come from `foo`). **svn copy** does not create intermediate directories unless you pass the `--parents` option.

svn move foo bar
This command is exactly the same as running **svn copy foo bar; svn delete foo**. That is, `bar` is scheduled for addition as a copy of `foo`, and `foo` is scheduled for removal. **svn move** does not create intermediate directories unless you pass the `--parents` option.

svn mkdir blort
This command is exactly the same as running **mkdir blort; svn add blort**. That is, a new directory named `blort` is created and scheduled for addition.

不通过工作副本修改版本库

有一些情况下会立刻提交目录树的修改到版本库，这只发生在子命令直接操作URL，而不是工作拷贝路径时。以特定的方式使用**svn mkdir**、**svn copy**、**svn move**和**svn delete**可以针对URL操作(并且不要忘记**svn import**只针对URL操作)。

URL operations behave in this manner because commands that operate on a working copy can use the working copy as a sort of “staging area” to set up your changes before committing them to the repository. Commands that operate on URLs don't have this luxury, so when you operate directly on a URL, any of the aforementioned actions represents an immediate commit.

2.4.3. 检查你的修改

当你完成修改，你需要提交他们到版本库，但是在此之前，检查一下做过什么修改是个好主意，通过提交前的检查，你可以整理一份精确的日志信息，你也可以发现你不小心修改的文件，给了你一次恢复修改的机会。此外，这是一个审查和仔细察看修改的好机会，你可通过命令**svn status**浏览所做的修改，通过**svn diff**检查修改的详细信息。

[2]当然没有任何东西是在版本库里被删除了—只是在版本库的`HEAD`里消失了，你可以通过检出(或者更新你的工作拷贝)你做出删除操作的前一个修订版本来找回所有的东西，详细请见???。

看！没有网络！

You can use the commands **svn status**, **svn diff**, and **svn revert** without any network access even if your repository *is* across the network. This makes it easy to manage your changes-in-progress when you are somewhere without a network connection, such as traveling on an airplane, riding a commuter train, or hacking on the beach. [3]

Subversion does this by keeping private caches of pristine versions of each versioned file inside the `.svn` administrative areas. This allows Subversion to report—and revert—local modifications to those files *without network access*. This cache (called the "text-base") also allows Subversion to send the user's local modifications during a commit to the server as a compressed *delta* (or "difference") against the pristine version. Having this cache is a tremendous benefit—even if you have a fast Internet connection, it's much faster to send only a file's changes rather than the whole file to the server.

Subversion has been optimized to help you with this task, and it is able to do many things without communicating with the repository. In particular, your working copy contains a hidden cached "pristine" copy of each version-controlled file within the `.svn` area. Because of this, Subversion can quickly show you how your working files have changed or even allow you to undo your changes without contacting the repository.

2.4.3.1. 查看你的修改概况

为了浏览修改的内容，你会使用这个**svn status**命令，在所有Subversion命令里，**svn status**可能会是你用的最多的命令。

CVS 用户：控制另类的更新！

你也许使用**cvs update**来看你做了哪些修改，**svn status**会给你所有你做的改变—而不需要访问版本库，并且不会在不知情的情况下与其他用户作的更改比较。

In Subversion, **svn update** does just that—it updates your working copy with any changes committed to the repository since the last time you updated your working copy. You may have to break the habit of using the **update** command to see what local modifications you've made.

If you run **svn status** at the top of your working copy with no arguments, it will detect all file and tree changes you've made. Here are a few examples of the most common status codes that **svn status** can return. (Note that the text following `#` is not actually printed by **svn status**.)

```
?       scratch.c           # file is not under version control
A       stuff/loot/bloo.h   # file is scheduled for addition
C       stuff/loot/lump.c   # file has textual conflicts from an update
D       stuff/fish.c        # file is scheduled for deletion
M       bar.c               # the content in bar.c has local modifications
```

[3] And you don't have a WLAN card. Thought you got us, huh?

在这种格式下，**svn status**打印6列字符，紧跟一些空格，接着是文件或者目录名。第一列告诉一个文件或目录的状态或它的内容，返回代码如下：

`A item`
预定加入到版本库的文件、目录或符号链的`item`。

`C item`
文件`item`发生冲突，在从服务器更新时与工作拷贝(如果更新时没有解决)的本地版本发生交迭，在你提交到版本库前，必须手工的解决冲突。

`D item`
文件、目录或是符号链`item`预定从版本库中删除。

`M item`
文件`item`的内容被修改了。

如果你传递一个路径给**svn status**，它只给你这个项目的信息：

```
$ svn status stuff/fish.c
D      stuff/fish.c
```

svn status也有一个`--verbose(-v)`选项，它可以显示工作拷贝中的所有项目，即使没有改变过的：

```
$ svn status -v
M               44        23    sally     README
                44        30    sally     INSTALL
M               44        20    harry     bar.c
                44        18    ira       stuff
                44        35    harry     stuff/trout.c
D               44        19    ira       stuff/fish.c
                44        21    sally     stuff/things
A                0         ?     ?        stuff/things/bloo.h
                44        36    harry     stuff/things/gloo.c
```

这是**svn status**的“加长形式”，第一列保持相同，第二列显示一个工作版本号，第三和第四列显示最后一次修改的版本号和修改人(这些列不会与我们刚才提到的字符混淆)。

上面所有的**svn status**调用并没有联系版本库，只是与`.svn`中的原始数据进行比较的结果，最后，是`--show-updates(-u)`选项，它将会联系版本库为已经过时的数据添加新信息：

```
$ svn status -u -v
M      *        44        23    sally     README
M               44        20    harry     bar.c
       *        44        35    harry     stuff/trout.c
D               44        19    ira       stuff/fish.c
A                0         ?     ?        stuff/things/bloo.h
```

```
Status against revision:   46
```

Notice the two asterisks: if you were to run **`svn update`** at this point, you would receive changes to `README` and `trout.c`. This tells you some very useful information—you'll need to update and get the server changes on `README` before you commit, or the repository will reject your commit for being out of date (more on this subject later).

关于文件和目录，**svn status**可以比我们的展示显示更多的内容，**svn status**完整的描述可以看svn status。

2.4.3.2. 检查你的本地修改的详情

Another way to examine your changes is with the **svn diff** command. You can find out *exactly* how you've modified things by running **`svn diff`** with no arguments, which prints out file changes in *unified diff format*:

```
$ svn diff
Index: bar.c
===================================================================
--- bar.c (revision 3)
+++ bar.c (working copy)
@@ -1,7 +1,12 @@
+#include <sys/types.h>
+#include <sys/stat.h>
+#include <unistd.h>
+
+#include <stdio.h>

 int main(void) {
-  printf("Sixty-four slices of American Cheese...\n");
+  printf("Sixty-five slices of American Cheese...\n");
 return 0;
 }

Index: README
===================================================================
--- README (revision 3)
+++ README (working copy)
@@ -193,3 +193,4 @@
+Note to self:  pick up laundry.

Index: stuff/fish.c
===================================================================
--- stuff/fish.c (revision 1)
+++ stuff/fish.c (working copy)
-Welcome to the file known as 'fish'.
-Information on fish will be here soon.
```

```
Index: stuff/things/bloo.h
===================================================================
--- stuff/things/bloo.h (revision 8)
+++ stuff/things/bloo.h (working copy)
+Here is a new file to describe
+things about bloo.
```

svn diff命令通过比较你的文件和.svn的“原始”文件来输出信息，预定要增加的文件会显示所有增加的文本，要删除的文件会显示所有要删除的文本。

输出的格式为统一区别格式(unified diff format)，删除的行前面加一个-，添加的行前面有一个+，**svn diff**命令也打印文件名和打补丁需要的信息，所以你可以通过重定向一个区别文件来生成“补丁”：

```
$ svn diff > patchfile
```

You could, for example, email the patch file to another developer for review or testing prior to a commit.

Subversion uses its internal diff engine, which produces unified diff format, by default. If you want diff output in a different format, specify an external diff program using `--diff-cmd` and pass any flags you'd like to it using the `--extensions` (`-x`) option. For example, to see local differences in file `foo.c` in context output format while ignoring case differences, you might run **`svn diff --diff-cmd /usr/bin/diff --extensions '-i' foo.c`**.

2.4.4. 取消本地修改

假定我们在看**svn diff**的输出，你发现对某个文件的所有修改都是错误的，或许你根本不应该修改这个文件，或者是从开头重新修改会更加容易。

这是使用**svn revert**的好机会：

```
$ svn revert README
Reverted 'README'
```

Subversion把文件恢复到未修改的状态，叫做.svn目录的“原始”拷贝，应该知道**svn revert**可以恢复任何预定要做的操作，举个例子，你不再想添加一个文件：

```
$ svn status foo
?      foo

$ svn add foo
A         foo

$ svn revert foo
Reverted 'foo'
```

```
$ svn status foo
?      foo
```

注意

svn revert *item* has exactly the same effect as deleting *item* from your working copy and then running **svn update -r BASE *item***. However, if you're reverting a file, **svn revert** has one very noticeable difference—it doesn't have to communicate with the repository to restore your file.

或许你不小心删除了一个文件：

```
$ svn status README

$ svn delete README
D         README

$ svn revert README
Reverted 'README'

$ svn status README
```

2.4.5. 解决冲突 (合并别人的修改)

We've already seen how **svn status -u** can predict conflicts. Suppose you run **svn update** and some interesting things occur:

```
$ svn update
U  INSTALL
G  README
Conflict discovered in 'bar.c'.
Select: (p) postpone, (df) diff-full, (e) edit,
        (h) help for more options:
```

U和G没必要关心，文件干净的接受了版本库的变化，文件标示为U表明本地没有修改，文件已经根据版本库更新。G标示合并，标示本地已经修改过，与版本库没有重迭的地方，已经合并。

但是下面两行是叫做交互式的冲突解决特性(Subversion 1.5最新的)的一部分，这意味着服务器的变更和你的重叠了，而你有机会解决这个冲突，最常用的选项会显示，但是你可以输入*h*显示全部选项：

```
…
  (p)  postpone     - mark the conflict to be resolved later
  (df) diff-full    - show all changes made to merged file
  (e)  edit         - change merged file in an editor
```

```
  (r)  resolved    - accept merged version of file
  (mf) mine-full   - accept my version of entire file (ignore their change
  (tf) theirs-full - accept their version of entire file (lose my changes)
  (l)  launch      - launch external tool to resolve conflict
  (h)  help        - show this list
```

在我们详细查看每个选项含义之前，让我们简短的回顾一下所有这些选项。

(p)ostpone
　让文件在更新完成之后保持冲突状态。

(d)iff
　使用标准区别格式显示base修订版本和冲突文件本身的区别。

(e)dit
　用你喜欢的编辑器打开冲突的文件，编辑器是环境变量EDITOR设置的。

(r)esolved
　完成文件编辑之后，通知**svn**你已经解决了文件的冲突，它必须接受当前的内容—从本质上讲就是你已经“解决了”冲突。

(m)ine-(f)ull
　丢弃新从服务器接收的变更，并只使用你查看文件的本地修改。

(t)heirs-(f)ull
　丢弃你对查看文件的本地修改，只使用从服务器新接收的变更。

(l)aunch
　启动一个外置程序来执行冲突解决，这需要一些预先的准备。

(h)elp
　显示所有在冲突解决时可能使用的命令。

我们现在会更详细的覆盖这些命令，根据关联功能对其进行分组。

2.4.5.1. 交互式的查看冲突区别

Before deciding how to attack a conflict interactively, odds are that you'd like to see exactly what is in conflict, and the *diff* command (**d**) is what you'll use for this:

```
…
Select: (p) postpone, (df) diff-full, (e) edit,
        (h)elp for more options : d
--- .svn/text-base/sandwich.txt.svn-base      Tue Dec 11 21:33:57 2007
+++ .svn/tmp/tempfile.32.tmp     Tue Dec 11 21:34:33 2007
@@ -1 +1,5 @@
-Just buy a sandwich.
+<<<<<<< .mine
+Go pick up a cheesesteak.
```

```
+=======
+Bring me a taco!
+>>>>>>> .r32
…
```

区别内容的第一行显示的是工作拷贝(BASE修订版本)以前的内容，下一行内容是你的修改，最后的一行内容是你刚从服务器(通常是HEAD修订版本)接收的。手上有了信息，你已经准备好了做下一个动作。

2.4.5.2. 交互式的解决冲突区别

交互式解决冲突有四个不同的方式—两个是允许你选择性的合并和编辑修改，两个是允许你简单的选一个文件的版本并继续。

If you wish to choose some combination of your local changes, you can use the “edit” command (**e**) to manually edit the file with conflict markers in a text editor (determined by the EDITOR environment variable). Editing the file by hand in your favorite text editor is a somewhat low-tech way of remedying conflicts (see 第 2.4.5.4 节 “手工合并冲突” for a walkthrough), so some people like to use fancy graphical merge tools instead.

To use a merge tool, you need to either set the SVN_MERGE environment variable or define the merge-tool-cmd option in your Subversion configuration file (see 第 7.1.3 节 “配置选项” for more details). Subversion will pass four arguments to the merge tool: the BASE revision of the file, the revision of the file received from the server as part of the update, the copy of the file containing your local edits, and the merged copy of the file (which contains conflict markers). If your merge tool is expecting arguments in a different order or format, you'll need to write a wrapper script for Subversion to invoke. After you've edited the file, if you're satisfied with the changes you've made, you can tell Subversion that the edited file is no longer in conflict by using the “resolve” command (r).

If you decide that you don't need to merge any changes, but just want to accept one version of the file or the other, you can either choose your changes (a.k.a. “mine”) by using the “mine-full” command (**mf**) or choose theirs by using the “theirs-full” command (**tf**).

2.4.5.3. 延后解决冲突

This may sound like an appropriate section for avoiding marital disagreements, but it's actually still about Subversion, so read on. If you're doing an update and encounter a conflict that you're not prepared to review or resolve, you can type **p** to postpone resolving a conflict on a file-by-file basis when you run **svn update**. If you're running an update and don't want to resolve any conflicts, you can pass the --non-interactive option to **svn update**, and any file in conflict will be marked with a C automatically.

但是C表示冲突，说明服务器上的改动同你的改动冲突了，你需要在更新完成自己手工去解决。当你选择延后解决冲突，**svn**通常会做三件事来辅助通知你解决冲突。

- Subversion在更新时打印C标记，并且标记这个文件已冲突。
- If Subversion considers the file to be mergeable, it places *conflict markers*—special strings of text that delimit the “sides” of the conflict—into the file to visibly demonstrate the overlapping areas.

(Subversion uses the `svn:mime-type` property to decide whether a file is capable of contextual, line-based merging. See 第 3.3.1 节 “文件内容类型” to learn more.)

- 对于每一个冲突的文件，Subversion放置三个额外的未版本化文件到你的工作拷贝：

  `filename.mine`
  This is your file as it existed in your working copy before you updated your working copy—that is, without conflict markers. This file has only your latest changes in it. (If Subversion considers the file to be unmergeable, the `.mine` file isn't created, since it would be identical to the working file.)

  `filename.r`*`OLDREV`*
  这是你的做更新操作以前的BASE版本文件，就是你在上次更新之后未作更改的版本。

  `filename.r`*`NEWREV`*
  这是你的Subversion客户端从服务器刚刚收到的版本，这个文件对应版本库的HEAD版本。

  这里OLDREV是你的`.svn`目录中的修订版本号，NEWREV是版本库中HEAD的版本号。

For example, Sally makes changes to the file `sandwich.txt`, but does not yet commit those changes. Meanwhile, Harry commits changes to that same file. Sally updates her working copy before committing and she gets a conflict, which she postpones:

```
$ svn update
Conflict discovered in 'sandwich.txt'.
Select: (p) postpone, (df) diff-full, (e) edit,
        (h)elp for more options : p
C  sandwich.txt
Updated to revision 2.
$ ls -1
sandwich.txt
sandwich.txt.mine
sandwich.txt.r1
sandwich.txt.r2
```

At this point, Subversion will *not* allow Sally to commit the file `sandwich.txt` until the three temporary files are removed:

```
$ svn commit -m "Add a few more things"
svn: Commit failed (details follow):
svn: Aborting commit: '/home/sally/svn-work/sandwich.txt' remains in confl
```

如果你延办一个冲突，你需要在Subversion允许你提交你的修改之前解决冲突，你可以通过**svn resolve**命令和`--accept`选项的多个参数的一个完成。

如果你希望选择上次检出后修改之前的文件版本，选择*`base`*参数。

如果你希望选择只包含你修改的版本，选择*`mine-full`*参数。

如果你希望选择最近从服务器更新的版本(因此会丢弃你的所以编辑)，选择*theirs-full*参数。

然而，如果你希望

svn resolve removes the three temporary files and accepts the version of the file that you specified with the `--accept` option, and Subversion no longer considers the file to be in a state of conflict:

```
$ svn resolve --accept working sandwich.txt
Resolved conflicted state of 'sandwich.txt'
```

2.4.5.4. 手工合并冲突

第一次尝试解决冲突让人感觉很害怕，但经过一点训练，它简单的像是骑着车子下坡。

Here's an example. Due to a miscommunication, you and Sally, your collaborator, both edit the file `sandwich.txt` at the same time. Sally commits her changes, and when you go to update your working copy, you get a conflict and you're going to have to edit `sandwich.txt` to resolve the conflict. First, let's take a look at the file:

```
$ cat sandwich.txt
Top piece of bread
Mayonnaise
Lettuce
Tomato
Provolone
<<<<<<< .mine
Salami
Mortadella
Prosciutto
=======
Sauerkraut
Grilled Chicken
>>>>>>> .r2
Creole Mustard
Bottom piece of bread
```

The strings of less-than signs, equals signs, and greater-than signs are conflict markers and are not part of the actual data in conflict. You generally want to ensure that those are removed from the file before your next commit. The text between the first two sets of markers is composed of the changes you made in the conflicting area:

```
<<<<<<< .mine
Salami
Mortadella
Prosciutto
=======
```

后两组之间的是Sally提交的修改冲突：

```
=======
Sauerkraut
Grilled Chicken
>>>>>>> .r2
```

Usually you won't want to just delete the conflict markers and Sally's changes—she's going to be awfully surprised when the sandwich arrives and it's not what she wanted. This is where you pick up the phone or walk across the office and explain to Sally that you can't get sauerkraut from an Italian deli. [4] Once you've agreed on the changes you will commit, edit your file and remove the conflict markers:

```
Top piece of bread
Mayonnaise
Lettuce
Tomato
Provolone
Salami
Mortadella
Prosciutto
Creole Mustard
Bottom piece of bread
```

Now use **svn resolve**, and you're ready to commit your changes:

```
$ svn resolve --accept working sandwich.txt
Resolved conflicted state of 'sandwich.txt'
$ svn commit -m "Go ahead and use my sandwich, discarding Sally's edits."
```

Note that **svn resolve**, unlike most of the other commands we deal with in this chapter, requires that you explicitly list any filenames that you wish to resolve. In any case, you want to be careful and use **svn resolve** only when you're certain that you've fixed the conflict in your file—once the temporary files are removed, Subversion will let you commit the file even if it still contains conflict markers.

记住，如果你修改冲突时感到混乱，你可以参考subversion生成的三个文件—包括你未作更新的文件。你也可以使用三方交互合并工具检验这三个文件。

2.4.5.5. 丢弃你的修改而接收新获取的修订版本

If you get a conflict and decide that you want to throw out your changes, you can run **svn resolve --accept theirs-full *CONFLICTED-PATH*** and Subversion will discard your edits and remove the temporary files:

```
$ svn update
```

[4]如果你向他们询问，他们非常有理由把你带到城外的铁轨上。

```
Conflict discovered in 'sandwich.txt'.
Select: (p) postpone, (df) diff-full, (e) edit,
        (h) help for more options: p
C    sandwich.txt
Updated to revision 2.
$ ls sandwich.*
sandwich.txt  sandwich.txt.mine  sandwich.txt.r2  sandwich.txt.r1
$ svn resolve --accept theirs-full sandwich.txt
Resolved conflicted state of 'sandwich.txt'
```

2.4.5.6. Punting: Using svn revert

If you decide that you want to throw out your changes and start your edits again (whether this occurs after a conflict or anytime), just revert your changes:

```
$ svn revert sandwich.txt
Reverted 'sandwich.txt'
$ ls sandwich.*
sandwich.txt
```

Note that when you revert a conflicted file, you don't have to use **svn resolve**.

2.4.6. 提交你的修改

最后！你的修改结束了，你合并了服务器上所有的修改，你准备好提交修改到版本库。

The **svn commit** command sends all of your changes to the repository. When you commit a change, you need to supply a *log message* describing your change. Your log message will be attached to the new revision you create. If your log message is brief, you may wish to supply it on the command line using the `--message` (or `-m`) option:

```
$ svn commit -m "Corrected number of cheese slices."
Sending        sandwich.txt
Transmitting file data .
Committed revision 3.
```

然而，如果你把写日志信息当作工作的一部分，你也许会希望告诉Subversion通过一个文件名得到日志信息，使用`--file`(`-F`)选项：

```
$ svn commit -F logmsg
Sending        sandwich.txt
Transmitting file data .
Committed revision 4.
```

If you fail to specify either the `--message` or `--file` option, Subversion will automatically launch your favorite editor (see the information on `editor-cmd` in 第 7.1.3.2 节 “配置”) for composing a log message.

提示

If you're in your editor writing a commit message and decide that you want to cancel your commit, you can just quit your editor without saving changes. If you've already saved your commit message, simply delete the text, save again, and then abort:

```
$ svn commit
Waiting for Emacs...Done

Log message unchanged or not specified
(a)bort, (c)ontinue, (e)dit
a
$
```

The repository doesn't know or care whether your changes make any sense as a whole; it checks only to make sure nobody else has changed any of the same files that you did when you weren't looking. If somebody *has* done that, the entire commit will fail with a message informing you that one or more of your files are out of date:

```
$ svn commit -m "Add another rule"
Sending        rules.txt
svn: Commit failed (details follow):
svn: File '/sandwich.txt' is out of date
…
```

(错误信息的精确措辞依赖于网络协议和你使用的服务器，但对于所有的情况，其思想完全一样。)

At this point, you need to run **`svn update`**, deal with any merges or conflicts that result, and attempt your commit again.

That covers the basic work cycle for using Subversion. Subversion offers many other features that you can use to manage your repository and working copy, but most of your day-to-day use of Subversion will involve only the commands that we've discussed so far in this chapter. We will, however, cover a few more commands that you'll use fairly often.

2.5. 检验历史

你的版本库就像是一台时间机器，它记录了所有提交的修改，允许你检查文件或目录以及相关元数据的历史。通过一个Subversion命令你可以根据时间或修订号取出一个过去的版本(或者恢复现在的工作拷贝)，然而，有时候我们只是想看看历史而不想回到历史。

Several commands can provide you with historical data from the repository:

svn log
: Shows you broad information: log messages with date and author information attached to revisions and which paths changed in each revision

svn diff
Shows line-level details of a particular change

svn cat
Retrieves a file as it existed in a particular revision number and displays it on your screen

svn list
Displays the files in a directory for any given revision

2.5.1. 产生历史修改列表

To find information about the history of a file or directory, use the **svn log** command. **svn log** will provide you with a record of who made changes to a file or directory, at what revision it changed, the time and date of that revision, and—if it was provided—the log message that accompanied the commit:

```
$ svn log
------------------------------------------------------------------------
r3 | sally | 2008-05-15 23:09:28 -0500 (Thu, 15 May 2008) | 1 line

Added include lines and corrected # of cheese slices.
------------------------------------------------------------------------
r2 | harry | 2008-05-14 18:43:15 -0500 (Wed, 14 May 2008) | 1 line

Added main() methods.
------------------------------------------------------------------------
r1 | sally | 2008-05-10 19:50:31 -0500 (Sat, 10 May 2008) | 1 line

Initial import
------------------------------------------------------------------------
```

注意日志信息缺省根据时间逆序排列，如果希望察看特定顺序的一段修订版本或者单一版本，使用`--revision` (`-r`)选项：

```
$ svn log -r 5:19    # shows logs 5 through 19 in chronological order

$ svn log -r 19:5    # shows logs 5 through 19 in reverse order

$ svn log -r 8       # shows log for revision 8
```

你也可以检查单个文件或目录的日志历史，举个例子：

```
$ svn log foo.c
…
$ svn log http://foo.com/svn/trunk/code/foo.c
…
```

这样只会显示这个工作文件(或者URL)做过修订的版本的日志信息。

为什么 svn log 不会显示我刚刚提交的内容?

If you make a commit and immediately type **svn log** with no arguments, you may notice that your most recent commit doesn't show up in the list of log messages. This is due to a combination of the behavior of **svn commit** and the default behavior of **svn log**. First, when you commit changes to the repository, **svn** bumps only the revision of files (and directories) that it commits, so usually the parent directory remains at the older revision (See 第 1.3.5.1 节 “更新和提交是分开的” for an explanation of why). **svn log** then defaults to fetching the history of the directory at its current revision, and thus you don't see the newly committed changes. The solution here is to either update your working copy or explicitly provide a revision number to **svn log** by using the `--revision` (`-r`) option.

如果你希望得到目录和文件更多的信息，你可以对**svn log**命令使用`--verbose` (`-v`)开关，因为Subversion允许移动和复制文件和目录，所以跟踪路径修改非常重要，在详细模式下，**svn log** 输出中会包括一个路径修改的历史：

```
$ svn log -r 8 -v
------------------------------------------------------------------------
r8 | sally | 2008-05-21 13:19:25 -0500 (Wed, 21 May 2008) | 1 line
Changed paths:
   M /trunk/code/foo.c
   M /trunk/code/bar.h
   A /trunk/code/doc/README

Frozzled the sub-space winch.

------------------------------------------------------------------------
```

svn log也有一个`--quiet` (`-q`)选项，会禁止日志信息的主要部分，当与`--verbose`结合使用，仅会显示修改的文件名。

为什么 svn log 给我一个空的回应?

当使用Subversion一些时间后，许多用户会遇到这种情况：

```
$ svn log -r 2
------------------------------------------------------------------------
$
```

At first glance, this seems like an error. But recall that while revisions are repository-wide, **svn log** operates on a path in the repository. If you supply no path, Subversion uses the current working directory as the default target. As a result, if you're operating in a subdirectory of your working copy and attempt to see the log of a revision in which neither that directory nor any of its children was changed, Subversion will show you an empty log. If you want to see what changed in that revision, try pointing **svn log** directly at the topmost URL of your repository, as in **`svn log -r 2 http://svn.collab.net/repos/svn`**.

2.5.2. 检查历史修改详情

We've already seen **svn diff** before—it displays file differences in unified diff format; we used it to show the local modifications made to our working copy before committing to the repository.

事实上，**svn diff**有三种不同的用法：

- 检查本地修改
- 比较工作拷贝与版本库
- Comparing repository revisions

2.5.2.1. 检查本地修改

As we've seen, invoking **`svn diff`** with no options will compare your working files to the cached "pristine" copies in the `.svn` area:

```
$ svn diff
Index: rules.txt
===================================================================
--- rules.txt (revision 3)
+++ rules.txt (working copy)
@@ -1,4 +1,5 @@
 Be kind to others
 Freedom = Responsibility
 Everything in moderation
-Chew with your mouth open
+Chew with your mouth closed
+Listen when others are speaking
$
```

2.5.2.2. 比较工作副本和版本库

If a single `--revision` (`-r`) number is passed, your working copy is compared to the specified revision in the repository:

```
$ svn diff -r 3 rules.txt
Index: rules.txt
===================================================================
--- rules.txt (revision 3)
+++ rules.txt (working copy)
@@ -1,4 +1,5 @@
 Be kind to others
 Freedom = Responsibility
```

```
 Everything in moderation
-Chew with your mouth open
+Chew with your mouth closed
+Listen when others are speaking
$
```

2.5.2.3. Comparing repository revisions

If two revision numbers, separated by a colon, are passed via `--revision` (`-r`), the two revisions are directly compared:

```
$ svn diff -r 2:3 rules.txt
Index: rules.txt
===================================================================
--- rules.txt (revision 2)
+++ rules.txt (revision 3)
@@ -1,4 +1,4 @@
 Be kind to others
-Freedom = Chocolate Ice Cream
+Freedom = Responsibility
 Everything in moderation
 Chew with your mouth open
$
```

A more convenient way of comparing one revision to the previous revision is to use the `--change` (`-c`) option:

```
$ svn diff -c 3 rules.txt
Index: rules.txt
===================================================================
--- rules.txt (revision 2)
+++ rules.txt (revision 3)
@@ -1,4 +1,4 @@
 Be kind to others
-Freedom = Chocolate Ice Cream
+Freedom = Responsibility
 Everything in moderation
 Chew with your mouth open
$
```

最后，即使你在本机没有工作拷贝，还是可以比较版本库的修订版本，只需要在命令行中输入合适的URL：

```
$ svn diff -c 5 http://svn.example.com/repos/example/trunk/text/rules.txt
…
$
```

2.5.3. 浏览版本库

通过**svn cat**和**svn list**，你可以在未修改工作修订版本的情况下查看文件和目录的内容，实际上，你甚至也不需要有一个工作拷贝。

2.5.3.1. svn cat

如果你只是希望检查一个过去的版本而不希望察看它们的区别，使用**svn cat**：

```
$ svn cat -r 2 rules.txt
Be kind to others
Freedom = Chocolate Ice Cream
Everything in moderation
Chew with your mouth open
$
```

你可以重定向输出到一个文件：

```
$ svn cat -r 2 rules.txt > rules.txt.v2
$
```

2.5.3.2. svn list

svn list可以在不下载文件到本地目录的情况下来察看目录中的文件：

```
$ svn list http://svn.collab.net/repos/svn
README
branches/
clients/
tags/
trunk/
```

如果你希望察看详细信息，你可以使用`--verbose`(`-v`) 参数：

```
$ svn list -v http://svn.collab.net/repos/svn
  20620 harry            1084 Jul 13  2006 README
  23339 harry                 Feb 04 01:40 branches/
  21282 sally                 Aug 27 09:41 developer-resources/
  23198 harry                 Jan 23 17:17 tags/
  23351 sally                 Feb 05 13:26 trunk/
```

这些列告诉你文件和目录最后修改的修订版本、做出修改的用户、如果是文件还会有文件的大小，最后是修改日期和项目的名字。

警告

The **svn list** command with no arguments defaults to the *repository URL* of the current working directory, *not* the local working copy directory. After all, if you want a listing of your local directory, you could use just plain **ls** (or any reasonable non-Unixy equivalent).

2.5.4. 获得旧的版本库快照

除了以上的命令，你可以使用带参数`--revision`的**svn update**和**svn checkout**来使整个工作拷贝“回到过去”[5]：

```
$ svn checkout -r 1729 # Checks out a new working copy at r1729
…
$ svn update -r 1729 # Updates an existing working copy to r1729
…
```

提示

许多Subversion新手使用前面的**svn update**实例来“回退”修改，但是你不能提交修改，你获得有新修订版本的过时工作拷贝也是没有用的。关于如何“回退”，我们可以看第 4.3.5 节 “找回删除的项目”。

Lastly, if you're building a release and wish to bundle up your files from Subversion but don't want those pesky `.svn` directories in the way, you can use **svn export** to create a local copy of all or part of your repository sans `.svn` directories. As with **svn update** and **svn checkout**, you can also pass the `--revision` option to **svn export**:

```
$ svn export http://svn.example.com/svn/repos1 # Exports latest revision
…
$ svn export http://svn.example.com/svn/repos1 -r 1729
# Exports revision r1729
…
```

2.6. 有时你只需要清理

现在我们已经覆盖了使用Subversion的日常任务，我们会检阅一些工作拷贝相关的管理任务。

2.6.1. 处理你的工作副本

Subversion doesn't track either the state or the existence of working copies on the server, so there's no server overhead to keeping working copies around. Likewise, there's no need to let the server know that you're going to delete a working copy.

如果你还是喜欢使用工作拷贝，直到你再次使用它之前，把其保留在磁盘没有任何错误，任何时候一个**svn update**命令可以让使用的文件成为最新。

[5]看到了吧？我们说过Subversion是一个时间机器。

However, if you're definitely not going to use a working copy again, you can safely delete the entire thing, but you'd be well served to take a look through the working copy for unversioned files. To find these files, run **`svn status`** and review any files that are prefixed with a `?` to make certain that they're not of importance. After you're done reviewing, you can safely delete your working copy.

2.6.2. 从中断中恢复

When Subversion modifies your working copy (or any information within `.svn`), it tries to do so as safely as possible. Before changing the working copy, Subversion writes its intentions to a logfile. Next, it executes the commands in the logfile to apply the requested change, holding a lock on the relevant part of the working copy while it works—to prevent other Subversion clients from accessing the working copy mid-change. Finally, Subversion removes the logfile. Architecturally, this is similar to a journaled filesystem. If a Subversion operation is interrupted (e.g, if the process is killed or if the machine crashes), the logfiles remain on disk. By reexecuting the logfiles, Subversion can complete the previously started operation, and your working copy can get itself back into a consistent state.

And this is exactly what **svn cleanup** does: it searches your working copy and runs any leftover logs, removing working copy locks in the process. If Subversion ever tells you that some part of your working copy is "locked," this is the command that you should run. Also, **svn status** will display an `L` next to locked items:

```
$ svn status
  L     somedir
M       somedir/foo.c

$ svn cleanup
$ svn status
M       somedir/foo.c
```

Don't confuse these working copy locks with the ordinary locks that Subversion users create when using the lock-modify-unlock model of concurrent version control; see the sidebar 锁定的三种含义 for clarification.

2.7. 总结

Now we've covered most of the Subversion client commands. Notable exceptions are those dealing with branching and merging (see 第 4 章 分支与合并) and properties (see 第 3.2 节 "属性"). However, you may want to take a moment to skim through 第 9 章 *Subversion* 完全参考 to get an idea of all the different commands that Subversion has—and how you can use them to make your work easier.

第 3 章 高级主题

If you've been reading this book chapter by chapter, from start to finish, you should by now have acquired enough knowledge to use the Subversion client to perform the most common version control operations. You understand how to check out a working copy from a Subversion repository. You are comfortable with submitting and receiving changes using the **svn commit** and **svn update** operations. You've probably even developed a reflex that causes you to run the **svn status** command almost unconsciously. For all intents and purposes, you are ready to use Subversion in a typical environment.

但是Subversion的特性并没有止于“普通的版本控制操作”，它也有一些超越了与版本库传递文件和目录修改以外的功能。

本章重点介绍了一些很重要但不是经常使用的Subversion特性，本章假定你熟悉Subversion对文件和目录的基本版本操作能力，如果你还没有阅读这些内容，或者是需要一个复习，我们建议你重读第 1 章 基本概念和第 2 章 基本使用，一旦你已经掌握了基础知识和本章的内容，你会变成Subversion的超级用户！

3.1. 版本清单

就像你在第 1.3.3 节 “修订版本”见到的，Subversion的修订版本号码非常直接—就是随提交增大的整数。尽管如此，不会花很长时间你就会忘记每个修订版本的修改，但幸运的是，典型的Subvesion工作流程中一般不会要求你提供任意的修订版本号。在需要输入修订版本号时，通常或者是你在一个提交邮件中看到了一个修订版本，或者是在其他Subversion命令的输出结果中，或者是任何上下文环境得到某个版本号码的情况下。

但是有时候，你需要精确指定一个时间，而无法记住或者记录了某个版本，这时除了使用修订版本号码，**svn**允许使用其他形式来指定修订版本—修订版本关键字和修订版本日期。

注意

当用来指定修订版本范围时，不同形式的Subversion修订版本可以混合匹配。例如，你可以*REV1*是修订版本关键字，*REV2*是修订版本号，或者是*REV1*是日期，而*REV2*是修订版本关键字，等等。不同的修订版本指定符是等价的，所以你可以在冒号两边任意使用。

3.1.1. 修订版本关键字

Subversion客户端可以理解一些修订版本关键字，这些关键字可以用来代替`--revision`(`-r`)的数字参数，这会被Subversion解释到特定修订版本号：

`HEAD`
: 版本库中最新的(或者是“最年轻的”)版本。

`BASE`
: 工作拷贝中一个条目的修订版本号，如果这个版本在本地修改了，则这里指的是这个条目在本地未修改的版本。

COMMITTED
项目最近修改的修订版本，与BASE相同或更早。

PREV
The revision immediately *before* the last revision in which an item changed. Technically, this boils down to COMMITTED−1.

因为可以从描述中得到，关键字PREV，BASE和COMMITTED只在引用工作拷贝路径时使用，而不能用于版本库URL，而关键字HEAD则可以用于两种路径类型。

下面是一些修订版本关键字的例子：

```
$ svn diff -r PREV:COMMITTED foo.c
# shows the last change committed to foo.c

$ svn log -r HEAD
# shows log message for the latest repository commit

$ svn diff -r HEAD
# compares your working copy (with all of its local changes) to the
# latest version of that tree in the repository

$ svn diff -r BASE:HEAD foo.c
# compares the unmodified version of foo.c with the latest version of
# foo.c in the repository

$ svn log -r BASE:HEAD
# shows all commit logs for the current versioned directory since you
# last updated

$ svn update -r PREV foo.c
# rewinds the last change on foo.c, decreasing foo.c's working revision

$ svn diff -r BASE:14 foo.c
# compares the unmodified version of foo.c with the way foo.c looked
# in revision 14
```

3.1.2. 版本日期

在版本控制系统以外，修订版本号码是没有意义的，但是有时候你需要将时间和历史修订版本号关联。为此，--revision(-r)选项接受使用花括号({和})包裹的日期输入，Subversion支持标准ISO-8601日期和时间格式，也支持一些其他的。下面是一些例子。(记住使用引号括起所有包含空格的日期。)

```
$ svn checkout -r {2006-02-17}
$ svn checkout -r {15:30}
$ svn checkout -r {15:30:00.200000}
```

```
$ svn checkout -r {"2006-02-17 15:30"}
$ svn checkout -r {"2006-02-17 15:30 +0230"}
$ svn checkout -r {2006-02-17T15:30}
$ svn checkout -r {2006-02-17T15:30Z}
$ svn checkout -r {2006-02-17T15:30-04:00}
$ svn checkout -r {20060217T1530}
$ svn checkout -r {20060217T1530Z}
$ svn checkout -r {20060217T1530-0500}
…
```

当你指定一个日期，Subversion会在版本库找到接近这个日期的最近版本，并且对这个版本继续操作：

```
$ svn log -r {2006-11-28}
------------------------------------------------------------------------
r12 | ira | 2006-11-27 12:31:51 -0600 (Mon, 27 Nov 2006) | 6 lines
…
```

Subversion 会早一天吗?

如果你只是指定了日期而没有时间(举个例子2006-11-27)，你也许会以为Subversion会给你11-27号最后的版本，相反，你会得到一个26号版本，甚至更早。记住Subversion会根据你的日期找到最新的版本，如果你给一个日期，而没有给时间，像2006-11-27，Subversion会假定时间是00:00:00，所以在27号找不到任何版本。

如果你希望查询包括27号，你既可以使用({"2006-11-27 23:59"})，或是直接使用第二天({2006-11-28})。

你可以使用时间段，Subversion会找到这段时间的所有版本：

```
$ svn log -r {2006-11-20}:{2006-11-29}
…
```

警告

因为一个版本的时间戳是作为一个属性存储的—不是版本化的，而是可以编辑的属性(见第 3.2 节 “属性”)—版本号的时间戳可以被修改，从而建立一个虚假的年代表，也可以被完全删除。Subversion正确转化修订版本日期到修订版本的能力依赖于修订版本时间戳顺序排列—修订版本越年轻，则时间戳越年轻。如果顺序没有被维护，你会发现使用日期指定修订版本不会返回你期望的数据。

3.2. 属性

我们已经详细讲述了Subversion存储和检索版本库中不同版本的文件和目录的细节，并且用了好几个章节来论述这个工具的基本功能。如果对于版本化的支持到此为止，从版本控制的角度来看Subversion已经完整了。

但不仅仅如此。

In addition to versioning your directories and files, Subversion provides interfaces for adding, modifying, and removing versioned metadata on each of your versioned directories and files. We refer to this metadata as *properties*, and they can be thought of as two-column tables that map property names to arbitrary values attached to each item in your working copy. Generally speaking, the names and values of the properties can be whatever you want them to be, with the constraint that the names must contain only ASCII characters. And the best part about these properties is that they, too, are versioned, just like the textual contents of your files. You can modify, commit, and revert property changes as easily as you can file content changes. And the sending and receiving of property changes occurs as part of your typical commit and update operations—you don't have to change your basic processes to accommodate them.

注意

Subversion自己保留了一组名称以`svn:`开头的属性，现在已经有了一些在用的属性，所以在你根据需要创建自定义属性时，需要避免这些前缀开头的名称，否则，Subversion的新版本可能会采用同名的属性来满足新的特性，而其含义可能会完全不同。

Subversion的属性也可以在别的地方出现，就像文件和目录可能附加有任意的属性名和值，每个修订版本作为一个整体也可以附加任意的属性，也有同样的限制—可读的文本名称和任何你希望的二进制值，主要的区别是修订版本属性不是版本化的，换句话说，如果你修改，删除一个修订版本属性，在Subversion领域内没有办法恢复到以前的值。

Subversion不关心如何使用属性，但是要求你不要使用`svn:`为前缀的属性名，这是Subversion自己使用的命名空间，Subversion使用了版本化的和未版本化的属性。文件和目录上的特定版本化属性都有特别的意义或效果，或者是提供了修订版本的一些信息。一些修订版本属性会在提交时自动附加到修订版本上，包含了修订版本的信息。大多数这些属性会作为普通的主题在后面提及，关于Subversion预定义的属性的详细列表可以看第 9.10 节 “Subversion 属性”。

在本小节，我们将会检验这个工具—不仅是对Subversion的用户，也对Subversion本身—对于属性的支持。你会学到与属性相关的**svn**子命令，和属性怎样影响你的普通Subversion工作流。

3.2.1. 为什么需要属性?

就像Subversion使用属性保存其包含的文件、目录和修订版本的附加信息，你也会发现属性有一些类似的使用，你会发现如果在数据附近有个地方保存自定义元数据会非常有用。

假设你希望设计一个存放许多数码照片的网站，会显示标题和缩略图。现在你的图片会经常修改，所以你希望能够让这个站点尽量自动处理这些事情，这些照片会很大，所以作为网站，你希望为访问者提供相似的缩略图。

现在，你可以利用这些功能使用传统文件。你可以有一个`image123.jpg`和一个对应的`image123-thumbnail.jpg`在同一个目录里，有时候你希望保持文件名相同，你可以使用不同的目录，如`thumbnails/image123.jpg`。你可以用一种相似的样式来保存你的标

题和时间戳，同原始图像文件分开。每个新图片的添加都会成倍的增加混乱，很快你的目录树会是一团糟。

现在考虑使用Subversion文件的属性的方式来管理这个站点，想象我们有一个单独的图像文件`image123.jpg`，然后这个文件的属性集包括`caption`、`datestamp`甚至`thumbnail`。现在你的工作拷贝目录看起来更容易管理—实际上，它看起来只有图像文件，但是你的自动化脚本知道得更多，它们知道可以用**svn**(更好的选择是使用Subversion的语言绑定—见第 8.3 节 “使用 API”)来挖掘更多的站点显示需要的额外信息，而不必去阅读一个索引文件或者是玩一个路径处理的游戏。

注意

While Subversion places few restrictions on the names and values you use for properties, it has not been designed to optimally carry large property values or large sets of properties on a given file or directory. Subversion commonly holds all the property names and values associated with a single item in memory at the same time, which can cause detrimental performance or failed operations when extremely large property sets are used.

自定义修订版本属性也经常被使用，一个常见的用法是一个包含问题跟踪ID的属性，可能是因为这个修改修正了这个ID的问题。另外一些人用属性来存放更容易记的修订版本名称—记住修订版本1935是一个完全测试的版本是很困难的，但是如果在修订版本上设置一个值为`all passing`的`test-results`属性，这就有了一个有用的信息。

可搜索性 (或者，为什么不使用属性)

For all their utility, Subversion properties—or, more accurately, the available interfaces to them—have a major shortcoming: while it is a simple matter to *set* a custom property, *finding* that property later is a whole different ball of wax.

为了定位一个自定义属性通常要线形访问版本库的所有修订版本，向每个修订版本询问，“你们有我找的属性吗？”尝试查找自定义版本化属性也是同样的痛苦，通常需要在整个工作拷贝递归调用**svn propget**。在你的情况下，可能不会比遍历所有修订版本差。但也在性能和成功可能性里留下了许多悬念，特别是当你需要从版本库的根开始搜索时。

因为这个原因，你会选择—特别是在修订版本属性用例—简单的添加你的元数据到修订版本日志信息，使用一些政策驱动(并且是编程强制的)且可以通过**svn log**快速解析的格式。如下的Subversion日志信息会很常见：

```
Issue(s): IZ2376, IZ1919
Reviewed by:  sally

This fixes a nasty segfault in the wort frabbing process
…
```

但是现在依然有一些不幸，Subversion不支持日志信息模版机制，虽然这样对用户与日志嵌入的修订版本元数据保持一致有很大帮助。

3.2.2. 操作属性

The **svn** program affords a few ways to add or modify file and directory properties. For properties with short, human-readable values, perhaps the simplest way to add a new property is to specify the property name and value on the command line of the **svn propset** subcommand:

```
$ svn propset copyright '(c) 2006 Red-Bean Software' calc/button.c
property 'copyright' set on 'calc/button.c'
$
```

But we've been touting the flexibility that Subversion offers for your property values. And if you are planning to have a multiline textual, or even binary, property value, you probably do not want to supply that value on the command line. So the **svn propset** subcommand takes a `--file` (`-F`) option for specifying the name of a file that contains the new property value.

```
$ svn propset license -F /path/to/LICENSE calc/button.c
property 'license' set on 'calc/button.c'
$
```

对于属性名称也有一些限制，属性名必须以一个字符、一个冒号(:)或下划线(_)开始，之后你可以使用数字，横线(-)和句号(.)。[1]

作为**propset**命令的补充，**svn**提供了一个**propedit**命令，这个命令使用定制的编辑器程序(见第 7.1.3.2 节 “配置”)来添加和修改属性。当你运行这个命令，**svn**调用你的编辑器程序打开一个临时文件，文件中保存当前的属性值(或者是空文件，如果你正在添加新的属性)。然后你只需要修改为你想要的值，保存临时文件，然后离开编辑器程序。如果Subversion发现你已经修改了属性值，就会接受新值，如果你未作任何修改而离开，不会产生属性修改操作：

```
$ svn propedit copyright calc/button.c  ### exit the editor without change
No changes to property 'copyright' on 'calc/button.c'
$
```

我们也应该注意到，像其它**svn**子命令一样，这些关联的属性可以一次添加到多个路径上，这样就可以通过一个命令修改一组文件的属性。例如，我们可以：

```
$ svn propset copyright '(c) 2006 Red-Bean Software' calc/*
property 'copyright' set on 'calc/Makefile'
property 'copyright' set on 'calc/button.c'
property 'copyright' set on 'calc/integer.c'
...
$
```

如果不能方便的得到存储的属性值，那么属性的添加和编辑操作也不会很容易，所以**svn**提供了两个子命令来显示文件和目录存储的属性名和值。**svn proplist**命令会列出路径上存在的所

[1]如果你熟悉XML，其实这就是XML的"Name"语法的ASCII子集。

有属性名称，一旦你知道了某个节点的属性名称，你可以用**svn propget**获取它的值，这个命令获取给定的路径(或者是一组路径)和属性名称，打印这个属性的值到标准输出。

```
$ svn proplist calc/button.c
Properties on 'calc/button.c':
  copyright
  license
$ svn propget copyright calc/button.c
(c) 2006 Red-Bean Software
```

There's even a variation of the **proplist** command that will list both the name and the value for all of the properties. Simply supply the `--verbose` (`-v`) option.

```
$ svn proplist -v calc/button.c
Properties on 'calc/button.c':
  copyright : (c) 2006 Red-Bean Software
  license : ================================================================
Copyright (c) 2006 Red-Bean Software.  All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions, and the recipe for Fitz's famous
red-beans-and-rice.
…
```

The last property-related subcommand is **propdel**. Since Subversion allows you to store properties with empty values, you can't remove a property altogether using **svn propedit** or **svn propset**. For example, this command will *not* yield the desired effect:

```
$ svn propset license '' calc/button.c
property 'license' set on 'calc/button.c'
$ svn proplist -v calc/button.c
Properties on 'calc/button.c':
  copyright : (c) 2006 Red-Bean Software
  license :
$
```

你需要用**propdel**来删除属性，语法与其它与属性命令相似：

```
$ svn propdel license calc/button.c
property 'license' deleted from 'calc/button.c'.
$ svn proplist -v calc/button.c
Properties on 'calc/button.c':
```

```
  copyright : (c) 2006 Red-Bean Software
$
```

Remember those unversioned revision properties? You can modify those, too, using the same **svn** subcommands that we just described. Simply add the `--revprop` command-line parameter and specify the revision whose property you wish to modify. Since revisions are global, you don't need to specify a target path to these property-related commands so long as you are positioned in a working copy of the repository whose revision property you wish to modify. Otherwise, you can simply provide the URL of any path in the repository of interest (including the repository's root URL). For example, you might want to replace the commit log message of an existing revision. [2] If your current working directory is part of a working copy of your repository, you can simply run the **svn propset** command with no target path:

```
$ svn propset svn:log '* button.c: Fix a compiler warning.' -r11 --revprop
property 'svn:log' set on repository revision '11'
$
```

But even if you haven't checked out a working copy from that repository, you can still effect the property change by providing the repository's root URL:

```
$ svn propset svn:log '* button.c: Fix a compiler warning.' -r11 --revprop
              http://svn.example.com/repos/project
property 'svn:log' set on repository revision '11'
$
```

注意，修改这些未版本化的属性的能力一定要明确的添加给版本库管理员(见第 5.4.2 节 “修正提交消息”)。因为属性没有版本化，如果编辑的时候不小心，就会冒丢失信息的风险，版本库管理员可以设置方法来防范这种意外，缺省情况下，修改未版本化的属性是禁止的。

提示

用户必须在可能的情况下使用**svn propedit**，而不是**svn propset**。然而这两个命令的结果是相同的，前一个会允许他们查看修改以前的内容，可以帮助用户验证，实际上，作出他们所期望的修改，当修改未版本化修订版本属性时，这一点特别需要。另外，这个命令也可以通过文本编辑器或命令行轻松的修改多行属性。

3.2.3. 属性和 Subversion 工作流程

现在你已经熟悉了所有与属性相关的**svn**子命令，让我们看看属性修改如何影响Subversion的工作流。我们前面提到过，文件和目录的属性是版本化的，这一点类似于版本化的文件内容。后果之一，就是Subversion具有了同样的机制来合并—用干净或者冲突的方式—其他人的修改应用到你的修改。

就像文件内容，你的属性修改是本地修改，只有使用**svn commit**命令提交后才会保存到版本库中，属性修改也可以很容易的取消—**svn revert**命令会恢复你的文件和目录为编辑前状态，

[2] 修正提交日志信息的拼写错误，文法错误和“简单的错误”是`--revprop`选项最常见用例。

包括内容、属性和其它的信息。另外，你可以使用**svn status**和**svn diff**接受感兴趣的文件和目录属性的状态信息。

```
$ svn status calc/button.c
 M      calc/button.c
$ svn diff calc/button.c
Property changes on: calc/button.c
___________________________________________________________________
Name: copyright
   + (c) 2006 Red-Bean Software

$
```

Notice how the **status** subcommand displays `M` in the second column instead of the first. That is because we have modified the properties on `calc/button.c`, but not its textual contents. Had we changed both, we would have seen `M` in the first column, too. (We cover **svn status** in 第 2.4.3.1 节 “查看你的修改概况”).

属性冲突

与文件内容一样，本地的属性修改也会同别人的提交冲突，如果你更新你的工作拷贝目录并且接收到有资源属性修改与你的修改冲突，Subversion会报告资源处于冲突状态。

```
$ svn update calc
M  calc/Makefile.in
Conflict for property 'linecount' discovered on 'calc/button.c'.
Select: (p) postpone, (df) diff-full, (e) edit,
        (s) show all options: p
 C calc/button.c
Updated to revision 143.
$
```

Subversion也会在冲突资源的同一个目录创建一个`.prej`扩展名的文件，保存冲突的细节。你一定要检查这个文件的内容来决定如何解决冲突，在你解决冲突之前，你会在使用**svn status**时看到这个资源的输出的第二列是一个`C`，提交本地修改的尝试会失败。

```
$ svn status calc
 C     calc/button.c
?      calc/button.c.prej
$ cat calc/button.c.prej
Trying to change property 'linecount' from '1267' to '1301',
but property has been locally changed from '1267' to '1256'.
$
```

为了解决属性冲突，只需要确定冲突的属性保存了它们应该的值，然后使用**svn resolved**命令告诉Subversion你已经手工解决了问题。

You might also have noticed the nonstandard way that Subversion currently displays property differences. You can still use **svn diff** and redirect its output to create a usable patch file. The **patch** program will ignore property patches—as a rule, it ignores any noise it can't understand. This does, unfortunately, mean that to fully apply a patch generated by **svn diff**, any property modifications will need to be applied by hand.

3.2.4. 自动设置属性

Properties are a powerful feature of Subversion, acting as key components of many Subversion features discussed elsewhere in this and other chapters—textual diff and merge support, keyword substitution, newline translation, and so on. But to get the full benefit of properties, they must be set on the right files and directories. Unfortunately, that step can be easily forgotten in the routine of things, especially since failing to set a property doesn't usually result in an obvious error (at least compared to, say, failing to add a file to version control). To help your properties get applied to the places that need them, Subversion provides a couple of simple but useful features.

当你使用**svn add**或是**svn import**准备加入一个版本控制的文件时，Subversion会自动运行一个基本探测来检查文件是包含了可读还是不可读的内容，首先，在支持执行允许位的操作系统，Subversion会自动会为设置执行位的文件设置`svn:executable`属性(更多信息见第 3.3.2 节 “文件的可执行性”)。

Second, Subversion tries to determine the file's MIME type. If you've configured a `mime-types-files` runtime configuration parameter, Subversion will try to find a MIME type mapping in that file for your file's extension. If it finds such a mapping, it will set your file's `svn:mime-type` property to the MIME type it found. If no mapping file is configured, or no mapping for your file's extension could be found, Subversion runs a very basic heuristic to determine whether the file contains nontextual content. If so, it automatically sets the `svn:mime-type` property on that file to `application/octet-stream` (the generic “this is a collection of bytes” MIME type). Of course, if Subversion guesses incorrectly, or if you wish to set the `svn:mime-type` property to something more precise—perhaps `image/png` or `application/x-shockwave-flash`—you can always remove or edit that property. (For more on Subversion's use of MIME types, see 第 3.3.1 节 “文件内容类型” later in this chapter.)

Subversion also provides, via its runtime configuration system (see 第 7.1 节 “运行配置区”), a more flexible automatic property setting feature that allows you to create mappings of filename patterns to property names and values. Once again, these mappings affect adds and imports, and can not only override the default MIME type decision made by Subversion during those operations, but can also set additional Subversion or custom properties, too. For example, you might create a mapping that says that anytime you add JPEG files—ones whose names match the pattern `*.jpg`—Subversion should automatically set the `svn:mime-type` property on those files to `image/jpeg`. Or perhaps any files that match `*.cpp` should have `svn:eol-style` set to `native`, and `svn:keywords` set to `Id`. Automatic property support is perhaps the handiest property-related tool in the Subversion toolbox. See 第 7.1.3.2 节 “配置” for more about configuring that support.

3.3. 文件移植性

幸运的是，对于许多在不同操作系统下工作的用户，Subversion命令行程序的行为方式几乎完全一致，如果你知道在一个平台上如何运行**svn**，你也就学会了在其他平台上运行。

然而，这一点在本软件的其他几类地方或Subversion保持的实际文件并不一定都是正确的。例如，在一个Windows系统，“文本文件”的定义与Linux环境下的类似，但是也有区别—行结束的字符串并不相同。当然也有其他的区别，Unix平台支持(Subversion也支持)符号链；Windows不知吃，Unix使用文件系统执行位来检测可执行性；而Windows使用文件扩展名。

因为Subversion不是要将世界上的所有此类事情统一起来，所以我们最好是尽可能让我们在多个计算机和操作系统上使用版本化文件和目录时能够更简单，本节描述了Subversion是如何做的。

3.3.1. 文件内容类型

Subversion同很多应用一样利用多用途网际邮件扩展(MIME)内容类型，`svn:mime-type`属性为Subversion的许多目的服务，除了保存一个文件的MIME分类以外，这个`svn:mime-type`属性值也描述了一些Subversion自己使用的行为特性。

识别文件类型

大多数现代操作系统通过文件名的扩展名推断文件的类型和格式，例如，以`.txt`为后缀的文件通常被认为是可读的，不需要通过复杂的解码处理就可以被阅读。对于后缀名为`.png`文件，则被认为是Portable Network Graphics类型—不是可读的格式，只能通过识别PNG格式并且可以将信息转化为光栅的软件使用。

Unfortunately, some of those extensions have changed their meanings over time. When personal computers first appeared, a file named `README.DOC` would have almost certainly been a plain-text file, just like today's `.txt` files. But by the mid-1990s, you could almost bet that a file of that name would not be a plain-text file at all, but instead a Microsoft Word document in a proprietary, non-human-readable format. But this change didn't occur overnight—there was certainly a period of confusion for computer users over what exactly they had in hand when they saw a `.DOC` file. [3]

The popularity of computer networking cast still more doubt on the mapping between a file's name and its content. With information being served across networks and generated dynamically by server-side scripts, there was often no real file per se, and therefore no filename. Web servers, for example, needed some other way to tell browsers what they were downloading so that the browser could do something intelligent with that information, whether that was to display the data using a program registered to handle that datatype or to prompt the user for where on the client machine to store the downloaded data.

Eventually, a standard emerged for, among other things, describing the contents of a data stream. In 1996, RFC 2045 was published. It was the first of five RFCs describing MIME. It describes the concept of media types and subtypes and recommends a syntax for the representation of those types. Today, MIME media types—or “MIME types”—are used almost universally across email applications, web servers, and other software as the de facto mechanism for clearing up the file content confusion.

举个例子，一个好处就是Subversion在更新时通常可以提供基于上下文的行为基础的合并，如果一个文件`svn:mime-type`属性设置为非文本的MIME类型(通常是那些不是`text/`开头的

[3]你认为那样过于粗狂？在同一个时代里，WordPerfect也使用`.DOC`作为它们私有文件格式的扩展名!

类型，但也有例外)，Subversion会假定这个文件保存了二进制内容—也就是不可读的数据。一个好处就是Subversion通常在更新工作拷贝时提供了一个前后相关的以行为基础的修改合并，但是对于二进制数据文件，没有“行”的概念，所以对这些文件，Subversion不会在更新时尝试执行合并操作，相反，任何时候你在本地已经修改的一个二进制文件有了更新，你的文件扩展名会修改为.orig，然后Subversion保存一个新的工作拷贝文件来保存更新时得到的修改，但原来的文件名已经不是你自己的本地修改。这个行为模式是用来保护用户在对不可文本合并的文件尝试执行文本的合并时失败的情形。

警告

当svn:mime-type属性的值设置为非文本的值时，会根据其它属性导致一些非预期的行为。例如，因为行结束符的概念(因此，行结束符转化)，使得应用到非文本文件时没有意义，Subversion会防止你为文件设置svn:eol-style属性，但是如果你递归执行属性设置，这可能不是很清楚了，因为这时Subversion会默默的略过不适合特定属性的文件。

从Subversion 1.5开始，用户可以配置一个新的mime-types-file运行中配置参数，可以作为MIME类型映射文件的定位，Subversion会根据映射文件来确定新添加和导入文件的MIME类型。

另外，如果设置了svn:mime-type属性，Subversion的Apache模块会使用这个值来在HTTP头里输入Content-type:，这给了web浏览器如何显示版本库的一个文件提供了至关重要的线索。

3.3.2. 文件的可执行性

在多数操作系统，执行一个文件或命令的能力是由执行位管理的，这些位缺省是关闭的，必须由用户根据需要显式的指定，但是记住应该为哪些检出的文件设置可执行位会是一件很麻烦的事情，所以Subversion提供了svn:executable这个属性来保持打开执行位，在工作拷贝得到这些文件时设置执行位。

这个属性对于没有可执行权限位的文件系统无效，如FAT32和NTFS。 [4]也就是说，尽管它没有定义的值，在设置这个属性时，Subversion会强制它的值为*，最后，这个属性只对文件有效，目录无效。

3.3.3. 行结束字符序列

Unless otherwise noted using a versioned file's svn:mime-type property, Subversion assumes the file contains human-readable data. Generally speaking, Subversion uses this knowledge only to determine whether contextual difference reports for that file are possible. Otherwise, to Subversion, bytes are bytes.

这意味着缺省情况下，Subversion不会关注任何行结束标记*(end-of-*line，EOL)，不幸的是不同的操作系统在文本文件使用不同的行结束标志，举个例子，Windows平台下的A编辑工具使用一对ASCII控制字符—回车(CR)和一个换行(LF)，而Unix软件，只使用一个LF来表示一个行的结束。

[4]Windows文件系统使用文件扩展名(如.EXE、.BAT和.COM)来标示可执行文件。

并不是所有操作系统的工具准备好了理解与本地行结束样式不一样的行结束格式，一个常见的结果是Unix程序会把Windows文件中的CR当作一个不同的字符(通常表现为^M)，而Windows程序会把Unix文件合并为一个非常大的行，因为没有发现标志行结束的回车加换行(或者是CRLF)字符。

对外来EOL标志的敏感会让在多种操作系统分享文件的人们感到沮丧，例如，考虑有一个源代码文件，开发者会在Windows和Unix系统上编辑这个文件，如果所有的用户使用的工具可以展示文件的行结束，那就没有问题了。

But in practice, many common tools either fail to properly read a file with foreign EOL markers, or convert the file's line endings to the native style when the file is saved. If the former is true for a developer, he has to use an external conversion utility (such as **dos2unix** or its companion, **unix2dos**) to prepare the file for editing. The latter case requires no extra preparation. But both cases result in a file that differs from the original quite literally on every line! Prior to committing his changes, the user has two choices. Either he can use a conversion utility to restore the modified file to the same line-ending style that it was in before his edits were made, or he can simply commit the file—new EOL markers and all.

这个情景的结局看起来像是要浪费时间对提交的文件作不必要的修改，浪费时间是痛苦的，但是如果提交修改了文件的每一行，判断文件修改了哪一行会是一件复杂的工作，bug在哪一行修正的？哪一行导致了语法错误？

这个问题的解决方案是svn:eol-style属性，当这个属性设置为一个正确的值时，Subversion使用它来判断针对行结束样式执行何种特殊的操作，而不会随着多种操作系统的每次提交而发生剧烈变化，正确的值有：

native
: 这会导致保存EOL标志的文件使用Subversion运行的操作系统的本地编码，换句话说，如果一个Windows用户取出一个工作拷贝包含的文件设置native的属性为svn:eol-style，这个文件会使用CRLF的EOL标志，一个Unix用户取出相同的文件会看到他的文件使用LF的EOL标志。

  注意Subversion实际上使用LF的EOL标志，而不会考略操作系统，尽管这对用户来说是透明的。

CRLF
: 这会导致这个文件使用CRLF序列作为EOL标志，不管使用何种操作系统。

LF
: 这会导致文件使用LF字符作为EOL标志，不管使用何种操作系统。

CR
: This causes the file to contain CR characters for EOL markers, regardless of the operating system in use. This line-ending style is not very common.

3.4. 忽略未版本控制的条目

在任何工作拷贝，将版本化文件和目录与没有也不准备版本化的文件分开会是非常常见的情况。文本编辑器的备份文件会将目录搞乱，代码编译过程中生成的中间文件，甚至最终文件

也不是你希望版本化的，用户在见到这些文件和目录(经常是版本控制工作拷贝中)的任何时候都会将他们删除。

期望让Subversion的工作拷贝摆脱混乱保持干净是可笑的，实际上Subversion将工作拷贝是普通目录作为它的一项特性。但是这些没有版本化的文件和目录会给Subversion用户带来一些烦恼，例如，因为**svn add**和**svn import**命令都是会递归执行的，并不知道哪些文件你不希望版本化，很容易意外的添加一些文件。因为**svn status**会报告工作拷贝中包括未版本化文件和目录的信息，如果这种文件很多，它的输出会变得非常嘈杂。

So Subversion provides two ways for telling it which files you would prefer that it simply disregard. One of the ways involves the use of Subversion's runtime configuration system (see 第 7.1 节 “运行配置区”), and therefore applies to all the Subversion operations that make use of that runtime configuration—generally those performed on a particular computer or by a particular user of a computer. The other way makes use of Subversion's directory property support and is more tightly bound to the versioned tree itself, and therefore affects everyone who has a working copy of that tree. Both of the mechanisms use *file patterns* (strings of literal and special wildcard characters used to match against filenames) to decide which files to ignore.

Subversion运行配置系统提供一个`global-ignores`选项，其中的值是空格分开的文件名模式(或glob)。这些模式会应用到可以添加到版本控制的候选者，也就是**svn status**显示出来的未版本化文件。如果文件名与其中的某个模式匹配，Subversion会当这个文件不存在。这个文件模式最好是全局不期望版本化的模式，例如编辑器Emacs的备份文件`*~`和`.*~`。

Subversion 中的文件模式?

文件模式(也叫做*globs*或*shell wildcard patterns*)用来匹配文件名的字符串，通常是用来在无需输入完成文件名的情况下快速选择类似文件的一个子集，这个模式包含两种字符：普通字符，是与潜在匹配的明确比较，和特殊通配符，用来解释不同的匹配目标。

文件模式语法有不同的类型，但是Unix系统实现中Subversion使用最常见的`fnmatch`系统函数，支持如下的通配符，为了你的便利有一些简短的描述：

`?`
: Matches any single character

`*`
: Matches any string of characters, including the empty string

`[`
: Begins a character class definition terminated by `]`, used for matching a subset of characters

你可以在Unix的shell提示符中看到同样的模式匹配，下面是一些不同方式使用模式的例子：

```
$ ls   ### the book sources
appa-quickstart.xml             ch06-server-configuration.xml
appb-svn-for-cvs-users.xml      ch07-customizing-svn.xml
appc-webdav.xml                 ch08-embedding-svn.xml
book.xml                        ch09-reference.xml
ch00-preface.xml                ch10-world-peace-thru-svn.xml
ch01-fundamental-concepts.xml   copyright.xml
ch02-basic-usage.xml            foreword.xml
ch03-advanced-topics.xml        images/
ch04-branching-and-merging.xml  index.xml
ch05-repository-admin.xml       styles.css
$ ls ch*   ### the book chapters
ch00-preface.xml                ch06-server-configuration.xml
ch01-fundamental-concepts.xml   ch07-customizing-svn.xml
ch02-basic-usage.xml            ch08-embedding-svn.xml
ch03-advanced-topics.xml        ch09-reference.xml
ch04-branching-and-merging.xml  ch10-world-peace-thru-svn.xml
ch05-repository-admin.xml
$ ls ch?0-*   ### the book chapters whose numbers end in zero
ch00-preface.xml  ch10-world-peace-thru-svn.xml
$ ls ch0[3578]-*   ### the book chapters that Mike is responsible for
ch03-advanced-topics.xml   ch07-customizing-svn.xml
ch05-repository-admin.xml  ch08-embedding-svn.xml
$
```

文件模式匹配可能比我们这里描述更复杂一点，但是基本的使用水平会适合大多数Subversion的用户。

如果是在版本化目录上发现svn:ignore属性，其内容是一列以行分割的文件模式，Subversion用来判断在这个目录下对象是否被忽略。这些模式不会覆盖在运行配置设置的全局忽略，而是向其添加忽略模式。不像全局忽略选项，在svn:ignore属性中设置的值只会应用到其设置的目录，而不会应用到其子目录。svn:ignore属性是告诉Subversion在每个用户的工作拷贝对应目录忽略相同的文件的好方法，例如编译输出或—使用一个本书相关的例子—本书从DocBook XML文件生成的HTML、PDF或PostScript。

注意

Subversion对于忽略文件模式的支持仅限于将未版本化文件和目录添加到版本控制时，如果一个文件已经在Subversion控制下，忽略模式机制不会再有效果，不要期望Subversion会阻止你提交一个符合忽略条件的修改—Subversion一直认为它是版本化的对象。

CVS 用户的忽略模式

Subversion的svn:ignore属性与CVS的.cvsignore文件的语法和功能非常类似，实际上，如果你移植一个CVS的工作拷贝到Subversion，你可以直接使用.cvsignore作为**svn propset**输入文件参数：

```
$ svn propset svn:ignore -F .cvsignore .
property 'svn:ignore' set on '.'
$
```

但是CVS和Subversion处理忽略模式的方式有一些不同，这两个系统在不同的时候使用忽略模式，忽略模式应用的对象也由微小的不同，另外Subversion不可以使用!模式来去取消忽略模式。

全局忽略模式只是一种个人喜好，可能更接近于用户的特定工具链，而不是特定工作拷贝的需要，所以余下的小节将关注svn:ignore属性和它的使用。

假定你的**svn status**有如下输出：

```
$ svn status calc
 M      calc/button.c
?       calc/calculator
?       calc/data.c
?       calc/debug_log
?       calc/debug_log.1
?       calc/debug_log.2.gz
?       calc/debug_log.3.gz
```

In this example, you have made some property modifications to `button.c`, but in your working copy, you also have some unversioned files: the latest `calculator` program that you've compiled from your source code, a source file named `data.c`, and a set of debugging output logfiles. Now, you know that your build system always results in the `calculator` program being generated. [5] And you know

[5]这不是编译系统的基本功能吗?

that your test suite always leaves those debugging logfiles lying around. These facts are true for all working copies of this project, not just your own. And you know that you aren't interested in seeing those things every time you run **svn status**, and you are pretty sure that nobody else is interested in them either. So you use **svn propedit svn:ignore calc** to add some ignore patterns to the `calc` directory. For example, you might add this as the new value of the `svn:ignore` property:

```
calculator
debug_log*
```

当你添加完这些属性，你会在`calc`目录有一个本地修改，但是注意你的**svn status**输出有什么其他的不同：

```
$ svn status
 M      calc
 M      calc/button.c
?       calc/data.c
```

现在，所有多余的输出不见了！当然，你的计算器程序和所有的日志文件还在工作拷贝中，Subversion仅仅是不再提醒你它们的存在和未版本化。现在所有讨厌的噪音都已经不再显示，只留下了你感兴趣的条目—如你忘记添加到版本控制的源代码文件`data.c`。

当然，不仅仅只有这种简略的工作拷贝状态输出，如果想查看被忽略的文件，可以使用Subversion的`--no-ignore`选项：

```
$ svn status --no-ignore
 M      calc
 M      calc/button.c
I       calc/calculator
?       calc/data.c
I       calc/debug_log
I       calc/debug_log.1
I       calc/debug_log.2.gz
I       calc/debug_log.3.gz
```

我们在前面提到过，**svn add**和**svn import**也会使用这个忽略模式列表，这两个操作都包括了询问Subversion来开始管理一组文件和目录。比强制用户挑拣目录树中那个文件要纳入版本控制的方式更好，Subversion使用忽略模式来检测那个文件不应该在大的迭代添加和导入操作中进入版本控制系统。再次说明，操作Subversion文件和目录时你可以使用`--no-ignore`选项忽略这个忽略列表。

提示

Even if `svn:ignore` is set, you may run into problems if you use shell wildcards in a command. Shell wildcards are expanded into an explicit list of targets before Subversion operates on them, so running **svn *SUBCOMMAND* *** is just like running **svn *SUBCOMMAND* file1 file2 file3** .... In the case of the **svn add** command, this

has an effect similar to passing the `--no-ignore` option. So instead of using a wildcard, use **`svn add --force .`** to do a bulk scheduling of unversioned things for addition. The explicit target will ensure that the current directory isn't overlooked because of being already under version control, and the `--force` option will cause Subversion to crawl through that directory, adding unversioned files while still honoring the `svn:ignore` property and `global-ignores` runtime configuration variable. Be sure to also provide the `--depth files` option to the **svn add** command if you don't want a fully recursive crawl for things to add.

3.5. 关键字替换

Subversion has the ability to substitute *keywords*—pieces of useful, dynamic information about a versioned file—into the contents of the file itself. Keywords generally provide information about the last modification made to the file. Because this information changes each time the file changes, and more importantly, just *after* the file changes, it is a hassle for any process except the version control system to keep the data completely up to date. Left to human authors, the information would inevitably grow stale.

举个例子，你有一个文档希望显示最后修改的日期，你需要麻烦每个作者提交之前做这件事情，也要修改文档的一部分来描述何时作的修改，但是迟早会有人忘记做这件事，不选择简单的告诉Subversion来执行替换`LastChangedDate`关键字的操作，你通过在目标位置放置一个*keyword anchor*来控制关键字插入的位置，这个anchor只是一个格式为`$`*`KeywordName`*`$`字符串。

All keywords are case-sensitive where they appear as anchors in files: you must use the correct capitalization for the keyword to be expanded. You should consider the value of the `svn:keywords` property to be case-sensitive, too—certain keyword names will be recognized regardless of case, but this behavior is deprecated.

Subversion定义了用来替换的关键字列表，这个列表保存了如下五个关键字，有一些也包括了可用的别名：

`Date`
: 这个关键字保存了文件最后一次在版本库修改的日期，看起来类似于`$Date: 2006-07-22 21:42:37 -0700 (Sat, 22 Jul 2006) $`，它也可以用`LastChangedDate`来指定。

`Revision`
: 这个关键字描述了这个文件最后一次修改的修订版本，看起来像`$Revision: 144 $`，也可以通过`LastChangedRevision`或者`Rev`引用。

`Author`
: 这个关键字描述了最后一个修改这个文件的用户，看起来类似`$Author: harry $`，也可以用`LastChangedBy`来指定。

`HeadURL`
: 这个关键字描述了这个文件在版本库最新版本的完全URL，看起来类似`$HeadURL: http://svn.collab.net/repos/trunk/README $`，可以缩写为`URL`。

`Id`
: 这个关键字是其他关键字一个压缩组合，它看起来就像`$Id: calc.c 148 2006-07-28 21:30:43Z sally $`，可以解释为文件`calc.c`上一次修改的修订版本号是148，时间是2006年7月28日，作者是`sally`。这个关键字中显示的日期是UTC的，与`Date`关键字不一样(使用本地时区)。

Several of the preceding descriptions use the phrase “last known” or similar wording. Keep in mind that keyword expansion is a client-side operation, and your client “knows” only about changes that have occurred in the repository when you update your working copy to include those changes. If you never update your working copy, your keywords will never expand to different values even if those versioned files are being changed regularly in the repository.

只在你的文件增加关键字anchor不会做什么特别的事情，Subversion不会尝试对你的文件内容执行文本替换，除非明确的被告知这样做，毕竟，你可以撰写一个关于如何使用关键字的文档[6]，你不希望Subversion会替换你漂亮的关于不需要替换的关键字anchor实例!

To tell Subversion whether to substitute keywords on a particular file, we again turn to the property-related subcommands. The `svn:keywords` property, when set on a versioned file, controls which keywords will be substituted on that file. The value is a space-delimited list of keyword names or aliases.

举个例子，假定你有一个版本化的文件`weather.txt`，内容如下:

```
Here is the latest report from the front lines.
$LastChangedDate$
$Rev$
Cumulus clouds are appearing more frequently as summer approaches.
```

当没有`svn:keywords`属性设置到这个文件，Subversion不会有任何特别操作，现在让我们允许`LastChangedDate`关键字的替换。

```
$ svn propset svn:keywords "Date Author" weather.txt
property 'svn:keywords' set on 'weather.txt'
$
```

现在你已经对`weather.txt`的属性作了修改，你会看到文件的内容没有改变(除非你之前做了一些属性设置)，注意这个文件包含了`Rev`的关键字anchor，但我们没有在属性值中包括这个关键字，Subversion会高兴的忽略替换这个文件中的关键字，也不会替换`svn:keywords`属性中没有出现的关键字。

Immediately after you commit this property change, Subversion will update your working file with the new substitute text. Instead of seeing your keyword anchor `$LastChangedDate$`, you'll see its substituted result. That result also contains the name of the keyword and continues to be delimited by the dollar sign (`$`) characters. And as we predicted, the `Rev` keyword was not substituted because we didn't ask for it to be.

[6] … 或者可能是一本书的一个小节 …

注意我们设置svn:keywords属性为“Date Author”，关键字anchor使用别名$LastChangedDate$并且正确的扩展。

```
Here is the latest report from the front lines.
$LastChangedDate: 2006-07-22 21:42:37 -0700 (Sat, 22 Jul 2006) $
$Rev$
Cumulus clouds are appearing more frequently as summer approaches.
```

如果有其他人提交了weather.txt的修改，你的此文件的拷贝还会显示同样的替换关键字值—直到你更新你的工作拷贝，此时你的weather.txt重的关键字将会被替换来反映最新的提交信息。

$GlobalRev$ 是什么?

新用户经常为如何使用Rev关键字迷惑，自从版本库有了单独的全局增长的修订版本号码，许多人以为Rev关键字是反映修订版本号码的，但实际上Rev是文件最后修改的修订版本，而不是最后更新的。理解这一点，会减少一些混淆，但是还有一些挫折—如果没有Subversion关键字的支持，你怎么才能在你的文件自动得到全局修订版本号?

为此你需要外置处理，Subversion中有一个工具**svnversion**就是为此设计。**svnversion**遍历你的工作拷贝，然后输出它发现的修订版本，你可以使用这个程序，外加一些工具，将修订版本信息嵌入到你的文件。关于**svnversion**的更多信息，见第 9.7 节 “svnversion”。

Subversion 1.2引入了另一种关键字的语法，提供了额外和有用的，尽管是非典型的功能。你现在可以告诉Subversion为替代的关键字维护一个固定长度(从消耗字节的观点)，通过在关键字名后使用双冒号(::)，然后紧跟一组空格，你就定义了固定宽度。当Subversion使用替代值代替你的关键字，只会替换这些空白字符，保持关键字字段长度保持不变，如果替代值比定义的字段短，会有替代字段后保留空格；如果替代值太长，就会在最后的美元符号终止符前用井号(#)截断。

例如，你有一篇文档，其中一段是一些反映Subversion关键字的表格数据，使用原始的Subversion关键字替换语法，你的文件或许像这样：

```
$Rev$:      Revision of last commit
$Author$:   Author of last commit
$Date$:     Date of last commit
```

现在，表格看起来很漂亮，但是当你提交文件(当然，关键字替换功能已打开)，你会看到：

```
$Rev: 12 $:      Revision of last commit
$Author: harry $:  Author of last commit
$Date: 2006-03-15 02:33:03 -0500 (Wed, 15 Mar 2006) $:    Date of last com
```

The result is not so beautiful. And you might be tempted to then adjust the file after the substitution so that it again looks tabular. But that holds only as long as the keyword values are the same width. If the

last committed revision rolls into a new place value (say, from 99 to 100), or if another person with a longer username commits the file, stuff gets all crooked again. However, if you are using Subversion 1.2 or later, you can use the new fixed-length keyword syntax and define some field widths that seem sane, so your file might look like this:

```
$Rev::                  $:  Revision of last commit
$Author::               $:  Author of last commit
$Date::                 $:  Date of last commit
```

你提交这个文件的修改，这一次Subversion注意到了新的固定长度的关键字语法，根据你在双冒号之间指定的空格长度调整格式，并且紧跟一个美元符号。经过替换，字段的长度没有发生变化—Rev和Author多了一些空格，而较长的Date字段被一个分号截断：

```
$Rev:: 13               $:  Revision of last commit
$Author:: harry         $:  Author of last commit
$Date:: 2006-03-15 0#$:  Date of last commit
```

固定长度关键字在执行复杂文件格式的替换中非常易用，也可以处理那些很难通过其他程序(例如Microsoft Office文档)进行修改的文件。

警告

需要意识到，因为关键字字段的长度是以字节为单位，可能会破坏多字节值，例如一个用户名包含多字节的UTF-8字符，可能会遭遇从某个字符中间截断的情况，从字节角度看仅仅是一种截断，但是从UTF-8字符串角度看可能是错误和曲解的，当载入文件时，破坏的UTF-8文本可能导致整个文件的破坏，整个文件无法操作。所以，当限制关键字为固定大小时，需要选择一个可以扩展的大小。

3.6. 稀疏目录

By default, most Subversion operations on directories act in a recursive manner. For example, **svn checkout** creates a working copy with every file and directory in the specified area of the repository, descending recursively through the repository tree until the entire structure is copied to your local disk. Subversion 1.5 introduces a feature called *sparse directories* (or *shallow checkouts*) that allows you to easily check out a working copy—or a portion of a working copy—more shallowly than full recursion, with the freedom to bring in previously ignored files and subdirectories at a later time.

For example, say we have a repository with a tree of files and directories with names of the members of a human family with pets. (It's an odd example, to be sure, but bear with us.) A regular **svn checkout** operation will give us a working copy of the whole tree:

```
$ svn checkout file:///var/svn/repos mom
A    mom/son
A    mom/son/grandson
A    mom/daughter
A    mom/daughter/granddaughter1
```

```
A    mom/daughter/granddaughter1/bunny1.txt
A    mom/daughter/granddaughter1/bunny2.txt
A    mom/daughter/granddaughter2
A    mom/daughter/fishie.txt
A    mom/kitty1.txt
A    mom/doggie1.txt
Checked out revision 1.
$
```

Now, let's check out the same tree again, but this time we'll ask Subversion to give us only the topmost directory with none of its children at all:

```
$ svn checkout file:///var/svn/repos mom-empty --depth empty
Checked out revision 1
$
```

Notice that we added to our original **svn checkout** command line a new `--depth` option. This option is present on many of Subversion's subcommands and is similar to the `--non-recursive` (`-N`) and `--recursive` (`-R`) options. In fact, it combines, improves upon, supercedes, and ultimately obsoletes these two older options. For starters, it expands the supported degrees of depth specification available to users, adding some previously unsupported (or inconsistently supported) depths. Here are the depth values that you can request for a given Subversion operation:

`--depth empty`
: Include only the immediate target of the operation, not any of its file or directory children.

`--depth files`
: Include the immediate target of the operation and any of its immediate file children.

`--depth immediates`
: Include the immediate target of the operation and any of its immediate file or directory children. The directory children will themselves be empty.

`--depth infinity`
: Include the immediate target, its file and directory children, its children's children, and so on to full recursion.

Of course, merely combining two existing options into one hardly constitutes a new feature worthy of a whole section in our book. Fortunately, there is more to this story. This idea of depth extends not just to the operations you perform with your Subversion client, but also as a description of a working copy citizen's *ambient depth*, which is the depth persistently recorded by the working copy for that item. Its key strength is this very persistence—the fact that it is *sticky*. The working copy remembers the depth you've selected for each item in it until you later change that depth selection; by default, Subversion commands operate on the working copy citizens present, regardless of their selected depth settings.

提示

You can check the recorded ambient depth of a working copy using the **svn info** command. If the ambient depth is anything other than infinite recursion, **svn info** will display a line describing that depth value:

```
$ svn info mom-immediates | grep '^Depth:'
Depth: immediates
$
```

Our previous examples demonstrated checkouts of infinite depth (the default for **svn checkout**) and empty depth. Let's look now at examples of the other depth values:

```
$ svn checkout file:///var/svn/repos mom-files --depth files
A    mom-files/kitty1.txt
A    mom-files/doggie1.txt
Checked out revision 1.
$ svn checkout file:///var/svn/repos mom-immediates --depth immediates
A    mom-immediates/son
A    mom-immediates/daughter
A    mom-immediates/kitty1.txt
A    mom-immediates/doggie1.txt
Checked out revision 1.
$
```

As described, each of these depths is something more than only the target, but something less than full recursion.

We've used **svn checkout** as an example here, but you'll find the `--depth` option present on many other Subversion commands, too. In those other commands, depth specification is a way to limit the scope of an operation to some depth, much like the way the older `--non-recursive` (`-N`) and `--recursive` (`-R`) options behave. This means that when operating on a working copy of some depth, while requesting an operation of a shallower depth, the operation is limited to that shallower depth. In fact, we can make an even more general statement: given a working copy of any arbitrary—even mixed—ambient depth, and a Subversion command with some requested operational depth, the command will maintain the ambient depth of the working copy members while still limiting the scope of the operation to the requested (or default) operational depth.

In addition to the `--depth` option, the **svn update** and **svn switch** subcommands also accept a second depth-related option: `--set-depth`. It is with this option that you can change the sticky depth of a working copy item. Watch what happens as we take our empty-depth checkout and gradually telescope it deeper using **svn update --set-depth *NEW-DEPTH TARGET***:

```
$ svn update --set-depth files mom-empty
A    mom-empty/kittie1.txt
A    mom-empty/doggie1.txt
Updated to revision 1.
$ svn update --set-depth immediates mom-empty
A    mom-empty/son
A    mom-empty/daughter
Updated to revision 1.
```

```
$ svn update --set-depth infinity mom-empty
A    mom-empty/son/grandson
A    mom-empty/daughter/granddaughter1
A    mom-empty/daughter/granddaughter1/bunny1.txt
A    mom-empty/daughter/granddaughter1/bunny2.txt
A    mom-empty/daughter/granddaughter2
A    mom-empty/daughter/fishie1.txt
Updated to revision 1.
$
```

随着我们逐渐的增加我们的depth选择，版本库给我们目录树的片段。

在我们的例子里，我们只操作我们工作拷贝的根，修改其周围的depth值，但是我们可以独立的修改工作拷贝任何子目录的depth值。小心的使用这个能力允许我们充实工作拷贝树的一部分，而让其他部分不需参与(因此是特性名称的“稀疏”部分)。下面是我们可能如何构建分支的一部分的例子，为另一个分支开启完全的递归，保持其他部分是被修剪的(不在磁盘里)。

```
$ rm -rf mom-empty
$ svn checkout file:///var/svn/repos mom-empty --depth empty
Checked out revision 1.
$ svn update --set-depth empty mom-empty/son
A    mom-empty/son
Updated to revision 1.
$ svn update --set-depth empty mom-empty/daughter
A    mom-empty/daughter
Updated to revision 1.
$ svn update --set-depth infinity mom-empty/daughter/granddaughter1
A    mom-empty/daughter/granddaughter1
A    mom-empty/daughter/granddaughter1/bunny1.txt
A    mom-empty/daughter/granddaughter1/bunny2.txt
Updated to revision 1.
$
```

Fortunately, having a complex collection of ambient depths in a single working copy doesn't complicate the way you interact with that working copy. You can still make, revert, display, and commit local modifications in your working copy without providing any new options (including `--depth` and `--set-depth`) to the relevant subcommands. Even **svn update** works as it does elsewhere when no specific depth is provided—it updates the working copy targets that are present while honoring their sticky depths.

You might at this point be wondering, “So what? When would I use this?” One scenario where this feature finds utility is tied to a particular repository layout, specifically where you have many related or codependent projects or software modules living as siblings in a single repository location (`trunk/project1`, `trunk/project2`, `trunk/project3`, etc.). In such scenarios, it might be the case that you personally care about only a handful of those projects—maybe some primary project and a few other modules on which it depends. You can check out individual working copies of all of these things, but those working copies are disjoint and, as a result, it can be cumbersome to perform operations

across several or all of them at the same time. The alternative is to use the sparse directories feature, building out a single working copy that contains only the modules you care about. You'd start with an empty-depth checkout of the common parent directory of the projects, and then update with infinite depth only the items you wish to have, like we demonstrated in the previous example. Think of it like an opt-in system for working copy citizens.

Subversion 1.5's implementation of shallow checkouts is good but does not support a couple of interesting behaviors. First, you cannot de-telescope a working copy item. Running **`svn update --set-depth empty`** in an infinite-depth working copy will not have the effect of discarding everything but the topmost directory—it will simply error out. Second, there is no depth value to indicate that you wish an item to be explicitly excluded. You have to do implicit exclusion of an item by including everything else.

3.7. 锁定

Subversion's copy-modify-merge version control model lives and dies on its data merging algorithms—specifically on how well those algorithms perform when trying to resolve conflicts caused by multiple users modifying the same file concurrently. Subversion itself provides only one such algorithm: a three-way differencing algorithm that is smart enough to handle data at a granularity of a single line of text. Subversion also allows you to supplement its content merge processing with external differencing utilities (as described in 第 7.4.2 节 “外置 diff3”), some of which may do an even better job, perhaps providing granularity of a word or a single character of text. But common among those algorithms is that they generally work only on text files. The landscape starts to look pretty grim when you start talking about content merges of nontextual file formats. And when you can't find a tool that can handle that type of merging, you begin to run into problems with the copy-modify-merge model.

Let's look at a real-life example of where this model runs aground. Harry and Sally are both graphic designers working on the same project, a bit of marketing collateral for an automobile mechanic. Central to the design of a particular poster is an image of a car in need of some bodywork, stored in a file using the PNG image format. The poster's layout is almost finished, and both Harry and Sally are pleased with the particular photo they chose for their damaged car—a baby blue 1967 Ford Mustang with an unfortunate bit of crumpling on the left front fender.

Now, as is common in graphic design work, there's a change in plans, which causes the car's color to be a concern. So Sally updates her working copy to `HEAD`, fires up her photo-editing software, and sets about tweaking the image so that the car is now cherry red. Meanwhile, Harry, feeling particularly inspired that day, decides that the image would have greater impact if the car also appears to have suffered greater impact. He, too, updates to `HEAD`, and then draws some cracks on the vehicle's windshield. He manages to finish his work before Sally finishes hers, and after admiring the fruits of his undeniable talent, he commits the modified image. Shortly thereafter, Sally is finished with the car's new finish and tries to commit her changes. But, as expected, Subversion fails the commit, informing Sally that her version of the image is now out of date.

这里就是麻烦的地方，如果Harry和Sally修改的是文本文件，她只需要简单得更新工作拷贝，接收Harry的修改。在最坏的情况下，他们会修改文件的同一部分，Sally需要人工解决冲突。但是现在不是文本文件—而是二进制图像，没法估计合并的结果会是什么样子的，已存的软件不可能从基线图像分离出Harry和Sally的工作，并组合出一个挡风玻璃坏掉的红色Mustang。

很显然，如果能够将Harry和Sally的工作串行话事情会变得平滑，也就是说Harry可以等到Sally的红车然后再画上破坏的挡风玻璃，或者Sally在破坏之后改变颜色。就像在第 1.2.3 节 “拷贝–修改–合并”方案讨论的，如果Harry和Sally之间有完美的交流，就不会有这种问题发生。[7]但是作为一种版本控制系统，实际上是一种交流的形式，使得软件遵循非并行编辑的串行化也不是一件坏事，这里Subversion实现了锁定–修改–解锁模型，这里我们要讨论Subversion的锁定特性，与其他版本控制系统的“保留检出”机制类似。

Subversion's locking feature exists ultimately to minimize wasted time and effort. By allowing a user to programmatically claim the exclusive right to change a file in the repository, that user can be reasonably confident that any energy he invests on unmergeable changes won't be wasted—his commit of those changes will succeed. Also, because Subversion communicates to other users that serialization is in effect for a particular versioned object, those users can reasonably expect that the object is about to be changed by someone else. They, too, can then avoid wasting their time and energy on unmergeable changes that won't be committable due to eventual out-of-dateness.

当我们引用Subversion锁定特性时，这是在讨论一个处理版本化文件的行为特性[8](声明对一个文件排他性修改特权)，包括对文件的锁定和解锁(释放排他性修改权限)，察看包括文件被谁锁定的报告，以及提醒企图修改锁定文件的用户。在本小节，我们会覆盖锁定特性的大部分内容。

> **“锁定”的三种含义**
>
> 在本小节，和几乎本书的每一个地方“lock”和“locking”描述了一种避免用户之间冲突提交的排他机制，但是很不幸，Subversion中还有另外两种锁，因此需要在本书格外关心。
>
> The second is *working copy locks*, used internally by Subversion to prevent clashes between multiple Subversion clients operating on the same working copy. This is the sort of lock indicated by an `L` in the third column of **svn status** output, and removed by the **svn cleanup** command, as described in 第 2.6 节 “有时你只需要清理”.
>
> Third, there are *database locks*, used internally by the Berkeley DB backend to prevent clashes between multiple programs trying to access the database. This is the sort of lock whose unwanted persistence after an error can cause a repository to be “wedged,” as described in 第 5.4.4 节 “Berkeley DB 恢复”.
>
> 在发生问题之前你完全可以忘记上面两种锁，在本书，“锁定”意味着第一种锁，除非是在从上下文中十分明确或明确指出的。

3.7.1. 创建锁定

In the Subversion repository, a *lock* is a piece of metadata that grants exclusive access to one user to change a file. This user is said to be the *lock owner*. Each lock also has a unique identifier, typically a long string of characters, known as the *lock token*. The repository manages locks, ultimately handling

[7]对于Harry和Sally的好莱坞同名人来说交流也不是那么差的药，也是关于那一点。
[8]Subversion目前不允许锁定目录。

their creation, enforcement, and removal. If any commit transaction attempts to modify or delete a locked file (or delete one of the parent directories of the file), the repository will demand two pieces of information—that the client performing the commit be authenticated as the lock owner, and that the lock token has been provided as part of the commit process as a form of proof that the client knows which lock it is using.

为了描述锁的产生，我们回到前面那个关于多个图形设计师共同工作的例子，Harry决定修改一个JPEG图像，为了防止其他用户此时提交这个文件的修改(也是警告别人他正在修改它)，他使用**svn lock**命令锁定了版本库中的这个文件：

```
$ svn lock banana.jpg -m "Editing file for tomorrow's release."
'banana.jpg' locked by user 'harry'.
$
```

The preceding example demonstrates a number of new things. First, notice that Harry passed the `--message` (`-m`) option to **svn lock**. Similar to **svn commit**, the **svn lock** command can take comments—via either `--message` (`-m`) or `--file` (`-F`)—to describe the reason for locking the file. Unlike **svn commit**, however, **svn lock** will not demand a message by launching your preferred text editor. Lock comments are optional, but still recommended to aid communication.

Second, the lock attempt succeeded. This means that the file wasn't already locked, and that Harry had the latest version of the file. If Harry's working copy of the file had been out of date, the repository would have rejected the request, forcing Harry to **svn update** and reattempt the locking command. The locking command would also have failed if the file had already been locked by someone else.

就像你看到的，**svn lock**打印了锁定成功的确认信息。此时，通过**svn status**和**svn info**的输出我们可以看到文件已经锁定。

```
$ svn status
     K banana.jpg

$ svn info banana.jpg
Path: banana.jpg
Name: banana.jpg
URL: http://svn.example.com/repos/project/banana.jpg
Repository UUID: edb2f264-5ef2-0310-a47a-87b0ce17a8ec
Revision: 2198
Node Kind: file
Schedule: normal
Last Changed Author: frank
Last Changed Rev: 1950
Last Changed Date: 2006-03-15 12:43:04 -0600 (Wed, 15 Mar 2006)
Text Last Updated: 2006-06-08 19:23:07 -0500 (Thu, 08 Jun 2006)
Properties Last Updated: 2006-06-08 19:23:07 -0500 (Thu, 08 Jun 2006)
Checksum: 3b110d3b10638f5d1f4fe0f436a5a2a5
Lock Token: opaquelocktoken:0c0f600b-88f9-0310-9e48-355b44d4a58e
Lock Owner: harry
```

```
Lock Created: 2006-06-14 17:20:31 -0500 (Wed, 14 Jun 2006)
Lock Comment (1 line):
Editing file for tomorrow's release.

$
```

svn info命令不会联系版本库，当对工作拷贝路径应用**svn info**命令时，可以揭示令牌的一个重要事实—它们缓存在工作拷贝。有锁定令牌是非常重要的，这给了工作拷贝权利利用这个锁的能力。**svn status**会在文件后面显示一个K(locKed的缩写)，表明了拥有锁定令牌。

> **关于锁定令牌**
>
> A lock token isn't an authentication token, so much as an *authorization* token. The token isn't a protected secret. In fact, a lock's unique token is discoverable by anyone who runs **svn info *URL***. A lock token is special only when it lives inside a working copy. It's proof that the lock was created in that particular working copy, and not somewhere else by some other client. Merely authenticating as the lock owner isn't enough to prevent accidents.
>
> For example, suppose you lock a file using a computer at your office, but leave work for the day before you finish your changes to that file. It should not be possible to accidentally commit changes to that same file from your home computer later that evening simply because you've authenticated as the lock's owner. In other words, the lock token prevents one piece of Subversion-related software from undermining the work of another. (In our example, if you really need to change the file from an alternative working copy, you would need to *break* the lock and relock the file.)

现在Harry已经锁定了banana.jpg，Sally不能修改或删除这个文件：

```
$ svn delete banana.jpg
D         banana.jpg
$ svn commit -m "Delete useless file."
Deleting       banana.jpg
svn: Commit failed (details follow):
svn: Server sent unexpected return value (423 Locked) in response to DELET
 request for '/repos/project/!svn/wrk/64bad3a9-96f9-0310-818a-df4224ddc35
banana.jpg'
$
```

但是，当完成了香蕉的黄色渐变，就可以提交文件的修改，因为认证为锁定的拥有者，也因为他的工作拷贝有正确的锁定令牌：

```
$ svn status
M    K banana.jpg
$ svn commit -m "Make banana more yellow"
Sending        banana.jpg
Transmitting file data .
```

```
Committed revision 2201.
$ svn status
$
```

Notice that after the commit is finished, **svn status** shows that the lock token is no longer present in the working copy. This is the standard behavior of **svn commit**—it searches the working copy (or list of targets, if you provide such a list) for local modifications and sends all the lock tokens it encounters during this walk to the server as part of the commit transaction. After the commit completes successfully, all of the repository locks that were mentioned are released—*even on files that weren't committed*. This is meant to discourage users from being sloppy about locking or from holding locks for too long. If Harry haphazardly locks 30 files in a directory named `images` because he's unsure of which files he needs to change, yet changes only four of those files, when he runs **`svn commit images`**, the process will still release all 30 locks.

自动释放锁定的特性可以通过**svn commit**的`--no-unlock`选项关闭，当你要提交文件，同时期望继续修改而必须保留锁定时非常有用。这个特性也可以半永久性的设定，方法是设置运行中`config`文件(见第 7.1 节 “运行配置区”)的`no-unlock = yes`。

当然，锁定一个文件不会强制一个人要提交修改，任何时候都可以通过运行**svn unlock**命令释放锁定：

```
$ svn unlock banana.c
'banana.c' unlocked.
```

3.7.2. 发现锁定

When a commit fails due to someone else's locks, it's fairly easy to learn about them. The easiest way is to run **`svn status --show-updates`**:

```
$ svn status -u
M               23   bar.c
M    O          32   raisin.jpg
       *        72   foo.h
Status against revision:     105
$
```

In this example, Sally can see not only that her copy of `foo.h` is out of date, but also that one of the two modified files she plans to commit is locked in the repository. The `O` symbol stands for “Other,” meaning that a lock exists on the file and was created by somebody else. If she were to attempt a commit, the lock on `raisin.jpg` would prevent it. Sally is left wondering who made the lock, when, and why. Once again, **svn info** has the answers:

```
$ svn info http://svn.example.com/repos/project/raisin.jpg
Path: raisin.jpg
Name: raisin.jpg
URL: http://svn.example.com/repos/project/raisin.jpg
```

```
Repository UUID: edb2f264-5ef2-0310-a47a-87b0ce17a8ec
Revision: 105
Node Kind: file
Last Changed Author: sally
Last Changed Rev: 32
Last Changed Date: 2006-01-25 12:43:04 -0600 (Sun, 25 Jan 2006)
Lock Token: opaquelocktoken:fc2b4dee-98f9-0310-abf3-653ff3226e6b
Lock Owner: harry
Lock Created: 2006-02-16 13:29:18 -0500 (Thu, 16 Feb 2006)
Lock Comment (1 line):
Need to make a quick tweak to this image.
$
```

Just as you can use **svn info** to examine objects in the working copy, you can also use it to examine objects in the repository. If the main argument to **svn info** is a working copy path, then all of the working copy's cached information is displayed; any mention of a lock means that the working copy is holding a lock token (if a file is locked by another user or in another working copy, **svn info** on a working copy path will show no lock information at all). If the main argument to **svn info** is a URL, the information reflects the latest version of an object in the repository, and any mention of a lock describes the current lock on the object.

So in this particular example, Sally can see that Harry locked the file on February 16 to “make a quick tweak.” It being June, she suspects that he probably forgot all about the lock. She might phone Harry to complain and ask him to release the lock. If he's unavailable, she might try to forcibly break the lock herself or ask an administrator to do so.

3.7.3. 解除和偷窃锁定

版本库锁定并不是神圣不可侵犯的—在Subversion的缺省配置状态，不只是创建者可以释放锁定，任何人都可以。当有其他人期望消灭锁定时，我们称之为打破锁定。

从管理员的位子上很容易打破锁定，**svnlook**和**svnadmin**程序都有能力从版本库直接显示和删除锁定。(关于这些工具的信息可以看第 5.4.1 节 “管理员的工具箱”。)

```
$ svnadmin lslocks /var/svn/repos
Path: /project2/images/banana.jpg
UUID Token: opaquelocktoken:c32b4d88-e8fb-2310-abb3-153ff1236923
Owner: frank
Created: 2006-06-15 13:29:18 -0500 (Thu, 15 Jun 2006)
Expires:
Comment (1 line):
Still improving the yellow color.

Path: /project/raisin.jpg
UUID Token: opaquelocktoken:fc2b4dee-98f9-0310-abf3-653ff3226e6b
Owner: harry
Created: 2006-02-16 13:29:18 -0500 (Thu, 16 Feb 2006)
```

```
Expires:
Comment (1 line):
Need to make a quick tweak to this image.

$ svnadmin rmlocks /var/svn/repos /project/raisin.jpg
Removed lock on '/project/raisin.jpg'.
$
```

The more interesting option is to allow users to break each other's locks over the network. To do this, Sally simply needs to pass the `--force` to the **svn unlock** command:

```
$ svn status -u
M              23   bar.c
M    O         32   raisin.jpg
       *       72   foo.h
Status against revision:     105
$ svn unlock raisin.jpg
svn: 'raisin.jpg' is not locked in this working copy
$ svn info raisin.jpg | grep URL
URL: http://svn.example.com/repos/project/raisin.jpg
$ svn unlock http://svn.example.com/repos/project/raisin.jpg
svn: Unlock request failed: 403 Forbidden (http://svn.example.com)
$ svn unlock --force http://svn.example.com/repos/project/raisin.jpg
'raisin.jpg' unlocked.
$
```

Sally初始的unlock命令失败了，因为她直接在自己的工作拷贝上运行了**svn unlock**，而这里没有锁定令牌。为了直接从版本库删除锁定，她需要给**svn unlock**传递URL参数，她的这一次尝试又失败了，因为她不是锁定的拥有者(也没有锁定令牌)。当她使用了`--force`选项后，认证和授权的要求被忽略了，远程的锁定被打破了。

Simply breaking a lock may not be enough. In the running example, Sally may not only want to break Harry's long-forgotten lock, but relock the file for her own use. She can accomplish this by using **svn unlock** with `--force` and then **svn lock** back-to-back, but there's a small chance that somebody else might lock the file between the two commands. The simpler thing to do is to *steal* the lock, which involves breaking and relocking the file all in one atomic step. To do this, Sally passes the `--force` option to **svn lock**:

```
$ svn lock raisin.jpg
svn: Lock request failed: 423 Locked (http://svn.example.com)
$ svn lock --force raisin.jpg
'raisin.jpg' locked by user 'sally'.
$
```

In any case, whether the lock is broken or stolen, Harry may be in for a surprise. Harry's working copy still contains the original lock token, but that lock no longer exists. The lock token is said to be *defunct*. The lock represented by the lock token has either been broken (no longer in the repository) or

stolen (replaced with a different lock). Either way, Harry can see this by asking **svn status** to contact the repository:

```
$ svn status
     K raisin.jpg
$ svn status -u
     B         32   raisin.jpg
$ svn update
  B  raisin.jpg
$ svn status
$
```

If the repository lock was broken, then **svn status --show-updates** displays a `B` (Broken) symbol next to the file. If a new lock exists in place of the old one, then a `T` (sTolen) symbol is shown. Finally, **svn update** notices any defunct lock tokens and removes them from the working copy.

锁定策略

Different systems have different notions of how strict a lock should be. Some folks argue that locks must be strictly enforced at all costs, releasable only by the original creator or administrator. They argue that if anyone can break a lock, chaos runs rampant and the whole point of locking is defeated. The other side argues that locks are first and foremost a communication tool. If users are constantly breaking each other's locks, it represents a cultural failure within the team and the problem falls outside the scope of software enforcement.

Subversion缺省是比较“宽松的”方式，但也允许管理员创建钩子脚本来建立严格的控制策略。具体来说，`pre-lock`和`pre-unlock`钩子允许管理员决定什么时候创建和释放锁定。根据锁定是否已经存在，这两个钩子脚本可以决定是否允许特定用户打破或窃取锁定。也有`post-lock`和`post-unlock`钩子，可以用来发送锁定动作的通知邮件。关于版本库钩子的更多信息可以看第 5.3.2 节 “实现版本库钩子”。

3.7.4. 锁定交流

我们已经见到了如何利用**svn lock**和**svn unlock**来创建、释放、打破和窃取锁定，这就满足了顺序访问文件的要求，但是浪费时间这个大问题该如何呢？

For example, suppose Harry locks an image file and then begins editing it. Meanwhile, miles away, Sally wants to do the same thing. She doesn't think to run **svn status --show-updates**, so she has no idea that Harry has already locked the file. She spends hours editing the file, and when she tries to commit her change, she discovers that either the file is locked or that she's out of date. Regardless, her changes aren't mergeable with Harry's. One of these two people has to throw away his or her work, and a lot of time has been wasted.

Subversion's solution to this problem is to provide a mechanism to remind users that a file ought to be locked *before* the editing begins. The mechanism is a special property: `svn:needs-lock`. If that property is attached to a file (regardless of its value, which is irrelevant), Subversion will try to

use filesystem-level permissions to make the file read-only—unless, of course, the user has explicitly locked the file. When a lock token is present (as a result of using **svn lock**), the file becomes read/write. When the lock is released, the file becomes read-only again.

The theory, then, is that if the image file has this property attached, Sally would immediately notice something is strange when she opens the file for editing: many applications alert users immediately when a read-only file is opened for editing, and nearly all would prevent her from saving changes to the file. This reminds her to lock the file before editing, whereby she discovers the preexisting lock:

```
$ /usr/local/bin/gimp raisin.jpg
gimp: error: file is read-only!
$ ls -l raisin.jpg
-r--r--r--   1 sally   sally   215589 Jun  8 19:23 raisin.jpg
$ svn lock raisin.jpg
svn: Lock request failed: 423 Locked (http://svn.example.com)
$ svn info http://svn.example.com/repos/project/raisin.jpg | grep Lock
Lock Token: opaquelocktoken:fc2b4dee-98f9-0310-abf3-653ff3226e6b
Lock Owner: harry
Lock Created: 2006-06-08 07:29:18 -0500 (Thu, 08 June 2006)
Lock Comment (1 line):
Making some tweaks.  Locking for the next two hours.
$
```

提示

我们鼓励用户和管理员都应该给不能根据上下文的文件添加`svn:needs-lock`属性，这是鼓励好的锁定习惯和防止浪费的主要技术手段。

需要注意到这个属性是依赖于锁定系统的交流工具，不管是否有这个属性，文件都可以锁定。相反的，无论有没有这个属性，并不会要求提交需要首先锁定文件。

这个系统并不是毫无瑕疵，即使有这个属性，只读提醒也有可能失效。有些程序“偷偷的篡改了”文件的只读属性，悄无声息的允许用户编辑和保存文件，不幸的是，Subversion对此无能为力—即使到了现今，还是没有任何工具能够代替人与人的良好交流。[9]

3.8. 外部定义

有时候创建一个由多个不同检出得到的工作拷贝是非常有用的，举个例子，你或许希望不同的子目录来自不同的版本库位置，或者是不同的版本库。你可以手工设置这样一个工作拷贝—使用**svn checkout**来创建这种你需要的嵌套的工作拷贝结构。但是如果这个结构对所有的用户是很重要的，每个用户需要执行同样的检出操作。

很幸运，Subversion提供了外部定义的支持，一个外部定义是一个本地路经到URL的影射—也有可能一个特定的修订版本—一些版本化的资源。在Subversion你可以使用`svn:externals`属性来定义外部定义，你可以用**svn propset**或**svn propedit**(见第 3.2.2 节“操作属性”)创建和修改这个属性。它可以设置到任何版本化的路经，它的值是一个多行的

[9]除非是，或许一个经典的火神精神融合。

子目录，可选的修订版本标记和完全有效的Subversion版本库URL的列表(相对于设置属性的版本化目录)。

svn:externals的方便之处是这个属性设置到版本化的路径后，任何人可以从那个目录取出一个工作拷贝，同样得到外部定义的好处。换句话说，一旦一个人努力来定义这些嵌套的工作拷贝检出，其他任何人不需要再麻烦了—Subversion会在原先的工作拷贝检出之后，也会检出外部工作拷贝。

警告

外部定义的相对目标子目录不需要存在于你的或其它用户的系统中—Subversion会在检出工作拷贝时创建这些文件。实际上，你一定不要使用外部定义来产生已经在版本控制的路径。

You also get in the externals definition design all the regular benefits of Subversion properties. The definitions are versioned. If you need to change an externals definition, you can do so using the regular property modification subcommands. When you commit a change to the svn:externals property, Subversion will synchronize the checked-out items against the changed externals definition when you next run **svn update**. The same thing will happen when others update their working copies and receive your changes to the externals definition.

提示

因为svn:externals的值是多行的，所以我们强烈建议使用**svn propedit**，而不是使用**svn propset**。

Subversion releases prior to 1.5 honor an externals definition format that is a multiline table of subdirectories (relative to the versioned directory on which the property is set), optional revision flags, and fully qualified, absolute Subversion repository URLs. An example of this might looks as follows:

```
$ svn propget svn:externals calc
third-party/sounds             http://svn.example.com/repos/sounds
third-party/skins -r148        http://svn.example.com/skinproj
third-party/skins/toolkit -r21 http://svn.example.com/skin-maker
```

注意前一个外部定义实例，当有人取出了一个calc目录的工作拷贝，Subversion会继续来取出外部定义的项目。

```
$ svn checkout http://svn.example.com/repos/calc
A  calc
A  calc/Makefile
A  calc/integer.c
A  calc/button.c
Checked out revision 148.

Fetching external item into calc/third-party/sounds
A  calc/third-party/sounds/ding.ogg
```

```
A  calc/third-party/sounds/dong.ogg
A  calc/third-party/sounds/clang.ogg
…
A  calc/third-party/sounds/bang.ogg
A  calc/third-party/sounds/twang.ogg
Checked out revision 14.

Fetching external item into calc/third-party/skins
…
```

As of Subversion 1.5, though, a new format of the `svn:externals` property is supported. Externals definitions are still multiline, but the order and format of the various pieces of information have changed. The new syntax more closely mimics the order of arguments you might pass to **svn checkout**: the optional revision flags come first, then the external Subversion repository URL, and finally the relative local subdirectory. Notice, though, that this time we didn't say “fully qualified, absolute Subversion repository URLs.” That's because the new format supports relative URLs and URLs that carry peg revisions. The previous example of an externals definition might, in Subversion 1.5, look like the following:

```
$ svn propget svn:externals calc
      http://svn.example.com/repos/sounds third-party/sounds
-r148 http://svn.example.com/skinproj third-party/skins
-r21  http://svn.example.com/skin-maker third-party/skins/toolkit
```

Or, making use of the peg revision syntax (which we describe in detail in 第 3.9 节 “Peg 和实施修订版本”), it might appear as:

```
$ svn propget svn:externals calc
http://svn.example.com/repos/sounds third-party/sounds
http://svn.example.com/skinproj@148 third-party/skins
http://svn.example.com/skin-maker@21 third-party/skins/toolkit
```

提示

你一定要要慎重考虑在所有的外部定义中使用明确的修订版本，这样做意味着你已经决定了何时拖出外部信息不同的快照，和精确的拖出哪个快照。除了不会受到第三方版本库的意外修改的影响以外，当你的工作拷贝回溯到以前的版本库时，使用明确的修订版本号会让外部定义回到以前的那个修订版本，也意味着外部定义的工作拷贝更新会匹配以前修订版本的样子。对于软件项目，这可能是编译复杂代码基的老快照成功和失败的区别。

For most repositories, these three ways of formatting the externals definitions have the same ultimate effect. They all bring the same benefits. Unfortunately, they all bring the same annoyances, too. Since the definitions shown use absolute URLs, moving or copying a directory to which they are attached will not affect what gets checked out as an external (though the relative local target subdirectory will, of course, move with the renamed directory). This can be confusing—even frustrating—in certain situations. For example, say you have a top-level directory named `my-project`, and you've created

an externals definition on one of its subdirectories (`my-project/some-dir`) that tracks the latest revision of another of its subdirectories (`my-project/external-dir`).

```
$ svn checkout http://svn.example.com/projects .
A    my-project
A    my-project/some-dir
A    my-project/external-dir
…
Fetching external item into 'my-project/some-dir/subdir'
Checked out external at revision 11.

Checked out revision 11.
$ svn propget svn:externals my-project/some-dir
subdir http://svn.example.com/projects/my-project/external-dir

$
```

现在你使用**svn move**将目录`my-project`改名，此刻，你的外部定义还是指向`my-project`目录，即使这个目录已经不存在了。

```
$ svn move -q my-project renamed-project
$ svn commit -m "Rename my-project to renamed-project."
Deleting       my-project
Adding         renamed-project

Committed revision 12.
$ svn update

Fetching external item into 'renamed-project/some-dir/subdir'
svn: Target path does not exist
$
```

Also, absolute URLs can cause problems with repositories that are available via multiple URL schemes. For example, if your Subversion server is configured to allow everyone to check out the repository over `http://` or `https://`, but only allow commits to come in via `https://`, you have an interesting problem on your hands. If your externals definitions use the `http://` form of the repository URLs, you won't be able to commit anything from the working copies created by those externals. On the other hand, if they use the `https://` form of the URLs, anyone who might be checking out via `http://` because his client doesn't support `https://` will be unable to fetch the external items. Be aware, too, that if you need to reparent your working copy (using **svn switch** with the `--relocate` option), externals definitions will *not* also be reparented.

Subversion 1.5 takes a huge step in relieving these frustrations. As mentioned earlier, the URLs used in the new externals definition format can be relative, and Subversion provides syntax magic for specifying multiple flavors of URL relativity.

`../`
: Relative to the URL of the directory on which the `svn:externals` property is set

`^/`
: Relative to the root of the repository in which the `svn:externals` property is versioned

`//`
: Relative to the scheme of the URL of the directory on which the `svn:externals` property is set

`/`
: Relative to the root URL of the server on which the `svn:externals` property is versioned

So, looking a fourth time at our previous externals definition example, and making use of the new absolute URL syntax in various ways, we might now see:

```
$ svn propget svn:externals calc
^/sounds third-party/sounds
/skinproj@148 third-party/skins
//svn.example.com/skin-maker@21 third-party/skins/toolkit
```

The support that exists for externals definitions in Subversion remains less than ideal, though. An externals definition can point only to directories, not to files. Also, the local subdirectory part of the definition cannot contain `..` parent directory indicators (such as `../../skins/myskin`). Perhaps most disappointingly, the working copies created via the externals definition support are still disconnected from the primary working copy (on whose versioned directories the `svn:externals` property was actually set). And Subversion still truly operates only on nondisjoint working copies. So, for example, if you want to commit changes that you've made in one or more of those external working copies, you must run **svn commit** explicitly on those working copies—committing on the primary working copy will not recurse into any external ones.

We've already mentioned some of the additional shortcomings of the old `svn:externals` format and how the new Subversion 1.5 format improves upon it. But be careful when making use of the new format that you don't inadvertently cause problems for other folks accessing your repository who are using older Subversion clients. While Subversion 1.5 clients will continue to recognize and support the original externals definition format, older clients will *not* be able to correctly parse the new format.

Besides the **svn checkout**, **svn update**, **svn switch**, and **svn export** commands which actually manage the *disjoint* (or disconnected) subdirectories into which externals are checked out, the **svn status** command also recognizes externals definitions. It displays a status code of `X` for the disjoint external subdirectories, and then recurses into those subdirectories to display the status of the external items themselves. You can pass the `--ignore-externals` option to any of these subcommands to disable externals definition processing.

3.9. Peg 和实施修订版本

文件和目录的拷贝、改名和移动能力使你可以创建一个项目，然后删除它，然后在同一个位置添加一个新的—这是在我们的计算机中经常发生的操作，而你的版本控制系统不应该成为你这样操作的障碍。Subversion的文件管理操作是这样的开放，提供了几乎和普通文件一样的操作版本化文件的灵活性，但是灵活意味着在整个版本库的生命周期中，一个给定的版本化的资源可能会出现在许多不同的路径，一个给定的路径会展示给我们许多完全不同的版本化资源。当然这些功能也增加了你与这些路径和资源交互的难度。

Subversion可以非常聪明的注意到一个对象的包括一个“地址改变”历史变化，举个例子，如果你询问一个曾经上周改过名的文件的所有的日志信息，Subversion会很高兴提供所有的日志—重命名发生的修订版本，外加相关版本之前和之后的修订版本日志，所以大多数时间里，你不需要考虑这些事情，但是偶尔，Subversion会需要你的帮助来清除混淆。

这个最简单的例子发生在当一个目录或者文件从版本控制中删除时，然后一个新的同样名字目录或者文件添加到版本控制，显然你删除的和你后来添加的不是同样的东西，它们仅仅是有同样的路径，例如`/trunk/object`。什么，这意味着询问Subversion来查看`/trunk/object`的历史？你是询问当前这个位置的东西还是你在这个位置删除的那个对象？你是希望询问对这个对象的所有操作还是这个路径的所有对象？很明显，Subversion需要线索知道你真实的想法。

And thanks to moves, versioned object history can get far more twisted than even that. For example, you might have a directory named `concept`, containing some nascent software project you've been toying with. Eventually, though, that project matures to the point that the idea seems to actually have some wings, so you do the unthinkable and decide to give the project a name. [10] Let's say you called your software Frabnaggilywort. At this point, it makes sense to rename the directory to reflect the project's new name, so `concept` is renamed to `frabnaggilywort`. Life goes on, Frabnaggilywort releases a 1.0 version and is downloaded and used daily by hordes of people aiming to improve their lives.

这是一个美好的故事，但是没有在这里结束，作为主办人，你一定想到了另一件事，所以你创建了一个目录叫做`concept`，周期重新开始。实际上，这个循环在几年里开始了多次，每一个想法从使用旧的`concept`目录开始，然后有时在想法成熟之后重新命名，有时你放弃了这个注意而删除了这个目录。或者更加变态一点，或许你把`concept`改成其他名字之后又因为一些原因重新改回`concept`。

In scenarios like these, attempting to instruct Subversion to work with these reused paths can be a little like instructing a motorist in Chicago's West Suburbs to drive east down Roosevelt Road and turn left onto Main Street. In a mere 20 minutes, you can cross “Main Street” in Wheaton, Glen Ellyn, and Lombard. And no, they aren't the same street. Our motorist—and our Subversion—need a little more detail to do the right thing.

在1.1版本，Subversion提供了一种方法来说明你所指是哪一个街道，叫做*peg*修订版本，通过这个修订版本我们可以唯一确定一条历史线路，因为一个版本化的文件会在任何时间占用某个路径—路径和peg修订版本的合并是可以指定一个历史的特定线路。Peg修订版本可以在Subversion命令行客户端中用*at*语法指定，之所以使用这个名称是因为会在关联的修订版本的路径后面追加一个“at符号”(`@`)。

But what of the `--revision` (`-r`) of which we've spoken so much in this book? That revision (or set of revisions) is called the *operative revision* (or *operative revision range*). Once a particular line of history has been identified using a path and peg revision, Subversion performs the requested operation using the operative revision(s). To map this to our Chicagoland streets analogy, if we are told to go to 606 N. Main Street in Wheaton, [11] we can think of “Main Street” as our path and “Wheaton” as our peg revision. These two pieces of information identify a unique path that can be traveled (north or south on Main Street), and they keep us from traveling up and down the wrong Main Street in search of our destination. Now we throw in “606 N.” as our operative revision of sorts, and we know *exactly* where to go.

[10] “你不是被期望去命名它，一旦你取了名字，你开始与之联系在一起。” — Mike Wazowski

[11] 伊利诺伊州惠顿主大道606号市惠顿离市中心，让它作为惠顿的“历史”中心，看起来是恰当的…。

Peg 修订版本算法

The Subversion command-line client performs the peg revision algorithm any time it needs to resolve possible ambiguities in the paths and revisions provided to it. Here's an example of such an invocation:

```
$ svn command -r OPERATIVE-REV item@PEG-REV
```

If *OPERATIVE-REV* is older than *PEG-REV*, the algorithm is as follows:

1. 来到修订版本*PEG-REV*，找到*item*，在版本库定位到一个唯一的对象。
2. 追踪对象的历史背景(通过任何可能的改名)来到修订版本*OPERATIVE-REV*的祖先。
3. Perform the requested action on that ancestor, wherever it is located, or whatever its name might be or might have been at that time.

但是如果*OPERATIVE-REV*比*PEG-REV*更年轻时会怎么样？这为定位*OPERATIVE-REV*中的路径的理论问题增加了一些复杂性，因为在*PEG-REV*和*OPERATIVE-REV*之间，路径在历史中可以出现多次(由于拷贝操作)，而且那还不是全部—Subversion不会保存向前跟踪历史的足够信息，所以算法会有一点不同：

1. 来到修订版本*PEG-REV*，找到*item*，在版本库定位到一个唯一的对象。
2. Trace the object's history backward (through any possible renames) to its ancestor in the revision *PEG-REV*.
3. Verify that the object's location (path-wise) in *PEG-REV* is the same as it is in *OPERATIVE-REV*. If that's the case, at least the two locations are known to be directly related, so perform the requested action on the location in *OPERATIVE-REV*. Otherwise, relatedness was not established, so error out with a loud complaint that no viable location was found. (Someday, we expect that Subversion will be able to handle this usage scenario with more flexibility and grace.)

注意，即使你没有明确提供peg修订版本或操作修订版本，他们依然是存在的。为了使用的简便，对于工作拷贝项目的缺省peg修订版本是BASE，而版本库URL的缺省值是HEAD。当没有提供操作修订版本时，缺省是与peg修订版本一样。

Say that long ago we created our repository, and in revision 1 we added our first `concept` directory, plus an `IDEA` file in that directory talking about the concept. After several revisions in which real code was added and tweaked, we, in revision 20, renamed this directory to `frabnaggilywort`. By revision 27, we had a new concept, a new `concept` directory to hold it, and a new `IDEA` file to describe it. And then five years and thousands of revisions flew by, just like they would in any good romance story.

Now, years later, we wonder what the `IDEA` file looked like back in revision 1. But Subversion needs to know whether we are asking about how the *current* file looked back in revision 1, or whether we are asking for the contents of whatever file lived at `concepts/IDEA` in revision 1. Certainly those

questions have different answers, and because of peg revisions, you can ask those questions. To find out how the current `IDEA` file looked in that old revision, you run:

```
$ svn cat -r 1 concept/IDEA
svn: Unable to find repository location for 'concept/IDEA' in revision 1
```

当然，在这个例子里，当前的IDEA文件在修订版本1中并不存在，所以Subversion给出一个错误，这个上面的命令是长的peg修订版本命令一个缩写，扩展的写法是：

```
$ svn cat -r 1 concept/IDEA@BASE
svn: Unable to find repository location for 'concept/IDEA' in revision 1
```

当执行时，它包含期望的结果。

The perceptive reader is probably wondering at this point whether the peg revision syntax causes problems for working copy paths or URLs that actually have at signs in them. After all, how does **svn** know whether `news@11` is the name of a directory in my tree or just a syntax for "revision 11 of `news`"? Thankfully, while **svn** will always assume the latter, there is a trivial workaround. You need only append an at sign to the end of the path, such as `news@11@`. **svn** cares only about the last at sign in the argument, and it is not considered illegal to omit a literal peg revision specifier after that at sign. This workaround even applies to paths that end in an at sign—you would use `filename@@` to talk about a file named `filename@`.

然后让我们询问另一个问题—在修订版本1，占据concepts/IDEA路径的文件的内容到底是什么？我们会使用一个明确的peg修订版本来帮助我们完成。

```
$ svn cat concept/IDEA@1
The idea behind this project is to come up with a piece of software
that can frab a naggily wort.  Frabbing naggily worts is tricky
business, and doing it incorrectly can have serious ramifications, so
we need to employ over-the-top input validation and data verification
mechanisms.
```

注意我们这一次没有提供操作修订版本，那是因为如果没有指定操作修订版本，Subversion假定缺省的操作修订版本是peg修订版本。

正像你看到的，这看起来是正确的输出，这些文本甚至提到"frabbing naggily worts"，所以这就是现在叫做Frabnaggilywort项目的那个文件，实际上，我们可以使用显示的peg修订版本和实施修订版本的组合核实这一点。我们知道在HEAD，Frabnaggilywort项目坐落在frabnaggilywort目录，所以我们指定我们希望看到HEAD的frabnaggilywort/IDEA路经在历史上的修订版本1的内容。

```
$ svn cat -r 1 frabnaggilywort/IDEA@HEAD
The idea behind this project is to come up with a piece of software
that can frab a naggily wort.  Frabbing naggily worts is tricky
```

```
business, and doing it incorrectly can have serious ramifications, so
we need to employ over-the-top input validation and data verification
mechanisms.
```

而且peg修订版本和实施修订版本也不需要这样琐碎，举个例子，我们的frabnaggilywort已经在HEAD删除，但我们知道在修订版本20它是存在的，我们希望知道IDEA从修订版本4到10的区别，我们可以使用peg修订版本20和IDEA文件的修订版本20的URL的组合，然后使用4到10作为我们的实施修订版本范围。

```
$ svn diff -r 4:10 http://svn.red-bean.com/projects/frabnaggilywort/IDEA@2
Index: frabnaggilywort/IDEA
===================================================================
--- frabnaggilywort/IDEA (revision 4)
+++ frabnaggilywort/IDEA (revision 10)
@@ -1,5 +1,5 @@
-The idea behind this project is to come up with a piece of software
-that can frab a naggily wort.  Frabbing naggily worts is tricky
-business, and doing it incorrectly can have serious ramifications, so
-we need to employ over-the-top input validation and data verification
-mechanisms.
+The idea behind this project is to come up with a piece of
+client-server software that can remotely frab a naggily wort.
+Frabbing naggily worts is tricky business, and doing it incorrectly
+can have serious ramifications, so we need to employ over-the-top
+input validation and data verification mechanisms.
```

幸运的是，几乎所有的人不会面临如此复杂的情形，但是如果是，记住peg修订版本是帮助Subversion清除混淆的额外提示。

3.10. 修改列表

It is commonplace for a developer to find himself working at any given time on multiple different, distinct changes to a particular bit of source code. This isn't necessarily due to poor planning or some form of digital masochism. A software engineer often spots bugs in his peripheral vision while working on some nearby chunk of source code. Or perhaps he's halfway through some large change when he realizes the solution he's working on is best committed as several smaller logical units. Often, these logical units aren't nicely contained in some module, safely separated from other changes. The units might overlap, modifying different files in the same module, or even modifying different lines in the same file.

Developers can employ various work methodologies to keep these logical changes organized. Some use separate working copies of the same repository to hold each individual change in progress. Others might choose to create short-lived feature branches in the repository and use a single working copy that is constantly switched to point to one such branch or another. Still others use **diff** and **patch** tools to back up and restore uncommitted changes to and from patch files associated with each change. Each of these methods has its pros and cons, and to a large degree, the details of the changes being made heavily influence the methodology used to distinguish them.

Subversion 1.5 brings a new *changelists* feature that adds yet another method to the mix. Changelists are basically arbitrary labels (currently at most one per file) applied to working copy files for the express purpose of associating multiple files together. Users of many of Google's software offerings are familiar with this concept already. For example, Gmail doesn't provide the traditional folders-based email organization mechanism. In Gmail, you apply arbitrary labels to emails, and multiple emails can be said to be part of the same group if they happen to share a particular label. Viewing only a group of similarly labeled emails then becomes a simple user interface trick. Many other Web 2.0 sites have similar mechanisms—consider the “tags” used by sites such as YouTube and Flickr, “categories” applied to blog posts, and so on. Folks understand today that organization of data is critical, but that how that data is organized needs to be a flexible concept. The old files-and-folders paradigm is too rigid for some applications.

Subversion's changelist support allows you to create changelists by applying labels to files you want to be associated with that changelist, remove those labels, and limit the scope of the files on which its subcommands operate to only those bearing a particular label. In this section, we'll look in detail at how to do these things.

3.10.1. 创建和更新修改列表

You can create, modify, and delete changelists using the **svn changelist** command. More accurately, you use this command to set or unset the changelist association of a particular working copy file. A changelist is effectively created the first time you label a file with that changelist; it is deleted when you remove that label from the last file that had it. Let's examine a usage scenario that demonstrates these concepts.

Harry is fixing some bugs in the calculator application's mathematics logic. His work leads him to change a couple of files:

```
$ svn status
M       integer.c
M       mathops.c
$
```

While testing his bug fix, Harry notices that his changes bring to light a tangentially related bug in the user interface logic found in `button.c`. Harry decides that he'll go ahead and fix that bug, too, as a separate commit from his math fixes. Now, in a small working copy with only a handful of files and few logical changes, Harry can probably keep his two logical change groupings mentally organized without any problem. But today he's going to use Subversion's changelists feature as a special favor to the authors of this book.

Harry first creates a changelist and associates with it the two files he's already changed. He does this by using the **svn changelist** command to assign the same arbitrary changelist name to those files:

```
$ svn changelist math-fixes integer.c mathops.c
Path 'integer.c' is now a member of changelist 'math-fixes'.
Path 'mathops.c' is now a member of changelist 'math-fixes'.
$ svn status
```

```
--- Changelist 'math-fixes':
M      integer.c
M      mathops.c
$
```

就像你看到的，你的**svn status**反映了新的分组。

Harry now sets off to fix the secondary UI problem. Since he knows which file he'll be changing, he assigns that path to a changelist, too. Unfortunately, Harry carelessly assigns this third file to the same changelist as the previous two files:

```
$ svn changelist math-fixes button.c
Path 'button.c' is now a member of changelist 'math-fixes'.
$ svn status

--- Changelist 'math-fixes':
       button.c
M      integer.c
M      mathops.c
$
```

Fortunately, Harry catches his mistake. At this point, he has two options. He can remove the changelist association from `button.c`, and then assign a different changelist name:

```
$ svn changelist --remove button.c
Path 'button.c' is no longer a member of a changelist.
$ svn changelist ui-fix button.c
Path 'button.c' is now a member of changelist 'ui-fix'.
$
```

Or, he can skip the removal and just assign a new changelist name. In this case, Subversion will first warn Harry that `button.c` is being removed from the first changelist:

```
$ svn changelist ui-fix button.c
svn: warning: Removing 'button.c' from changelist 'math-fixes'.
Path 'button.c' is now a member of changelist 'ui-fix'.
$ svn status

--- Changelist 'ui-fix':
       button.c

--- Changelist 'math-fixes':
M      integer.c
M      mathops.c
$
```

Harry now has two distinct changelists present in his working copy, and **svn status** will group its output according to these changelist determinations. Notice that even though Harry hasn't yet modified `button.c`, it still shows up in the output of **svn status** as interesting because it has a changelist assignment. Changelists can be added to and removed from files at any time, regardless of whether they contain local modifications.

Harry now fixes the user interface problem in `button.c`.

```
$ svn status

--- Changelist 'ui-fix':
M      button.c

--- Changelist 'math-fixes':
M      integer.c
M      mathops.c
$
```

3.10.2. 用修改列表作为操作过滤器

The visual grouping that Harry sees in the output of **svn status** as shown in our previous section is nice, but not entirely useful. The **status** command is but one of many operations that he might wish to perform on his working copy. Fortunately, many of Subversion's other operations understand how to operate on changelists via the use of the `--changelist` option.

When provided with a `--changelist` option, Subversion commands will limit the scope of their operation to only those files to which a particular changelist name is assigned. If Harry now wants to see the actual changes he's made to the files in his `math-fixes` changelist, he *could* explicitly list only the files that make up that changelist on the **svn diff** command line.

```
$ svn diff integer.c mathops.c
Index: integer.c
===================================================================
--- integer.c (revision 1157)
+++ integer.c (working copy)
…
Index: mathops.c
===================================================================
--- mathops.c (revision 1157)
+++ mathops.c (working copy)
…
$
```

That works okay for a few files, but what if Harry's change touched 20 or 30 files? That would be an annoyingly long list of explicitly named files. Now that he's using changelists, though, Harry can avoid explicitly listing the set of files in his changelist from now on, and instead provide just the changelist name:

```
$ svn diff --changelist math-fixes
Index: integer.c
===================================================================
--- integer.c (revision 1157)
+++ integer.c (working copy)
…
Index: mathops.c
===================================================================
--- mathops.c (revision 1157)
+++ mathops.c (working copy)
…
$
```

And when it's time to commit, Harry can again use the `--changelist` option to limit the scope of the commit to files in a certain changelist. He might commit his user interface fix by doing the following:

```
$ svn ci -m "Fix a UI bug found while working on math logic." \
      --changelist ui-fix
Sending        button.c
Transmitting file data .
Committed revision 1158.
$
```

In fact, the **svn commit** command provides a second changelists-related option: `--keep-changelists`. Normally, changelist assignments are removed from files after they are committed. But if `--keep-changelists` is provided, Subversion will leave the changelist assignment on the committed (and now unmodified) files. In any case, committing files assigned to one changelist leaves other changelists undisturbed.

```
$ svn status

--- Changelist 'math-fixes':
M      integer.c
M      mathops.c
$
```

注意

The `--changelist` option acts only as a filter for Subversion command targets, and will not add targets to an operation. For example, on a commit operation specified as **`svn commit /path/to/dir`**, the target is the directory `/path/to/dir` and its children (to infinite depth). If you then add a changelist specifier to that command, only those files in and under `/path/to/dir` that are assigned that changelist name will be considered as targets of the commit—the commit will not include files located elsewhere (such is in `/path/to/another-dir`), regardless of their changelist assignment, even if they are part of the same working copy as the operation's target(s).

Even the **svn changelist** command accepts the `--changelist` option. This allows you to quickly and easily rename or remove a changelist:

```
$ svn changelist math-bugs --changelist math-fixes --depth infinity .
svn: warning: Removing 'integer.c' from changelist 'math-fixes'.
Path 'integer.c' is now a member of changelist 'math-bugs'.
svn: warning: Removing 'mathops.c' from changelist 'math-fixes'.
Path 'mathops.c' is now a member of changelist 'math-bugs'.
$ svn changelist --remove --changelist math-bugs --depth infinity .
Path 'integer.c' is no longer a member of a changelist.
Path 'mathops.c' is no longer a member of a changelist.
$
```

Finally, you can specify multiple instances of the `--changelist` option on a single command line. Doing so limits the operation you are performing to files found in any of the specified changesets.

3.10.3. 修改列表的限制

Subversion's changelist feature is a handy tool for grouping working copy files, but it does have a few limitations. Changelists are artifacts of a particular working copy, which means that changelist assignments cannot be propagated to the repository or otherwise shared with other users. Changelists can be assigned only to files—Subversion doesn't currently support the use of changelists with directories. Finally, you can have at most one changelist assignment on a given working copy file. Here is where the blog post category and photo service tag analogies break down—if you find yourself needing to assign a file to multiple changelists, you're out of luck.

3.11. 网络模型

在某些情况下，你需要理解Subversion客户端如何与服务器通讯。Subversion网络层是抽象的，意味着Subversion客户端不管其操作的对象都会使用相同的行为方式，不管是使用HTTP协议(`http://`)与Apache HTTP服务器通讯或是使用自定义Subversion协议(`svn://`)与**svnserve**通讯，基本的网络模型是相同的。在本小节，我们要解释网络模型基础，包括Subversion如何管理认证和授权信息。

3.11.1. 请求和响应

The Subversion client spends most of its time managing working copies. When it needs information from a remote repository, however, it makes a network request, and the server responds with an appropriate answer. The details of the network protocol are hidden from the user—the client attempts to access a URL, and depending on the URL scheme, a particular protocol is used to contact the server (see the sidebar 版本库的 URL).

提示

Run **`svn --version`** to see which URL schemes and protocols the client knows how to use.

When the server process receives a client request, it often demands that the client identify itself. It issues an authentication challenge to the client, and the client responds by providing *credentials* back to the server. Once authentication is complete, the server responds with the original information that the client asked for. Notice that this system is different from systems such as CVS, where the client preemptively offers credentials (“logs in”) to the server before ever making a request. In Subversion, the server “pulls” credentials by challenging the client at the appropriate moment, rather than the client “pushing” them. This makes certain operations more elegant. For example, if a server is configured to allow anyone in the world to read a repository, the server will never issue an authentication challenge when a client attempts to **svn checkout**.

If the particular network requests issued by the client result in a new revision being created in the repository (e.g., **svn commit**), Subversion uses the authenticated username associated with those requests as the author of the revision. That is, the authenticated user's name is stored as the value of the `svn:author` property on the new revision (see 第 9.10 节 “Subversion 属性”). If the client was not authenticated (i.e., if the server never issued an authentication challenge), the revision's `svn:author` property is empty.

3.11.2. 客户端凭证缓存

Many servers are configured to require authentication on every request. This would be a big annoyance to users if they were forced to type their passwords over and over again. Fortunately, the Subversion client has a remedy for this—a built-in system for caching authentication credentials on disk. By default, whenever the command-line client successfully responds to a server's authentication challenge, it saves the credentials in the user's private runtime configuration area (`~/.subversion/auth/` on Unix-like systems or `%APPDATA%/Subversion/auth/` on Windows; see 第 7.1 节 “运行配置区” for more details about the runtime configuration system). Successful credentials are cached on disk and keyed on a combination of the server's hostname, port, and authentication realm.

When the client receives an authentication challenge, it first looks for the appropriate credentials in the user's disk cache. If seemingly suitable credentials are not present, or if the cached credentials ultimately fail to authenticate, the client will, by default, fall back to prompting the user for the necessary information.

十分关心安全的人们一定会想“把密码缓存在磁盘？太可怕了，永远不要这样做！”

Subversion开发者认识到这种关注的正确性，所以Subversion使用操作系统和环境提供的机制来减少泄露这些信息的风险，下面是在大多数平台上这种含义的列表：

- On Windows 2000 and later, the Subversion client uses standard Windows cryptography services to encrypt the password on disk. Because the encryption key is managed by Windows and is tied to the user's own login credentials, only the user can decrypt the cached password. (Note that if the user's Windows account password is reset by an administrator, all of the cached passwords become undecipherable. The Subversion client will behave as though they don't exist, prompting for passwords when required.)

- Similarly, on Mac OS X, the Subversion client stores all repository passwords in the login keyring (managed by the Keychain service), which is protected by the user's account password. User preference settings can impose additional policies, such as requiring that the user's account password be entered each time the Subversion password is used.

- For other Unix-like operating systems, no standard “keychain” services exist. However, the `auth/` caching area is still permission-protected so that only the user (owner) can read data from it, not the world at large. The operating system's own file permissions protect the passwords.

Of course, for the truly paranoid, none of these mechanisms meets the test of perfection. So for those folks willing to sacrifice convenience for the ultimate in security, Subversion provides various ways of disabling its credentials caching system altogether.

你可以关闭凭证缓存，只需要一个简单的命令，使用参数`--no-auth-cache`:

```
$ svn commit -F log_msg.txt --no-auth-cache
Authentication realm: <svn://host.example.com:3690> example realm
Username:  joe
Password for 'joe':

Adding         newfile
Transmitting file data .
Committed revision 2324.

# password was not cached, so a second commit still prompts us

$ svn delete newfile
$ svn commit -F new_msg.txt
Authentication realm: <svn://host.example.com:3690> example realm
Username:  joe
…
```

或许，你希望永远关闭凭证缓存，你可以编辑你的运行运行配置区的`config`文件，只需要把`store-auth-creds`设置为`no`，这样在影响的主机上的Subversion操作就不会有凭证缓存在磁盘。通过修改系统级的运行配置区，这个功能也会影响到本机的所有用户(详细内容见第 7.1.1 节 “配置区布局”)。

```
[auth]
store-auth-creds = no
```

有时候，用户希望从磁盘缓存删除特定的凭证，为此你可以浏览到`auth/`区域，删除特定的缓存文件，凭证都是作为一个单独的文件缓存，如果你打开每一个文件，你会看到键和值，`svn:realmstring`描述了这个文件关联的特定服务器的域:

```
$ ls ~/.subversion/auth/svn.simple/
5671adf2865e267db74f09ba6f872c28
3893ed123b39500bca8a0b382839198e
5c3c22968347b390f349ff340196ed39

$ cat ~/.subversion/auth/svn.simple/5671adf2865e267db74f09ba6f872c28

K 8
```

```
username
V 3
joe
K 8
password
V 4
blah
K 15
svn:realmstring
V 45
<https://svn.domain.com:443> Joe's repository
END
```

一旦你定位了正确的缓存文件，只需要删除它。

One last word about **svn**'s authentication behavior, specifically regarding the `--username` and `--password` options. Many client subcommands accept these options, but it is important to understand that using these options does *not* automatically send credentials to the server. As discussed earlier, the server "pulls" credentials from the client when it deems necessary; the client cannot "push" them at will. If a username and/or password are passed as options, they will be presented to the server only if the server requests them. These options are typically used to authenticate as a different user than Subversion would have chosen by default (such as your system login name) or when trying to avoid interactive prompting (such as when calling **svn** from a script).

注意

A common mistake is to misconfigure a server so that it never issues an authentication challenge. When users pass `--username` and `--password` options to the client, they're surprised to see that they're never used; that is, new revisions still appear to have been committed anonymously!

这里是Subversion客户端在收到认证请求的时候的行为方式最终总结：

1. 首先，检查用户是否通过命令选项(`--username`和/或`--password`)指定了任何凭证信息，如果没有，或者这些选项没有认证成功，然后
2. If no command-line credentials were provided, or the provided ones were invalid, the client looks up the server's hostname, port, and realm in the runtime configuration's `auth/` area, to see whether appropriate credentials are cached there. If so, it attempts to use those credentials to authenticate.
3. 最终，如果前一种机制未能够为服务器成功认证用户，客户端返回并提示用户输入正确的凭证(除非使用`--non-interactive`选项或客户端对等的方式)。

如果客户端通过以上的任何一种方式成功认证，它会尝试在磁盘缓存凭证(除非用户已经关闭了这种行为方式，在前面提到过。)

3.12. 总结

After reading this chapter, you should have a firm grasp on some of Subversion's features that, while perhaps not used *every* time you interact with your version control system, are certainly handy to know

about. But don't stop here! Read on to the following chapter, where you'll learn about branches, tags, and merging. Then you'll have nearly full mastery of the Subversion client. Though our lawyers won't allow us to promise you anything, this additional knowledge could make you measurably more cool. [12]

[12]No purchase necessary. Certains terms and conditions apply. No guarantee of coolness—implicit or otherwise—exists. Mileage may vary.

第 4 章 分支与合并

“君子务本”

—孔子

Branching, tagging, and merging are concepts common to almost all version control systems. If you're not familiar with these ideas, we provide a good introduction in this chapter. If you are familiar, hopefully you'll find it interesting to see how Subversion implements them.

Branching is a fundamental part of version control. If you're going to allow Subversion to manage your data, this is a feature you'll eventually come to depend on. This chapter assumes that you're already familiar with Subversion's basic concepts (第 1 章 基本概念).

4.1. 什么是分支?

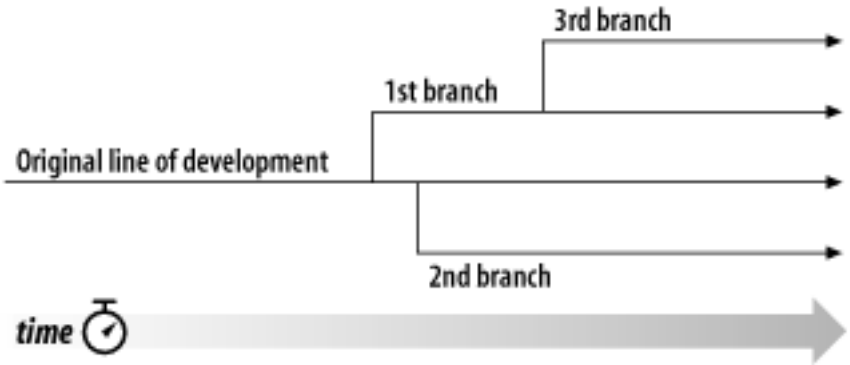
假设你的工作是维护本公司一个部门的手册文档，一天，另一个部门问你要相同的手册，但一些地方会有“区别”，因为他们有不同的需要。

这种情况下你会怎样做？显而易见的方法是：作一个版本的拷贝，然后分别维护两个版本，只要任何一个部门告诉要做一些小修改，你必须选择在对应的版本进行更改。

你也许希望在两个版本同时作修改，举个例子，你在第一个版本发现了一个拼写错误，很显然这个错误也会出现在第二个版本里。两份文档几乎相同，毕竟，只有许多特定的微小区别。

这是分支的基本概念—正如它的名字，开发的一条线独立于另一条线，如果回顾历史，可以发现两条线分享共同的历史，一个分支总是从一个备份开始的，从那里开始，发展自己独有的历史(见 图 4.1 “分支与开发”)。

图 **4.1.** 分支与开发



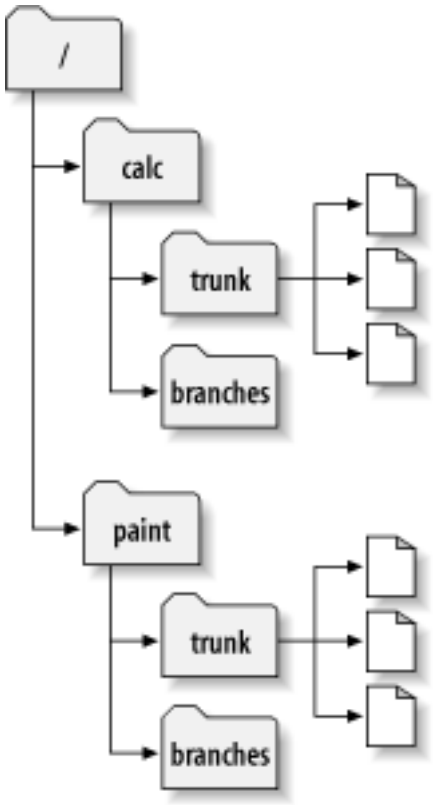
Subversion has commands to help you maintain parallel branches of your files and directories. It allows you to create branches by copying your data, and remembers that the copies are related to one another. It also helps you duplicate changes from one branch to another. Finally, it can make portions of your working copy reflect different branches so that you can “mix and match” different lines of development in your daily work.

4.2. 使用分支

At this point, you should understand how each commit creates an entirely new filesystem tree (called a “revision”) in the repository. If you don't, go back and read about revisions in 第 1.3.3 节 “修订版本”.

对于本章节，我们会回到第 1 章 基本概念的同一个例子，还记得你和你的合作者Sally分享一个包含两个项目的版本库，`paint`和`calc`。注意图 4.2 “开始规划版本库”，然而，现在每个项目的都有一个`trunk`和`branches`子目录，它们存在的理由很快就会清晰起来。

图 **4.2.** 开始规划版本库



像以前一样，假定Sally和你都有“calc”项目的一份拷贝，更准确地说，你有一份`/calc/trunk`的工作拷贝，这个项目的所有的文件在这个子目录里，而不是在`/calc`下，因为你的小组决定使用`/calc/trunk`作为开发使用的“主线”。

Let's say that you've been given the task of implementing a large software feature. It will take a long time to write, and will affect all the files in the project. The immediate problem is that you don't want to interfere with Sally, who is in the process of fixing small bugs here and there. She's depending on the fact that the latest version of the project (in `/calc/trunk`) is always usable. If you start committing your changes bit by bit, you'll surely break things for Sally (and other team members as well).

One strategy is to crawl into a hole: you and Sally can stop sharing information for a week or two. That is, start gutting and reorganizing all the files in your working copy, but don't commit or update until you're completely finished with the task. There are a number of problems with this, though. First,

it's not very safe. Most people like to save their work to the repository frequently, should something bad accidentally happen to their working copy. Second, it's not very flexible. If you do your work on different computers (perhaps you have a working copy of `/calc/trunk` on two different machines), you'll need to manually copy your changes back and forth or just do all the work on a single computer. By that same token, it's difficult to share your changes in progress with anyone else. A common software development “best practice” is to allow your peers to review your work as you go. If nobody sees your intermediate commits, you lose potential feedback and may end up going down the wrong path for weeks before another person on your team notices. Finally, when you're finished with all your changes, you might find it very difficult to remerge your final work with the rest of the company's main body of code. Sally (or others) may have made many other changes in the repository that are difficult to incorporate into your working copy—especially if you run **svn update** after weeks of isolation.

最佳方案是创建你自己的分支，或者是版本库的开发线。这允许你保存破坏了一半的工作而不打扰别人，尽管你仍可以选择性的同你的合作者分享信息，你将会看到这是怎样工作的。

4.2.1. 创建分支

Creating a branch is very simple—you make a copy of the project in the repository using the **svn copy** command. Subversion is able to copy not only single files, but whole directories as well. In this case, you want to make a copy of the `/calc/trunk` directory. Where should the new copy live? Wherever you wish—it's a matter of project policy. Let's say that your team has a policy of creating branches in the `/calc/branches` area of the repository, and you want to name your branch `my-calc-branch`. You'll want to create a new directory, `/calc/branches/my-calc-branch`, which begins its life as a copy of `/calc/trunk`.

You may already have seen **svn copy** used to copy one file to another within a working copy. But it can also be used to do a “remote” copy entirely within the repository. Just copy one URL to another:

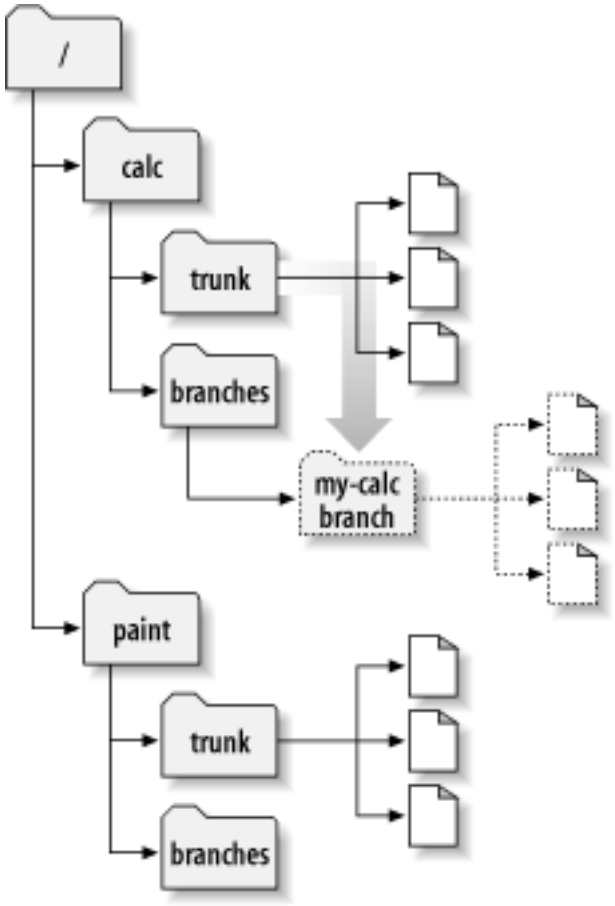
```
$ svn copy http://svn.example.com/repos/calc/trunk \
           http://svn.example.com/repos/calc/branches/my-calc-branch \
      -m "Creating a private branch of /calc/trunk."

Committed revision 341.
```

This command causes a near-instantaneous commit in the repository, creating a new directory in revision 341. The new directory is a copy of `/calc/trunk`. This is shown in 图 4.3 “版本库与复制”. [1] While it's also possible to create a branch by using **svn copy** to duplicate a directory within the working copy, this technique isn't recommended. It can be quite slow, in fact! Copying a directory on the client side is a linear-time operation, in that it actually has to duplicate every file and subdirectory on the local disk. Copying a directory on the server, however, is a constant-time operation, and it's the way most people create branches.

[1] Subversion不支持跨版本库的拷贝，当使用**svn copy**或者**svn move**直接操作URL时你只能在同一个版本库内操作。

图 **4.3.** 版本库与复制



廉价复制

Subversion's repository has a special design. When you copy a directory, you don't need to worry about the repository growing huge—Subversion doesn't actually duplicate any data. Instead, it creates a new directory entry that points to an *existing* tree. If you're an experienced Unix user, you'll recognize this as the same concept behind a hard link. As further changes are made to files and directories beneath the copied directory, Subversion continues to employ this hard link concept where it can. It duplicates data only when it is necessary to disambiguate different versions of objects.

这就是为什么经常听到Subversion用户谈论“廉价的拷贝”，与目录的大小无关—这个操作会使用很少的时间，事实上，这个特性是Subversion提交工作的基础：每一次版本都是前一个版本的一个“廉价的拷贝”，只有少数项目修改了。(要阅读更多关于这部分的内容，访问Subversion网站并且阅读设计文档中的“bubble up”方法)。

Of course, these internal mechanics of copying and sharing data are hidden from the user, who simply sees copies of trees. The main point here is that copies are cheap, both in time and in space. If you create a branch entirely within the repository (by running **`svn copy URL1 URL2`**), it's a quick, constant-time operation. Make branches as often as you want.

4.2.2. 在分支上工作

现在你已经在项目里建立分支了，你可以取出一个新的工作拷贝来开始使用：

```
$ svn checkout http://svn.example.com/repos/calc/branches/my-calc-branch
A  my-calc-branch/Makefile
A  my-calc-branch/integer.c
A  my-calc-branch/button.c
Checked out revision 341.
```

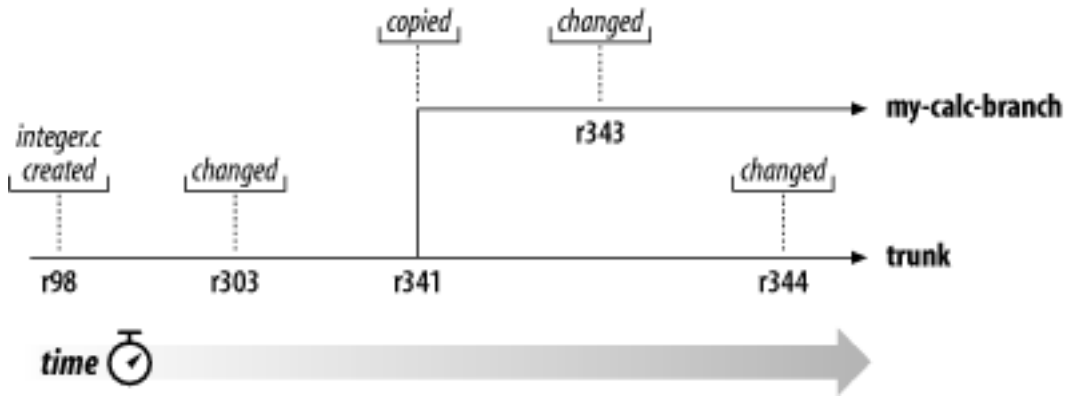
There's nothing special about this working copy; it simply mirrors a different directory in the repository. When you commit changes, however, Sally won't see them when she updates, because her working copy is of `/calc/trunk`. (Be sure to read 第 4.5 节 “使用分支” later in this chapter: the **svn switch** command is an alternative way of creating a working copy of a branch.)

我们假定本周就要过去了，如下的提交发生：

- 你修改了`/calc/branches/my-calc-branch/button.c`，生成修订版本342。
- 你修改了`/calc/branches/my-calc-branch/integer.c`，生成修订版本343。
- Sally修改了`/calc/trunk/integer.c`，生成了修订版本344。

Now two independent lines of development (shown in 图 4.4 “一个文件的分支历史”) are happening on `integer.c`.

图 **4.4.** 一个文件的分支历史



当你看到`integer.c`的改变时，你会发现很有趣：

```
$ pwd
/home/user/my-calc-branch

$ svn log -v integer.c
```

```
------------------------------------------------------------------------
r343 | user | 2002-11-07 15:27:56 -0600 (Thu, 07 Nov 2002) | 2 lines
Changed paths:
   M /calc/branches/my-calc-branch/integer.c

* integer.c:  frozzled the wazjub.

------------------------------------------------------------------------
r341 | user | 2002-11-03 15:27:56 -0600 (Thu, 07 Nov 2002) | 2 lines
Changed paths:
   A /calc/branches/my-calc-branch (from /calc/trunk:340)

Creating a private branch of /calc/trunk.

------------------------------------------------------------------------
r303 | sally | 2002-10-29 21:14:35 -0600 (Tue, 29 Oct 2002) | 2 lines
Changed paths:
   M /calc/trunk/integer.c

* integer.c:  changed a docstring.

------------------------------------------------------------------------
r98 | sally | 2002-02-22 15:35:29 -0600 (Fri, 22 Feb 2002) | 2 lines
Changed paths:
   A /calc/trunk/integer.c

* integer.c:  adding this file to the project.

------------------------------------------------------------------------
```

Notice that Subversion is tracing the history of your branch's `integer.c` all the way back through time, even traversing the point where it was copied. It shows the creation of the branch as an event in the history, because `integer.c` was implicitly copied when all of `/calc/trunk/` was copied. Now look at what happens when Sally runs the same command on her copy of the file:

```
$ pwd
/home/sally/calc

$ svn log -v integer.c
------------------------------------------------------------------------
r344 | sally | 2002-11-07 15:27:56 -0600 (Thu, 07 Nov 2002) | 2 lines
Changed paths:
   M /calc/trunk/integer.c

* integer.c:  fix a bunch of spelling errors.

------------------------------------------------------------------------
r303 | sally | 2002-10-29 21:14:35 -0600 (Tue, 29 Oct 2002) | 2 lines
```

```
Changed paths:
   M /calc/trunk/integer.c

* integer.c:  changed a docstring.

------------------------------------------------------------------------
r98 | sally | 2002-02-22 15:35:29 -0600 (Fri, 22 Feb 2002) | 2 lines
Changed paths:
   A /calc/trunk/integer.c

* integer.c:  adding this file to the project.

------------------------------------------------------------------------
```

sally看到她自己的344修订，你做的343修改她看不到，从Subversion看来，两次提交只是影响版本库中不同位置上的两个文件。然而，Subversion显示了两个文件有共同的历史，在分支拷贝之前，他们使用同一个文件，所以你和Sally都看到版本号303到98的修改。

4.2.3. 分支背后的关键概念

You should remember two important lessons from this section. First, Subversion has no internal concept of a branch—it knows only how to make copies. When you copy a directory, the resultant directory is only a “branch” because *you* attach that meaning to it. You may think of the directory differently, or treat it differently, but to Subversion it's just an ordinary directory that happens to carry some extra historical information.

Second, because of this copy mechanism, Subversion's branches exist as *normal filesystem directories* in the repository. This is different from other version control systems, where branches are typically defined by adding extra-dimensional “labels” to collections of files. The location of your branch directory doesn't matter to Subversion. Most teams follow a convention of putting all branches into a `/branches` directory, but you're free to invent any policy you wish.

4.3. 基本合并

现在你与Sally在同一个项目的并行分支上工作：你在私有分支上，而Sally在主干(*trunk*)或者叫做开发主线上。

由于有众多的人参与项目，大多数人拥有主干拷贝是很正常的，任何人如果进行一个长周期的修改会使得主干陷入混乱，所以通常的做法是建立一个私有分支，提交修改到自己的分支，直到这阶段工作结束。

所以，好消息就是你和Sally不会互相打扰，坏消息是有时候分离会太远。记住“闭门造车”策略的问题，当你完成你的分支后，可能因为太多冲突，已经无法轻易合并你的分支和主干的修改。

相反，在你工作的时候你和Sally仍然可以继续分享修改，这依赖于你决定什么值得分享，Subversion给你在分支间选择性“拷贝”修改的能力，当你完成了分支上的所有工作，

所有的分支修改可以被拷贝回到主干。在Subversion术语里，这种从一个分支复制修改到另一个分支的活动称为合并，它是使用**svn merge**命令执行。

In the examples that follow, we're assuming that both your Subversion client and server are running Subversion 1.5 (or later). If either client or server is older than version 1.5, things are more complicated: the system won't track changes automatically, and you'll have to use painful manual methods to achieve similar results. That is, you'll always need to use the detailed merge syntax to specify specific ranges of revisions to replicate (see 第 4.4.2 节 “合并的语法：完整的描述” later in this chapter), and take special care to keep track of what's already been merged and what hasn't. For this reason, we *strongly* recommend that you make sure your client and server are at least at version 1.5.

4.3.1. 变更集

Before we proceed further, we should warn you that there's going to be a lot of discussion of “changes” in the pages ahead. A lot of people experienced with version control systems use the terms “change” and “changeset” interchangeably, and we should clarify what Subversion understands as a *changeset*.

Everyone seems to have a slightly different definition of changeset, or at least a different expectation of what it means for a version control system to have one. For our purposes, let's say that a changeset is just a collection of changes with a unique name. The changes might include textual edits to file contents, modifications to tree structure, or tweaks to metadata. In more common speak, a changeset is just a patch with a name you can refer to.

In Subversion, a global revision number N names a tree in the repository: it's the way the repository looked after the Nth commit. It's also the name of an implicit changeset: if you compare tree N with tree N−1, you can derive the exact patch that was committed. For this reason, it's easy to think of revision N as not just a tree, but a changeset as well. If you use an issue tracker to manage bugs, you can use the revision numbers to refer to particular patches that fix bugs—for example, “this issue was fixed by r9238.” Somebody can then run **`svn log -r 9238`** to read about the exact changeset that fixed the bug, and run **`svn diff -c 9238`** to see the patch itself. And (as you'll see shortly) Subversion's **svn merge** command is able to use revision numbers. You can merge specific changesets from one branch to another by naming them in the merge arguments: passing **`-c 9238`** to **svn merge** would merge changeset r9238 into your working copy.

4.3.2. 保持分支同步

Continuing with our running example, let's suppose that a week has passed since you started working on your private branch. Your new feature isn't finished yet, but at the same time you know that other people on your team have continued to make important changes in the project's `/trunk`. It's in your best interest to replicate those changes to your own branch, just to make sure they mesh well with your changes. In fact, this is a best practice: frequently keeping your branch in sync with the main development line helps prevent “surprise” conflicts when it comes time for you to fold your changes back into the trunk.

Subversion is aware of the history of your branch and knows when it divided away from the trunk. To replicate the latest, greatest trunk changes to your branch, first make sure your working copy of the branch is “clean”—that it has no local modifications reported by **svn status**. Then simply run:

```
$ pwd
/home/user/my-calc-branch

$ svn merge http://svn.example.com/repos/calc/trunk
--- Merging r345 through r356 into '.':
U    button.c
U    integer.c
```

This basic syntax—**svn merge *URL***—tells Subversion to merge all recent changes from the URL to the current working directory (which is typically the root of your working copy). After running the prior example, your branch working copy now contains new local modifications, and these edits are duplications of all of the changes that have happened on the trunk since you first created your branch:

```
$ svn status
 M     .
M      button.c
M      integer.c
```

At this point, the wise thing to do is look at the changes carefully with **svn diff**, and then build and test your branch. Notice that the current working directory ("`.`") has also been modified; the **svn diff** will show that its `svn:mergeinfo` property has been either created or modified. This is important merge-related metadata that you should *not* touch, since it will be needed by future **svn merge** commands. (We'll learn more about this metadata later in the chapter.)

After performing the merge, you might also need to resolve some conflicts (just as you do with **svn update**) or possibly make some small edits to get things working properly. (Remember, just because there are no *syntactic* conflicts doesn't mean there aren't any *semantic* conflicts!) If you encounter serious problems, you can always abort the local changes by running **svn revert . -R** (which will undo all local modifications) and start a long "what's going on?" discussion with your collaborators. If things look good, however, you can submit these changes into the repository:

```
$ svn commit -m "Merged latest trunk changes to my-calc-branch."
Sending        .
Sending        button.c
Sending        integer.c
Transmitting file data ..
Committed revision 357.
```

At this point, your private branch is now "in sync" with the trunk, so you can rest easier knowing that as you continue to work in isolation, you're not drifting too far away from what everyone else is doing.

为什么不使用补丁?

也许你的脑中会出现一个问题，特别如果你是Unix用户，为什么非要使用**svn merge**？为什么不简单的使用操作系统的**patch**命令来进行相同的工作？例如：

```
$ cd my-calc-branch
$ svn diff -r 341:HEAD http://svn.example.com/repos/calc/trunk > patchfi
$ patch -p0  < patchfile
Patching file integer.c using Plan A...
Hunk #1 succeeded at 147.
Hunk #2 succeeded at 164.
Hunk #3 succeeded at 241.
Hunk #4 succeeded at 249.
done
```

在这种情况下，确实没有区别，但是**svn merge**有超越**patch**的特别能力，使用**patch**对文件格式有一定的限制，它只能针对文件内容，没有方法表现目录树的修改，例如添加、删除或是改名。如果Sally的修改包括增加一个新的目录，**svn diff**不会注意到这些，**svn diff**只会输出有限的补丁格式，所以有些问题无法表达。

The **svn merge** command, however, can express changes in tree structure and properties by directly applying them to your working copy. Even more important, this command records the changes that have been duplicated to your branch so that Subversion is aware of exactly which changes exist in each location (see 第 4.3.3 节 “合并信息和预览”.) This is a critical feature that makes branch management usable; without it, users would have to manually keep notes on which sets of changes have or haven't been merged yet.

Suppose that another week has passed. You've committed more changes to your branch, and your comrades have continued to improve the trunk as well. Once again, you'd like to replicate the latest trunk changes to your branch and bring yourself in sync. Just run the same merge command again!

```
$ svn merge http://svn.example.com/repos/calc/trunk
--- Merging r357 through r380 into '.':
U    integer.c
U    Makefile
A    README
```

Subversion knows which trunk changes you've already replicated to your branch, so it carefully replicates only those changes you don't yet have. Once again, you'll have to build, test, and **svn commit** the local modifications to your branch.

What happens when you finally finish your work, though? Your new feature is done, and you're ready to merge your branch changes back to the trunk (so your team can enjoy the bounty of your labor). The process is simple. First, bring your branch in sync with the trunk again, just as you've been doing all along:

```
$ svn merge http://svn.example.com/repos/calc/trunk
```

```
--- Merging r381 through r385 into '.':
U    button.c
U    README

$ # build, test, ...

$ svn commit -m "Final merge of trunk changes to my-calc-branch."
Sending        .
Sending        button.c
Sending        README
Transmitting file data ..
Committed revision 390.
```

Now, you use **svn merge** to replicate your branch changes back into the trunk. You'll need an up-to-date working copy of `/trunk`. You can do this by either doing an **svn checkout**, dredging up an old trunk working copy from somewhere on your disk, or using **svn switch** (see 第 4.5 节 “使用分支”.) However you get a trunk working copy, remember that it's a best practice to do your merge into a working copy that has *no* local edits and has been recently updated (i.e., is not a mixture of local revisions). If your working copy isn't “clean” in these ways, you can run into some unnecessary conflict-related headaches and **svn merge** will likely return an error.

Once you have a clean working copy of the trunk, you're ready to merge your branch back into it:

```
$ pwd
/home/user/calc-trunk

$ svn update  # (make sure the working copy is up to date)
At revision 390.

$ svn merge --reintegrate http://svn.example.com/repos/calc/branches/my-ca
--- Merging differences between repository URLs into '.':
U    button.c
U    integer.c
U    Makefile
 U   .

$ # build, test, verify, ...

$ svn commit -m "Merge my-calc-branch back into trunk!"
Sending        .
Sending        button.c
Sending        integer.c
Sending        Makefile
Transmitting file data ..
Committed revision 391.
```

Congratulations, your branch has now been remerged back into the main line of development. Notice our use of the `--reintegrate` option this time around. The option is critical for reintegrating

changes from a branch back into its original line of development—don't forget it! It's needed because this sort of “merge back” is a different sort of work than what you've been doing up until now. Previously, we had been asking **svn merge** to grab the “next set” of changes from one line of development (the trunk) and duplicate them to another (your branch). This is fairly straightforward, and each time Subversion knows how to pick up where it left off. In our prior examples, you can see that first it merges the ranges 345:356 from trunk to branch; later on, it continues by merging the next contiguously available range, 356:380. When doing the final sync, it merges the range 380:385.

When merging your branch back to the trunk, however, the underlying mathematics is quite different. Your feature branch is now a mishmosh of both duplicated trunk changes and private branch changes, so there's no simple contiguous range of revisions to copy over. By specifying the `--reintegrate` option, you're asking Subversion to carefully replicate *only* those changes unique to your branch. (And in fact, it does this by comparing the latest trunk tree with the latest branch tree: the resulting difference is exactly your branch changes!)

Now that your private branch is merged to trunk, you may wish to remove it from the repository:

```
$ svn delete http://svn.example.com/repos/calc/branches/my-calc-branch \
      -m "Remove my-calc-branch."
Committed revision 392.
```

But wait! Isn't the history of that branch valuable? What if somebody wants to audit the evolution of your feature someday and look at all of your branch changes? No need to worry. Remember that even though your branch is no longer visible in the `/branches` directory, its existence is still an immutable part of the repository's history. A simple **svn log** command on the `/branches` URL will show the entire history of your branch. Your branch can even be resurrected at some point, should you desire (see 第 4.3.5 节 “找回删除的项目”).

In Subversion 1.5, once a `--reintegrate` merge is done from branch to trunk, the branch is no longer usable for further work. It's not able to correctly absorb new trunk changes, nor can it be properly reintegrated to trunk again. For this reason, if you want to keep working on your feature branch, we recommend destroying it and then re-creating it from the trunk:

```
$ svn delete http://svn.example.com/repos/calc/branches/my-calc-branch \
      -m "Remove my-calc-branch."
Committed revision 392.

$ svn copy http://svn.example.com/repos/calc/trunk \
           http://svn.example.com/repos/calc/branches/new-branch
      -m "Create a new branch from trunk."
Committed revision 393.

$ cd my-calc-branch

$ svn switch http://svn.example.com/repos/calc/branches/new-branch
Updated to revision 393.
```

The final command in the prior example—**svn switch**—is a way of updating an existing working copy to reflect a different repository directory. We'll discuss this more in 第 4.5 节 “使用分支”.

4.3.3. 合并信息和预览

The basic mechanism Subversion uses to track changesets—that is, which changes have been merged to which branches—is by recording data in properties. Specifically, merge data is tracked in the `svn:mergeinfo` property attached to files and directories. (If you're not familiar with Subversion properties, now is the time to skim 第 3.2 节 “属性”.)

You can examine the property, just like any other:

```
$ cd my-calc-branch
$ svn propget svn:mergeinfo .
/trunk:341-390
```

It is *not* recommended that you change the value of this property yourself, unless you really know what you're doing. This property is automatically maintained by Subversion whenever you run **svn merge**. Its value indicates which changes (at a given path) have been replicated into the directory in question. In this case, the path is `/trunk` and the directory which has received the specific changes is `/branches/my-calc-branch`.

There's also a subcommand, **svn mergeinfo**, which can be helpful in seeing not only which changesets a directory has absorbed, but also which changesets it's still eligible to receive. This gives a sort of preview of the next set of changes that **svn merge** will replicate to your branch.

```
$ cd my-calc-branch

# Which changes have already been merged from trunk to branch?
$ svn mergeinfo http://svn.example.com/repos/calc/trunk
r341
r342
r343
…
r388
r389
r390

# Which changes are still eligible to merge from trunk to branch?
$ svn mergeinfo http://svn.example.com/repos/calc/trunk --show-revs eligi
r391
r392
r393
r394
r395
```

The **svn mergeinfo** command requires a “source” URL (where the changes would be coming from), and takes an optional “target” URL (where the changes would be merged to). If no target URL is

given, it assumes that the current working directory is the target. In the prior example, because we're querying our branch working copy, the command assumes we're interested in receiving changes to `/branches/mybranch` from the specified trunk URL.

Another way to get a more precise preview of a merge operation is to use the `--dry-run` option:

```
$ svn merge http://svn.example.com/repos/calc/trunk --dry-run
U    integer.c

$ svn status
#  nothing printed, working copy is still unchanged.
```

The `--dry-run` option doesn't actually apply any local changes to the working copy. It shows only status codes that *would* be printed in a real merge. It's useful for getting a “high-level” preview of the potential merge, for those times when running **svn diff** gives too much detail.

提示

After performing a merge operation, but before committing the results of the merge, you can use **`svn diff --depth=empty /path/to/merge/target`** to see only the changes to the immediate target of your merge. If your merge target was a directory, only property differences will be displayed. This is a handy way to see the changes to the `svn:mergeinfo` property recorded by the merge operation, which will remind you about what you've just merged.

Of course, the best way to preview a merge operation is to just do it! Remember, running **svn merge** isn't an inherently risky thing (unless you've made local modifications to your working copy—but we've already stressed that you shouldn't be merging into such an environment). If you don't like the results of the merge, simply run **`svn revert . -R`** to revert the changes from your working copy and retry the command with different options. The merge isn't final until you actually **svn commit** the results.

提示

While it's perfectly fine to experiment with merges by running **svn merge** and **svn revert** over and over, you may run into some annoying (but easily bypassed) roadblocks. For example, if the merge operation adds a new file (i.e., schedules it for addition), **svn revert** won't actually remove the file; it simply unschedules the addition. You're left with an unversioned file. If you then attempt to run the merge again, you may get conflicts due to the unversioned file “being in the way.” Solution? After performing a revert, be sure to clean up the working copy and remove unversioned files and directories. The output of **svn status** should be as clean as possible, ideally showing no output.

4.3.4. 取消修改

svn merge另一个常用的做法是取消已经做得提交，假设你愉快的在`/calc/trunk`工作，你发现303版本对`integer.c`的修改完全错了，它不应该被提交，你可以使用**svn merge**来“取消”这个工作拷贝上所作的操作，然后提交本地修改到版本库，你要做得只是指定一个相反的区别。(你可以通过指定`--revision 303:302--change -303`

```
$ svn merge -c -303 http://svn.example.com/repos/calc/trunk
--- Reverse-merging r303 into 'integer.c':
U    integer.c

$ svn status
 M      .
M       integer.c

$ svn diff
…
# verify that the change is removed
…

$ svn commit -m "Undoing change committed in r303."
Sending        integer.c
Transmitting file data .
Committed revision 350.
```

As we mentioned earlier, one way to think about a repository revision is as a specific changeset. By using the `-r` option, you can ask **svn merge** to apply a changeset, or a whole range of changesets, to your working copy. In our case of undoing a change, we're asking **svn merge** to apply changeset #303 to our working copy *backward*.

记住回滚修改和任何一个**svn merge**命令都一样，所以你应该使用**svn status**或是**svn diff**来确定你的工作处于期望的状态中，然后使用**svn commit**来提交，提交之后，这个特定修改集不会反映到HEAD版本了。

Again, you may be thinking: well, that really didn't undo the commit, did it? The change still exists in revision 303. If somebody checks out a version of the `calc` project between revisions 303 and 349, she'll still see the bad change, right?

是的，这是对的。当我们说“删除”一个修改时，我们只是说从HEAD删除，原始的修改还保存在版本库历史中，在多数情况下，这是足够好的。大多数人只是对追踪HEAD版本感兴趣，在一些特定情况下，你也许希望毁掉所有提交的证据(或许某个人提交了一个秘密文件)，这不是很容易的，因为Subversion设计用来不丢失任何信息，每个修订版本都是依赖其它修订版本的不可变目录树 ，从历史删除一个版本会导致多米诺效应，会在后面的版本导致混乱甚至会影响所有的工作拷贝。[2]

4.3.5. 找回删除的项目

版本控制系统非常重要的一个特性就是它的信息从不丢失，即使当你删除了文件或目录，它也许从HEAD版本消失了 ，但这个对象依然存在于历史的早期版本 ，一个新手经常问到的问题是“怎样找回我的文件和目录？”。

The first step is to define exactly *which* item you're trying to resurrect. Here's a useful metaphor: you can think of every object in the repository as existing in a sort of two-dimensional coordinate system.

[2]Subversion项目有计划，不管用什么方式，会有一天要实现**svnadmin obliterate**命令来进行永久删除操作，而此时可以看第 5.4.1.3 节 “svndumpfilter”找到可行的方案。

The first coordinate is a particular revision tree, and the second coordinate is a path within that tree. So every version of your file or directory can be defined by a specific coordinate pair. (Remember the "peg revision" syntax—foo.c@224—mentioned back in 第 3.9 节 "Peg 和实施修订版本" .)

First, you might need to use **svn log** to discover the exact coordinate pair you wish to resurrect. A good strategy is to run **`svn log --verbose`** in a directory that used to contain your deleted item. The `--verbose` (`-v`) option shows a list of all changed items in each revision; all you need to do is find the revision in which you deleted the file or directory. You can do this visually, or by using another tool to examine the log output (via **grep**, or perhaps via an incremental search in an editor).

```
$ cd parent-dir
$ svn log -v
…
------------------------------------------------------------------------
r808 | joe | 2003-12-26 14:29:40 -0600 (Fri, 26 Dec 2003) | 3 lines
Changed paths:
   D /calc/trunk/real.c
   M /calc/trunk/integer.c

Added fast fourier transform functions to integer.c.
Removed real.c because code now in double.c.
…
```

在这个例子里，你可以假定你正在找已经删除了的文件`real.c`，通过查找父目录的历史 ，你知道这个文件在808版本被删除，所以存在这个对象的版本在此之前 。结论：你想从版本807找回`/calc/trunk/real.c`。

以上是最重要的部分—重新找到你需要恢复的对象。现在你已经知道该恢复的文件，而你有两种选择。

One option is to use **svn merge** to apply revision 808 "in reverse." (We already discussed how to undo changes in 第 4.3.4 节 "取消修改" .) This would have the effect of re-adding `real.c` as a local modification. The file would be scheduled for addition, and after a commit, the file would again exist in `HEAD`.

In this particular example, however, this is probably not the best strategy. Reverse-applying revision 808 would not only schedule `real.c` for addition, but the log message indicates that it would also undo certain changes to `integer.c`, which you don't want. Certainly, you could reverse-merge revision 808 and then **svn revert** the local modifications to `integer.c`, but this technique doesn't scale well. What if 90 files were changed in revision 808?

A second, more targeted strategy is not to use **svn merge** at all, but rather to use the **svn copy** command. Simply copy the exact revision and path "coordinate pair" from the repository to your working copy:

```
$ svn copy http://svn.example.com/repos/calc/trunk/real.c@807 ./real.c

$ svn status
A  +   real.c
```

```
$ svn commit -m "Resurrected real.c from revision 807, /calc/trunk/real.c.
Adding         real.c
Transmitting file data .
Committed revision 1390.
```

加号标志表明这个项目不仅仅是计划增加中，而且还“包含了历史”，Subversion记住了它是从哪个拷贝过来的。在将来，对这个文件运行**svn log**会看到这个文件在版本807之前的历史，换句话说，`real.c`不是新的，而是原先删除的那一个的后代。这看起来是一个有用的功能，然而如果你希望不包含历史链接的恢复一个文件，这个技巧工作依然良好：

```
$ svn cat http://svn.example.com/repos/calc/trunk/real.c@807 > ./real.c

$ svn add real.c
A         real.c

$ svn commit -m "Re-created real.c from revision 807."
Adding         real.c
Transmitting file data .
Committed revision 1390.
```

尽管我们的例子告诉我们如何找回文件，对于恢复删除的目录也是一样的。也需要注意的是恢复不必发生在你的工作拷贝—完全可以在版本库发生：

```
$ svn copy http://svn.example.com/repos/calc/trunk/real.c@807 \
           http://svn.example.com/repos/calc/trunk/ \
      -m "Resurrect real.c from revision 807."
Committed revision 1390.

$ svn update
A    real.c
Updated to revision 1390.
```

4.4. 高级合并

Here ends the automated magic. Sooner or later, once you get the hang of branching and merging, you're going to have to ask Subversion to merge *specific* changes from one place to another. To do this, you're going to have to start passing more complicated arguments to **svn merge**. The next section describes the fully expanded syntax of the command and discusses a number of common scenarios that require it.

4.4.1. 摘录合并

Just as the term “changeset” is often used in version control systems, so is the term *cherrypicking*. This word refers to the act of choosing *one* specific changeset from a branch and replicating it to another. Cherrypicking may also refer to the act of duplicating a particular set of (not necessarily contiguous!)

changesets from one branch to another. This is in contrast to more typical merging scenarios, where the “next” contiguous range of revisions is duplicated automatically.

Why would people want to replicate just a single change? It comes up more often than you'd think. For example, let's go back in time and imagine that you haven't yet merged your private feature branch back to the trunk. At the water cooler, you get word that Sally made an interesting change to `integer.c` on the trunk. Looking over the history of commits to the trunk, you see that in revision 355 she fixed a critical bug that directly impacts the feature you're working on. You might not be ready to merge all the trunk changes to your branch just yet, but you certainly need that particular bug fix in order to continue your work.

```
$ svn diff -c 355 http://svn.example.com/repos/calc/trunk

Index: integer.c
===================================================================
--- integer.c (revision 354)
+++ integer.c (revision 355)
@@ -147,7 +147,7 @@
     case 6:  sprintf(info->operating_system, "HPFS (OS/2 or NT)"); break;
     case 7:  sprintf(info->operating_system, "Macintosh"); break;
     case 8:  sprintf(info->operating_system, "Z-System"); break;
-    case 9:  sprintf(info->operating_system, "CP/MM");
+    case 9:  sprintf(info->operating_system, "CP/M"); break;
     case 10:  sprintf(info->operating_system, "TOPS-20"); break;
     case 11:  sprintf(info->operating_system, "NTFS (Windows NT)"); break;
     case 12:  sprintf(info->operating_system, "QDOS"); break;
```

Just as you used **svn diff** in the prior example to examine revision 355, you can pass the same option to **svn merge**:

```
$ svn merge -c 355 http://svn.example.com/repos/calc/trunk
U    integer.c

$ svn status
M      integer.c
```

You can now go through the usual testing procedures before committing this change to your branch. After the commit, Subversion marks r355 as having been merged to the branch so that future “magic” merges that synchronize your branch with the trunk know to skip over r355. (Merging the same change to the same branch almost always results in a conflict!)

```
$ cd my-calc-branch

$ svn propget svn:mergeinfo .
/trunk:341-349,355
```

```
# Notice that r355 isn't listed as "eligible" to merge, because
# it's already been merged.
$ svn mergeinfo http://svn.example.com/repos/calc/trunk --show-revs eligib
r350
r351
r352
r353
r354
r356
r357
r358
r359
r360

$ svn merge http://svn.example.com/repos/calc/trunk
--- Merging r350 through r354 into '.':
 U    .
U    integer.c
U    Makefile
--- Merging r356 through r360 into '.':
 U    .
U    integer.c
U    button.c
```

This use case of replicating (or *backporting*) bug fixes from one branch to another is perhaps the most popular reason for cherrypicking changes; it comes up all the time, for example, when a team is maintaining a “release branch” of software. (We discuss this pattern in 第 4.8.1 节 “发布分支”.)

警告

Did you notice how, in the last example, the merge invocation caused two distinct ranges of merges to be applied? The **svn merge** command applied two independent patches to your working copy to skip over changeset 355, which your branch already contained. There's nothing inherently wrong with this, except that it has the potential to make conflict resolution trickier. If the first range of changes creates conflicts, you *must* resolve them interactively for the merge process to continue and apply the second range of changes. If you postpone a conflict from the first wave of changes, the whole merge command will bail out with an error message. [3]

A word of warning: while **svn diff** and **svn merge** are very similar in concept, they do have different syntax in many cases. Be sure to read about them in 第 9 章 *Subversion* 完全参考 for details, or ask **svn help**. For example, **svn merge** requires a working copy path as a target, that is, a place where it should apply the generated patch. If the target isn't specified, it assumes you are trying to perform one of the following common operations:

- 你希望合并目录修改到工作拷贝的当前目录。
- 你希望合并修改到你的当前工作目录的相同文件名的文件。

[3] At least, this is true in Subversion 1.5 at the time of this writing. This behavior may improve in future versions of Subversion.

如果你合并一个目录而没有指定特定的目标，**svn merge**假定第一种情况，在你的当前目录应用修改。如果你合并一个文件，而这个文件(或是一个有相同的名字文件)在你的当前工作目录存在，**svn merge**假定第二种情况，你想对这个同名文件使用合并。

4.4.2. 合并的语法：完整的描述

你已经看到了**svn merge**命令的例子，你将会看到更多，如果你对合并是如何工作的感到迷惑，这并不奇怪，很多人和你一样。许多新用户(特别是对版本控制很陌生的用户)会对这个命令的正确语法感到不知所措，不知道怎样和什么时候使用这个特性，不要害怕，这个命令实际上比你想象的简单！有一个简单的技巧来帮助你理解**svn merge**的行为。

The main source of confusion is the *name* of the command. The term “merge” somehow denotes that branches are combined together, or that some sort of mysterious blending of data is going on. That's not the case. A better name for the command might have been **svn diff-and-apply**, because that's all that happens: two repository trees are compared, and the differences are applied to a working copy.

If you're using **svn merge** to do basic copying of changes between branches, it will generally do the right thing automatically. For example, a command such as the following:

```
$ svn merge http://svn.example.com/repos/calc/some-branch
```

will attempt to duplicate any changes made on `some-branch` into your current working directory, which is presumably a working copy that shares some historical connection to the branch. The command is smart enough to only duplicate changes that your working copy doesn't yet have. If you repeat this command once a week, it will only duplicate the “newest” branch changes that happened since you last merged.

If you choose to use the **svn merge** command in all its full glory by giving it specific revision ranges to duplicate, the command takes three main arguments:

1. 初始的版本树(通常叫做比较的左边)，

2. 最终的版本树(通常叫做比较的右边)，

3. 一个接收区别的工作拷贝(通常叫做合并的目标)。

Once these three arguments are specified, the two trees are compared, and the differences are applied to the target working copy as local modifications. When the command is done, the results are no different than if you had hand-edited the files or run various **svn add** or **svn delete** commands yourself. If you like the results, you can commit them. If you don't like the results, you can simply **svn revert** all of the changes.

svn merge的语法允许非常灵活的指定三个必要的参数，如下是一些例子：

```
$ svn merge http://svn.example.com/repos/branch1@150 \
            http://svn.example.com/repos/branch2@212 \
            my-working-copy
```

```
$ svn merge -r 100:200 http://svn.example.com/repos/trunk my-working-copy

$ svn merge -r 100:200 http://svn.example.com/repos/trunk
```

The first syntax lays out all three arguments explicitly, naming each tree in the form *URL@REV* and naming the working copy target. The second syntax can be used as a shorthand for situations when you're comparing two different revisions of the same URL. The last syntax shows how the working copy argument is optional; if omitted, it defaults to the current directory.

While the first example shows the "full" syntax of **svn merge**, it needs to be used very carefully; it can result in merges which do not record any `svn:mergeinfo` metadata at all. The next section talks a bit more about this.

4.4.3. 不使用合并信息的合并

Subversion tries to generate merge metadata whenever it can, to make future invocations of **svn merge** smarter. There are still situations, however, where `svn:mergeinfo` data is not created or changed. Remember to be a bit wary of these scenarios:

合并无关的源
: If you ask **svn merge** to compare two URLs that aren't related to each other, a patch will still be generated and applied to your working copy, but no merging metadata will be created. There's no common history between the two sources, and future "smart" merges depend on that common history.

Merging from foreign repositories
: While it's possible to run a command such as **`svn merge -r 100:200 http://svn.foreignproject.com/repos/trunk`**, the resultant patch will also lack any historical merge metadata. At time of this writing, Subversion has no way of representing different repository URLs within the `svn:mergeinfo` property.

Using `--ignore-ancestry`
: If this option is passed to **svn merge**, it causes the merging logic to mindlessly generate differences the same way that **svn diff** does, ignoring any historical relationships. We discuss this later in the chapter in 第 4.4.7 节 "关注还是忽视祖先".

Applying reverse merges to a target's natural history
: Earlier in this chapter (第 4.3.4 节 "取消修改") we discussed how to use **svn merge** to apply a "reverse patch" as a way of rolling back changes. If this technique is used to undo a change to an object's personal history (e.g., commit r5 to the trunk, then immediately roll back r5 using **`svn merge . -c -5`**), this sort of merge doesn't affect the recorded mergeinfo. [4]

4.4.4. 合并冲突

就像**svn update**命令，**svn merge**会把修改应用到工作拷贝，因此它也会造成冲突，因为**svn merge**造成的冲突有时候会有些不同，本小节会解释这些区别。

[4]Interestingly, after rolling back a revision like this, we wouldn't be able to reapply the revision using **`svn merge . -c 5`**, since the mergeinfo would already list r5 as being applied. We would have to use the `--ignore-ancestry` option to make the merge command ignore the existing mergeinfo!

To begin with, assume that your working copy has no local edits. When you **svn update** to a particular revision, the changes sent by the server will always apply “cleanly” to your working copy. The server produces the delta by comparing two trees: a virtual snapshot of your working copy, and the revision tree you're interested in. Because the left hand side of the comparison is exactly equal to what you already have, the delta is guaranteed to correctly convert your working copy into the right hand tree.

但是**svn merge**没有这样的保证，会导致很多的混乱：用户可以询问服务器比较任何两个树，即使一个与工作拷贝毫不相关的！这意味着有潜在的人为错误，用户有时候会比较两个错误的树，创建的增量数据不会干净的应用，**svn merge**会尽力应用更多的增量数据，但是有一些部分也许会难以完成，就像Unix下**patch**命令有时候会报告“failed hunks”错误，**svn merge**会报告“skipped targets”：

```
$ svn merge -r 1288:1351 http://svn.example.com/repos/branch
U    foo.c
U    bar.c
Skipped missing target: 'baz.c'
U    glub.c
U    sputter.h

Conflict discovered in 'glorb.h'.
Select: (p) postpone, (df) diff-full, (e) edit,
        (h) help for more options:
```

In the previous example, it might be the case that `baz.c` exists in both snapshots of the branch being compared, and the resultant delta wants to change the file's contents, but the file doesn't exist in the working copy. Whatever the case, the “skipped” message means that the user is most likely comparing the wrong two trees; it's the classic sign of user error. When this happens, it's easy to recursively revert all the changes created by the merge (**`svn revert . --recursive`**), delete any unversioned files or directories left behind after the revert, and rerun **svn merge** with different arguments.

Also notice that the preceeding example shows a conflict happening on `glorb.h`. We already stated that the working copy has no local edits: how can a conflict possibly happen? Again, because the user can use **svn merge** to define and apply any old delta to the working copy, that delta may contain textual changes that don't cleanly apply to a working file, even if the file has no local modifications.

Another small difference between **svn update** and **svn merge** is the names of the full-text files created when a conflict happens. In 第 2.4.5 节 “解决冲突(合并别人的修改)”, we saw that an update produces files named `filename.mine`, `filename.rOLDREV`, and `filename.rNEWREV`. When **svn merge** produces a conflict, though, it creates three files named `filename.working`, `filename.left`, and `filename.right`. In this case, the terms “left” and “right” are describing which side of the double-tree comparison the file came from. In any case, these differing names will help you distinguish between conflicts that happened as a result of an update and ones that happened as a result of a merge.

4.4.5. 阻塞修改

Sometimes there's a particular changeset that you don't want to be automatically merged. For example, perhaps your team's policy is to do new development work on `/trunk`, but to be more conservative

about backporting changes to a stable branch you use for releasing to the public. On one extreme, you can manually cherrypick single changesets from the trunk to the branch—just the changes that are stable enough to pass muster. Maybe things aren't quite that strict, though; perhaps most of the time you'd like to just let **svn merge** automatically merge most changes from trunk to branch. In this case, you'd like a way to mask a few specific changes out, that is, prevent them from ever being automatically merged.

In Subversion 1.5, the only way to block a changeset is to make the system believe that the change has *already* been merged. To do this, one can invoke a merge command with the `--record-only` option:

```
$ cd my-calc-branch

$ svn propget svn:mergeinfo .
/trunk:1680-3305

# Let's make the metadata list r3328 as already merged.
$ svn merge -c 3328 --record-only http://svn.example.com/repos/calc/trunk

$ svn status
M     .

$ svn propget svn:mergeinfo .
/trunk:1680-3305,3328

$ svn commit -m "Block r3328 from being merged to the branch."
…
```

This technique works, but it's also a little bit dangerous. The main problem is that we're not clearly differentiating between the ideas of “I already have this change” and “I don't have this change.” We're effectively lying to the system, making it think that the change was previously merged. This puts the responsibility on you—the user—to remember that the change wasn't actually merged, it just wasn't wanted. There's no way to ask Subversion for a list of “blocked changelists.” If you want to track them (so that you can unblock them someday). you'll need to record them in a text file somewhere, or perhaps in an invented property. In Subversion 1.5, unfortunately, this is the only way to manage blocked revisions; the plans are to make a better interface for this in future versions.

4.4.6. 感知合并的日志和注解

One of the main features of any version control system is to keep track of who changed what, and when they did it. The **svn log** and **svn blame** commands are just the tools for this: when invoked on individual files, they show not only the history of changesets that affected the file, but also exactly which user wrote which line of code, and when she did it.

When changes start getting replicated between branches, however, things start to get complicated. For example, if you were to ask **svn log** about the history of your feature branch, it would show exactly every revision that ever affected the branch:

```
$ cd my-calc-branch
```

```
$ svn log -q
------------------------------------------------------------------------
r390 | user | 2002-11-22 11:01:57 -0600 (Fri, 22 Nov 2002) | 1 line
------------------------------------------------------------------------
r388 | user | 2002-11-21 05:20:00 -0600 (Thu, 21 Nov 2002) | 2 lines
------------------------------------------------------------------------
r381 | user | 2002-11-20 15:07:06 -0600 (Wed, 20 Nov 2002) | 2 lines
------------------------------------------------------------------------
r359 | user | 2002-11-19 19:19:20 -0600 (Tue, 19 Nov 2002) | 2 lines
------------------------------------------------------------------------
r357 | user | 2002-11-15 14:29:52 -0600 (Fri, 15 Nov 2002) | 2 lines
------------------------------------------------------------------------
r343 | user | 2002-11-07 13:50:10 -0600 (Thu, 07 Nov 2002) | 2 lines
------------------------------------------------------------------------
r341 | user | 2002-11-03 07:17:16 -0600 (Sun, 03 Nov 2002) | 2 lines
------------------------------------------------------------------------
r303 | sally | 2002-10-29 21:14:35 -0600 (Tue, 29 Oct 2002) | 2 lines
------------------------------------------------------------------------
r98 | sally | 2002-02-22 15:35:29 -0600 (Fri, 22 Feb 2002) | 2 lines
------------------------------------------------------------------------
```

But is this really an accurate picture of all the changes that happened on the branch? What's being left out here is the fact that revisions 390, 381, and 357 were actually the results of merging changes from the trunk. If you look at one of these logs in detail, the multiple trunk changesets that comprised the branch change are nowhere to be seen:

```
$ svn log -v -r 390
------------------------------------------------------------------------
r390 | user | 2002-11-22 11:01:57 -0600 (Fri, 22 Nov 2002) | 1 line
Changed paths:
   M /branches/my-calc-branch/button.c
   M /branches/my-calc-branch/README

Final merge of trunk changes to my-calc-branch.
```

We happen to know that this merge to the branch was nothing but a merge of trunk changes. How can we see those trunk changes as well? The answer is to use the `--use-merge-history` (`-g`) option. This option expands those “child” changes that were part of the merge.

```
$ svn log -v -r 390 -g
------------------------------------------------------------------------
r390 | user | 2002-11-22 11:01:57 -0600 (Fri, 22 Nov 2002) | 1 line
Changed paths:
   M /branches/my-calc-branch/button.c
   M /branches/my-calc-branch/README

Final merge of trunk changes to my-calc-branch.
```

```
------------------------------------------------------------------------
r383 | sally | 2002-11-21 03:19:00 -0600 (Thu, 21 Nov 2002) | 2 lines
Changed paths:
   M /branches/my-calc-branch/button.c
Merged via: r390

Fix inverse graphic error on button.
------------------------------------------------------------------------
r382 | sally | 2002-11-20 16:57:06 -0600 (Wed, 20 Nov 2002) | 2 lines
Changed paths:
   M /branches/my-calc-branch/README
Merged via: r390

Document my last fix in README.
```

By making the log operation use merge history, we see not just the revision we queried (r390), but also the two revisions that came along on the ride with it—a couple of changes made by Sally to the trunk. This is a much more complete picture of history!

The **svn blame** command also takes the `--use-merge-history` (`-g`) option. If this option is neglected, somebody looking at a line-by-line annotation of `button.c` may get the mistaken impression that you were responsible for the lines that fixed a certain error:

```
$ svn blame button.c
…
   390    user    retval = inverse_func(button, path);
   390    user    return retval;
   390    user    }
…
```

And while it's true that you did actually commit those three lines in revision 390, two of them were actually written by Sally back in revision 383:

```
$ svn blame button.c -g
…
G    383    sally   retval = inverse_func(button, path);
G    383    sally   return retval;
     390    user    }
…
```

Now we know who to *really* blame for those two lines of code!

4.4.7. 关注还是忽视祖先

When conversing with a Subversion developer, you might very likely hear reference to the term *ancestry*. This word is used to describe the relationship between two objects in a repository: if they're related to each other, one object is said to be an ancestor of the other.

举个例子，假设你提交版本100，包括对foo.c的修改，则foo.c@99是foo.c@100的一个“祖先”，另一方面，假设你在版本101删除这个文件，而在102版本提交一个同名的文件，在这个情况下，foo.c@99与foo.c@102看起来是关联的(有同样的路径)，但是事实上他们是完全不同的对象，它们并不共享同一个历史或者说“祖先”。

指出**svn diff**和**svn merge**区别的重要性在于，前一个命令忽略祖先，如果你询问**svn diff**来比较文件foo.c的版本99和102，你会看到行为基础的区别，diff命令只是盲目的比较两条路径，但是如果你使用**svn merge**是比较同样的两个对象，它会注意到他们是不关联的，而且首先尝试删除旧文件，然后添加新文件，输出会是一个删除紧接着一个增加：

```
D     foo.c
A     foo.c
```

大多数合并包括比较包括祖先关联的两条树，因此**svn merge**这样运作，然而，你也许会希望merge命令能够比较两个不相关的目录树，举个例子，你有两个目录树分别代表了供应方软件项目的不同版本(见第 4.9 节 “供方分支”)，如果你使用**svn merge**进行比较，你会看到第一个目录树被删除，而第二个树添加上！在这个情况下，你仅仅是希望**svn merge**以路径为基础比较两棵树，而忽略文件和目录的不相关性，当为合并命令添加--ignore-ancestry选项时，就会像**svn diff**一样工作。(相反，--notice-ancestry会导致**svn diff**像merge命令一样工作。)

4.4.8. 合并和移动

A common desire is to refactor source code, especially in Java-based software projects. Files and directories are shuffled around and renamed, often causing great disruption to everyone working on the project. Sounds like a perfect case to use a branch, doesn't it? Just create a branch, shuffle things around, and then merge the branch back to the trunk, right?

Alas, this scenario doesn't work so well right now and is considered one of Subversion's current weak spots. The problem is that Subversion's **svn update** command isn't as robust as it should be, particularly when dealing with copy and move operations.

When you use **svn copy** to duplicate a file, the repository remembers where the new file came from, but it fails to transmit that information to the client which is running **svn update** or **svn merge**. Instead of telling the client, “Copy that file you already have to this new location,” it sends down an entirely new file. This can lead to problems, especially because the same thing happens with renamed files. A lesser-known fact about Subversion is that it lacks “true renames”—the **svn move** command is nothing more than an aggregation of **svn copy** and **svn delete**.

例如，假定我们在一个私有分支工作，你将integer.c改名为whole.c，你这是在分支上创建了原来文件的一个拷贝，并且删除了原来的文件。同时，回到trunk，Sally提交了一些integer.c的修改，所以你需要将分支合并到主干：

```
$ cd calc/trunk

$ svn merge --reintegrate http://svn.example.com/repos/calc/branches/my-ca
```

```
--- Merging differences between repository URLs into '.':
D    integer.c
A    whole.c
U    .
```

第一眼看起来不是很差，但是很可能这不是你和Sally希望的，合并操作已经删除了最新版本的`integer.c`(包含了Sally最新的修改)，而且盲目的添加了你的`whole.c`文件—是旧版本的`integer.c`复制品。最终的结果是将你的“rename”合并到分支，并且从最新修订版本删除了Sally最近的修改。

This isn't true data loss. Sally's changes are still in the repository's history, but it may not be immediately obvious that this has happened. The moral of this story is that until Subversion improves, be very careful about merging copies and renames from one branch to another.

4.4.9. 阻塞不知道合并的客户端

If you've just upgraded your server to Subversion 1.5 or later, there's a significant risk that pre-1.5 Subversion clients can mess up your automated merge tracking. Why is this? When a pre-1.5 Subversion client performs **svn merge**, it doesn't modify the value of the `svn:mergeinfo` property at all. So the subsequent commit, despite being the result of a merge, doesn't tell the repository about the duplicated changes—that information is lost. Later on, when “merge-aware” clients attempt automatic merging, they're likely to run into all sorts of conflicts resulting from repeated merges.

If you and your team are relying on the merge-tracking features of Subversion, you may want to configure your repository to prevent older clients from committing changes. The easy way to do this is by inspecting the “capabilities” parameter in the `start-commit` hook script. If the client reports itself as having `mergeinfo` capabilities, the hook script can allow the commit to start. If the client doesn't report that capability, have the hook deny the commit. We'll learn more about hook scripts in the next chapter; see 第 5.3.2 节 “实现版本库钩子” and start-commit for details.

4.4.10. 合并跟踪的最终信息

The bottom line is that Subversion's merge-tracking feature has an extremely complex internal implementation, and the `svn:mergeinfo` property is the only window the user has into the machinery. Because the feature is relatively new, a numbers of edge cases and possible unexpected behaviors may pop up.

For example, sometimes mergeinfo will be generated when running a simple **svn copy** or **svn move** command. Sometimes mergeinfo will appear on files that you didn't expect to be touched by an operation. Sometimes mergeinfo won't be generated at all, when you expect it to. Furthermore, the management of mergeinfo metadata has a whole set of taxonomies and behaviors around it, such as “explicit” versus “implicit” mergeinfo, “operative” versus “inoperative” revisions, specific mechanisms of mergeinfo “elision,” and even “inheritance” from parent to child directories.

We've chosen not to cover these detailed topics in this book for a couple of reasons. First, the level of detail is absolutely overwhelming for a typical user. Second, as Subversion continues to improve, we feel that a typical user *shouldn't* have to understand these concepts; they'll eventually fade into

the background as pesky implementation details. All that said, if you enjoy this sort of thing, you can get a fantastic overview in a paper posted at CollabNet's website: `http://www.collab.net/community/subversion/articles/merge-info.html`.

For now, if you want to steer clear of bugs and odd behaviors in automatic merging, the CollabNet article recommends that you stick to these simple best practices:

- For short-term feature branches, follow the simple procedure described throughout 第 4.3 节 “基本合并”.
- For long-lived release branches (as described in 第 4.8 节 “常用分支模式”), perform merges only on the root of the branch, not on subdirectories.
- Never merge into working copies with a mixture of working revision numbers, or with “switched” subdirectories (as described next in 第 4.5 节 “使用分支”). A merge target should be a working copy which represents a *single* location in the repository at a single point in time.
- Don't ever edit the `svn:mergeinfo` property directly; use **svn merge** with the `--record-only` option to effect a desired change to the metadata (as demonstrated in 第 4.4.5 节 “阻塞修改”).
- Always make sure you have complete read access to all of your merge sources, and that your target working copy has no sparse directories.

4.5. 使用分支

svn switch命令改变存在的工作拷贝到另一个分支，然而这个命令在分支上工作时不是严格必要的，它只是提供了一个快捷方式。在前面的例子里，完成了私有分支的建立，你取出了新目录的工作拷贝，相反，你可以简单的告诉Subversion改变你的`/calc/trunk`的工作拷贝到分支的路径：

```
$ cd calc

$ svn info | grep URL
URL: http://svn.example.com/repos/calc/trunk

$ svn switch http://svn.example.com/repos/calc/branches/my-calc-branch
U   integer.c
U   button.c
U   Makefile
Updated to revision 341.

$ svn info | grep URL
URL: http://svn.example.com/repos/calc/branches/my-calc-branch
```

“Switching” a working copy that has no local modifications to a different branch results in the working copy looking just as it would if you'd done a fresh checkout of the directory. It's usually more efficient to use this command, because often branches differ by only a small degree. The server sends only the minimal set of changes necessary to make your working copy reflect the branch directory.

svn switch命令也可以带`--revision(-r)`参数，所以你不需要一直移动你的工作拷贝到分支的`HEAD`。

当然，许多项目比我们的`calc`要复杂的多，有更多的子目录，Subversion用户通常用如下的法则使用分支：

1. 拷贝整个项目的“trunk”目录到一个新的分支目录。

2. 只是转换工作拷贝的部分目录到分支。

In other words, if a user knows that the branch work needs to happen on only a specific subdirectory, she uses **svn switch** to move only that subdirectory to the branch. (Or sometimes users will switch just a single working file to the branch!) That way, the user can continue to receive normal “trunk” updates to most of her working copy, but the switched portions will remain immune (unless someone commits a change to her branch). This feature adds a whole new dimension to the concept of a “mixed working copy”—not only can working copies contain a mixture of working revisions, but they can also contain a mixture of repository locations as well.

如果你的工作拷贝包含许多来自不同版本库目录跳转的子树，它会工作如常。当你更新时，你会得到每一个目录适当的补丁，当你提交时，你的本地修改会一直作为一个单独的原子修改提交到版本库。

Note that while it's okay for your working copy to reflect a mixture of repository locations, these locations must all be within the *same* repository. Subversion repositories aren't yet able to communicate with one another; that feature is planned for the future. [5]

切换和更新

Have you noticed that the output of **svn switch** and **svn update** looks the same? The switch command is actually a superset of the update command.

当你运行**svn update**时，你会告诉版本库比较两个目录树，版本库这样做，并且返回给客户区别的描述，**svn switch**和**svn update**两个命令唯一区别就是**update**会一直比较同一路径。

That is, if your working copy is a mirror of `/calc/trunk`, **svn update** will automatically compare your working copy of `/calc/trunk` to `/calc/trunk` in the `HEAD` revision. If you're switching your working copy to a branch, **svn switch** will compare your working copy of `/calc/trunk` to some *other* branch directory in the `HEAD` revision.

换句话说，一个更新通过时间移动你的工作拷贝，一个转换通过时间和空间移动工作拷贝。

因为**svn switch**是**svn update**的一个变种，具有相同的行为，当新的数据到达时，任何工作拷贝的已经完成的本地修改会被保存。

[5]如果你不希望抛弃已存在的工作拷贝，你可以使用带`--relocate`选项的**svn switch**命令转换URL，更多信息和例子可以看svn switch。

提示

你是否发现你做出了复杂的修改(在/trunk的工作拷贝)，并突然发现，“这些修改必须在它们自己的分支？”处理这个问题的技术可以总结为两步：

```
$ svn copy http://svn.example.com/repos/calc/trunk \
           http://svn.example.com/repos/calc/branches/newbranch \
      -m "Create branch 'newbranch'."
Committed revision 353.
$ svn switch http://svn.example.com/repos/calc/branches/newbranch
At revision 353.
```

就像**svn update**命令，**svn switch**会保留工作拷贝的本地修改，此刻，你的工作拷贝反映到新建的分支上，而你的下一次**svn commit**会发送修改到服务器。

4.6. 标签

另一个常见的版本控制系统概念是标¾(*tag*)，一个标签只是一个项目某一时间的“快照”，在Subversion里这个概念无处不在—每一次提交的修订版本都是一个精确的快照。

然而人们希望更人性化的标签名称，像release-1.0。他们也希望可以对一个子目录快照，毕竟，记住release-1.0是修订版本4822的某一小部分不是件很容易的事。

4.6.1. 建立简单标签

Once again, **svn copy** comes to the rescue. If you want to create a snapshot of /calc/trunk exactly as it looks in the HEAD revision, make a copy of it:

```
$ svn copy http://svn.example.com/repos/calc/trunk \
           http://svn.example.com/repos/calc/tags/release-1.0 \
      -m "Tagging the 1.0 release of the 'calc' project."

Committed revision 902.
```

This example assumes that a /calc/tags directory already exists. (If it doesn't, you can create it using **svn mkdir**.) After the copy completes, the new release-1.0 directory is forever a snapshot of how the /trunk directory looked in the HEAD revision at the time you made the copy. Of course, you might want to be more precise about exactly which revision you copy, in case somebody else may have committed changes to the project when you weren't looking. So if you know that revision 901 of /calc/trunk is exactly the snapshot you want, you can specify it by passing -r 901 to the **svn copy** command.

But wait a moment: isn't this tag creation procedure the same procedure we used to create a branch? Yes, in fact, it is. In Subversion, there's no difference between a tag and a branch. Both are just ordinary directories that are created by copying. Just as with branches, the only reason a copied directory is a “tag” is because *humans* have decided to treat it that way: as long as nobody ever commits to the directory, it forever remains a snapshot. If people start committing to it, it becomes a branch.

If you are administering a repository, there are two approaches you can take to managing tags. The first approach is “hands off”: as a matter of project policy, decide where your tags will live, and make sure all users know how to treat the directories they copy. (That is, make sure they know not to commit to them.) The second approach is more paranoid: you can use one of the access control scripts provided with Subversion to prevent anyone from doing anything but creating new copies in the tags area (see 第 6 章 服务配置). The paranoid approach, however, isn't usually necessary. If a user accidentally commits a change to a tag directory, you can simply undo the change as discussed in the previous section. This is version control, after all!

4.6.2. 建立复杂标签

有时候你希望你的“快照”能够很复杂，而不只是一个单独修订版本的一个单独目录。

For example, pretend your project is much larger than our `calc` example: suppose it contains a number of subdirectories and many more files. In the course of your work, you may decide that you need to create a working copy that is designed to have specific features and bug fixes. You can accomplish this by selectively backdating files or directories to particular revisions (using **svn update** with the `-r` option liberally), by switching files and directories to particular branches (making use of **svn switch**), or even just by making a bunch of local changes. When you're done, your working copy is a hodgepodge of repository locations from different revisions. But after testing, you know it's the precise combination of data you need to tag.

Time to make a snapshot. Copying one URL to another won't work here. In this case, you want to make a snapshot of your exact working copy arrangement and store it in the repository. Luckily, **svn copy** actually has four different uses (which you can read about in 第 9 章 *Subversion* 完全参考), including the ability to copy a working copy tree to the repository:

```
$ ls
my-working-copy/

$ svn copy my-working-copy \
           http://svn.example.com/repos/calc/tags/mytag \
           -m "Tag my existing working copy state."

Committed revision 940.
```

Now there is a new directory in the repository, `/calc/tags/mytag`, which is an exact snapshot of your working copy—mixed revisions, URLs, local changes, and all.

Other users have found interesting uses for this feature. Sometimes there are situations where you have a bunch of local changes made to your working copy, and you'd like a collaborator to see them. Instead of running **svn diff** and sending a patch file (which won't capture directory, symlink, or property changes), you can use **svn copy** to “upload” your working copy to a private area of the repository. Your collaborator can then either check out a verbatim copy of your working copy or use **svn merge** to receive your exact changes.

While this is a nice method for uploading a quick snapshot of your working copy, note that this is *not* a good way to initially create a branch. Branch creation should be an event unto itself, and this method

conflates the creation of a branch with extra changes to files, all within a single revision. This makes it very difficult (later on) to identify a single revision number as a branch point.

4.7. 维护分支

你一定注意到了Subversion极度的灵活性，因为它用相同的底层机制(目录拷贝)实现了分支和标签，因为分支和标签是作为普通的文件系统出现，会让人们感到害怕，因为它太灵活了，在这个小节里，我们会提供安排和管理数据的一些建议。

4.7.1. 版本库布局

There are some standard, recommended ways to organize a repository. Most people create a `trunk` directory to hold the "main line" of development, a `branches` directory to contain branch copies, and a `tags` directory to contain tag copies. If a repository holds only one project, often people create these top-level directories:

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

如果一个版本库保存了多个项目，管理员会通过项目来布局(见第 5.2.1 节 “规划你的版本库结构”关于“项目根目录”)：

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

当然，你可以自由的忽略这些通常的布局方式，你可以创建任意的变化，只要是对你和你的项目有益，记住无论你选择什么，这不会是一种永久的承诺，你可以随时重新组织你的版本库。因为分支和标签都是普通的目录，**svn move**命令可以任意的改名和移动它们，从一种布局到另一种大概只是一系列服务器端的移动，如果你不喜欢版本库的组织方式，你可以任意修改目录结构。

记住，尽管移动目录非常容易，你必须体谅你的用户，你的修改会让你的用户感到迷惑，如果一个用户的拥有一个版本库目录的工作拷贝，你的**svn move**命令也许会删除最新的版本的这个路径，当用户运行**svn update**，会被告知这个工作拷贝引用的路径已经不再存在，用户需要强制使用**svn switch**转到新的位置。

4.7.2. 数据的生命周期

另一个Subversion模型的可爱特性是分支和标签可以有有限的生命周期，就像其它的版本化的项目，举个例子，假定你最终完成了`calc`项目你的个人分支上的所有工作，在合并了你的所有修改到`/calc/trunk`后，没有必要继续保留你的私有分支目录：

```
$ svn delete http://svn.example.com/repos/calc/branches/my-calc-branch \
             -m "Removing obsolete branch of calc project."

Committed revision 375.
```

And now your branch is gone. Of course, it's not really gone: the directory is simply missing from the HEAD revision, no longer distracting anyone. If you use **svn checkout**, **svn switch**, or **svn list** to examine an earlier revision, you'll still be able to see your old branch.

如果浏览你删除的目录还不足够，你可以把它找回来，恢复数据对Subversion来说很简单，如果你希望恢复一个已经删除的目录(或文件)到HEAD，仅需要使用**svn copy**来从旧的版本拷贝出来：

```
$ svn copy http://svn.example.com/repos/calc/branches/my-calc-branch@374 \
           http://svn.example.com/repos/calc/branches/my-calc-branch \
           -m "Restore my-calc-branch."

Committed revision 376.
```

在我们的例子里，你的个人分支只有一个相对短的生命周期：你会为修复一个Bug或实现一个小的特性来创建它，当任务完成，分支也该结束了。在软件开发过程中，有两个“主要的”分支一直存在很长的时间也是很常见的情况，举个例子，假定我们是发布一个稳定的calc项目的时候了，但我们仍会需要几个月的时间来修复Bug，你不希望添加新的特性，但你不希望告诉开发者停止开发，所以作为替代，你为软件创建了一个“稳定”分支，这个分支更改不会很多：

```
$ svn copy http://svn.example.com/repos/calc/trunk \
           http://svn.example.com/repos/calc/branches/stable-1.0 \
           -m "Creating stable branch of calc project."

Committed revision 377.
```

而且开发者可以自由的继续添加新的(试验的)特性到/calc/trunk，你可以宣布这样一种政策，只有bug修正提交到/calc/branches/stable-1.0，这样的话，人们继续在主干上工作，某个人会选择在稳定分支上做出一些Bug修正，甚至在稳定版本发布之后。你或许会在这个维护分支上工作很长时间—也就是说，你会一直继续为客户提供这个版本的支持，我们会在后面的部分讨论更多。

4.8. 常用分支模式

分支和**svn merge**有很多不同的用法，这个小节描述了最常见的用法。

版本控制在软件开发中广泛使用，这里是团队里程序员最常用的两种分支/合并模式的介绍，如果你不是使用Subversion软件开发，可随意跳过本小节，如果你是第一次使用版本控制的软件开发者，请更加注意，以下模式被许多老兵当作最佳实践，这个过程并不只是针对Subversion，在任何版本控制系统中都一样，但是在这里使用Subversion术语会感觉更方便一点。

4.8.1. 发布分支

Most software has a typical life cycle: code, test, release, repeat. There are two problems with this process. First, developers need to keep writing new features while quality assurance teams take time to test supposedly stable versions of the software. New work cannot halt while the software is tested. Second, the team almost always needs to support older, released versions of software; if a bug is discovered in the latest code, it most likely exists in released versions as well, and customers will want to get that bug fix without having to wait for a major new release.

这是版本控制可以做的帮助，典型的过程如下：

1. *Developers commit all new work to the trunk.* Day-to-day changes are committed to `/trunk`: new features, bug fixes, and so on.

2. *The trunk is copied to a "release" branch.* When the team thinks the software is ready for release (say, a 1.0 release), `/trunk` might be copied to `/branches/1.0`.

3. 项目组继续并行工作，一个小组开始对分支进行严酷的测试，同时另一个小组在`/trunk`继续新的工作(如，准备2.0)，如果一个bug在任何一个位置被发现，错误修正需要来回运送。然而这个过程有时候也会结束，例如分支已经为发布前的最终测试“停滞”了。

4. 分支已经作了标签并且发布，当测试结束，`/branches/1.0`作为引用快照已经拷贝到`/tags/1.0.0`，这个标签被打包发布给客户。

5. *The branch is maintained over time.* While work continues on `/trunk` for version 2.0, bug fixes continue to be ported from `/trunk` to `/branches/1.0`. When enough bug fixes have accumulated, management may decide to do a 1.0.1 release: `/branches/1.0` is copied to `/tags/1.0.1`, and the tag is packaged and released.

整个过程随着软件的成熟不断重复：当2.0完成，一个新的2.0分支被创建，测试、打标签和最终发布，经过许多年，版本库结束了许多版本发布，进入了“维护”模式，许多标签代表了最终的发布版本。

4.8.2. 特性分支

A *feature branch* is the sort of branch that's been the dominant example in this chapter (the one you've been working on while Sally continues to work on `/trunk`). It's a temporary branch created to work on a complex change without interfering with the stability of `/trunk`. Unlike release branches (which may need to be supported forever), feature branches are born, used for a while, merged back to the trunk, and then ultimately deleted. They have a finite span of usefulness.

还有，关于是否创建特性分支的项目政策也变化广泛，一些项目永远不使用特性分支：大家都可以提交到`/trunk`，好处是系统的简单—没有人需要知道分支和合并，坏处是主干会经常不稳定或者不可用，另外一些项目使用分支达到极限：没有修改曾经直接提交到主干，即使最细小的修改都要创建短暂的分支，然后小心的审核合并到主干，然后删除分支，这样系统保持主干一直稳定和可用，但是造成了巨大的负担。

Most projects take a middle-of-the-road approach. They commonly insist that `/trunk` compile and pass regression tests at all times. A feature branch is required only when a change requires a large

number of destabilizing commits. A good rule of thumb is to ask this question: if the developer worked for days in isolation and then committed the large change all at once (so that `/trunk` were never destabilized), would it be too large a change to review? If the answer to that question is “yes,” the change should be developed on a feature branch. As the developer commits incremental changes to the branch, they can be easily reviewed by peers.

最终，有一个问题就是怎样保持一个特性分支“同步”于工作中的主干，在前面提到过，在一个分支上工作数周或几个月是很有风险的，主干的修改也许会持续涌入，因为这一点，两条线的开发会区别巨大，合并分支回到主干会成为一个噩梦。

This situation is best avoided by regularly merging trunk changes to the branch. Make up a policy: once a week, merge the last week's worth of trunk changes to the branch.

在一些时候，你已经准备好了将“同步的”特性分支合并回到主干，为此，开始做一次将主干最新修改和分支的最终合并，这样以后，除了你的分支修改的部分，最新的分支和主干将会绝对一致，你或许需要使用`--reintegrate`选项合并回去：

```
$ cd trunk-working-copy

$ svn update
At revision 1910.

$ svn merge --reintegrate http://svn.example.com/repos/calc/branches/mybra
--- Merging differences between repository URLs into '.':
U    real.c
U    integer.c
A    newdirectory
A    newdirectory/newfile
 U   .
…
```

Another way of thinking about this pattern is that your weekly sync of trunk to branch is analogous to running **svn update** in a working copy, while the final merge step is analogous to running **svn commit** from a working copy. After all, what else *is* a working copy but a very shallow private branch? It's a branch that's capable of storing only one change at a time.

4.9. 供方分支

当开发软件时有这样一个情况，你版本控制的数据可能关联于或者是依赖于其他人的数据，通常来讲，你的项目的需要会要求你自己的项目对外部实体提供的数据保持尽可能最新的版本，同时不会牺牲稳定性，这种情况总是会出现—只要某个小组的信息对另一个小组的信息有直接的影响。

For example, software developers might be working on an application that makes use of a third-party library. Subversion has just such a relationship with the Apache Portable Runtime (APR) library (see 第 8.3.1 节 “Apache 可移植运行库”). The Subversion source code depends on the APR library for all its portability needs. In earlier stages of Subversion's development, the project closely tracked APR's changing API, always sticking to the “bleeding edge” of the library's code churn. Now that both

APR and Subversion have matured, Subversion attempts to synchronize with APR's library API only at well-tested, stable release points.

Now, if your project depends on someone else's information, you could attempt to synchronize that information with your own in several ways. Most painfully, you could issue oral or written instructions to all the contributors of your project, telling them to make sure they have the specific versions of that third-party information that your project needs. If the third-party information is maintained in a Subversion repository, you could also use Subversion's externals definitions to effectively “pin down” specific versions of that information to some location in your own working copy directory (see 第 3.8 节 “外部定义”).

但是有时候，你希望在你自己的版本控制系统维护一个针对第三方数据的自定义修改，回到软件开发的例子，程序员为了他们自己的目的会需要修改第三方库，这些修改会包括新的功能和bug修正，在成为第三方工具官方发布之前，只是内部维护。或者这些修改永远不会传给库的维护者，只是作为满足软件开发需要的单独的自定义修改存在。

Now you face an interesting situation. Your project could house its custom modifications to the third-party data in some disjointed fashion, such as using patch files or full-fledged alternative versions of files and directories. But these quickly become maintenance headaches, requiring some mechanism by which to apply your custom changes to the third-party code and necessitating regeneration of those changes with each successive version of the third-party code that you track.

这个问题的解决方案是使用供方分支，一个供方分支是一个目录树保存了第三方实体或供应方的信息，每一个供应方数据的版本吸收到你的项目叫做供方*drop*。

Vendor branches provide two benefits. First, by storing the currently supported vendor drop in your own version control system, you ensure that the members of your project never need to question whether they have the right version of the vendor's data. They simply receive that correct version as part of their regular working copy updates. Second, because the data lives in your own Subversion repository, you can store your custom changes to it in-place—you have no more need of an automated (or worse, manual) method for swapping in your customizations.

4.9.1. 常规的供方分支管理过程

Managing vendor branches generally works like this: first, you create a top-level directory (such as `/vendor`) to hold the vendor branches. Then you import the third-party code into a subdirectory of that top-level directory. You then copy that subdirectory into your main development branch (e.g., `/trunk`) at the appropriate location. You always make your local changes in the main development branch. With each new release of the code you are tracking, you bring it into the vendor branch and merge the changes into `/trunk`, resolving whatever conflicts occur between your local changes and the upstream changes.

也许一个例子有助于我们阐述这个算法，我们会使用这样一个场景，我们的开发团队正在开发一个计算器程序，与一个第三方的复杂数字运算库libcomplex关联。我们从供方分支的初始创建开始，并且导入供方drop，我们会把每株分支目录叫做`libcomplex`，我们的代码drop会进入到供方分支的子目录`current`，并且因为**svn import**创建所有的需要的中间父目录，我们可以使用一个命令完成这一步：

```
$ svn import /path/to/libcomplex-1.0 \
             http://svn.example.com/repos/vendor/libcomplex/current \
             -m 'importing initial 1.0 vendor drop'
…
```

我们现在在/vendor/libcomplex/current有了libcomplex当前版本的代码，现在我们为那个版本作标签(见第 4.6 节 “标签”)，然后拷贝它到主要开发分支，我们的拷贝会在calc项目目录创建一个新的目录libcomplex，它是这个我们将要进行自定义的供方数据的拷贝版本。

```
$ svn copy http://svn.example.com/repos/vendor/libcomplex/current  \
           http://svn.example.com/repos/vendor/libcomplex/1.0      \
           -m 'tagging libcomplex-1.0'
…
$ svn copy http://svn.example.com/repos/vendor/libcomplex/1.0  \
           http://svn.example.com/repos/calc/libcomplex        \
           -m 'bringing libcomplex-1.0 into the main branch'
…
```

我们取出我们项目的主分支—现在包括了第一个供方释放的拷贝—我们开始自定义libcomplex的代码，在我们知道之前，我们的libcomplex修改版本是已经与我们的计算器程序完全集成了。[6]

几周之后，libcomplex得开发者发布了一个新的版本—版本1.1—包括了我们很需要的一些特性和功能。我们很希望升级到这个版本，但不希望失去在当前版本所作的修改。我们本质上会希望把我们当前基线版本是的libcomplex1.0的拷贝替换为libcomplex 1.1，然后把前面自定义的修改应用到新的版本。但是实际上我们通过一个相反的方向解决这个问题，应用libcomplex从版本1.0到1.1的修改到我们修改的拷贝。

为了执行这个升级，我们取出一个我们供方分支的拷贝，替换current目录为新的libcomplex 1.1的代码，我们只是拷贝新文件到存在的文件上，或者是解压缩libcomplex 1.1的打包文件到我们存在的文件和目录。此时的目标是让我们的current目录只保留libcomplex 1.1的代码，并且保证所有的代码在版本控制之下，哦，我们希望在最小的版本控制历史扰动下完成这件事。

完成了这个从1.0到1.1的代码替换，**svn status**会显示文件的本地修改，或许也包括了一些未版本化或者丢失的文件，如果我们做了我们应该做的事情，未版本化的文件应该都是libcomplex在1.1新引入的文件—我们运行**svn add**来将它们加入到版本控制。丢失的文件是存在于1.1但是不是在1.1，在这些路径我们运行**svn delete**。最终一旦我们的当前的工作拷贝只是包括了libcomplex1.1的代码，我们可以提交这些改变目录和文件的修改。

我们的当前分支现在保存了新的供方drop，我们为这个新的版本创建一个新的标签1.1(就像我们为1.0版本drop所作的)，然后合并这从个标签前一个版本的区别到主要开发分支。

```
$ cd working-copies/calc
$ svn merge http://svn.example.com/repos/vendor/libcomplex/1.0      \
```

[6] And is entirely bug-free, of course!

```
            http://svn.example.com/repos/vendor/libcomplex/current  \
            libcomplex
… # resolve all the conflicts between their changes and our changes
$ svn commit -m 'merging libcomplex-1.1 into the main branch'
…
```

在这个琐碎的用例里，第三方工具的新版本会从一个文件和目录的角度来看，就像前一个版本。没有任何libcomplex源文件会被删除、被改名或是移动到别的位置—新的版本只会保存针对上一个版本的文本修改。在完美世界，我们对呢修改会干净得应用到库的新版本，不会产生任何并发和冲突。

But things aren't always that simple, and in fact it is quite common for source files to get moved around between releases of software. This complicates the process of ensuring that our modifications are still valid for the new version of code, and things can quickly degrade into a situation where we have to manually re-create our customizations in the new version. Once Subversion knows about the history of a given source file—including all its previous locations—the process of merging in the new version of the library is pretty simple. But we are responsible for telling Subversion how the source file layout changed from vendor drop to vendor drop.

4.9.2. svn_load_dirs.pl

Vendor drops that contain more than a few deletes, additions, and moves complicate the process of upgrading to each successive version of the third-party data. So Subversion supplies the **svn_load_dirs.pl** script to assist with this process. This script automates the importing steps we mentioned in the general vendor branch management procedure to make sure mistakes are minimized. You will still be responsible for using the merge commands to merge the new versions of the third-party data into your main development branch, but **svn_load_dirs.pl** can help you more quickly and easily arrive at that stage.

一句话，**svn_load_dirs.pl**是一个增强的**svn import**，具备了许多重要的特性：

- 它可以在任何有一个存在的版本库目录与一个外部的目录匹配时执行，会执行所有必要的添加和删除并且可以选则执行移动。
- 它可以用来操作一系列复杂的操作，如那些需要一个中间媒介的提交—如在操作之前重命名一个文件或者目录两次。
- 它可以随意的为新导入目录打上标签。
- 它可以随意为符合正则表达式的文件和目录添加任意的属性。

svn_load_dirs.pl takes three mandatory arguments. The first argument is the URL to the base Subversion directory to work in. This argument is followed by the URL—relative to the first argument—into which the current vendor drop will be imported. Finally, the third argument is the local directory to import. Using our previous example, a typical run of **svn_load_dirs.pl** might look like this:

```
$ svn_load_dirs.pl http://svn.example.com/repos/vendor/libcomplex \
                   current                                        \
                   /path/to/libcomplex-1.1
```

…

你可以说明你会希望**svn_load_dirs.pl**同时打上标签，这使用`-t`命令行选项，需要指定一个标签名，这个标签是第一个参数的一个相对URL。

```
$ svn_load_dirs.pl -t libcomplex-1.1                                  \
                   http://svn.example.com/repos/vendor/libcomplex \
                   current                                            \
                   /path/to/libcomplex-1.1
…
```

When you run **svn_load_dirs.pl**, it examines the contents of your existing "current" vendor drop and compares them with the proposed new vendor drop. In the trivial case, no files will be in one version and not the other, and the script will perform the new import without incident. If, however, there are discrepancies in the file layouts between versions, **svn_load_dirs.pl** will ask you how to resolve those differences. For example, you will have the opportunity to tell the script that you know that the file `math.c` in version 1.0 of libcomplex was renamed to `arithmetic.c` in libcomplex 1.1. Any discrepancies not explained by moves are treated as regular additions and deletions.

The script also accepts a separate configuration file for setting properties on files and directories matching a regular expression that are *added* to the repository. This configuration file is specified to **svn_load_dirs.pl** using the `-p` command-line option. Each line of the configuration file is a whitespace-delimited set of two or four values: a Perl-style regular expression against which to match the added path, a control keyword (either `break` or `cont`), and then optionally a property name and value.

```
\.png$              break   svn:mime-type   image/png
\.jpe?g$            break   svn:mime-type   image/jpeg
\.m3u$              cont    svn:mime-type   audio/x-mpegurl
\.m3u$              break   svn:eol-style   LF
.*                  break   svn:eol-style   native
```

For each added path, the configured property changes whose regular expression matches the path are applied in order, unless the control specification is `break` (which means that no more property changes should be applied to that path). If the control specification is `cont`—an abbreviation for `continue`—matching will continue with the next line of the configuration file.

Any whitespace in the regular expression, property name, or property value must be surrounded by either single or double quotes. You can escape quotes that are not used for wrapping whitespace by preceding them with a backslash (`\`) character. The backslash escapes only quotes when parsing the configuration file, so do not protect any other characters beyond what is necessary for the regular expression.

4.10. 总结

We covered a lot of ground in this chapter. We discussed the concepts of tags and branches and demonstrated how Subversion implements these concepts by copying directories with the **svn copy**

command. We showed how to use **svn merge** to copy changes from one branch to another or roll back bad changes. We went over the use of **svn switch** to create mixed-location working copies. And we talked about how one might manage the organization and lifetimes of branches in a repository.

Remember the Subversion mantra: branches and tags are cheap. So don't be afraid to use them when needed!

As a helpful reminder of all the operations we discussed, here is handy reference table you can consult as you begin to make use of branches.

表 **4.1.** 分支与合并命令

动作	**Command**
创建一个分支或标签	`svn copy URL1 URL2`
Switch a working copy to a branch or tag	`svn switch URL`
Synchronize a branch with trunk	`svn merge trunkURL; svn commit`
See merge history or eligible changesets	`svn mergeinfo target --from-source=URL`
Merge a branch back into trunk	`svn merge --reintegrate branchURL; svn commit`
复制特定的修改	`svn merge -c REV URL; svn commit`
合并一个范围的修改	`svn merge -r REV1:REV2 URL; svn commit`
Block a change from automatic merging	`svn merge -c REV --record-only URL; svn commit`
预览合并	`svn merge URL --dry-run`
Abandon merge results	`svn revert -R .`
Resurrect something from history	`svn copy URL@REV localPATH`
Undo a committed change	`svn merge -c -REV URL; svn commit`
Examine merge-sensitive history	`svn log -g; svn blame -g`
从工作拷贝创建一个标签	`svn copy . tagURL`
Rearrange a branch or tag	`svn mv URL1 URL2`
Remove a branch or tag	`svn rm URL`

第 5 章 版本库管理

Subversion版本库是保存任意数量项目版本化数据的中央仓库，因此，版本库成为管理员关注的对象。版本库的维护一般并不需要太多的关注，但为了避免一些潜在的问题和解决一些实际问题，理解怎样适当的配置和维护还是非常重要的。

在这一章里，我们将讨论如何建立和配置一个Subversion版本库，还会讨论版本库的维护，包括**svnlook**和**svnadmin**工具的使用实例。我们将说明一些常见的问题和错误，并提供一些安排版本库数据的建议。

If you plan to access a Subversion repository only in the role of a user whose data is under version control (i.e., via a Subversion client), you can skip this chapter altogether. However, if you are, or wish to become, a Subversion repository administrator, [1] this chapter is for you.

5.1. Subversion 版本库的定义

在进入版本库管理这块宽泛的主题之前，让我们进一步确定一下版本库的定义，它是怎样工作的？让人有什么感觉？它希望茶是热的还是冰的，加糖或柠檬吗？作为一名管理员，你应该既能够从物理具体细节的视角 – 版本库如何响应一个非Subversion的工具，也能够从逻辑视角 – 数据在版本库中如何展示。

Seen through the eyes of a typical file browser application (such as Windows Explorer) or command-line based filesystem navigation tools, the Subversion repository is just another directory full of stuff. There are some subdirectories with human-readable configuration files in them, some subdirectories with some not-so-human-readable data files, and so on. As in other areas of the Subversion design, modularity is given high regard, and hierarchical organization is preferred to cluttered chaos. So a shallow glance into a typical repository from a nuts-and-bolts perspective is sufficient to reveal the basic components of the repository:

```
$ ls repos
conf/  dav/  db/  format  hooks/  locks/  README.txt
```

下面是一个你看到列出目录的快速总揽。(不要因为术语陷入困境—这些组件的细节介绍可以从本章或其他章节找到。)

conf
: A directory containing configuration files

dav
: A directory provided to `mod_dav_svn` for its private housekeeping data

db
: The data store for all of your versioned data

format
: A file that contains a single integer that indicates the version number of the repository layout

[1] 这可能听起来很崇高，但我们所指的只是那些对管理别人工作拷贝数据之外的神秘领域感兴趣的人。

hooks
A directory full of hook script templates (and hook scripts themselves, once you've installed some)

locks
A directory for Subversion's repository lock files, used for tracking accessors to the repository

README.txt
A file whose contents merely inform its readers that they are looking at a Subversion repository

Of course, when accessed via the Subversion libraries, this otherwise unremarkable collection of files and directories suddenly becomes an implementation of a virtual, versioned filesystem, complete with customizable event triggers. This filesystem has its own notions of directories and files, very similar to the notions of such things held by real filesystems (such as NTFS, FAT32, ext3, etc.). But this is a special filesystem—it hangs these directories and files from revisions, keeping all the changes you've ever made to them safely stored and forever accessible. This is where the entirety of your versioned data lives.

5.2. 版本库开发策略

Due largely to the simplicity of the overall design of the Subversion repository and the technologies on which it relies, creating and configuring a repository are fairly straightforward tasks. There are a few preliminary decisions you'll want to make, but the actual work involved in any given setup of a Subversion repository is pretty basic, tending toward mindless repetition if you find yourself setting up multiples of these things.

Some things you'll want to consider beforehand, though, are:

- 你的版本库将要存放什么数据(或多个版本库)，这些数据如何组织?
- 版本库存放在哪里，如何被访问?
- 你需要什么类型的访问控制和版本库事件报告?
- 你希望使用哪种数据存储方式?

在本节，我们要尝试帮你回答这些问题。

5.2.1. 规划你的版本库结构

在Subversion版本库中，移动版本化的文件和目录不会损失任何信息，甚至也可以将版本库的的一组数据无损历史的移植到另一个版本库，但是这样一来那些经常访问版本库并且以为文件总是在同一个路径的用户可能会受到干扰。为将来着想，最好预先对你的版本库布局进行规划。以一种高效的“布局”开始项目，可以减少将来很多不必要的麻烦。

假如你是一个版本库管理员，需要向多个项目提供版本控制支持。那么，你首先要决定的是，用一个版本库支持多个项目，还是为每个项目建立一个版本库，还是两种方法的混合方式。

使用一个版本库支持多个项目有很多好处，最明显的无过于不需要维护好几个版本库。单一版本库就意味着只有一个钩子程序，只需要备份一个数据库，当Subversion进行不兼容升级

时，只需要一次转储和装载操作，等等。还有，你可以轻易的在项目之间移动数据，还不会损失任何历史版本信息。

单一版本库的缺点是，不同的项目通常都有不同的版本库触发事件需求，例如需要发送提交通知邮件到不同的邮件列表，需要不同的鉴定提交是否合法的定义。这些都不是不可逾越的问题，当然—之需要你的钩子程序能够察看版本库的布局，而不是假定整个版本库与同一组人关联。还有，别忘了Subversion的修订版本号是针对整个版本库的，这些号码没有任何魔力。即使最近没有对某个项目作出修改，版本库的修订版本号还是会因为其它项目的修改而不停的提升，许多人并不喜欢这样的事实。[2]

可以采用折中的办法。比如，可以把许多项目按照彼此之间的关联程度划分为几个组合，然后为每一个项目组合建立一个版本库。这样，在相关项目之间共享数据依旧很简单，而如果修订版本号有了变化，至少开发人员知道，改变的东西多少和他们有些关系。

After deciding how to organize your projects with respect to repositories, you'll probably want to think about directory hierarchies within the repositories themselves. Because Subversion uses regular directory copies for branching and tagging (see 第 4 章 分支与合并), the Subversion community recommends that you choose a repository location for each *project root*—the "topmost" directory that contains data related to that project—and then create three subdirectories beneath that root: `trunk`, meaning the directory under which the main project development occurs; `branches`, which is a directory in which to create various named branches of the main development line; and `tags`, which is a collection of tree snapshots that are created, and perhaps destroyed, but never changed. [3]

For example, your repository might look like this:

```
/
   calc/
      trunk/
      tags/
      branches/
   calendar/
      trunk/
      tags/
      branches/
   spreadsheet/
      trunk/
      tags/
      branches/
   ...
```

Note that it doesn't matter where in your repository each project root is. If you have only one project per repository, the logical place to put each project root is at the root of that project's respective repository. If you have multiple projects, you might want to arrange them in groups inside the repository, perhaps putting projects with similar goals or shared code in the same subdirectory, or maybe just grouping them alphabetically. Such an arrangement might look like this:

[2]无论是在忽略情况下建立或很少考虑过如何产生正确的软件开发矩阵，都不应该愚蠢的担心全局的修订版本号码，这不应该成为安排项目和版本库的理由。

[3]The `trunk`, `tags`, and `branches` trio is sometimes referred to as "the TTB directories."

```
/
   utils/
      calc/
         trunk/
         tags/
         branches/
      calendar/
         trunk/
         tags/
         branches/
      …
   office/
      spreadsheet/
         trunk/
         tags/
         branches/
      …
```

按照你认为合适的方式安排版本库的布局，Subversion自身并不强制或者偏好某一种布局形式，对于Subversion来说，目录就是目录。最后，在设计版本库布局的时候，不要忘了考虑一下项目参与者们的意见。

In the name of full disclosure, though, we'll mention another very common layout. In this layout, the `trunk`, `tags`, and `branches` directories live in the root directory of your repository, and your projects are in subdirectories beneath those, like so:

```
/
   trunk/
      calc/
      calendar/
      spreadsheet/
      …
   tags/
      calc/
      calendar/
      spreadsheet/
      …
   branches/
      calc/
      calendar/
      spreadsheet/
      …
```

There's nothing particularly incorrect about such a layout, but it may or may not seem as intuitive for your users. Especially in large, multiproject situations with many users, those users may tend to be familiar with only one or two of the projects in the repository. But the projects-as-branch-siblings approach tends to deemphasize project individuality and focus on the entire set of projects as a single

entity. That's a social issue, though. We like our originally suggested arrangement for purely practical reasons—it's easier to ask about (or modify, or migrate elsewhere) the entire history of a single project when there's a single repository path that holds the entire history—past, present, tagged, and branched—for that project and that project alone.

5.2.2. 决定在哪里与如何部署你的版本库

Before creating your Subversion repository, an obvious question you'll need to answer is where the thing is going to live. This is strongly connected to myriad other questions involving how the repository will be accessed (via a Subversion server or directly), by whom (users behind your corporate firewall or the whole world out on the open Internet), what other services you'll be providing around Subversion (repository browsing interfaces, email-based commit notification, etc.), your data backup strategy, and so on.

我们在第 6 章 服务配置覆盖了服务器的选择和配置，我们也提供一些可能会使你必须决定使用某种服务器的问题的答案。例如，特定的部署策略可能会需要从多个计算机通过远程文件系统访问版本库，这个情况下(下一小节会读到)要求你不能选择一种版本库后端数据存储方式，因为只有一种后端在这种场景下可以工作。

列出所有的Subversion可能的部署方法是不可能的，超出了本书的范围，我们只是简单的鼓励你使用这部分内容和参考材料验证你的想法，并预先计划。

5.2.3. 选择数据存储格式

在Subversion1.1中，版本库中有两种数据存储方式—通常叫做“后端”或其他容易混淆的名字，如“(版本化的)文件系统”，每一个版本库都会使用一种。一种是在Berkeley DB数据库中存储数据，我们称之为“BDB后端”；另一种是使用普通的文件，自定义格式，Subversion开发者根据习惯称之为*FSFS*[4] —一种使用本地操作系统文件存储数据的版本化文件系统直接实现—而不是通过某个数据库层或其他抽象层来保存数据。

表 5.1 “版本库数据存储对照表”从总体上比较了Berkeley DB和FSFS版本库。

表 **5.1.** 版本库数据存储对照表

分类	特性	**Berkeley DB**	**FSFS**
可靠性	数据完整性	When properly deployed, extremely reliable; Berkeley DB 4.4 brings auto-recovery	Older versions had some rarely demonstrated, but data-destroying bugs
	对操作中断的敏感	Very; crashes and permission problems can leave the database “楔住” requiring journaled recovery procedures	Quite insensitive

[4]通常读作“fuzz-fuzz”, 如果Jack Repenning说起这个问题。(本书，假定读者认为是“eff-ess-eff-ess”。)

分类	特性	**Berkeley DB**	**FSFS**
可用性	可只读加载	否	是
	存储平台无关	否	是
	可从网络文件系统访问	Generally, no	是
	组访问权处理	Sensitive to user umask problems; best if accessed by only one user	Works around umask problems
伸缩性	版本库磁盘使用情况	Larger (especially if logfiles aren't purged)	Smaller
	修订版本树的数量	Database; no problems	Some older native filesystems don't scale well with thousands of entries in a single directory
	有很多文件的目录	Slower	Faster
性能	检出最新的代码	No meaningful difference	No meaningful difference
	大的提交	Slower overall, but cost is amortized across the lifetime of the commit	Faster overall, but finalization delay may cause client timeouts

两种后端都有优点和缺点，没有一种更加“正式”，尽管新的FSFS在Subversion1.2成为缺省数据存储，两者用来存储版本化数据都是可靠的。但是就象你在表 5.1 “版本库数据存储对照表”看到的，FSFS后端在部署场景中提供了更多的灵活性，更灵活意味着你很难错误的配置。那些原因—加上不使用Berkeley DB意味着在这个系统有更少的组件—这就是为什么今天几乎所有的人都使用FSFS来创建新的版本库。

幸运的是，大多数访问Subversion的程序不会在意其所用的后端数据存储。而且你不必一定要使用你最初的数据存储方法—如果后来你改变了主意，Subversion提供了移植版本库数据到另一个版本库的方法，我们会在后面详细讨论。

下面的小节提供了数据存储类型更加详细的介绍。

5.2.3.1. Berkeley DB

When the initial design phase of Subversion was in progress, the developers decided to use Berkeley DB for a variety of reasons, including its open source license, transaction support, reliability, performance, API simplicity, thread safety, support for cursors, and so on.

Berkeley DB provides real transaction support—perhaps its most powerful feature. Multiple processes accessing your Subversion repositories don't have to worry about accidentally clobbering each other's data. The isolation provided by the transaction system is such that for any given operation, the Subversion repository code sees a static view of the database—not a database that is constantly changing at the hand of some other process—and can make decisions based on that view. If the decision

made happens to conflict with what another process is doing, the entire operation is rolled back as though it never happened, and Subversion gracefully retries the operation against a new, updated (and yet still static) view of the database.

Berkeley DB另一个强大的特性是热备份 – 不必“脱机”就可以备份数据库环境的能力。我们将会在第 5.4.8 节 “版本库备份”讨论如何备份你的版本库，能够不停止系统对版本库做全面备份的好处是显而易见的。

Berkeley DB同时是一个可信赖的数据库系统。Subversion利用了Berkeley DB可以记日志的便利，这意味着数据库先在磁盘上写一个日志文件，描述它将要做的修改，然后再做这些修改。这是为了确保如果如果任何地方出了差错，数据库系统能恢复到先前的检查点—一个日志文件认为没有错误的位置，重新开始事务直到数据恢复为一个可用的状态。关于Berkeley DB日志文件的更多信息请查看第 5.4.3 节 “管理磁盘空间”。

But every rose has its thorn, and so we must note some known limitations of Berkeley DB. First, Berkeley DB environments are not portable. You cannot simply copy a Subversion repository that was created on a Unix system onto a Windows system and expect it to work. While much of the Berkeley DB database format is architecture-independent, other aspects of the environment are not. Second, Subversion uses Berkeley DB in a way that will not operate on Windows 95/98 systems—if you need to house a BDB-backed repository on a Windows machine, stick with Windows 2000 or later.

然而Berkeley DB对于在网络共享上工作提出了一组规范，[5]大多数网络文件系统和应用没有实现这个要求，所以不能允许在网络共享上的BDB后端版本库被多个客户端同时访问(首先要知道版本库存放在网络共享上是非常普遍的)。

警告

如果你尝试在不顺从的远程文件系统上使用Berkeley DB，结果是不可预知的—你会立刻看到神秘的错误，或者是在发生隐含错误之后几个月之后才发现。你必须认真考虑在网络共享情况下使用FSFS数据存储。

最后，因为Berkeley DB的库直接链接到了Subversion中，它对于中断比典型的关系型数据库系统更为敏感。大多数SQL系统，举例来说，有一个主服务进程来协调对数据库表的访问。如果一个访问数据库的程序因为某种原因出现问题，数据库守护进程察觉到连接中断会做一些清理。因为数据库守护进程是唯一访问数据库表的进程，应用程序不需要担心访问许可的冲突。但是，这些情况与Berkeley DB不同。Subversion(和使用Subversion库的程序)直接访问数据库的表，这意味着如果有一个程序崩溃，就会使数据库处于一个暂时的不一致、不可访问的状态。当这种情况发生时，管理员需要让Berkeley DB恢复到一个检查点，这的确有点讨厌。除了崩溃的进程，还有一些情况能让版本库出现异常，比如程序在数据库文件的所有权或访问权限上发生冲突。

注意

Berkeley DB 4.4 brings (to Subversion 1.4 and later) the ability for Subversion to automatically and transparently recover Berkeley DB environments in need of such recovery. When a Subversion process attaches to a repository's Berkeley DB environment, it uses some process accounting mechanisms to detect any unclean disconnections by previous processes, performs any necessary recovery, and then continues on as though

[5]Berkeley DB需要底层的文件系统实现严格的POSIX锁定语法，更重要的是，将文件直接映射到内存的能力。

nothing happened. This doesn't completely eliminate instances of repository wedging, but it does drastically reduce the amount of human interaction required to recover from them.

So while a Berkeley DB repository is quite fast and scalable, it's best used by a single server process running as one user—such as Apache's **httpd** or **svnserve** (see 第 6 章 服务配置)—rather than accessing it as many different users via `file://` or `svn+ssh://` URLs. If you're accessing a Berkeley DB repository directly as multiple users, be sure to read 第 6.6 节 “支持多种版本库访问方法” later in this chapter.

5.2.3.2. FSFS

在2004年中期，另一种版本库存储系统慢慢形成了：一种不需要数据库的存储系统。FSFS版本库在单一文件中存储修订版本树，所以版本库中所有的修订版本都在一个子文件夹中有限的几个文件里。事务在单独的子目录中被创建，创建完成后，一个单独的事务文件被创建并移动到修订版本目录，这保证提交是原子性的。因为一个修订版本文件是持久不可改变的，版本库也可以做到“热”备份，就象Berkeley DB版本库一样。

The FSFS revision files describe a revision's directory structure, file contents, and deltas against files in other revision trees. Unlike a Berkeley DB database, this storage format is portable across different operating systems and isn't sensitive to CPU architecture. Because no journaling or shared-memory files are being used, the repository can be safely accessed over a network filesystem and examined in a read-only environment. The lack of database overhead also means the overall repository size is a bit smaller.

FSFS has different performance characteristics, too. When committing a directory with a huge number of files, FSFS is able to more quickly append directory entries. On the other hand, FSFS writes the latest version of a file as a delta against an earlier version, which means that checking out the latest tree is a bit slower than fetching the full-texts stored in a Berkeley DB HEAD revision. FSFS also has a longer delay when finalizing a commit, which could in extreme cases cause clients to time out while waiting for a response.

最重要的区别是当出现错误时FSFS不会楔住的能力。如果使用Berkeley DB的进程发生许可错误或突然崩溃，数据库会一直无法使用，直到管理员恢复。假如在应用FSFS版本库时发生同样的情况，版本库不会受到任何干扰，最坏情况下也就是会留下一些事务数据。

The only real argument against FSFS is its relative immaturity compared to Berkeley DB. Unlike Berkeley DB, which has years of history, its own dedicated development team, and, now, Oracle's mighty name attached to it, [6] FSFS is a newer bit of engineering. Prior to Subversion 1.4, it was still shaking out some pretty serious data integrity bugs, which, while triggered in only very rare cases, nonetheless did occur. That said, FSFS has quickly become the backend of choice for some of the largest public and private Subversion repositories, and it promises a lower barrier to entry for Subversion across the board.

5.3. 创建和配置你的版本库

Earlier in this chapter (in 第 5.2 节 “版本库开发策略”), we looked at some of the important decisions that should be made before creating and configuring your Subversion repository. Now, we

[6] Oracle在2006情人节购买了Sleepycat和它的旗舰软件Berkeley DB。

finally get to get our hands dirty! In this section, we'll see how to actually create a Subversion repository and configure it to perform custom actions when special repository events occur.

5.3.1. 创建版本库

创建一个Subversion版本库出乎寻常的简单。 Subversion提供的**svnadmin** 工具，有一个执行这个功能的子命令(**svnadmin create**)。

```
$ # Create a repository
$ svnadmin create /var/svn/repos
$
```

这样在目录`/path/to/repos`使用默认数据存储方式创建了一个新的版本库。在Subversion 1.2之前，缺省值是Berkeley DB；而现在是FSFS。你可以通过`--fs-type`参数明确地指定文件系统类型，可选的值包括`fsfs`和`bdb`。

```
$ # Create an FSFS-backed repository
$ svnadmin create --fs-type fsfs /var/svn/repos
$
```

```
# Create a Berkeley-DB-backed repository
$ svnadmin create --fs-type bdb /var/svn/repos
$
```

运行这个命令之后，你有了一个Subversion版本库。

提示

你可能已经注意到了，**svnadmin**命令的路径参数只是一个普通的文件系统路径，而不是一个**svn**客户端程序访问版本库时使用的URL。**svnadmin**和**svnlook**都被认为是服务器端工具—它们在版本库所在的机器上使用，用来检查或修改版本库，不能通过网络来执行任务。一个Subversion的新手通常会犯的错误，就是试图将URL(甚至“本地”`file:`路径)传给这两个程序。

这个命令在目录`/path/to/repos`创建了一个新的版本库。这个新的版本库会以修订版本版本0开始其生命周期，里面除了最上层的根目录(`/`)，什么都没有。刚开始，修订版本0有一个修订版本属性`svn:date`，设置为版本库创建的时间。

现在你有了一个版本库，可以用户化了。

警告

一般来说，版本库除了一小部分—例如配置文件和钩子脚本，你不要(也不需要)手动干预版本库。**svnadmin**工具应该足以用来处理对版本库的任何修改，或者你也可以使用第三方工具(比如Berkeley DB的工具包)来调整部分版本库。不要尝试通过处理版本库数据存储文件手工修改版本控制历史！

5.3.2. 实现版本库钩子

A *hook* is a program triggered by some repository event, such as the creation of a new revision or the modification of an unversioned property. Some hooks (the so-called “pre hooks”) run in advance of a repository operation and provide a means by which to both report what is about to happen and prevent it from happening at all. Other hooks (the “post hooks”) run after the completion of a repository event and are useful for performing tasks that examine—but don't modify—the repository. Each hook is handed enough information to tell what that event is (or was), the specific repository changes proposed (or completed), and the username of the person who triggered the event.

默认情况下，hooks子目录中包含各种版本库钩子模板。

```
$ ls repos/hooks/
post-commit.tmpl           post-unlock.tmpl   pre-revprop-change.tmpl
post-lock.tmpl             pre-commit.tmpl    pre-unlock.tmpl
post-revprop-change.tmpl   pre-lock.tmpl      start-commit.tmpl
$
```

对每种Subversion版本库支持的钩子的都有一个模板，通过查看这些脚本的内容，你能看到是什么事件触发了脚本及如何给传脚本传递数据。同时，这些模版也是如何使用这些脚本，结合Subversion支持的工具来完成有用任务的例子。要实际安装一个可用的钩子，你需要在repos/hooks目录下安装一些与钩子同名(如 **start-commit**或者**post-commit**)的可执行程序或脚本。

在Unix平台上，这意味着要提供一个与钩子同名的脚本或程序(可能是shell 脚本，Python 程序，编译过的c语言二进制文件或其他东西)。当然，脚本模板文件不仅仅是展示了一些信息—在Unix下安装钩子最简单的办法就是拷贝这些模板，并且去掉.tmpl扩展名，然后自定义钩子的内容，确定脚本是可运行的。Windows用文件的扩展名来决定一个程序是否可运行，所以你要使程序的基本名与钩子同名，同时，它的扩展名是Windows系统所能辨认的，例如exe、com和批处理的bat。

提示

由于安全原因，Subversion版本库在一个空环境中执行钩子脚本—就是没有设置任何环境变量，甚至没有$PATH或%PATH%。由于这个原因，许多管理员会感到很困惑，它们的钩子脚本手工运行时正常，可在Subversion中却不能运行。要注意，必须在你的钩子中设置好环境变量或为你的程序指定好绝对路径。

Subversion会试图以当前访问版本库的用户身份执行钩子。通常，对版本库的访问总是通过Apache HTTP服务器和mod_dav_svn进行，因此，执行钩子的用户就是运行Apache的用户。钩子本身需要具有操作系统级的访问许可，用户可以运行它。另外，其它被钩子直接或间接使用的文件或程序(包括Subversion版本库本身)也要被同一个用户访问。换句话说，要注意潜在的访问控制问题，它可能会让你的钩子无法按照你的目的顺利执行。

There are serveral hooks implemented by the Subversion repository, and you can get details about each of them in 第 9.11 节 “版本库钩子”. As a repository administrator, you'll need to decide which hooks you wish to implement (by way of providing an appropriately named and permissioned hook program), and how. When you make this decision, keep in mind the big picture of how your repository

is deployed. For example, if you are using server configuration to determine which users are permitted to commit changes to your repository, you don't need to do this sort of access control via the hook system.

There is no shortage of Subversion hook programs and scripts that are freely available either from the Subversion community itself or elsewhere. These scripts cover a wide range of utility—basic access control, policy adherence checking, issue tracker integration, email- or syndication-based commit notification, and beyond. Or, if you wish to write your own, see 第 8 章 嵌入 *Subversion*.

警告

尽管经过调整钩子脚本可以作任何事情，但钩子脚本的作者仍会受到一些限制：不要修改使用钩子脚本修改提交事务，因为使用钩子脚本自动修改错误或提交文件的政策违例的尝试会导致问题。Subversion会在客户端缓存对应的版本库数据，如果你这样修改了提交事务，这些缓存就进入了未知的状态，这种不一致会导致令人吃惊和预想不到的行为。作为对事物修改的替换，你可以简单的在`pre-commit`确认事物信息并且拒绝提交，如果这样满足不了需求，作为额外的奖赏，你的用户会学会小心顺从的工作习惯。

5.3.3. Berkeley DB 配置

A Berkeley DB environment is an encapsulation of one or more databases, logfiles, region files, and configuration files. The Berkeley DB environment has its own set of default configuration values for things such as the number of database locks allowed to be taken out at any given time, the maximum size of the journaling logfiles, and so on. Subversion's filesystem logic additionally chooses default values for some of the Berkeley DB configuration options. However, sometimes your particular repository, with its unique collection of data and access patterns, might require a different set of configuration option values.

The producers of Berkeley DB understand that different applications and database environments have different requirements, so they have provided a mechanism for overriding at runtime many of the configuration values for the Berkeley DB environment. BDB checks for the presence of a file named `DB_CONFIG` in the environment directory (namely, the repository's `db` subdirectory), and parses the options found in that file. Subversion itself creates this file when it creates the rest of the repository. The file initially contains some default options, as well as pointers to the Berkeley DB online documentation so that you can read about what those options do. Of course, you are free to add any of the supported Berkeley DB options to your `DB_CONFIG` file. Just be aware that while Subversion never attempts to read or interpret the contents of the file and makes no direct use of the option settings in it, you'll want to avoid any configuration changes that may cause Berkeley DB to behave in a fashion that is at odds with what Subversion might expect. Also, changes made to `DB_CONFIG` won't take effect until you recover the database environment (using **svnadmin recover**).

5.4. 版本库维护

Maintaining a Subversion repository can be daunting, mostly due to the complexities inherent in systems that have a database backend. Doing the task well is all about knowing the tools—what they are, when to use them, and how. This section will introduce you to the repository administration tools

provided by Subversion and discuss how to wield them to accomplish tasks such as repository data migration, upgrades, backups, and cleanups.

5.4.1. 管理员的工具箱

Subversion提供了一些用来创建、查看、修改和修复版本库的工具。让我们首先详细了解一下每个工具，然后，我们再看一下仅在Berkeley DB后端分发版本中提供的版本数据库工具。首先，我们简短检查一下Berkeley DB发布包含的针对版本库后端数据库的工具，不是Subversion本身提供的工具。

5.4.1.1. svnadmin

svnadmin程序是版本库管理员最好的朋友。除了提供创建Subversion版本库的功能，这个程序使你可以维护这些版本库。**svnadmin**的语法同其他Subversion命令类似：

```
$ svnadmin help
general usage: svnadmin SUBCOMMAND REPOS_PATH  [ARGS & OPTIONS ...]
Type 'svnadmin help <subcommand>' for help on a specific subcommand.
Type 'svnadmin --version' to see the program version and FS modules.

Available subcommands:
   crashtest
   create
   deltify
…
```

在本章的前面)(在第 5.3.1 节 “创建版本库”)，我们已经讨论了**svnadmin**的`create`子命令，本章后面我们会详细讲解大多数其他的子命令，关于所有的子命令你可以参考第 9.2 节 “svnadmin”。

5.4.1.2. svnlook

svnlook是Subversion提供的用来查看版本库中不同的修订版本和事务(正在产生的修订版本)。这个程序不会修改版本库内容 – 这是个“只读”的工具。**svnlook**通常用在版本库钩子程序中，用来记录版本库即将提交(用在**pre-commit**钩子时)或者已经提交的(用在**post-commit**钩子时)修改。版本库管理员可以将这个工具用于诊断。

svnlook的语法很直接：

```
$ svnlook help
general usage: svnlook SUBCOMMAND REPOS_PATH [ARGS & OPTIONS ...]
Note: any subcommand which takes the '--revision' and '--transaction'
      options will, if invoked without one of those options, act on
      the repository's youngest revision.
Type 'svnlook help <subcommand>' for help on a specific subcommand.
Type 'svnlook --version' to see the program version and FS modules.
…
```

几乎**svnlook**的每一个子命令都能操作修订版本或事务树，显示树本身的信息，或是它与版本库中上一个修订版本的不同。你可以用`--revision(-r)`和`--transaction(-t)`选项指定要查看的修订版本或事务。如果没有指定`--revision(-r)`和`--transaction(-t)`选项，**svnlook**会检查版本库最新的(或者说“HEAD”)修订版本。所以当19是位于`/path/to/repos`的版本库的最新版本时，如下的两个命令起到相同的效果：

```
$ svnlook info /var/svn/repos
$ svnlook info /var/svn/repos -r 19
```

这些子命令的唯一例外是**svnlook youngest**，它不需要任何选项，只会打印出版本库的最新修订版本号：

```
$ svnlook youngest /var/svn/repos
19
$
```

注意

请记住只能浏览未提交的事物，大多数版本库没有这样的事物，因为事物要么是已经提交的(也就是你可以`--revision(-r)`访问的修订版本)，要么是退出的和删除的。

svnlook的输出被设计为人和机器都易理解，拿**svnlook info**子命令举例来说：

```
$ svnlook info /var/svn/repos
sally
2002-11-04 09:29:13 -0600 (Mon, 04 Nov 2002)
27
Added the usual
Greek tree.
$
```

svnlook info的输出包含如下的内容，按照给定的顺序：

1. The author, followed by a newline

2. The date, followed by a newline

3. The number of characters in the log message, followed by a newline

4. The log message itself, followed by a newline

This output is human-readable, meaning items such as the datestamp are displayed using a textual representation instead of something more obscure (such as the number of nanoseconds since the Tastee Freez guy drove by). But the output is also machine-parsable—because the log message can contain multiple lines and be unbounded in length, **svnlook** provides the length of that message before the message itself. This allows scripts and other wrappers around this command to make intelligent

decisions about the log message, such as how much memory to allocate for the message, or at least how many bytes to skip in the event that this output is not the last bit of data in the stream.

svnlook还可以做很多别的查询：显示我们先前提到的信息的一些子集，递归显示版本目录树，报告指定的修订版本或事务中哪些路径曾经被修改过，显示对文件和目录做过的文本和属性的修改，等等。第 9.3 节 “svnlook”是**svnlook**命令能接受子命令的完全特性参考。

5.4.1.3. svndumpfilter

虽然在管理员的日常工作中并不会经常使用，不过**svndumpfilter**提供了一项特别有用的功能—可以简单快速的作为Subversion版本库历史的以路径为基础的过滤器。

svndumpfilter的语法如下：

```
$ svndumpfilter help
general usage: svndumpfilter SUBCOMMAND [ARGS & OPTIONS ...]
Type "svndumpfilter help <subcommand>" for help on a specific subcommand.
Type 'svndumpfilter --version' to see the program version.

Available subcommands:
   exclude
   include
   help (?, h)
```

There are only two interesting subcommands: **svndumpfilter exclude** and **svndumpfilter include**. They allow you to make the choice between implicit or explicit inclusion of paths in the stream. You can learn more about these subcommands and **svndumpfilter**'s unique purpose later in this chapter, in 第 5.4.6 节 “过滤版本库历史”.

5.4.1.4. svnsync

svnsync程序是Subversion 1.4版的新特性，提供了维护一个只读版本库镜像的全部功能。这个程序只有一个工作—将一个版本库的历史转移到另一个，尽管有几种方法，但这种方法的主要特点是可以远程操作—“源”，“目标”[7]版本库以及**svnsync**程序可能在不同的计算机上。

就像你期望的，**svnsync**的语法与本节提到的其他命令非常类似。

```
$ svnsync help
general usage: svnsync SUBCOMMAND DEST_URL  [ARGS & OPTIONS ...]
Type 'svnsync help <subcommand>' for help on a specific subcommand.
Type 'svnsync --version' to see the program version and RA modules.

Available subcommands:
   initialize (init)
   synchronize (sync)
```

[7]或者是，“sync”？

```
   copy-revprops
   help (?, h)
$
```

我们会在第 5.4.7 节 “版本库复制”详细讨论使用**svnsync**实现版本库复制。

5.4.1.5. fsfs-reshard.py

While not an official member of the Subversion toolchain, the **fsfs-reshard.py** script (found in the `tools/server-side` directory of the Subversion source distribution) is a useful performance tuning tool for administrators of FSFS-backed Subversion repositories. FSFS repositories contain files that describe the changes made in a single revision, and files that contain the revision properties associated with a single revision. Repositories created in versions of Subversion prior to 1.5 keep these files in two directories—one for each type of file. As new revisions are committed to the repository, Subversion drops more files into these two directories—over time, the number of these files in each directory can grow to be quite large. This has been observed to cause performance problems on certain network-based filesystems.

Subversion 1.5 creates FSFS-backed repositories using a slightly modified layout in which the contents of these two directories are *sharded*, or scattered across several subdirectories. This can greatly reduce the time it takes the system to locate any one of these files, and therefore increases the overall performance of Subversion when reading from the repository. The number of subdirectories used to house these files is configurable, though, and that's where **fsfs-reshard.py** comes in. This script reshuffles the repository's file structure into a new arrangement that reflects the requested number of sharding subdirectories. This is especially useful for converting an older Subversion repository into the new Subversion 1.5 sharded layout (which Subversion will not automatically do for you) or for fine-tuning an already sharded repository.

5.4.1.6. Berkeley DB 工具

If you're using a Berkeley DB repository, all of your versioned filesystem's structure and data live in a set of database tables within the `db/` subdirectory of your repository. This subdirectory is a regular Berkeley DB environment directory and can therefore be used in conjunction with any of the Berkeley database tools, typically provided as part of the Berkeley DB distribution.

For day-to-day Subversion use, these tools are unnecessary. Most of the functionality typically needed for Subversion repositories has been duplicated in the **svnadmin** tool. For example, **svnadmin list-unused-dblogs** and **svnadmin list-dblogs** perform a subset of what is provided by the Berkeley **db_archive** utility, and **svnadmin recover** reflects the common use cases of the **db_recover** utility.

当然，还有一些Berkeley DB工具有时是有用的。**db_load**和**db_dump**分别将Berkeley DB数据库中的键值对以特定的格式读写文件。Berkeley数据库本身不支持跨平台转移，这两个工具在这样的情况下就可以实现在平台间转移数据库的功能，而无需关心操作系统或机器架构。就像我们以前描述的，你可以使用**svnadmin dump**和**svnadmin load**实现类似的目的，但是**db_dump**和**db_load**可以更快一点，它们也可以协助Berkeley DB的hacker来篡改BDB后端的数据，这是Subversion工具不允许的。此外，**db_stat**工具能够提供关于Berkeley DB环境的许多有用信息，包括详细的锁定和存储子系统的统计信息。

关于Berkeley DB工具的更多信息，可以访问Oracle网站的Berkeley DB文档部分，在`http://www.oracle.com/technology/documentation/berkeley-db/db/`。

5.4.2. 修正提交消息

Sometimes a user will have an error in her log message (a misspelling or some misinformation, perhaps). If the repository is configured (using the `pre-revprop-change` hook; see 第 5.3.2 节 “实现版本库钩子”) to accept changes to this log message after the commit is finished, the user can “fix” her log message remotely using **svn propset** (see svn propset). However, because of the potential to lose information forever, Subversion repositories are not, by default, configured to allow changes to unversioned properties—except by an administrator.

如果管理员想要修改日志信息，那么可以使用**svnadmin setlog**命令。这个命令从指定的文件中读取信息，取代版本库中某个修订版本的日志信息(`svn:log`属性)。

```
$ echo "Here is the new,  correct log message" > newlog.txt
$ svnadmin setlog myrepos newlog.txt -r 388
```

即使是**svnadmin setlog**命令也受到限制。`pre-`和 `post-revprop-change`钩子同样会被触发，因此必须进行相应的设置才能允许修改非版本化属性。不过管理员可以使用**svnadmin setlog**命令的`--bypass-hooks`选项跳过钩子。

警告

不过需要注意的是，一旦跳过钩子也就跳过了钩子所提供的所有功能，比如邮件通知(通知属性有改动)、系统备份(可以用来跟踪非版本化的属性变更)等等。换句话说，要留心你所作出的修改，以及你作出修改的方式。

5.4.3. 管理磁盘空间

虽然存储器的价格在过去的几年里以让人难以致信的速度滑落，但是对于那些需要对大量数据进行版本管理的管理员们来说，磁盘空间的消耗依然是一个重要的因素。版本库每增加一个字节都意味着需要多一个字节的磁盘空间进行备份，对于多重备份来说，就需要消耗更多的磁盘空间。Berkeley DB版本库的主要存储机制是基于一个复杂的数据库系统建立的，因此了解一些数据性质是有意义的，比如哪些数据必须保持在线，哪些数据需要备份、哪些数据可以安全的删除等等。

5.4.3.1. 让 Subversion 节约磁盘空间

To keep the repository small, Subversion uses *deltification* (or deltified storage) within the repository itself. Deltification involves encoding the representation of a chunk of data as a collection of differences against some other chunk of data. If the two pieces of data are very similar, this deltification results in storage savings for the deltified chunk—rather than taking up space equal to the size of the original data, it takes up only enough space to say, “I look just like this other piece of data over here, except for the following couple of changes.” The result is that most of the repository data that tends to be bulky—namely, the contents of versioned files—is stored at a much smaller size than the original full-text representation of that data. And for repositories created with Subversion 1.4 or later, the space savings are even better—now those full-text representations of file contents are themselves compressed.

注意

由于Subversion版本库的增量化数据保存在单一Berkeley DB数据库文件中，减少数据的体积并不一定能够减小数据库文件的大小。但是，Berkeley DB会在内部记录未使用的数据库文件区域，并且在增加数据库文件大小之前会首先使用这些未使用的区域。因此，即使增量化技术不能立杆见影的节省磁盘空间，也可以极大的减慢数据库的膨胀速度。

5.4.3.2. 删除终止的事务

尽管不太常见，Subversion的提交进程也有失败，同时留下将要生成的修订版本—未提交的事物和所有随之的文件和目录修改。出现这种情况可能有以下原因：客户端的用户粗暴的结束了操作，操作过程中出现网络故障，等等。不管是什么原因，死亡的事务总是有可能会出现。这类事务不会产生什么负面影响，仅仅是消耗了一点点磁盘空间。不过，严厉的管理员总是希望能够将它们清除出去。

可以使用**svnadmin lstxns**命令列出当前的事务名。

```
$ svnadmin lstxns myrepos
19
3a1
a45
$
```

Each item in the resultant output can then be used with **svnlook** (and its `--transaction` (`-t`) option) to determine who created the transaction, when it was created, what types of changes were made in the transaction—information that is helpful in determining whether the transaction is a safe candidate for removal! If you do indeed want to remove a transaction, its name can be passed to **svnadmin rmtxns**, which will perform the cleanup of the transaction. In fact, **svnadmin rmtxns** can take its input directly from the output of **svnadmin lstxns**!

```
$ svnadmin rmtxns myrepos `svnadmin lstxns myrepos`
$
```

在按照上面例子中的方法清理版本库之前，你或许应该暂时关闭版本库和客户端的连接。这样在你开始清理之前，不会有正常的事务进入版本库。例 5.1 “txn-info.sh (报告异常事务)”中的shell脚本可以用来迅速获得版本库中异常事务的信息。

例 **5.1. txn-info.sh** (报告异常事务)

```
#!/bin/sh

### Generate informational output for all outstanding transactions in
### a Subversion repository.

REPOS="${1}"
if [ "x$REPOS" = x ] ; then
  echo "usage: $0 REPOS_PATH"
  exit
fi

for TXN in `svnadmin lstxns ${REPOS}`; do
  echo "---[ Transaction ${TXN} ]-------------------------------------------
  svnlook info "${REPOS}" -t "${TXN}"
done
```

The output of the script is basically a concatenation of several chunks of **svnlook info** output (see 第 5.4.1.2 节 “svnlook”) and will look something like this:

```
$ txn-info.sh myrepos
---[ Transaction 19 ]-------------------------------------------
sally
2001-09-04 11:57:19 -0500 (Tue,  04 Sep 2001)
0
---[ Transaction 3a1 ]------------------------------------------
harry
2001-09-10 16:50:30 -0500 (Mon,  10 Sep 2001)
39
Trying to commit over a faulty network.
---[ Transaction a45 ]------------------------------------------
sally
2001-09-12 11:09:28 -0500 (Wed,  12 Sep 2001)
0
$
```

一个废弃了很长时间的事务通常是提交错误或异常中断的结果。事务的时间戳可以提供给我们一些有趣的信息，比如一个进行了9个月的操作居然还是活动的等等。

In short, transaction cleanup decisions need not be made unwisely. Various sources of information—including Apache's error and access logs, Subversion's operational logs, Subversion revision history, and so on—can be employed in the decision-making process. And of course, an administrator can often simply communicate with a seemingly dead transaction's owner (via email, e.g.) to verify that the transaction is, in fact, in a zombie state.

5.4.3.3. 删除不使用的 Berkeley DB 日志文件

Until recently, the largest offender of disk space usage with respect to BDB-backed Subversion repositories were the logfiles in which Berkeley DB performs its prewrites before modifying the actual database files. These files capture all the actions taken along the route of changing the database from one state to another—while the database files, at any given time, reflect a particular state, the logfiles contain all of the many changes along the way *between* states. Thus, they can grow and accumulate quite rapidly.

Fortunately, beginning with the 4.2 release of Berkeley DB, the database environment has the ability to remove its own unused logfiles automatically. Any repositories created using **svnadmin** when compiled against Berkeley DB version 4.2 or later will be configured for this automatic logfile removal. If you don't want this feature enabled, simply pass the `--bdb-log-keep` option to the **svnadmin create** command. If you forget to do this or change your mind at a later time, simply edit the `DB_CONFIG` file found in your repository's `db` directory, comment out the line that contains the `set_flags DB_LOG_AUTOREMOVE` directive, and then run **svnadmin recover** on your repository to force the configuration changes to take effect. See 第 5.3.3 节 “Berkeley DB 配置” for more information about database configuration.

Without some sort of automatic logfile removal in place, logfiles will accumulate as you use your repository. This is actually somewhat of a feature of the database system—you should be able to recreate your entire database using nothing but the logfiles, so these files can be useful for catastrophic database recovery. But typically, you'll want to archive the logfiles that are no longer in use by Berkeley DB, and then remove them from disk to conserve space. Use the **svnadmin list-unused-dblogs** command to list the unused logfiles:

```
$ svnadmin list-unused-dblogs /var/svn/repos
/var/svn/repos/log.0000000031
/var/svn/repos/log.0000000032
/var/svn/repos/log.0000000033
…
$ rm `svnadmin list-unused-dblogs /var/svn/repos`
## disk space reclaimed!
```

警告

BDB-backed repositories whose logfiles are used as part of a backup or disaster recovery plan should *not* make use of the logfile autoremoval feature. Reconstruction of a repository's data from logfiles can only be accomplished only when *all* the logfiles are available. If some of the logfiles are removed from disk before the backup system has a chance to copy them elsewhere, the incomplete set of backed-up logfiles is essentially useless.

5.4.4. Berkeley DB 恢复

As mentioned in 第 5.2.3.1 节 “Berkeley DB”, a Berkeley DB repository can sometimes be left in a frozen state if not closed properly. When this happens, an administrator needs to rewind the database back into a consistent state. This is unique to BDB-backed repositories, though—if you are using FSFS-

backed ones instead, this won't apply to you. And for those of you using Subversion 1.4 with Berkeley DB 4.4 or later, you should find that Subversion has become much more resilient in these types of situations. Still, wedged Berkeley DB repositories do occur, and an administrator needs to know how to safely deal with this circumstance.

To protect the data in your repository, Berkeley DB uses a locking mechanism. This mechanism ensures that portions of the database are not simultaneously modified by multiple database accessors, and that each process sees the data in the correct state when that data is being read from the database. When a process needs to change something in the database, it first checks for the existence of a lock on the target data. If the data is not locked, the process locks the data, makes the change it wants to make, and then unlocks the data. Other processes are forced to wait until that lock is removed before they are permitted to continue accessing that section of the database. (This has nothing to do with the locks that you, as a user, can apply to versioned files within the repository; we try to clear up the confusion caused by this terminology collision in the sidebar 锁定的三种含义.)

在操作Subversion版本库的过程中，致命错误(如内存或硬盘空间不足)或异常中断可能会导致某个进程没能及时将锁解除。结果就是后端的数据库系统被“楔住”了。一旦发生这种情况，任何访问版本库的进程都会挂起(每个访问进程都在等待锁被解除，但是锁已经无法解除了)。

如果你的版本库出现这种情况，没什么好惊慌的。Berkeley DB的文件系统采用了数据库事务、检查点以及预写入日志等技术来确保只有灾难性的事件[8]才能永久性的破坏数据库环境。所以虽然一个过于稳重的版本库管理员通常都会按照某种方案进行大量的版本库离线备份，不过不要急着通知你的管理员进行恢复。

然后，使用下面的方法试着“恢复”你的版本库：

1. Make sure no processes are accessing (or attempting to access) the repository. For networked repositories, this also means shutting down the Apache HTTP Server or svnserve daemon.

2. 成为版本库的拥有者和管理员。这一点很重要，如果以其它用户的身份恢复版本库，可能会改变版本库文件的访问权限，导致在版本库“恢复”后依旧无法访问。

3. Run the command **`svnadmin recover /var/svn/repos`**. You should see output such as this:

   ```
   Repository lock acquired。
   Please wait; recovering the repository may take some time...

   Recovery completed.
   The latest repos revision is 19.
   ```

   此命令可能需要数分钟才能完成。

4. 重新启动服务进程。

This procedure fixes almost every case of repository wedging. Make sure that you run this command as the user that owns and manages the database, not just as `root`. Part of the recovery process might

[8] For example, hard drive + huge electromagnet = disaster.

involve re-creating from scratch various database files (shared memory regions, e.g.). Recovering as `root` will create those files such that they are owned by `root`, which means that even after you restore connectivity to your repository, regular users will be unable to access it.

如果因为某些原因，上面的方法没能成功的恢复版本库，那么你可以做两件事。首先，将破损的版本库保存到其它地方，然后从最新的备份中恢复版本库。然后，发送一封邮件到Subversion用户列表(地址是：<users@subversion.tigris.org>)，写清你所遇到的问题。对于Subversion的开发者来说，数据安全是最重要的问题。

5.4.5. 版本库数据的移植

Subversion文件系统将数据保存在许多数据库表中，而这些表的结构只有Subversion开发者们才了解(也只有他们才感兴趣)，不过，有些时候我们会想到把所有或一部分数据转移到另一个版本库。

Subversion provides such functionality by way of *repository dump streams*. A repository dump stream (often referred to as a "dump file" when stored as a file on disk) is a portable, flat file format that describes the various revisions in your repository—what was changed, by whom, when, and so on. This dump stream is the primary mechanism used to marshal versioned history—in whole or in part, with or without modification—between repositories. And Subversion provides the tools necessary for creating and loading these dump streams: the **svnadmin dump** and **svnadmin load** subcommands, respectively.

警告

While the Subversion repository dump format contains human-readable portions and a familiar structure (it resembles an RFC 822 format, the same type of format used for most email), it is *not* a plain-text file format. It is a binary file format, highly sensitive to meddling. For example, many text editors will corrupt the file by automatically converting line endings.

There are many reasons for dumping and loading Subversion repository data. Early in Subversion's life, the most common reason was due to the evolution of Subversion itself. As Subversion matured, there were times when changes made to the backend database schema caused compatibility issues with previous versions of the repository, so users had to dump their repository data using the previous version of Subversion and load it into a freshly created repository with the new version of Subversion. Now, these types of schema changes haven't occurred since Subversion's 1.0 release, and the Subversion developers promise not to force users to dump and load their repositories when upgrading between minor versions (such as from 1.3 to 1.4) of Subversion. But there are still other reasons for dumping and loading, including re-deploying a Berkeley DB repository on a new OS or CPU architecture, switching between the Berkeley DB and FSFS backends, or (as we'll cover later in this chapter in 第 5.4.6 节 "过滤版本库历史") purging versioned data from repository history.

注意

The Subversion repository dump format describes versioned repository changes only. It will not carry any information about uncommitted transactions, user locks on filesystem paths, repository or server configuration customizations (including hook scripts), and so on.

无论你是什么原因需要移植版本库历史，都可以直接使用**svnadmin dump**和**svnadmin load**。**svnadmin dump**命令会将版本库中的修订版本数据按照特定的格式输出到转储流中，转储数据会输出到标准输出，而提示信息会输出到标准错误。这就是说，可以将转储数据存储到文件中，而同时在终端窗口中监视运行状态，例如：

```
$ svnlook youngest myrepos
26
$ svnadmin dump myrepos > dumpfile
* Dumped revision 0.
* Dumped revision 1.
* Dumped revision 2.
…
* Dumped revision 25.
* Dumped revision 26.
```

At the end of the process, you will have a single file (`dumpfile` in the previous example) that contains all the data stored in your repository in the requested range of revisions. Note that **svnadmin dump** is reading revision trees from the repository just like any other "reader" process would (e.g., **svn checkout**), so it's safe to run this command at any time.

另一个命令，**svnadmin load**，从标准输入流中读取Subversion转储数据，并且高效的将数据转载到目标版本库中。这个命令的提示信息输出到标准输出流中：

```
$ svnadmin load newrepos < dumpfile
<<< Started new txn, based on original revision 1
     * adding path : A ... done.
     * adding path : A/B ... done.
     …
------- Committed new rev 1 (loaded from original rev 1) >>>

<<< Started new txn, based on original revision 2
     * editing path : A/mu ... done.
     * editing path : A/D/G/rho ... done.

------- Committed new rev 2 (loaded from original rev 2) >>>

…

<<< Started new txn, based on original revision 25
     * editing path : A/D/gamma ... done.

------- Committed new rev 25 (loaded from original rev 25) >>>

<<< Started new txn, based on original revision 26
     * adding path : A/Z/zeta ... done.
     * editing path : A/mu ... done.
```

```
------- Committed new rev 26 (loaded from original rev 26) >>>
```

load命令的结果就是添加一些新的修订版本—与使用普通Subversion客户端直接提交到版本库相同。正像一次简单的提交，你也可以使用钩子脚本在每次load的开始和结束执行一些操作。通过传递`--use-pre-commit-hook`和`--use-post-commit-hook`选项给**svnadmin load**，你可以告诉Subversion的对每一个加载修订版本执行pre-commit和post-commit钩子脚本，可以利用这个选项确保这种提交也能通过一般提交的检验。当然，你要小心使用这个选项，你一定不想接受一大堆提交邮件。你可以查看第 5.3.2 节 “实现版本库钩子”来得到更多相关信息。

Note that because **svnadmin** uses standard input and output streams for the repository dump and load processes, people who are feeling especially saucy can try things such as this (perhaps even using different versions of **svnadmin** on each side of the pipe):

```
$ svnadmin create newrepos
$ svnadmin dump oldrepos | svnadmin load newrepos
```

By default, the dump file will be quite large—much larger than the repository itself. That's because by default every version of every file is expressed as a full text in the dump file. This is the fastest and simplest behavior, and it's nice if you're piping the dump data directly into some other process (such as a compression program, filtering program, or loading process). But if you're creating a dump file for longer-term storage, you'll likely want to save disk space by using the `--deltas` option. With this option, successive revisions of files will be output as compressed, binary differences—just as file revisions are stored in a repository. This option is slower, but it results in a dump file much closer in size to the original repository.

之前我们提到**svnadmin dump**输出指定范围内的修订版本，使用`--revision (-r)`选项可以指定一个单独的修订版本，或者一个修订版本的范围。如果忽略这个选项，所有版本库中的修订版本都会被转储。

```
$ svnadmin dump myrepos -r 23 > rev-23.dumpfile
$ svnadmin dump myrepos -r 100:200 > revs-100-200.dumpfile
```

Subversion在转储修订版本时，仅会输出与前一个修订版本之间的差异，通过这些差异足以从前一个修订版本中重建当前的修订版本。换句话说，在转储文件中的每一个修订版本仅包含这个修订版本作出的修改。这个规则的唯一一个例外是当前**svnadmin dump**转储的第一个修订版本。

By default, Subversion will not express the first dumped revision as merely differences to be applied to the previous revision. For one thing, there is no previous revision in the dump file! And second, Subversion cannot know the state of the repository into which the dump data will be loaded (if it ever is). To ensure that the output of each execution of **svnadmin dump** is self-sufficient, the first dumped revision is, by default, a full representation of every directory, file, and property in that revision of the repository.

不过，这些都是可以改变的。如果转储时设置了`--incremental`选项，**svnadmin**会比较第一个转储的修订版本和版本库中前一个修订版本，就像对待其它转储的修订版本一样。转储

时也是一样，转储文件中将仅包含第一个转储的修订版本的增量信息。这样的好处是，可以创建几个连续的小体积的转储文件代替一个大文件，比如：

```
$ svnadmin dump myrepos -r 0:1000 > dumpfile1
$ svnadmin dump myrepos -r 1001:2000 --incremental > dumpfile2
$ svnadmin dump myrepos -r 2001:3000 --incremental > dumpfile3
```

这些转储文件可以使用下列命令装载到一个新的版本库中：

```
$ svnadmin load newrepos < dumpfile1
$ svnadmin load newrepos < dumpfile2
$ svnadmin load newrepos < dumpfile3
```

另一个有关的技巧是，可以使用`--incremental`选项在一个转储文件中增加新的转储修订版本。举个例子，可以使用`post-commit`钩子在每次新的修订版本提交后将其转储到文件中。或者，可以编写一个脚本，在每天夜里将所有新增的修订版本转储到文件中。这样，**svnadmin dump**命令就变成了很好的版本库备份工具，以防万一出现系统崩溃或其它灾难性事件。

The dump format can also be used to merge the contents of several different repositories into a single repository. By using the `--parent-dir` option of **svnadmin load**, you can specify a new virtual root directory for the load process. That means if you have dump files for three repositories—say `calc-dumpfile`, `cal-dumpfile`, and `ss-dumpfile`—you can first create a new repository to hold them all:

```
$ svnadmin create /var/svn/projects
$
```

然后在版本库中创建三个目录分别保存来自三个不同版本库的数据：

```
$ svn mkdir -m "Initial project roots" \
      file:///var/svn/projects/calc \
      file:///var/svn/projects/calendar \
      file:///var/svn/projects/spreadsheet
Committed revision 1.
$
```

最后，将转储文件分别装载到各自的目录中：

```
$ svnadmin load /var/svn/projects --parent-dir calc < calc-dumpfile
…
$ svnadmin load /var/svn/projects --parent-dir calendar < cal-dumpfile
…
$ svnadmin load /var/svn/projects --parent-dir spreadsheet < ss-dumpfile
…
```

```
$
```

我们再介绍一下Subversion版本库转储数据的最后一种用途——在不同的存储机制或版本控制系统之间转换。因为转储数据的格式的大部分是可以阅读的，所以使用这种格式描述变更集(每个变更集对应一个新的修订版本)会相对容易一些。事实上，**cvs2svn**工具(参见 第 B.11 节 “迁移 CVS 版本库到 Subversion”)正是将CVS版本库的内容转换为转储数据格式，如此才能将CVS版本库的数据导入Subversion版本库之中。

5.4.6. 过滤版本库历史

因为Subversion使用底层的二进制区别和压缩算法(也可以选择完全非透明数据库系统)储存各类数据，手工调整是不明智的，即使这样做并不困难，我们也不鼓励这样做。然而，一旦你的数据存进了版本库，Subversion没有提供删除数据的简单办法。[9]但是不可避免的，总会有些时候你需要处理版本库的历史数据。你也许想把一个不应该出现的文件从版本库中彻底清除(无论任何原因不应该在那个位置出现)。或者，你曾经用一个版本库管理多个工程，现在又想把它们分开。要完成这样的工作，管理员们需要更易于管理和扩展的方法表示版本库中的数据，Subversion版本库转储文件格式就是一个很好的选择。

就像我们在第 5.4.5 节 “版本库数据的移植”中说的，Subversion版本库转储文件记录了所有版本数据的变更信息，而且以易于阅读的格式保存。可以使用**svnadmin dump**命令生成转储文件，然后用**svnadmin load**命令生成一个新的版本库。(参见 第 5.4.5 节 “版本库数据的移植”)。转储文件易于阅读意味着你可以查看和修改它。当然，问题是如果你有一个运行了三年的版本库，那么生成的转储文件会很庞大，阅读和手工修改起来都会花费很多时间。

That's where **svndumpfilter** becomes useful. This program acts as a path-based filter for repository dump streams. Simply give it either a list of paths you wish to keep or a list of paths you wish to not keep, and then pipe your repository dump data through this filter. The result will be a modified stream of dump data that contains only the versioned paths you (explicitly or implicitly) requested.

Let's look at a realistic example of how you might use this program. Earlier in this chapter (see 第 5.2.1 节 “规划你的版本库结构”), we discussed the process of deciding how to choose a layout for the data in your repositories—using one repository per project or combining them, arranging stuff within your repository, and so on. But sometimes after new revisions start flying in, you rethink your layout and would like to make some changes. A common change is the decision to move multiple projects that are sharing a single repository into separate repositories for each project.

假设有一个包含三个项目的版本库： `calc`，`calendar`，和 `spreadsheet`。它们在版本库中的布局如下：

```
/
   calc/
      trunk/
      branches/
      tags/
   calendar/
      trunk/
      branches/
```

[9]那就是你是用版本控制的原因，对吗？

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

现在要把这三个项目转移到三个独立的版本库中。首先，转储整个版本库：

```
$ svnadmin dump /var/svn/repos > repos-dumpfile
* Dumped revision 0.
* Dumped revision 1.
* Dumped revision 2.
* Dumped revision 3.
…
$
```

然后，将转储文件三次送入过滤器，每次仅保留一个顶级目录，就可以得到三个转储文件：

```
$ svndumpfilter include calc < repos-dumpfile > calc-dumpfile
…
$ svndumpfilter include calendar < repos-dumpfile > cal-dumpfile
…
$ svndumpfilter include spreadsheet < repos-dumpfile > ss-dumpfile
…
$
```

At this point, you have to make a decision. Each of your dump files will create a valid repository, but will preserve the paths exactly as they were in the original repository. This means that even though you would have a repository solely for your `calc` project, that repository would still have a top-level directory named `calc`. If you want your `trunk`, `tags`, and `branches` directories to live in the root of your repository, you might wish to edit your dump files, tweaking the `Node-path` and `Node-copyfrom-path` headers so that they no longer have that first `calc/` path component. Also, you'll want to remove the section of dump data that creates the `calc` directory. It will look something like the following:

```
Node-path: calc
Node-action: add
Node-kind: dir
Content-length: 0
```

警告

If you do plan on manually editing the dump file to remove a top-level directory, make sure your editor is not set to automatically convert end-of-line characters to the native format (e.g., `\r\n` to `\n`), as the content will then not agree with the metadata. This will render the dump file useless.

All that remains now is to create your three new repositories, and load each dump file into the right repository, ignoring the UUID found in the dump stream:

```
$ svnadmin create calc
$ svnadmin load --ignore-uuid calc < calc-dumpfile
<<< Started new transaction, based on original revision 1
     * adding path : Makefile ... done.
     * adding path : button.c ... done.
…
$ svnadmin create calendar
$ svnadmin load --ignore-uuid calendar < cal-dumpfile
<<< Started new transaction, based on original revision 1
     * adding path : Makefile ... done.
     * adding path : cal.c ... done.
…
$ svnadmin create spreadsheet
$ svnadmin load --ignore-uuid spreadsheet < ss-dumpfile
<<< Started new transaction, based on original revision 1
     * adding path : Makefile ... done.
     * adding path : ss.c ... done.
…
$
```

svndumpfilter的两个子命令都可以通过选项设定如何处理“空”修订版本。如果某个指定的修订版本仅包含路径的更改，过滤器就会将它删除，因为当前为空的修订版本通常是无用的甚至是让人讨厌的。为了让用户有选择的处理这些修订版本，**svndumpfilter**提供了以下命令行选项：

`--drop-empty-revs`
   不生成任何空修订版本，忽略它们。

`--renumber-revs`
   如果空修订版本被剔除(通过使用`--drop-empty-revs`选项)，依次修改其它修订版本的编号，确保编号序列是连续的。

`--preserve-revprops`
   If empty revisions are not dropped, preserve the revision properties (log message, author, date, custom properties, etc.) for those empty revisions. Otherwise, empty revisions will contain only the original datestamp, and a generated log message that indicates that this revision was emptied by **svndumpfilter**.

尽管**svndumpfilter**十分有用，能节省大量的时间，但它却是把不折不扣的双刃剑。首先，这个工具对路径语义极为敏感。仔细检查转储文件中的路径是不是以斜线开头。也许`Node-path`和`Node-copyfrom-path`这两个头参数对你有些帮助。

```
…
Node-path: spreadsheet/Makefile
```

...

If the paths have leading slashes, you should include leading slashes in the paths you pass to **svndumpfilter include** and **svndumpfilter exclude** (and if they don't, you shouldn't). Further, if your dump file has an inconsistent usage of leading slashes for some reason, [10] you should probably normalize those paths so that they all have, or all lack, leading slashes.

Also, copied paths can give you some trouble. Subversion supports copy operations in the repository, where a new path is created by copying some already existing path. It is possible that at some point in the lifetime of your repository, you might have copied a file or directory from some location that **svndumpfilter** is excluding, to a location that it is including. To make the dump data self-sufficient, **svndumpfilter** needs to still show the addition of the new path—including the contents of any files created by the copy—and not represent that addition as a copy from a source that won't exist in your filtered dump data stream. But because the Subversion repository dump format shows only what was changed in each revision, the contents of the copy source might not be readily available. If you suspect that you have any copies of this sort in your repository, you might want to rethink your set of included/excluded paths, perhaps including the paths that served as sources of your troublesome copy operations, too.

Finally, **svndumpfilter** takes path filtering quite literally. If you are trying to copy the history of a project rooted at `trunk/my-project` and move it into a repository of its own, you would, of course, use the **svndumpfilter include** command to keep all the changes in and under `trunk/my-project`. But the resultant dump file makes no assumptions about the repository into which you plan to load this data. Specifically, the dump data might begin with the revision that added the `trunk/my-project` directory, but it will *not* contain directives that would create the `trunk` directory itself (because `trunk` doesn't match the include filter). You'll need to make sure that any directories that the new dump stream expects to exist actually do exist in the target repository before trying to load the stream into that repository.

5.4.7. 版本库复制

有许多场景下会存在一个Subversion版本库的版本历史与另一个完全相同。或许最明显的就是在主版本库因为硬件故障或网络已出或其他原因而不可用时，维护一个简单的备份版本库。其他的场景包括，部署一个镜像版本库来分流压力，作为软升级机制等等。

As of version 1.4, Subversion provides a program for managing scenarios such as these—**svnsync**. This works by essentially asking the Subversion server to "replay" revisions, one at a time. It then uses that revision information to mimic a commit of the same to another repository. Neither repository needs to be locally accessible to the machine on which **svnsync** is running—its parameters are repository URLs, and it does all its work through Subversion's Repository Access (RA) interfaces. All it requires is read access to the source repository and read/write access to the destination repository.

注意

When using **svnsync** against a remote source repository, the Subversion server for that repository must be running Subversion version 1.4 or later.

[10]尽管**svnadmin dump**对是否以斜线作为路径的开头有统一的规定—这个规定就是不以斜线作为路径的开头—其它生成转储文件的程序不一定会遵守这个规定。

假定你已经有了一个希望镜像的源版本库，下一步就是你要有一个作为镜像的目标版本库。目标版本库可以使用任意文件系统数据存储后端(见第 5.2.3 节 “选择数据存储格式”)，但是其中一定不能有历史版本。**svnsync**的通讯议对于源和目标版本库版本历史的不一致非常敏感，因此，虽然**svnsync**无法要求目标版本库是只读的，[11]最好的办法就是只允许镜像进程修改目标版本库内容。

警告

不要做出会对镜像版本库产生版本库历史偏移的修改，所有提交和版本库的属性修改必须是由**svnsync**执行的。

对于目标版本库的另一种需求是**svnsync**可以修改特定版本化属性。**svnsync**在目标版本库的修订版本0的特别属性上记录了簿记信息，因为**svnsync**在版本库的钩子系统的框架下工作的，版本库缺省的状态(关闭了版本库属性修改；见pre-revprop-change)是不够的。你会需要明确的实现pre-revprop-change钩子，而且你的脚本必须允许**svnsync**设置它的特别属性，有了这些准备工作，你就可以开始镜像版本库修订版本了。

提示

实现授权措施允许复制进程的操作，同时防止其他用户修改镜像版本库内容是一个好主意。

让我们在一个典型的镜像场景中浏览一下**svnsync**的使用，我们急着讨论实践推荐，但是如果你们不需要或者感到不适合你们的环境，你可以不必去关注。

As a service to the fine developers of our favorite version control system, we will be mirroring the public Subversion source code repository and exposing that mirror publicly on the Internet, hosted on a different machine than the one on which the original Subversion source code repository lives. This remote host has a global configuration that permits anonymous users to read the contents of repositories on the host, but requires users to authenticate to modify those repositories. (Please forgive us for glossing over the details of Subversion server configuration for the moment—those are covered thoroughly in 第 6 章 服务配置.) And for no other reason than that it makes for a more interesting example, we'll be driving the replication process from a third machine—the one that we currently find ourselves using.

首先，我们会创建一个作为镜像的版本库，下面两步需要我们能够通过shell访问镜像版本库的机器。一旦版本库配置完成，我们不必再直接碰它了。

```
$ ssh admin@svn.example.com \
      "svnadmin create /var/svn/svn-mirror"
admin@svn.example.com's password: ********
$
```

此刻，我们有了我们的版本库，因为我们服务器的配置，这个版本库现在“存在于”Internet。现在，因为除了复制进程我们不希望任何其他修改，我们需要将这个进程同其他可能的提交者区分开来。为此，我们的进程使用专用的用户，只有特定用户`syncuser`的提交和属性修改可以被执行。

[11]实际上，它不是真的完全只读，或者**svnsync**本身有时间将版本库历史拷入。

我们会使用版本库的钩子系统来允许复制进程完成我们的任务，我们通过实现两个版本库事件钩子pre-revprop-change和start-commit来强制这个过程。我们的`pre-revprop-change`钩子脚本可以在例 5.2 “镜像版本库的 pre-revprop-change 钩子”找到，只是验证尝试修改属性的用户是`syncuser`，如果是，则允许修改；否则，拒绝修改。

例 5.2. 镜像版本库的 pre-revprop-change 钩子

```
#!/bin/sh

USER="$3"

if [ "$USER" = "syncuser" ]; then exit 0; fi

echo "Only the syncuser user may change revision properties" >&2
exit 1
```

That covers revision property changes. Now we need to ensure that only the `syncuser` user is permitted to commit new revisions to the repository. We do this using a `start-commit` hook scripts such as the one in 例 5.3 “镜像版本库的 start-commit 钩子”.

例 5.3. 镜像版本库的 start-commit 钩子

```
#!/bin/sh

USER="$2"

if [ "$USER" = "syncuser" ]; then exit 0; fi

echo "Only the syncuser user may commit new revisions" >&2
exit 1
```

在安装了我们的钩子脚本和确定它们可以被Subversion服务器执行后，我们完成了镜像版本库的配置，现在我们开始实际的镜像。

对于**svnsync**，我们首先需要在目标版本库上注册源版本库，我们通过**svnsync initialize**实现这一步。提供的URL分别指向目标和源版本库，在Subversion1.4，这是必须的—只允许完全的版本库镜像。在Subversion1.5，你可以使用**svnsync**镜像版本库的子树。

```
$ svnsync help init
initialize (init): usage: svnsync initialize DEST_URL SOURCE_URL

Initialize a destination repository for synchronization from
another repository.
…
$ svnsync initialize http://svn.example.com/svn-mirror \
```

```
                        http://svn.collab.net/repos/svn \
                        --sync-username syncuser --sync-password syncpass
Copied properties for revision 0.
$
```

我们的目标版本库现在记住了它是Subversion公共源代码版本库的镜像，注意我们在**svnsync**提供了一个用户名和密码—这是我们的镜像版本库pre-revprop-change钩子的要求。

注意

svnsync的最初版本(在Subversion 1.4)有一些缺陷—用来认证的`--username`和`--password`命令行参数同时作用于源和目标版本库。显然，我们无法保证同步的用户认证信息是相同的，如果不一样，用户使用非交互模式(`--non-interactive`选项)来运行**svnsync**时会遇到这个问题。

This has been fixed in Subversion 1.5 with the introduction of two new pairs of options. Use `--source-username` and `--source-password` to provide authentication credentials for the source repository; use `--sync-username` and `--sync-password` to provide credentials for the destination repository. (The old `--username` and `--password` options still exist for compatibility, but we advise against using them.)

And now comes the fun part. With a single subcommand, we can tell **svnsync** to copy all the as-yet-unmirrored revisions from the source repository to the target. [12] The **svnsync synchronize** subcommand will peek into the special revision properties previously stored on the target repository, and determine both what repository it is mirroring as well as that the most recently mirrored revision was revision 0. Then it will query the source repository and determine what the latest revision in that repository is. Finally, it asks the source repository's server to start replaying all the revisions between 0 and that latest revision. As **svnsync** get the resultant response from the source repository's server, it begins forwarding those revisions to the target repository's server as new commits.

```
$ svnsync help synchronize
synchronize (sync): usage: svnsync synchronize DEST_URL

Transfer all pending revisions to the destination from the source
with which it was initialized.
…
$ svnsync synchronize http://svn.example.com/svn-mirror
Transmitting file data ........................................
Committed revision 1.
Copied properties for revision 1.
Transmitting file data ..
Committed revision 2.
Copied properties for revision 2.
Transmitting file data .....
Committed revision 3.
Copied properties for revision 3.
```

[12]要预先警告一下，尽管对于普通读者只需要几秒钟就可以理解下面的输出，而对于整个镜像过程花费的时间可能会非常长。

```
…
Transmitting file data ..
Committed revision 23406.
Copied properties for revision 23406.
Transmitting file data .
Committed revision 23407.
Copied properties for revision 23407.
Transmitting file data ....
Committed revision 23408.
Copied properties for revision 23408.
$
```

镜像修订版本有一点特别有趣，首先是到目标版本库的修订版本提交，然后跟着属性修改。这是因为最初的提交是通过用户`syncuser`执行的，而时间戳是提交的时间，而且Subversion底层的版本库访问接口不允许在提交时任意修改修订版本属性，所以**svnsync**会立即使用属性修改，将源版本库发现的所有修订版本属性拷贝到目标版本库，这其中就包括了修改作者和时间戳使之与源版本库一致的效果。

Also noteworthy is that **svnsync** performs careful bookkeeping that allows it to be safely interrupted and restarted without ruining the integrity of the mirrored data. If a network glitch occurs while mirroring a repository, simply repeat the **svnsync synchronize** command, and it will happily pick up right where it left off. In fact, as new revisions appear in the source repository, this is exactly what you to do to keep your mirror up to date.

关于 svnsync

svnsync needs to be able to set and modify revision properties on the mirror repository because those properties are part of the data it is tasked with mirroring. As those properties change in the source repository, those changes need to be reflected in the mirror repository, too. But **svnsync** also uses a set of custom revision properties—stored in revision 0 of the mirror repository—for its own internal bookkeeping. These properties contain information such as the URL and UUID of the source repository, plus some additional state-tracking information.

One of those pieces of state-tracking information is a flag that essentially just means “there's a synchronization in progress right now.” This is used to prevent multiple **svnsync** processes from colliding with each other while trying to mirror data to the same destination repository. Now, generally you won't need to pay any attention whatsoever to *any* of these special properties (all of which begin with the prefix `svn:sync-`). Occasionally, though, if a synchronization fails unexpectedly, Subversion never has a chance to remove this particular state flag. This causes all future synchronization attempts to fail because it appears that a synchronization is still in progress when, in fact, none is. Fortunately, recovering from this situation is as simple as removing the `svn:sync-lock` property which serves as this flag from revision 0 of the mirror repository:

```
$ svn propdel --revprop -r0 svn:sync-lock http://svn.example.com/svn-mir
property 'svn:sync-lock' deleted from repository revision 0
$
```

That **svnsync** stores the source repository URL in a bookkeeping property on the mirror repository is the reason why you have to specify that URL only once, during **svnsync init**. Future synchronization operations against that mirror simply consult the special `svn:sync-from-url` property stored on the mirror itself to know where to synchronize from. This value is used literally by the synchronization process, though. So while from within CollabNet's network you can perhaps access our example source URL as `http://svn/repos/svn` (because that first `svn` magically gets `.collab.net` appended to it by DNS voodoo), if you later need to update that mirror from another machine outside CollabNet's network, the synchronization might fail (because the hostname `svn` is ambiguous). For this reason, it's best to use fully qualified source repository URLs when initializing a mirror repository rather than those that refer to only hostnames or IP addresses (which can change over time). But here again, if you need an existing mirror to start referring to a different URL for the same source repository, you can change the bookkeeping property which houses that information:

```
$ svn propset --revprop -r0 svn:sync-from-url NEW-SOURCE-URL \
      http://svn.example.com/svn-mirror
property 'svn:sync-from-url' set on repository revision 0
$
```

Another interesting thing about these special bookkeeping properties is that **svnsync** will not attempt to mirror any of those properties when they are found in the source repository. The reason is probably obvious, but basically boils down to **svnsync** not being able to distinguish the special properties it has merely copied from the source repository from those it needs to consult and maintain for its own bookkeeping needs. This situation could occur if, for example, you were maintaining a mirror of a mirror of a third repository. When **svnsync** sees its own special properties in revision 0 of the source repository, it simply ignores them.

There is, however, one bit of inelegance in the process. Because Subversion revision properties can be changed at any time throughout the lifetime of the repository, and because they don't leave an audit trail that indicates when they were changed, replication processes have to pay special attention to them. If you've already mirrored the first 15 revisions of a repository and someone then changes a revision property on revision 12, **svnsync** won't know to go back and patch up its copy of revision 12. You'll need to tell it to do so manually by using (or with some additional tooling around) the **svnsync copy-revprops** subcommand, which simply rereplicates all the revision properties for a particular revision or range thereof.

```
$ svnsync help copy-revprops
copy-revprops: usage: svnsync copy-revprops DEST_URL [REV[:REV2]]

Copy the revision properties in a given range of revisions to the
destination from the source with which it was initialized.
…
$ svnsync copy-revprops http://svn.example.com/svn-mirror 12
Copied properties for revision 12.
$
```

版本库复制只是一个壳，你一定会希望利用这个进程的自动化。例如，如果我们的例子是一个“拖和推”设置，你或许希望在post-commit和post-revprop-change钩子实现中从你的主版本库将修改推倒一个或多个镜像，这样就可以近乎实时的保持镜像的时效性。

Also, while it isn't very commonplace to do so, **svnsync** does gracefully mirror repositories in which the user as whom it authenticates has only partial read access. It simply copies only the bits of the repository that it is permitted to see. Obviously, such a mirror is not useful as a backup solution.

In Subversion 1.5, **svnsync** grew the ability to also mirror a subset of a repository rather than the whole thing. The process of setting up and maintaining such a mirror is exactly the same as when mirroring a whole repository, except that instead of specifying the source repository's root URL when running **svnsync init**, you specify the URL of some subdirectory within that repository. Synchronization to that mirror will now copy only the bits that changed under that source repository subdirectory. There are some limitations to this support, though. First, you can't mirror multiple disjoint subdirectories of the source repository into a single mirror repository—you'd need to instead mirror some parent directory that is common to both. Second, the filtering logic is entirely path-based, so if the subdirectory you are mirroring was renamed at some point in the past, your mirror would contain only the revisions since the directory appeared at the URL you specified. And likewise, if the source subdirectory is renamed in the future, your synchronization processes will stop mirroring data at the point that the source URL you specified is no longer valid.

只要用户与版本库和镜像的交互继续，是可以有一个工作拷贝直接与这两个版本库交互。但是你需要跳出几个圈子才能做到这样。第一，你需要保证主和镜像版本库有相同的UUID(通常缺省不是相同)，本章后面第 5.4.9 节 “管理版本库的 UUID”详细讨论了这个问题。

Once the two repositories have the same UUID, you can use **svn switch** with the `--relocate` option to point your working copy to whichever of the repositories you wish to operate against, a process that is described in svn switch. There is a possible danger here, though, in that if the primary and mirror repositories aren't in close synchronization, a working copy up to date with, and pointing to,

the primary repository will, if relocated to point to an out-of-date mirror, become confused about the apparent sudden loss of revisions it fully expects to be present, and it will throw errors to that effect. If this occurs, you can relocate your working copy back to the primary repository and then either wait until the mirror repository is up to date, or backdate your working copy to a revision you know is present in the sync repository, and then retry the relocation.

最后我们需要意识到，**svnsync**只支持修订版本为基础的复制，只有版本库转储文件包含的内容会进行复制。因此**svnsync**也有了版本库转储流的此类限制，它没有包括诸如钩子实现，版本库或服务器配置数据，未提交事务或关于用户锁定版本库路径的信息，只有Subversion版本库转储文件格式在复制时包含这些信息。

5.4.8. 版本库备份

Despite numerous advances in technology since the birth of the modern computer, one thing unfortunately rings true with crystalline clarity—sometimes things go very, very awry. Power outages, network connectivity dropouts, corrupt RAM, and crashed hard drives are but a taste of the evil that Fate is poised to unleash on even the most conscientious administrator. And so we arrive at a very important topic—how to make backup copies of your repository data.

Subversion版本库管理有两种备份方法—完全和增量。一个完全的版本库备份包含了在重大灾难后重建版本库所需的所有信息，通常，这意味着对版本库目录(包括Berkeley DB或FSFS环境)的完全复制，增量备份的内容要少一些：只包含在上次备份后改变的部分。

As far as full backups go, the naïve approach might seem like a sane one, but unless you temporarily disable all other access to your repository, simply doing a recursive directory copy runs the risk of generating a faulty backup. In the case of Berkeley DB, the documentation describes a certain order in which database files can be copied that will guarantee a valid backup copy. A similar ordering exists for FSFS data. But you don't have to implement these algorithms yourself, because the Subversion development team has already done so. The **svnadmin hotcopy** command takes care of the minutia involved in making a hot backup of your repository. And its invocation is as trivial as the Unix **cp** or Windows **copy** operations:

```
$ svnadmin hotcopy /var/svn/repos /var/svn/repos-backup
```

The resultant backup is a fully functional Subversion repository, able to be dropped in as a replacement for your live repository should something go horribly wrong.

当进行Berkeley DB版本库的备份时，你可以指导**svnadmin hotcopy**清理源版本库中无用的Berkeley DB日志文件(见第 5.4.3.3 节 “删除不使用的 Berkeley DB 日志文件”)，只需要简单的在命令行里提供`--clean-logs`。

```
$ svnadmin hotcopy --clean-logs /var/svn/bdb-repos /var/svn/bdb-repos-bac
```

Additional tooling around this command is available, too. The `tools/backup/` directory of the Subversion source distribution holds the **hot-backup.py** script. This script adds a bit of backup management atop **svnadmin hotcopy**, allowing you to keep only the most recent configured number of backups of each repository. It will automatically manage the names of the backed-up repository

directories to avoid collisions with previous backups and will “rotate off” older backups, deleting them so that only the most recent ones remain. Even if you also have an incremental backup, you might want to run this program on a regular basis. For example, you might consider using **hot-backup.py** from a program scheduler (such as **cron** on Unix systems), which can cause it to run nightly (or at whatever granularity of time you deem safe).

Some administrators use a different backup mechanism built around generating and storing repository dump data. We described in 第 5.4.5 节 “版本库数据的移植” how to use **svnadmin dump** with the `--incremental` option to perform an incremental backup of a given revision or range of revisions. And of course, you can achieve a full backup variation of this by omitting the `--incremental` option to that command. There is some value in these methods, in that the format of your backed-up information is flexible—it's not tied to a particular platform, versioned filesystem type, or release of Subversion or Berkeley DB. But that flexibility comes at a cost, namely that restoring that data can take a long time—longer with each new revision committed to your repository. Also, as is the case with so many of the various backup methods, revision property changes that are made to already backed-up revisions won't get picked up by a nonoverlapping, incremental dump generation. For these reasons, we recommend against relying solely on dump-based backup approaches.

如你所见，几种备份方式都有各自的优点，最简单的方式是完全热备份，将会每次建立版本库的完美复制品，这意味着如果当你的活动版本库发生了什么事情，你可以用备份恢复。但不幸的是，如果你维护多个备份，每个完全的备份会吞噬掉和你的活动版本库同样的空间。与之相对照的是增量备份，能够快速生成小的备份，但是恢复过程将会很痛苦，通常要包括多个增量拷贝的应用。其他方法都有自己的特点，管理员需要在创建拷贝和恢复的代价之间寻求平衡。

The **svnsync** program (see 第 5.4.7 节 “版本库复制”) actually provides a rather handy middle-ground approach. If you are regularly synchronizing a read-only mirror with your main repository, in a pinch your read-only mirror is probably a good candidate for replacing that main repository if it falls over. The primary disadvantage of this method is that only the versioned repository data gets synchronized—repository configuration files, user-specified repository path locks, and other items that might live in the physical repository directory but not *inside* the repository's virtual versioned filesystem are not handled by **svnsync**.

在每一种备份情境下，版本库管理员需要意识到对未版本化的修订版本属性的修改对备份的影响，因为这些修改本身不会产生新的修订版本，所以不会触发post-commit的钩子程序，也不会触发pre-revprop-change和post-revprop-change的钩子。 [13]而且因为你可以改变修订版本的属性，而不需要遵照时间顺序—你可在任何时刻修改任何修订版本的属性—因此最新版本的增量备份不会捕捉到以前特定修订版本的属性修改。

通常说来，在每次提交时，只有妄想狂才会备份整个版本库，然而，假设一个给定的版本库拥有一些恰当粒度的冗余机制(如每次提交的邮件)。版本库管理员也许会希望将版本库的热备份引入到系统级的每夜备份，对大多数版本库，归档的提交邮件为保存资源提供了足够的冗余措施，至少对于最近的提交。但是它是你的数据—你喜欢怎样保护都可以。

Often, the best approach to repository backups is a diversified one that leverages combinations of the methods described here. The Subversion developers, for example, back up the Subversion source code repository nightly using **hot-backup.py** and an off-site **rsync** of those full backups; keep multiple archives of all the commit and property change notification emails; and have repository mirrors

[13]**svnadmin setlog**可以被绕过钩子程序被调用。

maintained by various volunteers using **svnsync**. Your solution might be similar, but should be catered to your needs and that delicate balance of convenience with paranoia. And whatever you do, validate your backups from time to time—what good is a spare tire that has a hole in it? While all of this might not save your hardware from the iron fist of Fate, [14] it should certainly help you recover from those trying times.

5.4.9. 管理版本库的 UUID

Subversion repositories have a universally unique identifier (UUID) associated with them. This is used by Subversion clients to verify the identity of a repository when other forms of verification aren't good enough (such as checking the repository URL, which can change over time). Most Subversion repository administrators rarely, if ever, need to think about repository UUIDs as anything more than a trivial implementation detail of Subversion. Sometimes, however, there is cause for attention to this detail.

As a general rule, you want the UUIDs of your live repositories to be unique. That is, after all, the point of having UUIDs. But there are times when you want the repository UUIDs of two repositories to be exactly the same. For example, if you make a copy of a repository for backup purposes, you want the backup to be a perfect replica of the original so that, in the event that you have to restore that backup and replace the live repository, users don't suddenly see what looks like a different repository. When dumping and loading repository history (as described earlier in 第 5.4.5 节 “版本库数据的移植”), you get to decide whether to apply the UUID encapsulated in the data dump stream to the repository in which you are loading the data. The particular circumstance will dictate the correct behavior.

There are a couple of ways to set (or reset) a repository's UUID, should you need to. As of Subversion 1.5, this is as simple as using the **svnadmin setuuid** command. If you provide this subcommand with an explicit UUID, it will validate that the UUID is well-formed and then set the repository UUID to that value. If you omit the UUID, a brand-new UUID will be generated for your repository.

```
$ svnlook uuid /var/svn/repos
cf2b9d22-acb5-11dc-bc8c-05e83ce5dbec
$ svnadmin setuuid /var/svn/repos   # generate a new UUID
$ svnlook uuid /var/svn/repos
3c3c38fe-acc0-11dc-acbc-1b37ff1c8e7c
$ svnadmin setuuid /var/svn/repos \
           cf2b9d22-acb5-11dc-bc8c-05e83ce5dbec  # restore the old UUID
$ svnlook uuid /var/svn/repos
cf2b9d22-acb5-11dc-bc8c-05e83ce5dbec
$
```

For folks using versions of Subversion earlier than 1.5, these tasks are a little more complicated. You can explicitly set a repository's UUID by piping a repository dump file stub that carries the new UUID specification through **svnadmin load --force-uuid *REPOS-PATH***.

```
$ svnadmin load --force-uuid /var/svn/repos <<EOF
```

[14] 你知道的—只是对各种变化莫测的问题的统称。

```
SVN-fs-dump-format-version: 2

UUID: cf2b9d22-acb5-11dc-bc8c-05e83ce5dbec
EOF
$ svnlook uuid /var/svn/repos
cf2b9d22-acb5-11dc-bc8c-05e83ce5dbec
$
```

Having older versions of Subversion generate a brand-new UUID is not quite as simple to do, though. Your best bet here is to find some other way to generate a UUID, and then explicitly set the repository's UUID to that value.

5.5. 移动和删除版本库

Subversion repository data is wholly contained within the repository directory. As such, you can move a Subversion repository to some other location on disk, rename a repository, copy a repository, or delete a repository altogether using the tools provided by your operating system for manipulating directories —**mv**, **cp -a**, and **rm -r** on Unix platforms; **copy**, **move**, and **rmdir /s /q** on Windows; vast numbers of mouse and menu gyrations in various graphical file explorer applications, and so on.

Of course, there's often still more to be done when trying to cleanly affect changes such as this. For example, you might need to update your Subversion server configuration to point to the new location of a relocated repository or to remove configuration bits for a now-deleted repository. If you have automated processes that publish information from or about your repositories, they may need to be updated. Hook scripts might need to be reconfigured. Users may need to be notified. The list can go on indefinitely, or at least to the extent that you've built processes and procedures around your Subversion repository.

In the case of a copied repository, you should also consider the fact that Subversion uses repository UUIDs to distinguish repositories. If you copy a Subversion repository using a typical shell recursive copy command, you'll wind up with two repositories that are identical in every way—including their UUIDs. In some circumstances, this might be desirable. But in the instances where it is not, you'll need to generate a new UUID for one of these identical repositories. See 第 5.4.9 节 “管理版本库的 UUID” for more about managing repository UUIDs.

5.6. 总结

By now you should have a basic understanding of how to create, configure, and maintain Subversion repositories. We introduced you to the various tools that will assist you with this task. Throughout the chapter, we noted common administration pitfalls and offered suggestions for avoiding them.

剩下的，就是由你决定在你的版本库中存放一些什么有趣的资料，并最终通过网络获得这些资料。下一章是关于网络的内容。

第 6 章 服务配置

一个Subversion的版本库可以和客户端同时运行在同一个机器上，使用`file:///`访问，但是一个典型的Subversion设置应该包括一个单独的服务器，可以被办公室的所有客户端访问—或者有可能是整个世界。

This chapter describes how to get your Subversion repository exposed outside its host machine for use by remote clients. We will cover Subversion's currently available server mechanisms, discussing the configuration and use of each. After reading this chapter, you should be able to decide which networking setup is right for your needs, as well as understand how to enable such a setup on your host computer.

6.1. 概述

Subversion的设计包括一个抽象的网络层，这意味着版本库可以通过各种服务器进程访问，而且客户端“版本库访问”的API允许程序员写出相关协议的插件，理论上讲，Subversion可以使用无限数量的网络协议实现，目前实践中只有两种服务器。

Apache是最流行的web服务器，通过使用**mod_dav_svn**模块，Apache可以访问版本库，并且可以使客户端使用HTTP的扩展协议WebDAV/DeltaV进行访问，因为Apache是一个非常易于扩展的web服务器，它提供了许多“易于获取的”特性，例如加密的SSL通讯，日志和与第三方工具的集成，以及内置的版本库web浏览功能。

在另一个角落是**svnserve**：一个更小，轻型的服务器程序，同客户端使用自定义的协议。因为协议是为Subversion专门设计的，并且是有状态的(不像HTTP)，它提供了更快的网络操作—但也有一些代价。它只理解CRAM-MD5的认证，然而它非常易于配置，是开始使用Subversion的小团队的最佳选择。

A third option is to use **svnserve** tunneled over an SSH connection. Even though this scenario still uses **svnserve**, it differs quite a bit in features from a traditional **svnserve** deployment. SSH is used to encrypt all communication. SSH is also used exclusively to authenticate, so real system accounts are required on the server host (unlike vanilla **svnserve**, which has its own private user accounts). Finally, because this setup requires that each user spawn a private, temporary **svnserve** process, it's equivalent (from a permissions point of view) to allowing a group of local users to all access the repository via `file://` URLs. Path-based access control has no meaning, since each user is accessing the repository database files directly.

表 6.1 “Subversion 服务器选项比较”是三种典型服务器部署的总结。

表 **6.1. Subversion** 服务器选项比较

特性	**Apache + mod_dav_svn**	**svnserve**	**svnserve over SSH**
认证选项	HTTP(S) basic auth, X.509 certificates, LDAP, NTLM, or any other mechanism	CRAM-MD5 by default; LDAP, NTLM, or any other mechanism available to SASL	SSH

特性	**Apache + mod_dav_svn**	**svnserve**	**svnserve over SSH**
	available to Apache httpd		
用户帐号选项	私有“users”文件，或其他Apache httpd(LDAP，SQL等等)支持的机制。	Private 'users' file, or other mechanisms available to SASL (LDAP, SQL, etc.)	System accounts
授权选项	Read/write access can be granted over the whole repository, or specified per path	Read/write access can be granted over the whole repository, or specified per path	Read/write access only grantable over the whole repository
加密	Available via optional SSL	Available via optional SASL features	Inherent in SSH connection
Logging	Full Apache logs of each HTTP request, with optional “高级” logging of general client operations	No logging	No logging
交互性	Accessible by other WebDAV clients	Talks only to svn clients	Talks only to svn clients
Web浏览能力	Limited built-in support, or via third-party tools such as ViewVC	Only via third-party tools such as ViewVC	Only via third-party tools such as ViewVC
Master-slave server replication	Transparent write-proxying available from slave to master	Can only create read-only slave servers	Can only create read-only slave servers
速度	Somewhat slower	Somewhat faster	Somewhat faster
初始设置	Somewhat complex	Extremely simple	Moderately simple

6.2. 选择一个服务器配置

那你应该用什么服务器？什么最好？

Obviously, there's no right answer to that question. Every team has different needs, and the different servers all represent different sets of trade-offs. The Subversion project itself doesn't endorse one server or another, or consider either server more “official” than another.

下面是你选择或者不选择某一个部署方式的原因。

6.2.1. svnserve 服务器

为什么你会希望使用它：

- 设置快速简单。

- 网络协议是有状态的，比WebDAV快很多。
- 不需要在服务器创建系统帐号。
- 不会在网络传输密码。

为什么你会希望避免它：

- By default, only one authentication method is available, the network protocol is not encrypted, and the server stores clear text passwords. (All these things can be changed by configuring SASL, but it's a bit more work to do.)
- 没有任何类型的日志，甚至是错误。
- No built-in web browsing. (You'd have to install a separate web server and repository browsing software to add this.)

6.2.2. svnserve over SSH

为什么你会希望使用它：

- The network protocol is stateful and noticeably faster than WebDAV.
- 你可以利用现有的SSH帐号和用户基础。
- 所有网络传输是加密的。

为什么你会希望避免它：

- Only one choice of authentication method is available.
- There is no logging of any kind, not even errors.
- It requires users to be in the same system group, or use a shared SSH key.
- If used improperly, it can lead to file permission problems.

6.2.3. Apache 的 HTTP 服务器

为什么你会希望使用它：

- It allows Subversion to use any of the numerous authentication systems already integrated with Apache.
- There is no need to create system accounts on the server.
- Full Apache logging is available.
- 网络传输可以通过SSL加密。
- HTTP(S) 通常可以穿越公司防火墙。

- Built-in repository browsing is available via web browser.
- The repository can be mounted as a network drive for transparent version control (see 第 C.2 节“自动版本化”).

为什么你会希望避免它：

- 比**svnserve**慢很多，因为HTTP是无状态的协议，需要更多的传递。
- 初始设置可能复杂

6.2.4. 推荐

In general, the authors of this book recommend a vanilla **svnserve** installation for small teams just trying to get started with a Subversion server; it's the simplest to set up and has the fewest maintenance issues. You can always switch to a more complex server deployment as your needs change.

下面是一些常见的建议和小技巧，基于多年对用户的支持：

- If you're trying to set up the simplest possible server for your group, a vanilla **svnserve** installation is the easiest, fastest route. Note, however, that your repository data will be transmitted in the clear over the network. If your deployment is entirely within your company's LAN or VPN, this isn't an issue. If the repository is exposed to the wide-open Internet, you might want to make sure that either the repository's contents aren't sensitive (e.g., it contains only open source code), or that you go the extra mile in configuring SASL to encrypt network communications.
- If you need to integrate with existing legacy identity systems (LDAP, Active Directory, NTLM, X.509, etc.), you must use either the Apache-based server or **svnserve** configured with SASL. If you absolutely need server-side logs of either server errors or client activities, an Apache-based server is your only option.
- 如果你已经决定使用Apache或**svnserve**，应该单独创建一个运行服务器进程的`svn`用户，也需要确定版本库目录属于`svn`用户。从安全的角度，这样很好的利用了操作系统的文件系统许可保护了版本库数据，只有Subversion服务进程可以修改其内容。
- If you have an existing infrastructure that is heavily based on SSH accounts, and if your users already have system accounts on your server machine, it makes sense to deploy an **svnserve**-over-SSH solution. Otherwise, we don't widely recommend this option to the public. It's generally considered safer to have your users access the repository via (imaginary) accounts managed by **svnserve** or Apache, rather than by full-blown system accounts. If your deep desire for encrypted communication still draws you to this option, we recommend using Apache with SSL or **svnserve** with SASL encryption instead.
- Do *not* be seduced by the simple idea of having all of your users access a repository directly via `file://` URLs. Even if the repository is readily available to everyone via a network share, this is a bad idea. It removes any layers of protection between the users and the repository: users can accidentally (or intentionally) corrupt the repository database, it becomes hard to take the repository offline for inspection or upgrade, and it can lead to a mess of file permission problems (see 第 6.6 节“支持多种版本库访问方法”). Note that this is also one of the reasons we warn against accessing repositories via `svn+ssh://` URLs—from a security standpoint, it's effectively the same as local

users accessing via `file://`, and it can entail all the same problems if the administrator isn't careful.

6.3. svnserve - 定制的服务器

svnserve是一个轻型的服务器，可以同客户端通过在TCP/IP基础上的自定义有状态协议通讯，客户端通过使用开头为`svn://`或者`svn+ssh://`**svnserve**的URL来访问一个**svnserve**服务器。这一小节将会解释运行**svnserve**的不同方式，客户端怎样实现服务器的认证，怎样配置版本库恰当的访问控制。

6.3.1. 调用服务器

有许多不同方法运行**svnserve**：

- 作为一个独立守护进程启动**svnserve**，监听请求。
- 当特定端口收到一个请求，就会使UNIX的**inetd**守护进程临时调用**svnserve**处理。
- 使用SSH在加密通道发起临时**svnserve**服务。
- 以Windows service服务方式运行**svnserve**。

6.3.1.1. svnserve 与守护进程

使用**svnserve**最简单的方式是作为独立“守护”进程运行，使用`-d`选项：

```
$ svnserve -d
$               # svnserve is now running, listening on port 3690
```

When running **svnserve** in daemon mode, you can use the `--listen-port` and `--listen-host` options to customize the exact port and hostname to “bind” to.

Once we successfully start **svnserve** as explained previously, it makes every repository on your system available to the network. A client needs to specify an *absolute* path in the repository URL. For example, if a repository is located at `/var/svn/project1`, a client would reach it via `svn://host.example.com/var/svn/project1`. To increase security, you can pass the `-r` option to **svnserve**, which restricts it to exporting only repositories below that path. For example:

```
$ svnserve -d -r /var/svn
…
```

使用`-r`可以有效地改变文件系统的根位置，客户端可以使用去掉前半部分的路径，留下的要短一些的(更加有提示性)URL：

```
$ svn checkout svn://host.example.com/project1
…
```

6.3.1.2. 通过 inetd 调用 svnserve

If you want **inetd** to launch the process, you need to pass the `-i` (`--inetd`) option. In the following example, we've shown the output from running `svnserve -i` at the command line, but note that this isn't how you actually start the daemon; see the paragraphs following the example for how to configure **inetd** to start **svnserve**.

```
$ svnserve -i
( success ( 1 2 ( ANONYMOUS ) ( edit-pipeline ) ) )
```

When invoked with the `--inetd` option, **svnserve** attempts to speak with a Subversion client via `stdin` and `stdout` using a custom protocol. This is the standard behavior for a program being run via **inetd**. The IANA has reserved port 3690 for the Subversion protocol, so on a Unix-like system you can add lines to `/etc/services` such as these (if they don't already exist):

```
svn             3690/tcp    # Subversion
svn             3690/udp    # Subversion
```

如果系统是使用经典的类Unix的**inetd**守护进程，你可以在`/etc/inetd.conf`添加这几行：

```
svn stream tcp nowait svnowner /usr/bin/svnserve svnserve -i
```

确定“svnowner”用户拥有访问版本库的适当权限，现在如果一个客户连接来到你的服务器的端口3690，**inetd**会产生一个**svnserve**进程来做服务。当然，你也可以添加`-r`到命令行，限制暴露出的版本库。

6.3.1.3. 通过隧道调用 svnserve

A third way to invoke **svnserve** is in tunnel mode, using the `-t` option. This mode assumes that a remote-service program such as **rsh** or **ssh** has successfully authenticated a user and is now invoking a private **svnserve** process *as that user*. (Note that you, the user, will rarely, if ever, have reason to invoke **svnserve** with the `-t` at the command line; instead, the SSH daemon does so for you.) The **svnserve** program behaves normally (communicating via `stdin` and `stdout`) and assumes that the traffic is being automatically redirected over some sort of tunnel back to the client. When **svnserve** is invoked by a tunnel agent like this, be sure that the authenticated user has full read and write access to the repository database files. It's essentially the same as a local user accessing the repository via `file://` URLs.

这个选项将在第 6.3.4 节 “穿越 SSH 隧道”详细讨论。

6.3.1.4. svnserve 与 Windows 服务

If your Windows system is a descendant of Windows NT (2000, 2003, XP, or Vista), you can run **svnserve** as a standard Windows service. This is typically a much nicer experience than running it as a standalone daemon with the `--daemon (-d)` option. Using daemon mode requires launching a console, typing a command, and then leaving the console window running indefinitely. A Windows

service, however, runs in the background, can start at boot time automatically, and can be started and stopped using the same consistent administration interface as other Windows services.

你需要使用命令行工具**SC.EXE**定义新的服务，就像**inetd**的配置行，你必须在Windows启动时指明**svnserve**的调用：

```
C:\> sc create svn
        binpath= "C:\svn\bin\svnserve.exe --service -r C:\repos"
        displayname= "Subversion Server"
        depend= Tcpip
        start= auto
```

这样定义了一个新的Windows服务，叫做“svn”，会在启动时(在这个例子里，根目录是C:\repos。)执行特定的**svnserve.exe**，可是前面这个例子产生了一些错误。

First, notice that the **svnserve.exe** program must always be invoked with the `--service` option. Any other options to **svnserve** must then be specified on the same line, but you cannot add conflicting options such as `--daemon (-d)`, `--tunnel`, or `--inetd (-i)`. Options such as `-r` or `--listen-port` are fine, though. Second, be careful about spaces when invoking the **SC.EXE** command: the `key= value` patterns must have no spaces between `key=` and must have exactly one space before the `value`. Lastly, be careful about spaces in your command line to be invoked. If a directory name contains spaces (or other characters that need escaping), place the entire inner value of `binpath` in double quotes, by escaping them:

```
C:\> sc create svn
        binpath= "\"C:\program files\svn\bin\svnserve.exe\" --service -r C
        displayname= "Subversion Server"
        depend= Tcpip
        start= auto
```

Also note that the word `binpath` is misleading—its value is a *command line*, not the path to an executable. That's why you need to surround it with quotes if it contains embedded spaces.

一旦定义了服务，就可以使用标准GUI工具(服务管理控制面板)进行停止、启动和查询，或者是通过命令行：

```
C:\> net stop svn
C:\> net start svn
```

The service can also be uninstalled (i.e., undefined) by deleting its definition: **`sc delete svn`**. Just be sure to stop the service first! The **SC.EXE** program has many other subcommands and options; run **`sc /?`** to learn more about it.

6.3.2. 内置的认证和授权

如果一个客户端连接到**svnserve**进程，如下事情会发生：

- 客户端选择特定的版本库。

- 服务器处理版本库的`conf/svnserve.conf`文件，并且执行里面定义的所有认证和授权政策。
- Depending on the defined policies, one of the following may occur:
  - 如果没有收到认证请求，客户端可能被允许匿名访问。
  - 客户端可以在任何认证时被要求。
  - 如果操作在“通道模式”，客户端会宣布自己已经在外部得到认证(通常通过SSH)。

svnserve服务器缺省只知道发送CRAM-MD5[1]认证请求，本质上讲，就是服务器发送一些数据到客户端，客户端使用MD5哈希算法创建这些数据组合密码的指纹，然后返回指纹，服务器执行同样的计算并且来计算结果的一致性，真正的密码并没有在互联网上传递。

If your **svnserve** server was built with SASL support, it not only knows how to send CRAM-MD5 challenges, but also likely knows a whole host of other authentication mechanisms. See 第 6.3.3 节 “让 svnserve 使用 SASL” later in this chapter to learn how to configure SASL authentication and encryption.

It's also possible, of course, for the client to be externally authenticated via a tunnel agent, such as **ssh**. In that case, the server simply examines the user it's running as, and uses this name as the authenticated username. For more on this, see the later section, 第 6.3.4 节 “穿越 SSH 隧道”.

As you've already guessed, a repository's `svnserve.conf` file is the central mechanism for controlling authentication and authorization policies. The file has the same format as other configuration files (see 第 7.1 节 “运行配置区”): section names are marked by square brackets (`[` and `]`), comments begin with hashes (`#`), and each section contains specific variables that can be set (`variable = value`). Let's walk through these files and learn how to use them.

6.3.2.1. 创建一个用户文件和认证域

For now, the `[general]` section of `svnserve.conf` has all the variables you need. Begin by changing the values of those variables: choose a name for a file that will contain your usernames and passwords and choose an authentication realm:

```
[general]
password-db = userfile
realm = example realm
```

`realm`是你定义的名称，这告诉客户端连接的“认证命名空间”，Subversion会在认证提示里显示，并且作为凭证缓存(见第 3.11.2 节 “客户端凭证缓存”。)的关键字(还有服务器的主机名和端口)，`password-db`参数指出了保存用户和密码列表文件，这个文件使用同样熟悉的格式，举个例子：

```
[users]
harry = foopassword
```

[1]见RFC 2195。

```
sally = barpassword
```

The value of `password-db` can be an absolute or relative path to the users file. For many admins, it's easy to keep the file right in the `conf/` area of the repository, alongside `svnserve.conf`. On the other hand, it's possible you may want to have two or more repositories share the same users file; in that case, the file should probably live in a more public place. The repositories sharing the users file should also be configured to have the same realm, since the list of users essentially defines an authentication realm. Wherever the file lives, be sure to set the file's read and write permissions appropriately. If you know which user(s) **svnserve** will run as, restrict read access to the users file as necessary.

6.3.2.2. 设置访问控制

There are two more variables to set in the `svnserve.conf` file: they determine what unauthenticated (anonymous) and authenticated users are allowed to do. The variables `anon-access` and `auth-access` can be set to the value `none`, `read`, or `write`. Setting the value to `none` prohibits both reading and writing; `read` allows read-only access to the repository, and `write` allows complete read/write access to the repository. For example:

```
[general]
password-db = userfile
realm = example realm

# anonymous users can only read the repository
anon-access = read

# authenticated users can both read and write
auth-access = write
```

实例中的设置实际上是参数的缺省值，你一定不要忘了设置它们，如果你希望更保守一点，你可以完全封锁匿名访问：

```
[general]
password-db = userfile
realm = example realm

# anonymous users aren't allowed
anon-access = none

# authenticated users can both read and write
auth-access = write
```

The server process understands not only these “blanket” access controls to the repository, but also finer-grained access restrictions placed on specific files and directories within the repository. To make use of this feature, you need to define a file containing more detailed rules, and then set the `authz-db` variable to point to it:

```
[general]
```

```
password-db = userfile
realm = example realm

# Specific access rules for specific locations
authz-db = authzfile
```

We discuss the syntax of the `authzfile` file in detail later in this chapter, in 第 6.5 节 “基于路径的授权”. Note that the `authz-db` variable isn't mutually exclusive with the `anon-access` and `auth-access` variables; if all the variables are defined at once, *all* of the rules must be satisfied before access is allowed.

6.3.3. 让 svnserve 使用 SASL

For many teams, the built-in CRAM-MD5 authentication is all they need from **svnserve**. However, if your server (and your Subversion clients) were built with the Cyrus Simple Authentication and Security Layer (SASL) library, you have a number of authentication and encryption options available to you.

> **什么是 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 **svn** client know how to make use of this library. It may or may not be available to you: if you're building Subversion yourself, you'll need to have at least version 2.1 of SASL installed on your system, and you'll need to make sure that it's detected during Subversion's build process. If you're using a prebuilt Subversion binary package, you'll have to check with the package maintainer as to whether SASL support was compiled in. SASL comes with a number of pluggable modules that represent different authentication systems: Kerberos (GSSAPI), NTLM, One-Time-Passwords (OTP), DIGEST-MD5, LDAP, Secure-Remote-Password (SRP), and others. Certain mechanisms may or may not be available to you; be sure to check which modules are provided.
>
> You can download Cyrus SASL (both code and documentation) from `http://asg.web.cmu.edu/sasl/sasl-library.html`.

Normally, when a subversion client connects to **svnserve**, the server sends a greeting that advertises a list of the capabilities it supports, and the client responds with a similar list of capabilities. If the server is configured to require authentication, it then sends a challenge that lists the authentication mechanisms available; the client responds by choosing one of the mechanisms, and then authentication is carried out in some number of round-trip messages. Even when SASL capabilities aren't present, the client and server inherently know how to use the CRAM-MD5 and ANONYMOUS mechanisms (see 第 6.3.2 节 “内置的认证和授权”). If server and client were linked against SASL, a number of other authentication mechanisms may also be available. However, you'll need to explicitly configure SASL on the server side to advertise them.

6.3.3.1. 使用 SASL 认证

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

```
[sasl]
use-sasl = true
```

Second, create a main SASL configuration file called `svn.conf` in a place where the SASL library can find it—typically in the directory where SASL plug-ins are located. You'll have to locate the plug-in directory on your particular system, such as `/usr/lib/sasl2/` or `/etc/sasl2/`. (Note that this is *not* the `svnserve.conf` file that lives within a repository!)

On a Windows server, you'll also have to edit the system registry (using a tool such as **regedit**) 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 keys inside it: a key called `SearchPath` (whose value is a path to the directory containing the SASL `sasl*.dll` plug-in libraries), and a key called `ConfFile` (whose value is a path to the parent directory containing the `svn.conf` file you created).

Because SASL provides so many different kinds of authentication mechanisms, it would be foolish (and far beyond the scope of this book) to try to describe every possible server-side configuration. Instead, we recommend that you read the documentation supplied in the `doc/` subdirectory of the SASL source code. It goes into great detail about every mechanism and how to configure the server appropriately for each. For the purposes of this discussion, we'll just demonstrate a simple example of configuring the DIGEST-MD5 mechanism. For example, if your `subversion.conf` (or `svn.conf`) file contains the following:

```
pwcheck_method: auxprop
auxprop_plugin: sasldb
sasldb_path: /etc/my_sasldb
mech_list: DIGEST-MD5
```

you've told SASL to advertise the DIGEST-MD5 mechanism to clients and to check user passwords against a private password database located at `/etc/my_sasldb`. A system administrator can then use the **saslpasswd2** program to add or modify usernames and passwords in the database:

```
$ saslpasswd2 -c -f /etc/my_sasldb -u realm username
```

A few words of warning: first, make sure the “realm” argument to **saslpasswd2** matches the same realm you've defined in your repository's `svnserve.conf` file; if they don't match, authentication will fail. Also, due to a shortcoming in SASL, the common realm must be a string with no space characters. Finally, if you decide to go with the standard SASL password database, make sure the **svnserve** program has read access to the file (and possibly write access as well, if you're using a mechanism such as OTP).

This is just one simple way of configuring SASL. Many other authentication mechanisms are available, and passwords can be stored in other places such as in LDAP or a SQL database. Consult the full SASL documentation for details.

Remember that if you configure your server to only allow certain SASL authentication mechanisms, this forces all connecting clients to have SASL support as well. Any Subversion client built without

SASL support (which includes all pre-1.5 clients) will be unable to authenticate. On the one hand, this sort of restriction may be exactly what you want (“My clients must all use Kerberos!”). However, if you still want non-SASL clients to be able to authenticate, be sure to advertise the CRAM-MD5 mechanism as an option. All clients are able to use CRAM-MD5, whether they have SASL capabilities or not.

6.3.3.2. SASL 加密

SASL is also able to perform data encryption if a particular mechanism supports it. The built-in CRAM-MD5 mechanism doesn't support encryption, but DIGEST-MD5 does, and mechanisms such as SRP actually require use of the OpenSSL library. 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.

6.3.4. 穿越 SSH 隧道

svnserve的内置认证(和SASL支持)可以非常容易得到，因为它避免了创建真实的系统帐号，另一方面，一些管理员已经创建好了SSH认证框架，在这种情况下，所有的项目用户已经拥有了系统帐号和有能力“SSH到”服务器。

SSH与**svnserve**结合很简单，客户端只需要使用`svn+ssh://`的URL模式来连接：

```
$ whoami
harry

$ svn list svn+ssh://host.example.com/repos/project
harryssh@host.example.com's password:  *****

foo
bar
baz
…
```

In this example, the Subversion client is invoking a local **ssh** process, connecting to `host.example.com`, authenticating as the user `harryssh` (according to SSH user configuration), then spawning a private **svnserve** process on the remote machine running as the user `harryssh`.

The **svnserve** command is being invoked in tunnel mode (`-t`), and its network protocol is being "tunneled" over the encrypted connection by **ssh**, the tunnel agent. If the client performs a commit, the authenticated username `harryssh` will be used as the author of the new revision.

这里要理解的最重要的事情是Subversion客户端不是连接到运行中的**svnserve**守护进程，这种访问方法不需要一个运行的守护进程，也不需要在必要时唤醒一个，它依赖于**ssh**来发起一个**svnserve**进程，然后网络断开后终止进程。

当使用`svn+ssh://`的URL访问版本库时，记住是**ssh**提示请求认证，而不是**svn**客户端程序。这意味着密码不会有自动缓存(见第 3.11.2 节 "客户端凭证缓存")，Subversion客户端通常会建立多个版本库的连接，但用户通常会因为密码缓存特性而没有注意到这一点，当使用`svn+ssh://`的URL时，用户会为**ssh**在每次建立连接时重复的询问密码感到讨厌，解决方案是用一个独立的SSH密码缓存工具，像类Unix系统的**ssh-agent**或者是Windows下的**pageant**。

When running over a tunnel, authorization is primarily controlled by operating system permissions to the repository's database files; it's very much the same as if Harry were accessing the repository directly via a `file://` URL. If multiple system users are going to be accessing the repository directly, you may want to place them into a common group, and you'll need to be careful about umasks (be sure to read 第 6.6 节 "支持多种版本库访问方法" later in this chapter). But even in the case of tunneling, you can still use the `svnserve.conf` file to block access, by simply setting `auth-access = read` or `auth-access = none`.[2]

你会认为SSH管道的故事该结束了，但还不是，Subversion允许你在运行配置文件`config`(见第 7.1 节 "运行配置区")创建一个自定义的管道行为方式，举个例子，假定你希望使用RSH而不是SSH，在`config`文件的`[tunnels]`部分作如下定义：

```
[tunnels]
rsh = rsh
```

And now, you can use this new tunnel definition by using a URL scheme that matches the name of your new variable: `svn+rsh://host/path`. When using the new URL scheme, the Subversion client will actually be running the command **`rsh host svnserve -t`** behind the scenes. If you include a username in the URL (e.g., `svn+rsh://username@host/path`), the client will also include that in its command (**`rsh username@host svnserve -t`**). But you can define new tunneling schemes to be much more clever than that:

```
[tunnels]
joessh = $JOESSH /opt/alternate/ssh -p 29934
```

This example demonstrates a couple of things. First, it shows how to make the Subversion client launch a very specific tunneling binary (the one located at `/opt/alternate/ssh`) with specific options. In this case, accessing an `svn+joessh://` URL would invoke the particular SSH binary with `-p 29934` as arguments—useful if you want the tunnel program to connect to a nonstandard port.

Second, it shows how to define a custom environment variable that can override the name of the tunneling program. Setting the `SVN_SSH` environment variable is a convenient way to override the

[2]请注意，使用**svnserve**的访问控制进行权限控制将会失去意义，因为用户已经直接访问到了版本库数据。

default SSH tunnel agent. But if you need to have several different overrides for different servers, each perhaps contacting a different port or passing a different set of options to SSH, you can use the mechanism demonstrated in this example. Now if we were to set the `JOESSH` environment variable, its value would override the entire value of the tunnel variable—**$JOESSH** would be executed instead of **`/opt/alternate/ssh -p 29934`**.

6.3.5. SSH 配置技巧

It's possible to control not only the way in which the client invokes **ssh**, but also to control the behavior of **sshd** on your server machine. In this section, we'll show how to control the exact **svnserve** command executed by **sshd**, as well as how to have multiple users share a single system account.

6.3.5.1. 初始设置

作为开始，定位到你启动**svnserve**的帐号的主目录，确定这个账户已经安装了一套SSH公开/私有密钥对，用户可以通过公开密钥认证，因为所有如下的技巧围绕着使用SSH`authorized_keys`文件，密码认证在这里不会工作。

如果这个文件还不存在，创建一个`authorized_keys`文件(在UNIX下通常是`~/.ssh/authorized_keys`)，这个文件的每一行描述了一个允许连接的公钥，这些行通常是下面的形式：

```
ssh-dsa AAAABtce9euch.... user@example.com
```

第一个字段描述了密钥的类型，第二个字段是未加密的密钥本身，第三个字段是注释。然而，这是一个很少人知道的事实，可以使用一个`command`来处理整行：

```
command="program" ssh-dsa AAAABtce9euch.... user@example.com
```

When the `command` field is set, the SSH daemon will run the named program instead of the typical tunnel-mode **svnserve** invocation that the Subversion client asks for. This opens the door to a number of server-side tricks. In the following examples, we abbreviate the lines of the file as:

```
command="program" TYPE KEY COMMENT
```

6.3.5.2. 控制调用的命令

因为我们可以指定服务器端执行的命令，我们很容易来选择运行一个特定的**svnserve**程序来并且传递给它额外的参数：

```
command="/path/to/svnserve -t -r /virtual/root" TYPE KEY COMMENT
```

在这个例子里，`/path/to/svnserve`也许会是一个**svnserve**程序的包裹脚本，会来设置umask(见第 6.6 节 “支持多种版本库访问方法”)。它也展示了怎样在虚拟根目录定位一

个**svnserve**，就像我们经常在使用守护进程模式下运行**svnserve**一样。这样做不仅可以把访问限制在系统的一部分，也可以使用户不需要在svn+ssh://URL里输入绝对路径。

It's also possible to have multiple users share a single account. Instead of creating a separate system account for each user, generate a public/private key pair for each person. Then place each public key into the authorized_users file, one per line, and use the --tunnel-user option:

```
  command="svnserve -t --tunnel-user=harry" TYPE1 KEY1 harry@example.com
  command="svnserve -t --tunnel-user=sally" TYPE2 KEY2 sally@example.com
```

This example allows both Harry and Sally to connect to the same account via public key authentication. Each of them has a custom command that will be executed; the --tunnel-user option tells **svnserve** to assume that the named argument is the authenticated user. Without --tunnel-user, it would appear as though all commits were coming from the one shared system account.

A final word of caution: giving a user access to the server via public-key in a shared account might still allow other forms of SSH access, even if you've set the command value in authorized_keys. For example, the user may still get shell access through SSH or be able to perform X11 or general port forwarding through your server. To give the user as little permission as possible, you may want to specify a number of restrictive options immediately after the command:

```
  command="svnserve -t --tunnel-user=harry",no-port-forwarding,no-agent-fo
arding,no-X11-forwarding,no-pty TYPE1 KEY1 harry@example.com
```

Note that this all must be on one line—truly on one line—since SSH authorized_keys files do not even allow the conventional backslash character (\) for line continuation. The only reason we've shown it with a line break is to fit it on the physical page of a book.

6.4. httpd - Apache 的 HTTP 服务器

The Apache HTTP Server is a “heavy-duty” network server that Subversion can leverage. Via a custom module, **httpd** makes Subversion repositories available to clients via the WebDAV/DeltaV protocol, which is an extension to HTTP 1.1 (see http://www.webdav.org/ for more information). This protocol takes the ubiquitous HTTP protocol that is the core of the World Wide Web, and adds writing—specifically, versioned writing—capabilities. The result is a standardized, robust system that is conveniently packaged as part of the Apache 2.0 software, supported by numerous operating systems and third-party products, and doesn't require network administrators to open up yet another custom port. [3] While an Apache-Subversion server has more features than **svnserve**, it's also a bit more difficult to set up. With flexibility often comes more complexity.

下面的讨论包括了对Apache配置指示的引用，给了一些使用这些指示的例子，详细地描述不在本章的范围之内，Apache小组维护了完美的文档，公开存放在他们的站点http://httpd.apache.org。例如，一个一般的配置参考位于 http://httpd.apache.org/docs-2.0/mod/directives.html。

[3]他们讨厌这样做。

同样，当你修改你的Apache设置，很有可能会出现一些错误，如果你还不熟悉Apache的日志子系统，你一定需要认识到这一点。在你的文件`httpd.conf`里会指定Apache生成的访问和错误日志(`CustomLog`和`ErrorLog`指示)的磁盘位置。Subversion的mod_dav_svn使用Apache的错误日志接口，你可以浏览这个文件的内容查看信息来查找难于发现的问题根源。

为什么是 Apache 2?

If you're a system administrator, it's very likely that you're already running the Apache web server and have some prior experience with it. At the time of this writing, Apache 1.3 is the more popular version of Apache. The world has been somewhat slow to upgrade to the Apache 2.x series for various reasons: some people fear change, especially changing something as critical as a web server. Other people depend on plug-in modules that work only against the Apache 1.3 API, and they are waiting for a 2.x port. Whatever the reason, many people begin to worry when they first discover that Subversion's Apache module is written specifically for the Apache 2 API.

对此问题的适当反应是：不需要担心，同时运行Apache 1.3和Apache 2非常简单，只需要安装到不同的位置，用Apache 2作为Subversion的专用服务器，并且不使用80端口，客户端可以访问版本库时在URL里指定端口：

```
$ svn checkout http://host.example.com:7382/repos/project
```

6.4.1. 先决条件

为了让你的版本库使用HTTP网络，你基本上需要两个包里的四个部分。你需要Apache **httpd**2.0和包括的**mod_dav** DAV模块，Subversion和与之一同分发的**mod_dav_svn**文件系统提供者模块，如果你有了这些组件，网络化你的版本库将非常简单，如：

- 配置好httpd 2.0，并且使用**mod_dav**启动
- Installing the **mod_dav_svn** backend to **mod_dav**, which uses Subversion's libraries to access the repository
- 配置你的`httpd.conf`来输出(或者说暴露)版本库

你可以通过从源代码编译**httpd**和Subversion来完成前两个项目，也可以通过你的系统上的已经编译好的二进制包来安装。最新的使用Apache HTTP的Subversion的编译方法和Apache的配置方式可以看Subversion源代码树根目录的`INSTALL`文件。

6.4.2. 基本的 Apache 配置

一旦你安装了必须的组件，剩下的工作就是在`httpd.conf`里配置Apache，使用`LoadModule`来加载mod_dav_svn模块，这个指示必须先与其它Subversion相关的其它配置出现，如果你的Apache使用缺省布局安装，你的**mod_dav_svn**模块一定在Apache安装目录(通常是在`/usr/local/apache2`)的`modules`子目录，`LoadModule`指示的语法很简单，影射一个名字到它的共享库的物理位置：

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

注意，如果**mod_dav**是作为共享对象编译(而不是静态链接到**httpd**程序)，你需要为它使用LoadModule语句，一定确定它在**mod_dav_svn**之前：

```
LoadModule dav_module         modules/mod_dav.so
LoadModule dav_svn_module     modules/mod_dav_svn.so
```

在你的配置文件后面的位置，你需要告诉Apache你在什么地方保存Subversion版本库(也许是多个)，位置指示有一个很像XML的符号，开始于一个开始标签，以一个结束标签结束，配合中间许多的其它配置。Location指示的目的是告诉Apache在特定的URL以及子URL下需要特殊的处理，如果是为Subversion准备的，你希望可以通过告诉Apache特定URL是指向版本化的资源，从而把支持转交给DAV层，你可以告诉Apache将所有路径部分(URL中服务器名称和端口之后的部分)以/repos/开头的URL交由DAV服务提供者处理。一个DAV服务提供者的版本库位于/absolute/path/to/repository，可以使用如下的httpd.conf语法：

```
<Location /repos>
  DAV svn
  SVNPath /var/svn/repository
</Location>
```

如果你计划支持多个具备相同父目录的Subversion版本库，你有另外的选择，SVNParentPath指示，来表示共同的父目录。举个例子，如果你知道会在/usr/local/svn下创建多个Subversion版本库，并且通过类似http://my.server.com/svn/repos1，http://my.server.com/svn/repos2的URL访问，你可以用后面例子中的httpd.conf配置语法：

```
<Location /svn>
  DAV svn

  # any "/svn/foo" URL will map to a repository /var/svn/foo
  SVNParentPath /var/svn
</Location>
```

Using the previous syntax, Apache will delegate the handling of all URLs whose path portions begin with /svn/ to the Subversion DAV provider, which will then assume that any items in the directory specified by the SVNParentPath directive are actually Subversion repositories. This is a particularly convenient syntax in that, unlike the use of the SVNPath directive, you don't have to restart Apache to create and network new repositories.

请确定当你定义新的Location，不会与其它输出的位置重叠。例如你的主要DocumentRoot是/www，不要把Subversion版本库输出到<Location /www/repos>，如果一个请求的URI是/www/repos/foo.c，Apache不知道是直接到repos/foo.c访问这个文件还是让**mod_dav_svn**代理从Subversion版本库返回foo.c。服务器返回的结果通常是301 Moved Permanently。

服务器名称和复制请求

Subversion利用COPY请求类型来执行服务器端的文件和目录拷贝，作为一个健全的Apache模块的一部分，拷贝源和拷贝的目标通常坐落在同一个机器上，为了满足这个需求，你或许需要告诉mod_dav服务器主机的名称，通常你可以使用httpd.conf的ServerName指示来完成此目的。

```
ServerName svn.example.com
```

If you are using Apache's virtual hosting support via the `NameVirtualHost` directive, you may need to use the `ServerAlias` directive to specify additional names by which your server is known. Again, refer to the Apache documentation for full details.

在本阶段，你一定要考虑访问权限问题，如果你已经作为普通的web服务器运行过Apache，你一定有了一些内容—网页、脚本和其他。这些项目已经配置了许多在Apache下可以工作的访问许可，或者更准确一点，允许Apache与这些文件一起工作。Apache当作为Subversion服务器运行时，同样需要正确的访问许可来读写你的Subversion版本库。

你会需要检验权限系统的设置满足Subversion的需求，同时不会把以前的页面和脚本搞乱。这或许意味着修改Subversion的访问许可来配合Apache服务器已经使用的工具，或者可能意味着需要使用httpd.conf的User和Group指示来指定Apache作为运行的用户和Subversion版本库的组。并不是只有一条正确的方式来设置许可，每个管理员都有不同的原因来以特定的方式操作，只需要意识到许可关联的问题经常在为Apache配置Subversion版本库的过程中被疏忽。

6.4.3. 认证选项

At this point, if you configured `httpd.conf` to contain something such as the following:

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

your repository is “anonymously” accessible to the world. Until you configure some authentication and authorization policies, the Subversion repositories that you make available via the `Location` directive will be generally accessible to everyone. In other words:

- 任何人可以使用Subversion客户端来从版本库URL取出一个工作拷贝(或者是它的子目录)。
- 任何人可以在浏览器输入版本库URL交互浏览的方式来查看版本库的最新修订版本。
- 任何人可以提交到版本库。

Of course, you might have already set up a `pre-commit` hook script to prevent commits (see 第 5.3.2 节 “实现版本库钩子”). But as you read on, you'll see that it's also possible to use Apache's built-in methods to restrict access in specific ways.

6.4.3.1. 配置 HTTP 认证

最简单的客户端认证方式是通过HTTP基本认证机制，简单的使用用户名和密码来验证一个用户所自称的身份，Apache提供了一个**htpasswd**工具来管理可接受的用户名和密码，这些就是你希望赋予Subversion特别权限的用户，让我们给Sally和Harry赋予提交权限，首先，我们需要添加他们到密码文件：

```
$ ### First time: use -c to create the file
$ ### Use -m to use MD5 encryption of the password, which is more secure
$ htpasswd -cm /etc/svn-auth-file harry
New password: *****
Re-type new password: *****
Adding password for user harry
$ htpasswd -m /etc/svn-auth-file sally
New password: *******
Re-type new password: *******
Adding password for user sally
$
```

Next, you need to add some more `httpd.conf` directives inside your `Location` block to tell Apache what to do with your new password file. The `AuthType` directive specifies the type of authentication system to use. In this case, we want to specify the `Basic` authentication system. `AuthName` is an arbitrary name that you give for the authentication domain. Most browsers will display this name in the pop-up dialog box when the browser is querying the user for her name and password. Finally, use the `AuthUserFile` directive to specify the location of the password file you created using **htpasswd**.

添加完这三个指示，你的`<Location>`区块一定像这个样子：

```
<Location /svn>
  DAV svn
  SVNParentPath /var/svn
  AuthType Basic
  AuthName "Subversion repository"
  AuthUserFile /etc/svn-auth-file
</Location>
```

这个`<Location>`区块还没有结束，还不能做任何有用的事情，它只是告诉Apache当需要授权时，要去向Subversion客户端索要用户名和密码。我们这里遗漏的，是一些告诉Apache什么样客户端需要授权的指示。哪里需要授权，Apache就会在哪里要求认证，最简单的方式是保护所有的请求，添加`Require valid-user`来告诉Apache任何请求需要认证的用户：

```
<Location /svn>
  DAV svn
  SVNParentPath /var/svn
  AuthType Basic
```

```
  AuthName "Subversion repository"
  AuthUserFile /etc/svn-auth-file
  Require valid-user
</Location>
```

一定要阅读后面的部分(第 6.4.4 节 “授权选项”)来得到Require的细节，和授权政策的其他设置方法。

One word of warning: HTTP Basic Auth passwords pass in very nearly plain text over the network, and thus are extremely insecure.

Another option is to not use Basic authentication, but to use Digest authentication instead. Digest authentication allows the server to verify the client's identity *without* passing the plain-text password over the network. Assuming that the client and server both know the user's password, they can verify that the password is the same by using it to apply a hashing function to a one-time bit of information. The server sends a small random-ish string to the client; the client uses the user's password to hash the string; the server then looks to see whether the hashed value is what it expected.

Configuring Apache for Digest authentication is also fairly easy, and only a small variation on our prior example. Be sure to consult Apache's documentation for full details.

```
<Location /svn>
  DAV svn
  SVNParentPath /var/svn
  AuthType Digest
  AuthName "Subversion repository"
  AuthDigestDomain /svn/
  AuthUserFile /etc/svn-auth-file
  Require valid-user
</Location>
```

If you're looking for maximum security, public key cryptography is the best solution. It may be best to use some sort of SSL encryption, so that clients authenticate via `https://` instead of `http://`; at a bare minimum, you can configure Apache to use a self-signed server certificate. [4] Consult Apache's documentation (and OpenSSL documentation) about how to do that.

6.4.3.2. SSL 证书管理

商业应用需要越过公司防火墙的版本库访问，防火墙需要小心的考虑非认证用户“吸取”他们的网络流量的情况，SSL让那种形式的关注更不容易导致敏感数据泄露。

If a Subversion client is compiled to use OpenSSL, it gains the ability to speak to an Apache server via `https://` URLs. The Neon library used by the Subversion client is not only able to verify server certificates, but can also supply client certificates when challenged. When the client and server have exchanged SSL certificates and successfully authenticated one another, all further communication is encrypted via a session key.

[4]当使用自签名的服务器时仍会遭受“中间人”攻击，但是与偷取未保护的密码相比，这样的攻击比一个偶然的获取要艰难许多。

It's beyond the scope of this book to describe how to generate client and server certificates and how to configure Apache to use them. Many other books, including Apache's own documentation, describe this task. But what we *can* cover here is how to manage server and client certificates from an ordinary Subversion client.

当通过`https://`与Apache通讯时，一个Subversion客户端可以接收两种类型的信息：

- 一个服务器证书
- 一个客户端证书的要求

If the client receives a server certificate, it needs to verify that it trusts the certificate: is the server really who it claims to be? The OpenSSL library does this by examining the signer of the server certificate, or *certificate authority* (CA). If OpenSSL is unable to automatically trust the CA, or if some other problem occurs (such as an expired certificate or hostname mismatch), the Subversion command-line client will ask you whether you want to trust the server certificate anyway:

```
$ svn list https://host.example.com/repos/project

Error validating server certificate for 'https://host.example.com:443':
 - The certificate is not issued by a trusted authority. Use the
   fingerprint to validate the certificate manually!
Certificate information:
 - Hostname: host.example.com
 - Valid: from Jan 30 19:23:56 2004 GMT until Jan 30 19:23:56 2006 GMT
 - Issuer: CA, example.com, Sometown, California, US
 - Fingerprint: 7d:e1:a9:34:33:39:ba:6a:e9:a5:c4:22:98:7b:76:5c:92:a0:9c:7

(R)eject, accept (t)emporarily or accept (p)ermanently?
```

This dialogue should look familiar; it's essentially the same question you've probably seen coming from your web browser (which is just another HTTP client like Subversion). If you choose the `(p)`ermanent option, the server certificate will be cached in your private runtime `auth/` area in just the same way your username and password are cached (see 第 3.11.2 节 “客户端凭证缓存”). If cached, Subversion will automatically trust this certificate in future negotiations.

你的运行中`servers`文件也会给你能力可以让Subversion客户端自动信任特定的CA，包括全局的或是每主机为基础的，只需要设置`ssl-authority-files`为一组逗号隔开的PEM加密的CA证书列表：

```
[global]
ssl-authority-files = /path/to/CAcert1.pem;/path/to/CAcert2.pem
```

许多OpenSSL安装包括一些预先定义好的可以普遍信任的“缺省的”CA，为了让Subversion客户端自动信任这些标准权威，设置`ssl-trust-default-ca`为`true`。

When talking to Apache, a Subversion client might also receive a challenge for a client certificate. Apache is asking the client to identify itself: is the client really who it says it is? If all goes correctly, the Subversion client sends back a private certificate signed by a CA that Apache trusts. A client certificate

is usually stored on disk in encrypted format, protected by a local password. When Subversion receives this challenge, it will ask you for a path to the certificate and the password that protects it:

```
$ svn list https://host.example.com/repos/project

Authentication realm: https://host.example.com:443
Client certificate filename: /path/to/my/cert.p12
Passphrase for '/path/to/my/cert.p12':  ********
…
```

注意这个客户端证书是一个“p12”文件，为了让Subversion使用客户端证书，它必须是运输标准的PKCS#12格式，大多数浏览器可以导入和导出这种格式的证书，另一个选择是用OpenSSL命令行工具来转化存在的证书为PKCS#12格式。

再次，运行中servers文件允许你为每个主机自动响应这种要求，单个或两条信息可以用运行参数来描述：

```
[groups]
examplehost = host.example.com

[examplehost]
ssl-client-cert-file = /path/to/my/cert.p12
ssl-client-cert-password = somepassword
```

一旦你设置了ssl-client-cert-file和 ssl-client-cert-password参数，Subversion客户端可以自动响应客户端证书请求而不会打扰你。[5]

6.4.4. 授权选项

此刻，你已经配置了认证，但是没有配置授权，Apache可以要求用户认证并且确定身份，但是并没有说明这个身份的怎样允许和限制，这个部分描述了两种控制访问版本库的策略。

6.4.4.1. 整体访问控制

最简单的访问控制形式是授权特定用户为只读版本库访问或者是读/写访问版本库。

你可以通过在<Location>区块添加Require valid-user指示来限制所有的版本库操作，使用我们前面的例子，这意味着只有客户端只可以是harry或者sally，而且他们必须提供正确的用户名及对应密码，这样允许对Subversion版本库做任何事：

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

  # how to authenticate a user
```

[5]更多有安全意识的人不会希望在运行中servers文件保存客户端证书密码。

```
  AuthType Basic
  AuthName "Subversion repository"
  AuthUserFile /path/to/users/file

  # only authenticated users may access the repository
  Require valid-user
</Location>
```

有时候，你不需要这样严密，举个例子，Subversion自己在http://svn.collab.net/repos/svn的源代码允许全世界的人执行版本库的只读操作(例如检出我们的工作拷贝和使用浏览器浏览版本库)，但是限定只有认证用户可以执行写操作。为了执行特定的限制，你可以使用Limit和LimitExcept配置指示，就像Location指示，这个区块有开始和结束标签，你需要在<Location>中添加这个指示。

在Limit和LimitExcept中使用的参数是可以被这个区块影响的HTTP请求类型，举个例子，如果你希望禁止所有的版本库访问，只是保留当前支持的只读操作，你可以使用LimitExcept指示，并且使用GET，PROPFIND，OPTIONS和REPORT请求类型参数，然后前面提到过的Require valid-user指示将会在<LimitExcept>区块中而不是在<Location>区块。

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

  # how to authenticate a user
  AuthType Basic
  AuthName "Subversion repository"
  AuthUserFile /path/to/users/file

  # For any operations other than these, require an authenticated user.
  <LimitExcept GET PROPFIND OPTIONS REPORT>
    Require valid-user
  </LimitExcept>
</Location>
```

这里只是一些简单的例子，想看关于Apache访问控制Require指示的更深入信息，可以查看Apache文档中的教程集http://httpd.apache.org/docs-2.0/misc/tutorials.html中的Security部分。

6.4.4.2. 每目录访问控制

也可以使用Apache的httpd模块**mod_authz_svn**更加细致的设置访问权限，这个模块收集客户端传递过来的不同的晦涩的URL信息，询问**mod_dav_svn**来解码，然后根据在配置文件定义的访问政策来裁决请求。

如果你从源代码创建Subversion，**mod_authz_svn**会自动附加到**mod_dav_svn**，许多二进制分发版本也会自动安装，为了验证它是安装正确，确定它是在httpd.conf的LoadModule指示中的**mod_dav_svn**后面：

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

为了激活这个模块，你需要配置你的Location区块的AuthzSVNAccessFile指示，指定保存路径中的版本库访问政策的文件。(一会儿我们将会讨论这个文件的格式。)

Apache非常的灵活，你可以从三种模式里选择一种来配置你的区块，作为开始，你选择一种基本的配置模式。(下面的例子非常简单；见Apache自己的文档中的认证和授权选项来查看更多的细节。)

最简单的区块是允许任何人可以访问，在这个场景里，Apache决不会发送认证请求，所有的用户作为“匿名”对待(见例 6.1 “匿名访问的配置样例” 。)

例 6.1. 匿名访问的配置样例

```
<Location /repos>
  DAV svn
  SVNParentPath /var/svn

  # our access control policy
  AuthzSVNAccessFile /path/to/access/file
</Location>
```

在另一个极端，你可以配置为拒绝所有人的认证，所有客户端必须提供证明自己身份的证书，你通过Require valid-user指示来阻止无条件的认证，并且定义一种认证的手段。(见例 6.2 “认证访问的配置样例” 。)

例 6.2. 认证访问的配置样例

```
<Location /repos>
  DAV svn
  SVNParentPath /var/svn

  # our access control policy
  AuthzSVNAccessFile /path/to/access/file

  # only authenticated users may access the repository
  Require valid-user

  # how to authenticate a user
  AuthType Basic
  AuthName "Subversion repository"
  AuthUserFile /path/to/users/file
</Location>
```

第三种流行的模式是允许认证和匿名用户的组合，举个例子，许多管理员希望允许匿名用户读取特定的版本库路径，但希望只有认证用户可以读(或者写)更多敏感的区域，在这个设置里，所有的用户开始时用匿名用户访问版本库，如果你的访问控制策略在任何时候要求一个真实的用户名，Apache将会要求认证客户端，为此，你可以同时使用`Satisfy Any`和`Require valid-user`指示。(见例 6.3 “混合认证/匿名访问的配置样例” 。)

例 **6.3.** 混合认证/匿名访问的配置样例

```
<Location /repos>
  DAV svn
  SVNParentPath /var/svn

  # our access control policy
  AuthzSVNAccessFile /path/to/access/file

  # try anonymous access first, resort to real
  # authentication if necessary.
  Satisfy Any
  Require valid-user

  # how to authenticate a user
  AuthType Basic
  AuthName "Subversion repository"
  AuthUserFile /path/to/users/file
</Location>
```

Once you've settled on one of these three basic `httpd.conf` templates, you need to create your file containing access rules for particular paths within the repository. We describe this later in this chapter, in 第 6.5 节 “基于路径的授权” .

6.4.4.3. 禁用基于路径的检查

The **mod_dav_svn** module goes through a lot of work to make sure that data you've marked “unreadable” doesn't get accidentally leaked. This means it needs to closely monitor all of the paths and file-contents returned by commands such as **svn checkout** and **svn update**. If these commands encounter a path that isn't readable according to some authorization policy, the path is typically omitted altogether. In the case of history or rename tracing—for example, running a command such as **`svn cat -r OLD foo.c`** on a file that was renamed long ago—the rename tracking will simply halt if one of the object's former names is determined to be read-restricted.

All of this path checking can sometimes be quite expensive, especially in the case of **svn log**. When retrieving a list of revisions, the server looks at every changed path in each revision and checks it for readability. If an unreadable path is discovered, it's omitted from the list of the revision's changed paths (normally seen with the `--verbose` option), and the whole log message is suppressed. Needless to say, this can be time-consuming on revisions that affect a large number of files. This is the cost of security: even if you haven't configured a module such as **mod_authz_svn** at all, the **mod_dav_svn**

module is still asking Apache **httpd** to run authorization checks on every path. The **mod_dav_svn** module has no idea what authorization modules have been installed, so all it can do is ask Apache to invoke whatever might be present.

On the other hand, there's also an escape hatch of sorts, which allows you to trade security features for speed. If you're not enforcing any sort of per-directory authorization (i.e., not using **mod_authz_svn** or similar module), you can disable all of this path checking. In your `httpd.conf` file, use the `SVNPathAuthz` directive as shown in 例 6.4 “禁用所有的路径检查”.

例 **6.4.** 禁用所有的路径检查

```
<Location /repos>
  DAV svn
  SVNParentPath /var/svn

  SVNPathAuthz off
</Location>
```

The `SVNPathAuthz` directive is “on” by default. When set to “off,” all path-based authorization checking is disabled; **mod_dav_svn** stops invoking authorization checks on every path it discovers.

6.4.5. 额外的糖果

我们已经覆盖了关于认证和授权的Apache和 **mod_dav_svn**的大多数选项，但是Apache还提供了许多很好的特性。

6.4.5.1. 版本库浏览

使用Apache/WebDAV配置Subversion版本库时一个非常有用的好处是可以用普通的浏览器察看最新的版本库文件，因为Subversion使用URL来鉴别版本库版本化的资源，版本库使用的HTTP为基础的URL也可以直接输入到Web浏览器中，你的浏览器会发送一个`GET`请求到URL，根据访问的URL是指向一个版本化的目录还是文件，mod_dav_svn会负责列出目录列表或者是文件内容。

因为URL不能确定你所希望看到的资源的版本，**mod_dav_svn**会一直返回最新的版本，这样会有一些美妙的副作用，你可以直接把Subversion的URL传递给文档作为引用，这些URL会一直指向文档最新的材料，当然，你也可以在别的网站作为超链使用这些URL。

我可以看到老的修订版本吗?

通过一个普通的浏览器?一句话:不可以,至少是当你只使用**mod_dav_svn**作为唯一的工具时。

你的Web浏览器只会说普通的HTTP,也就是说它只会GET公共的URL,这个URL代表了最新版本的文件和目录,根据WebDAV/DeltaV规范,每种服务器定义了一种私有的URL语法来代表老的资源的版本,这个语法对客户端是不透明的,为了得到老的版本,一个客户端必须通过一种规范过程来“发现”正确的URL;这个过程包括执行一系列WebDAV PROPFIND请求和理解DeltaV概念,这些事情一般是你的web浏览器做不了的。

So, to answer the question, one obvious way to see older revisions of files and directories is by passing the `--revision` (`-r`) argument to the **svn list** and **svn cat** commands. To browse old revisions with your web browser, however, you can use third-party software. A good example of this is ViewVC (`http://viewvc.tigris.org/`). ViewVC was originally written to display CVS repositories through the Web, [6] and the latest releases are able to understand Subversion repositories as well.

6.4.5.1.1. 正确的 MIME 类型

当浏览Subversion版本库时,web浏览器通过从Apache的HTTP GET返回内容中查看`Content-Type:`头可以知道如何渲染文件的线索,这个值是一种MIME类型。默认情况下,Apache告诉浏览器所有的版本库文件都是缺省的MIME类型,通常是`text/plain`,这样有时候会让人沮丧,如果一个用户希望版本库文件能够更有意义的渲染—例如一个`foo.html`,在浏览时最好能够按照HTML方式渲染。

To make this happen, you need only to make sure that your files have the proper `svn:mime-type` set. We discuss this in more detail in 第 3.3.1 节 “文件内容类型”, and you can even configure your client to automatically attach proper `svn:mime-type` properties to files entering the repository for the first time; see 第 3.2.4 节 “自动设置属性”.

So in our example, if one were to set the `svn:mime-type` property to `text/html` on file `foo.html`, Apache would properly tell your web browser to render the file as HTML. One could also attach proper `image/*` MIME-type properties to image files and ultimately get an entire web site to be viewable directly from a repository! There's generally no problem with this, as long as the web site doesn't contain any dynamically generated content.

6.4.5.1.2. 定制外观

你通常会在版本化的文件的URL之外得到更多地用处—毕竟那里是有趣的内容存在的地方,但是你会偶尔浏览一个Subversion的目录列表,你会很快发现展示列表生成的HTML非常基本,并且一定没有在外观上(或者是有趣上)下功夫,为了自定义这些目录显示,Subversion提供了一个XML目录特性,一个单独的`SVNIndexXSLT`指示在你的`httpd.conf`文件版本库的`Location`块里,它将会指导mod_dav_svn在显示目录列表的时候生成XML输出,并且引用你选择的XSLT样式表文件:

[6] Back then, it was called ViewCVS.

```
<Location /svn>
  DAV svn
  SVNParentPath /var/svn
  SVNIndexXSLT "/svnindex.xsl"
  …
</Location>
```

Using the `SVNIndexXSLT` directive and a creative XSLT stylesheet, you can make your directory listings match the color schemes and imagery used in other parts of your web site. Or, if you'd prefer, you can use the sample stylesheets provided in the Subversion source distribution's `tools/xslt/` directory. Keep in mind that the path provided to the `SVNIndexXSLT` directory is actually a URL path—browsers need to be able to read your stylesheets to make use of them!

6.4.5.1.3. 列出版本库

如果你通过 `SVNParentPath`指示从一个URL维护一组版本库，也可以让Apache在浏览器显示所有存在的版本库，只需要通过`SVNListParentPath`指示激活：

```
<Location /svn>
  DAV svn
  SVNParentPath /var/svn
  SVNListParentPath on
  …
</Location>
```

如果一个用户将浏览器指向`http://host.example.com/svn/`，她一定会看到`/var/svn`下所有的Subversion版本库，很明显这是一件安全问题，所以这个特性默认是关闭的。

6.4.5.2. Apache 日志

Because Apache is an HTTP server at heart, it contains fantastically flexible logging features. It's beyond the scope of this book to discuss all of the ways logging can be configured, but we should point out that even the most generic `httpd.conf` file will cause Apache to produce two logs: `error_log` and `access_log`. These logs may appear in different places, but are typically created in the logging area of your Apache installation. (On Unix, they often live in `/usr/local/apache2/logs/`.)

`error_log`描述了所有Apache运行中的内部错误，`access_log`记录了Apache接收到的所有HTTP请求，这个日志很容易查看，例如包括Subversion客户端的IP地址，哪些用户正确认证和请求成功还是失败。

不幸的是，因为HTTP是无状态协议，即使最简单的Subversion客户端操作会产生多个网络请求，很难通过查看`access_log`来确定用户的操作—大多数操作看起来像是一系列神秘的`PROPPATCH`、`GET`、`PUT`和`REPORT`请求。更糟糕的是，许多客户段操作会发送几乎完全相同的一系列请求，所以更加难以区分。

`mod_dav_svn`会成为一个辅助，通过激活“operational logging”属性，你可以告诉`mod_dav_svn`创建另外的日志文件，来描述你的客户端都做了哪些高级操作。

To do this, you need to make use of Apache's `CustomLog` directive (which is explained in more detail in Apache's own documentation). Be sure to invoke this directive *outside* your Subversion `Location` block:

```
<Location /svn>
  DAV svn
  …
</Location>

CustomLog logs/svn_logfile "%t %u %{SVN-ACTION}e" env=SVN-ACTION
```

In this example, we're asking Apache to create a special logfile, `svn_logfile`, in the standard Apache `logs` directory. The `%t` and `%u` variables are replaced by the time and username of the request, respectively. The really important parts are the two instances of `SVN-ACTION`. When Apache sees that variable, it substitutes the value of the `SVN-ACTION` environment variable, which is automatically set by **mod_dav_svn** whenever it detects a high-level client action.

So, instead of having to interpret a traditional `access_log` like this:

```
[26/Jan/2007:22:25:29 -0600] "PROPFIND /svn/calc/!svn/vcc/default HTTP/1.1
[26/Jan/2007:22:25:29 -0600] "PROPFIND /svn/calc/!svn/bln/59 HTTP/1.1" 207
[26/Jan/2007:22:25:29 -0600] "PROPFIND /svn/calc HTTP/1.1" 207 647
[26/Jan/2007:22:25:29 -0600] "REPORT /svn/calc/!svn/vcc/default HTTP/1.1"
[26/Jan/2007:22:25:31 -0600] "OPTIONS /svn/calc HTTP/1.1" 200 188
[26/Jan/2007:22:25:31 -0600] "MKACTIVITY /svn/calc/!svn/act/e6035ef7-5df0-
…
```

you can peruse a much more intelligible `svn_logfile` like this:

```
[26/Jan/2007:22:24:20 -0600] - get-dir /tags r1729 props
[26/Jan/2007:22:24:27 -0600] - update /trunk r1729 depth=infinity send-cop
[26/Jan/2007:22:25:29 -0600] - status /trunk/foo r1729 depth=infinity
[26/Jan/2007:22:25:31 -0600] sally commit r1730
```

完全记录的日志动作见“高级日志”一节。

6.4.5.3. 通过代理写

One of the nice advantages of using Apache as a Subversion server is that it can be set up for simple replication. For example, suppose that your team is distributed across four offices around the globe. The Subversion repository can exist only in one of those offices, which means the other three offices will not enjoy accessing it—they're likely to experience significantly slower traffic and response times when updating and committing code. A powerful solution is to set up a system consisting of one *master* Apache server and several *slave* Apache servers. If you place a slave server in each office, users can check out a working copy from whichever slave is closest to them. All read requests go to their local slave. Write requests get automatically routed to the single master server. When the commit

completes, the master then automatically “pushes” the new revision to each slave server using the **svnsync** replication tool.

This configuration creates a huge perceptual speed increase for your users, because Subversion client traffic is typically 80–90% read requests. And if those requests are coming from a *local* server, it's a huge win.

In this section, we'll walk you through a standard setup of this single-master/multiple-slave system. However, keep in mind that your servers must be running at least Apache 2.2.0 (with **mod_proxy** loaded) and Subversion 1.5 (**mod_dav_svn**).

6.4.5.3.1. 配置服务器

First, configure your master server's `httpd.conf` file in the usual way. Make the repository available at a certain URI location, and configure authentication and authorization however you'd like. After that's done, configure each of your “slave” servers in the exact same way, but add the special `SVNMasterURI` directive to the block:

```
<Location /svn>
  DAV svn
  SVNPath /var/svn/repos
  SVNMasterURI http://master.example.com/svn
  …
</Location>
```

This new directive tells a slave server to redirect all write requests to the master. (This is done automatically via Apache's **mod_proxy** module.) Ordinary read requests, however, are still serviced by the slaves. Be sure that your master and slave servers all have matching authentication and authorization configurations; if they fall out of sync, it can lead to big headaches.

Next, we need to deal with the problem of infinite recursion. With the current configuration, imagine what will happen when a Subversion client performs a commit to the master server. After the commit completes, the server uses **svnsync** to replicate the new revision to each slave. But because **svnsync** appears to be just another Subversion client performing a commit, the slave will immediately attempt to proxy the incoming write request back to the master! Hilarity ensues.

The solution to this problem is to have the master push revisions to a different `<Location>` on the slaves. This location is configured to *not* proxy write requests at all, but to accept normal commits from (and only from) the master's IP address:

```
<Location /svn-proxy-sync>
  DAV svn
  SVNPath /var/svn/repos
  Order deny,allow
  Deny from all
  # Only let the server's IP address access this Location:
  Allow from 10.20.30.40
```

```
  …
</Location>
```

6.4.5.3.2. 设置复制

Now that you've configured your `Location` blocks on master and slaves, you need to configure the master to replicate to the slaves. This is done the usual way— using **svnsync**. If you're not familiar with this tool, see 第 5.4.7 节 “版本库复制” for details.

First, make sure that each slave repository has a `pre-revprop-change` hook script which allows remote revision property changes. (This is standard procedure for being on the receiving end of **svnsync**.) Then log into the master server and configure each of the slave repository URIs to receive data from the master repository on the local disk:

```
$ svnsync init http://slave1.example.com/svn-proxy-sync file://var/svn/rep
Copied properties for revision 0.
$ svnsync init http://slave2.example.com/svn-proxy-sync file://var/svn/rep
Copied properties for revision 0.
$ svnsync init http://slave3.example.com/svn-proxy-sync file://var/svn/rep
Copied properties for revision 0.

# Perform the initial replication

$ svnsync sync http://slave1.example.com/svn-proxy-sync
Transmitting file data ....
Committed revision 1.
Copied properties for revision 1.
Transmitting file data .......
Committed revision 2.
Copied properties for revision 2.
…

$ svnsync sync http://slave2.example.com/svn-proxy-sync
Transmitting file data ....
Committed revision 1.
Copied properties for revision 1.
Transmitting file data .......
Committed revision 2.
Copied properties for revision 2.
…

$ svnsync sync http://slave3.example.com/svn-proxy-sync
Transmitting file data ....
Committed revision 1.
Copied properties for revision 1.
Transmitting file data .......
Committed revision 2.
Copied properties for revision 2.
```

...

After this is done, we configure the master server's `post-commit` hook script to invoke **svnsync** on each slave server:

```
#!/bin/sh
# Post-commit script to replicate newly committed revision to slaves

svnsync sync http://slave1.example.com/svn-proxy-sync > /dev/null 2>&1
svnsync sync http://slave2.example.com/svn-proxy-sync > /dev/null 2>&1
svnsync sync http://slave3.example.com/svn-proxy-sync > /dev/null 2>&1
```

The extra bits on the end of each line aren't necessary, but they're a sneaky way to allow the sync commands to run in the background so that the Subversion client isn't left waiting forever for the commit to finish. In addition to this `post-commit` hook, you'll need a `post-revprop-change` hook as well so that when a user, say, modifies a log message, the slave servers get that change also:

```
#!/bin/sh
# Post-revprop-change script to replicate revprop-changes to slaves

REV=${2}
svnsync copy-revprops http://slave1.example.com/svn-proxy-sync ${REV} > /d
svnsync copy-revprops http://slave2.example.com/svn-proxy-sync ${REV} > /d
svnsync copy-revprops http://slave3.example.com/svn-proxy-sync ${REV} > /d
```

The only thing we've left out here is what to do about locks. Because locks are strictly enforced by the master server (the only place where commits happen), we don't technically need to do anything. Many teams don't use Subversion's locking features at all, so it may be a nonissue for you. However, if lock changes aren't replicated from master to slaves, it means that clients won't be able to query the status of locks (e.g., **`svn status -u`** will show no information about repository locks). If this bothers you, you can write `post-lock` and `post-unlock` hook scripts that run **svn lock** and **svn unlock** on each slave machine, presumably through a remote shell method such as SSH. That's left as an exercise for the reader!

6.4.5.3.3. 告诫

Your master/slave replication system should now be ready to use. A couple of words of warning are in order, however. Remember that this replication isn't entirely robust in the face of computer or network crashes. For example, if one of the automated **svnsync** commands fails to complete for some reason, the slaves will begin to fall behind. For example, your remote users will see that they've committed revision 100, but then when they run **svn update**, their local server will tell them that revision 100 doesn't yet exist! Of course, the problem will be automatically fixed the next time another commit happens and the subsequent **svnsync** is successful—the sync will replicate all waiting revisions. But still, you may want to set up some sort of out-of-band monitoring to notice synchronization failures and force **svnsync** to run when things go wrong.

我们可以为 svnserve 设置复制吗?

If you're using **svnserve** instead of Apache as your server, you can certainly configure your repository's hook scripts to invoke **svnsync** as we've shown here, thereby causing automatic replication from master to slaves. Unfortunately, at the time of this writing there is no way to make slave **svnserve** servers automatically proxy write requests back to the master server. This means your users would only be able to check out read-only working copies from the slave servers. You'd have to configure your slave servers to disallow write access completely. This might be useful for creating read-only “mirrors” of popular open source projects, but it's not a transparent proxying system.

6.4.5.4. 其它的 Apache 特性

Several of the features already provided by Apache in its role as a robust web server can be leveraged for increased functionality or security in Subversion as well. The Subversion client is able to use SSL (the Secure Sockets Layer, discussed earlier). If your Subversion client is built to support SSL, it can access your Apache server using `https://` and enjoy a high-quality encrypted network session.

同样有用的是Apache和Subversion关系的一些特性，像可以指定自定义的端口(而不是缺省的HTTP的80)或者是一个Subversion可以被访问的虚拟主机名，或者是通过HTTP代理服务器访问的能力，这些特性都是Neon所支持的，所以Subversion轻易得到这些支持。

最后，因为**mod_dav_svn**是使用一个半完成的WebDAV/DeltaV方言，所以通过第三方的DAV客户端访问也是可能的，几乎所有的现代操作系统(Win32、OS X和Linux)都有把DAV服务器影射为普通的网络“共享”的内置能力，这是一个复杂的主题；察看附录 C, *WebDAV* 和自动版本来得到更多细节。

Note that there are a number of other small tweaks one can make to **mod_dav_svn** that are too obscure to mention in this chapter. For a complete list of all `httpd.conf` directives that **mod_dav_svn** responds to, see “指示”一节.

6.5. 基于路径的授权

Apache和**svnserve**都可以给用户赋予(或拒绝)访问许可，通常是对整个版本库：一个用户可以读版本库(或不)，而且他可以写版本库(或不)。如果可能，也可以定义细粒度的访问规则。一组用户可以有版本库的一个目录的读写权限，但是没有其它的；另一个目录可以是只对一少部分用户可读。

Both servers use a common file format to describe these path-based access rules. In the case of Apache, one needs to load the **mod_authz_svn** module and then add the `AuthzSVNAccessFile` directive (within the `httpd.conf` file) pointing to your own rules file. (For a full explanation, see 第 6.4.4.2 节 “每目录访问控制”.) If you're using **svnserve**, you need to make the `authz-db` variable (within `svnserve.conf`) point to your rules file.

你真的需要基于路径的访问控制吗?

许多第一次设置Subversion的管理员会在未经太多的思考的情况下轻易选择使用路径为基础的访问控制，管理员通常知道团队的成员工作在哪个项目，所以很容易确定赋予哪些团队访问哪些目录，不能访问哪些目录。这看起来是很自然的事情，它满足了管理员紧密控制版本库访问的愿望。

注意，这个特性通常有一些看不见(和可见的)代价。可见的，需要更多的工作来确信用户对某个路径有读写权限；在一些情况下，会是非常大的性能损失。不可见的，考虑你创建的文化，大多数情况下，因为特定用户不能够在特定目录提交修改，所以社会契约不必通过技术来加强。团队有时候可以自然的互相协作；一些人会通过为他人提交不能正常工作目录的内容的方法帮助别人，你设置了一种不期望交流的障碍。你也要建立一套项目开发、新人加入等活动的规则，还有很多额外的工作。

Remember that this is a version control system! Even if somebody accidentally commits a change to something she shouldn't, it's easy to undo the change. And if a user commits to the wrong place with deliberate malice, it's a social problem anyway, and that the problem needs to be dealt with outside Subversion.

So, before you begin restricting users' access rights, ask yourself whether there's a real, honest need for this, or whether it's just something that “sounds good” to an administrator. Decide whether it's worth sacrificing some server speed, and remember that there's very little risk involved; it's bad to become dependent on technology as a crutch for social problems. [7]

作为一个思考的例子，考虑Subversion项目本身有允许某个用户可以在那个目录提交的设置，而只是通过社交方式规定。这是一个社区信任的好模型，特别是对开源项目。当然，有时候需要正统的路径为基础的访问控制；在公司中，例如，只有部分数据是敏感的，只允许以小组人可以访问。

当你的服务器知道去查找规则文件时，就是需要定义规则的时候了。

The syntax of the file is the same familiar one used by `svnserve.conf` and the runtime configuration files. Lines that start with a hash (`#`) are ignored. In its simplest form, each section names a repository and path within it, as well as the authenticated usernames are the option names within each section. The value of each option describes the user's level of access to the repository path: either `r` (read-only) or `rw` (read/write). If the user is not mentioned at all, no access is allowed.

To be more specific: the value of the section names is either of the form `[repos-name:path]` or of the form `[path]`. If you're using the `SVNParentPath` directive, it's important to specify the repository names in your sections. If you omit them, a section such as `[/some/dir]` will match the path `/some/dir` in *every* repository. If you're using the `SVNPath` directive, however, it's fine to only define paths in your sections—after all, there's only one repository.

```
[calc:/branches/calc/bug-142]
harry = rw
sally = r
```

[7]本书的共同主题!

在第一个例子里，用户harry对calc版本库中/branches/calc/bug-142具备完全的读写权利，但是用户sally只有读权利，任何其他用户禁止访问这个目录。

Of course, permissions are inherited from parent to child directory. That means we can specify a subdirectory with a different access policy for Sally:

```
[calc:/branches/calc/bug-142]
harry = rw
sally = r

# give sally write access only to the 'testing' subdir
[calc:/branches/calc/bug-142/testing]
sally = rw
```

Now Sally can write to the `testing` subdirectory of the branch, but can still only read other parts. Harry, meanwhile, continues to have complete read/write access to the whole branch.

也可以通过继承规则明确的的拒绝某人的访问，只需要设置用户名参数为空：

```
[calc:/branches/calc/bug-142]
harry = rw
sally = r

[calc:/branches/calc/bug-142/secret]
harry =
```

In this example, Harry has read/write access to the entire `bug-142` tree, but has absolutely no access at all to the `secret` subdirectory within it.

提示

The thing to remember is that the most specific path always matches first. The server tries to match the path itself, and then the parent of the path, then the parent of that, and so on. The net effect is that mentioning a specific path in the access file will always override any permissions inherited from parent directories.

缺省情况下，没有人对版本库有任何访问，这意味着如果你已经从一个空文件开始，你会希望给所有用户对版本库根目录具备读权限，你可以使用星号(*)实现，用来代表“所有用户”：

```
[/]
* = r
```

This is a common setup; notice that no repository name is mentioned in the section name. This makes all repositories world-readable to all users. Once all users have read access to the repositories, you can give explicit `rw` permission to certain users on specific subdirectories within specific repositories.

The asterisk variable (`*`) is also worth special mention because it's the *only* pattern that matches an anonymous user. If you've configured your server block to allow a mixture of anonymous and authenticated access, all users start out accessing anonymously. The server looks for a `*` value defined for the path being accessed; if it can't find one, it demands real authentication from the client.

访问文件也允许你定义一组的用户，很像Unix的`/etc/group`文件：

```
[groups]
calc-developers = harry, sally, joe
paint-developers = frank, sally, jane
everyone = harry, sally, joe, frank, sally, jane
```

组可以被赋予通用户一样的访问权限，使用“at”(`@`)前缀来加以区别：

```
[calc:/projects/calc]
@calc-developers = rw

[paint:/projects/paint]
jane = r
@paint-developers = rw
```

Another important fact is that the *first* matching rule is the one which gets applied to a user. In the prior example, even though Jane is a member of the `paint-developers` group (which has read/write access), the `jane = r` rule will be discovered and matched before the group rule, thus denying Jane write access.

组中也可以定义为包含其它的组：

```
[groups]
calc-developers = harry, sally, joe
paint-developers = frank, sally, jane
everyone = @calc-developers, @paint-developers
```

Subversion 1.5 brings another useful feature to the access file syntax: username aliases. Some authentication systems expect and carry relatively short usernames of the sorts we've been describing here—`harry`, `sally`, `joe`, and so on. But other authentication systems—such as those which use LDAP stores or SSL client certificates—may carry much more complex usernames. For example, Harry's username in an LDAP-protected system might be `CN=Harold Hacker,OU=Engineers,DC=red-bean,DC=com`. With usernames like that, the access file can become quite bloated with long or obscure usernames that are easy to mistype. Fortunately, username aliases allow you to have to type the correct complex username only once, in a statement which assigns to it a more easily digestable alias.

```
[aliases]
harry = CN=Harold Hacker,OU=Engineers,DC=red-bean,DC=com
```

```
sally = CN=Sally Swatterbug,OU=Engineers,DC=red-bean,DC=com
joe = CN=Gerald I. Joseph,OU=Engineers,DC=red-bean,DC=com
…
```

Once you've defined a set of aliases, you can refer to the users elsewhere in the access file via their aliases in all the same places you could have instead used their actual usernames. Simply prepend an ampersand to the alias to distinguish it from a regular username:

```
[groups]
calc-developers = &harry, &sally, &joe
paint-developers = &frank, &sally, &jane
everyone = @calc-developers, @paint-developers
```

You might also choose to use aliases if your users' usernames change frequently. Doing so allows you to need to update only the aliases table when these username changes occur, instead of doing global-search-and-replace operations on the whole access file.

部分可读性和检出

If you're using Apache as your Subversion server and have made certain subdirectories of your repository unreadable to certain users, you need to be aware of a possible nonoptimal behavior with **svn checkout**.

When the client requests a checkout or update over HTTP, it makes a single server request and receives a single (often large) server response. When the server receives the request, that is the *only* opportunity Apache has to demand user authentication. This has some odd side effects. For example, if a certain subdirectory of the repository is readable only by user Sally, and user Harry checks out a parent directory, his client will respond to the initial authentication challenge as Harry. As the server generates the large response, there's no way it can resend an authentication challenge when it reaches the special subdirectory; thus the subdirectory is skipped altogether, rather than asking the user to reauthenticate as Sally at the right moment. In a similar way, if the root of the repository is anonymously world-readable, the entire checkout will be done without authentication—again, skipping the unreadable directory, rather than asking for authentication partway through.

6.6. 支持多种版本库访问方法

你已经看到了一个版本库可以用多种方式访问，但是可以—或者说安全的—用几种方式同时并行的访问你的版本库吗？回答是可以，倘若你有一些深谋远虑的使用。

在任何给定的时间，这些进程会要求读或者写访问你的版本库：

- 常规的系统用户使用Subversion客户端(客户端程序本身)通过`file://`URL直接访问版本库
- 常规的系统用户连接使用SSH调用的访问版本库的**svnserve**进程(就像它们自己运行一样)；

- 一个**svnserve**进程—是一个守护进程或是通过**inetd**启动的—作为一个固定的用户运行
- 一个Apache **httpd**进程，以一个固定用户运行

The most common problem administrators run into is repository ownership and permissions. Does every process (or user) in the preceding list have the rights to read and write the repository's underlying data files? Assuming you have a Unix-like operating system, a straightforward approach might be to place every potential repository user into a new `svn` group, and make the repository wholly owned by that group. But even that's not enough, because a process may write to the database files using an unfriendly umask—one that prevents access by other users.

So the next step beyond setting up a common group for repository users is to force every repository-accessing process to use a sane umask. For users accessing the repository directly, you can make the **svn** program into a wrapper script that first runs **`umask 002`** and then runs the real **svn** client program. You can write a similar wrapper script for the **svnserve** program, and add a **`umask 002`** command to Apache's own startup script, `apachectl`. For example:

```
$ cat /usr/bin/svn

#!/bin/sh

umask 002
/usr/bin/svn-real "$@"
```

另一个在类Unix系统下常见的问题是，当版本库在使用时，BerkeleyDB有时候创建一个新的日志文件来记录它的东西，即使这个版本库是完全由**svn**组拥有，这个新创建的文件不是必须被同一个组拥有，这给你的用户造成了更多地许可问题。一个好的工作区应该设置组的SUID字节到版本库的`db`目录，这会导致所有新创建的日志文件拥有同父目录相同的组拥有者。

Once you've jumped through these hoops, your repository should be accessible by all the necessary processes. It may seem a bit messy and complicated, but the problems of having multiple users sharing write access to common files are classic ones that are not often elegantly solved.

Fortunately, most repository administrators will never *need* to have such a complex configuration. Users who wish to access repositories that live on the same machine are not limited to using `file://` access URLs—they can typically contact the Apache HTTP server or **svnserve** using `localhost` for the server name in their `http://` or `svn://` URL. And maintaining multiple server processes for your Subversion repositories is likely to be more of a headache than necessary. We recommend that you choose a single server that best meets your needs and stick with it!

svn+ssh 服务器检查列表

It can be quite tricky to get a bunch of users with existing SSH accounts to share a repository without permissions problems. If you're confused about all the things that you (as an administrator) need to do on a Unix-like system, here's a quick checklist that resummarizes some of the topics discussed in this section:

- 所有的SSH用户需要能够读写版本库，把所有的SSH用户放到同一个组里。
- 让那个组拥有整个版本库。
- 设置组的访问许可为读/写。
- Your users need to use a sane umask when accessing the repository, so make sure **svnserve** (`/usr/bin/svnserve`, or wherever it lives in `$PATH`) is actually a wrapper script that runs **`umask 002`** and executes the real **svnserve** binary.
- **svnlook**和**svnadmin**的使用类似，使用健全的umask或者使用前面提到的包裹程序。

第 7 章 定制你的 Subversion 体验

版本控制可以成为复杂的主题，和科学一样充满艺术性，为解决事情能提供了无数的方法。贯穿这本书，你已经阅读许多Subversion命令行子命令，以及可以改变运行方式的选项，在本章我们要查看一些自定义Subversion工作的方法—设置Subversion运行配置，使用外置帮助程序，Subversion与操作系统配置的地区交互等等。

7.1. 运行配置区

Subversion provides many optional behaviors that the user can control. Many of these options are of the kind that a user would wish to apply to all Subversion operations. So, rather than forcing users to remember command-line arguments for specifying these options and to use them for every operation they perform, Subversion uses configuration files, segregated into a Subversion configuration area.

The Subversion *configuration area* is a two-tiered hierarchy of option names and their values. Usually, this boils down to a special directory that contains *configuration files* (the first tier), which are just text files in standard INI format (with “sections” providing the second tier). You can easily edit these files using your favorite text editor (such as Emacs or vi), and they contain directives read by the client to determine which of several optional behaviors the user prefers.

7.1.1. 配置区布局

The first time the **svn** command-line client is executed, it creates a per-user configuration area. On Unix-like systems, this area appears as a directory named `.subversion` in the user's home directory. On Win32 systems, Subversion creates a folder named `Subversion`, typically inside the `Application Data` area of the user's profile directory (which, by the way, is usually a hidden directory). However, on this platform, the exact location differs from system to system and is dictated by the Windows Registry. [1] We will refer to the per-user configuration area using its Unix name, `.subversion`.

In addition to the per-user configuration area, Subversion also recognizes the existence of a system-wide configuration area. This gives system administrators the ability to establish defaults for all users on a given machine. Note that the system-wide configuration area alone does not dictate mandatory policy—the settings in the per-user configuration area override those in the system-wide one, and command-line arguments supplied to the **svn** program have the final word on behavior. On Unix-like platforms, the system-wide configuration area is expected to be the `/etc/subversion` directory; on Windows machines, it looks for a `Subversion` directory inside the common `Application Data` location (again, as specified by the Windows Registry). Unlike the per-user case, the **svn** program does not attempt to create the system-wide configuration area.

The per-user configuration area currently contains three files—two configuration files (`config` and `servers`), and a `README.txt` file, which describes the INI format. At the time of their creation, the files contain default values for each of the supported Subversion options, mostly commented out and grouped with textual descriptions about how the values for the key affect Subversion's behavior.

[1] `APPDATA`环境变量指向`Application Data`目录，所以你可以通过`%APPDATA%\Subversion`引用用户配置区目录。

To change a certain behavior, you need only to load the appropriate configuration file into a text editor, and to modify the desired option's value. If at any time you wish to have the default configuration settings restored, you can simply remove (or rename) your configuration directory and then run some innocuous **svn** command, such as **`svn --version`**. A new configuration directory with the default contents will be created.

用户配置区也缓存了认证信息，`auth`目录下的子目录中缓存了一些Subversion支持的各种认证方法的信息，这个目录需要相应的用户权限才可以访问。

7.1.2. 配置和 Windows 注册表

In addition to the usual INI-based configuration area, Subversion clients running on Windows platforms may also use the Windows Registry to hold the configuration data. The option names and their values are the same as in the INI files. The "file/section" hierarchy is preserved as well, though addressed in a slightly different fashion—in this schema, files and sections are just levels in the Registry key tree.

Subversion的系统配置值保存在键`HKEY_LOCAL_MACHINE\Software\Tigris.org\Subversion`下。举个例子，`global-ignores`选项位于`config`文件的`miscellany`小节，在Windows注册表中，则位于`HKEY_LOCAL_MACHINE\Software\Tigris.org\Subversion\Config\Miscellany\global-ignores`。用户配置值存放在`HKEY_CURRENT_USER\Software\Tigris.org\Subversion`下。

Registry-based configuration options are parsed *before* their file-based counterparts, so they are overridden by values found in the configuration files. In other words, Subversion looks for configuration information in the following locations on a Windows system; lower-numbered locations take precedence over higher-numbered locations:

1. 命令行选项
2. 用户INI配置文件
3. 用户注册表值
4. 系统INI配置文件
5. 系统注册表值

此外，虽然Windows注册表不支持"注释掉"这种概念，但是Subversion会忽略所有以井号(`#`)开始的字符，这允许你快速的取消一个选项而不需要删除整个注册表键，明显简化了恢复选项的过程。

The **svn** command-line client never attempts to write to the Windows Registry and will not attempt to create a default configuration area there. You can create the keys you need using the **REGEDIT** program. Alternatively, you can create a `.reg` file (such as the one in 例 7.1 "注册表条目(.reg)文件样例"), and then double-click on that file's icon in the Explorer shell, which will cause the data to be merged into your Registry.

例 **7.1.** 注册表条目**(.reg)**文件样例

```
REGEDIT4

[HKEY_LOCAL_MACHINE\Software\Tigris.org\Subversion\Servers\groups]

[HKEY_LOCAL_MACHINE\Software\Tigris.org\Subversion\Servers\global]
"#http-proxy-host"=""
"#http-proxy-port"=""
"#http-proxy-username"=""
"#http-proxy-password"=""
"#http-proxy-exceptions"=""
"#http-timeout"="0"
"#http-compression"="yes"
"#neon-debug-mask"=""
"#ssl-authority-files"=""
"#ssl-trust-default-ca"=""
"#ssl-client-cert-file"=""
"#ssl-client-cert-password"=""

[HKEY_CURRENT_USER\Software\Tigris.org\Subversion\Config\auth]
"#store-passwords"="yes"
"#store-auth-creds"="yes"

[HKEY_CURRENT_USER\Software\Tigris.org\Subversion\Config\helpers]
"#editor-cmd"="notepad"
"#diff-cmd"=""
"#diff3-cmd"=""
"#diff3-has-program-arg"=""

[HKEY_CURRENT_USER\Software\Tigris.org\Subversion\Config\tunnels]

[HKEY_CURRENT_USER\Software\Tigris.org\Subversion\Config\miscellany]
"#global-ignores"="*.o *.lo *.la #*# .*.rej *.rej .*~ *~ .#* .DS_Store"
"#log-encoding"=""
"#use-commit-times"=""
"#no-unlock"=""
"#enable-auto-props"=""

[HKEY_CURRENT_USER\Software\Tigris.org\Subversion\Config\auto-props]
```

上面例子里显示的.reg文件中，包含了一些最常用的配置选项和它们的缺省值。注意，上面的例子中不仅包含了系统设置(关于网络代理相关的选项)，也包含了用户设置(指定的编辑器程序，是否保存密码，以及其它选项)。同时要注意的是，所有选项都注释掉了，要启用其中的选项，只需删除该选项名称前面的井号(#)，然后设置相应的值就可以了。

7.1.3. 配置选项

In this section, we will discuss the specific runtime configuration options that Subversion currently supports.

7.1.3.1. 服务器

servers文件保存了Subversion关于网络层的配置选项，这个文件有两个特别的小节：groups 和global。groups小节是一个交叉引用表，其中的关键字是servers文件中其它小节的名字，值则是*globs*格式的，也就是包含通配符的字符序列，对应于接收Subversion请求的主机名。

```
[groups]
beanie-babies = *.red-bean.com
collabnet = svn.collab.net

[beanie-babies]
…

[collabnet]
…
```

当通过网络访问Subversion服务器时，客户端会设法匹配正在尝试连接的服务器名字和groups小节中的glob名称，如果发现匹配，Subversion会在servers文件中查找对应于这个glob名称的小节，并从该小节中去读取真实的网络配置设置。

如果没有能够匹配到groups中的glob名称，global小节中的选项就会发生作用。global小节中的选项与其他小节一样(当然是除了groups小节)，这些选项是：

http-proxy-exceptions
: 这里指定了一组逗号分割的列表，其内容是无须代理服务器可以直接访问的版本库主机名模式，模式语法与Unix的shell中的文件名相同，其中任何匹配的版本库主机不会通过代理访问。

http-proxy-host
: 代理服务器的详细主机名，是HTTP为基础的Subversion请求必须通过的，缺省值为空，意味着Subversion不会尝试通过代理服务器进行HTTP请求，而会尝试直接连接目标机器。

http-proxy-port
: 代理服务器的详细端口，缺省值为空。

http-proxy-username
: 代理服务器的用户名，缺省值为空。

http-proxy-password
: 代理服务器的密码，缺省为空。

http-timeout
: 等待服务器响应的时间，以秒为单位，如果你的网络速度较慢，导致Subversion的操作超时，你可以加大这个数值，缺省值是0，意思是让HTTP库Neon使用自己的缺省值。

`http-compression`
: This specifies whether Subversion should attempt to compress network requests made to DAV-ready servers. The default value is `yes` (though compression will occur only if that capability is compiled into the network layer). Set this to `no` to disable compression, such as when debugging network transmissions.

`http-library`
: Subversion provides a pair of repository access modules that understand its WebDAV network protocol. The original one, which shipped with Subversion 1.0, is `libsvn_ra_neon` (though back then it was called `libsvn_ra_dav`). Newer Subversion versions also provide `libsvn_ra_serf`, which uses a different underlying implementation and aims to support some of the newer HTTP concepts.

  At this point, `libsvn_ra_serf` is still considered experimental, though it appears to work in the common cases quite well. To encourage experimentation, Subversion provides the `http-library` runtime configuration option to allow users to specify (generally, or in a per-server-group fashion) which WebDAV access module they'd prefer to use—`neon` or `serf`.

`http-auth-types`
: This option is a semicolon-delimited list of authentication types supported by the Neon-based WebDAV repository access modules. Valid members of this list are `basic`, `digest`, and `negotiate`.

`neon-debug-mask`
: 只是一个整形的掩码，底层的HTTP库Neon用来选择产生调试的输出，缺省值是`0`，意思是关闭所有的调试输出，关于Subversion使用Neon的详细信息，见第 8 章 嵌入 *Subversion*。

`ssl-authority-files`
: 这是一个分号分割的路径和文件列表，这些文件包含了Subversion客户端在用HTTPS访问时可以接受的认证授权(或者CA)证书。

`ssl-trust-default-ca`
: 如果你希望Subversion可以自动相信OpenSSL携带的缺省的CA，可以设置为`yes`。

`ssl-client-cert-file`
: 如果一个主机(或是一些主机)需要一个SSL客户端证书，你会收到一个提示说需要证书的路径。通过设置这个路径你的Subversion客户端可以自动找到你的证书而不会打扰你。没有标准的存放位置；Subversion会从任何你指定的路径得到这个文件。

`ssl-client-cert-password`
: If your SSL client certificate file is encrypted by a passphrase, Subversion will prompt you for the passphrase whenever the certificate is used. If you find this annoying (and don't mind storing the password in the `servers` file), you can set this variable to the certificate's passphrase. You won't be prompted anymore.

7.1.3.2. 配置

其它的Subversion运行选项保存在`config`文件中，这些运行选项与网络连接无关，只是一些正在使用的选项，但是为了应对未来的扩展，也按小节划分成组。

auth小节保存了Subversion相关的认证和授权的设置，它包括：

store-passwords
: 这告诉Subversion是否缓存服务器认证要求时用户提供的密码，缺省值是yes。设置为no可以关闭在存盘的密码缓存，你可以通过**svn**的--no-auth-cache命令行参数(那些支持这个参数的子命令)来覆盖这个设置，详细信息请见第 3.11.2 节 “客户端凭证缓存”。

store-auth-creds
: This setting is the same as store-passwords, except that it enables or disables on-disk caching of *all* authentication information: usernames, passwords, server certificates, and any other types of cacheable credentials.

helpers小节控制完成Subversion任务的外部程序，正确的选项包括：

editor-cmd
: This specifies the program Subversion will use to query the user for certain types of textual metadata or when interactively resolving conflicts. See 第 7.3 节 “使用外置编辑器” for more details on using external text editors with Subversion.

diff-cmd
: 这里是比较程序的绝对路径，当Subversion生成了“diff”输出时(例如当使用**svn diff**命令)就会使用，缺省Subversion会使用一个内置的比较库—设置这个参数会强制它使用外部程序执行这个任务，此类程序的更多信息见第 7.4 节 “使用外置比较与合并工具”。

diff3-cmd
: 这指定了一个三向的比较程序，Subversion使用这个程序来合并用户和从版本库接受的修改，缺省Subversion会使用一个内置的比较库—设置这个参数会导致它会使用外部程序执行这个任务，此类程序的更多信息见第 7.4 节 “使用外置比较与合并工具”。

diff3-has-program-arg
: 如果diff3-cmd选项设置的程序接受一个--diff-program命令行参数，这个标记必须设置为true。

merge-tool-cmd
: This specifies the program that Subversion will use to perform three-way merge operations on your versioned files. See 第 7.4 节 “使用外置比较与合并工具” for more details on using such programs.

tunnels小节允许你定义一个**svnserve**和svn://客户端连接使用的管道模式，更多细节见第 6.3.4 节 “穿越 SSH 隧道”。

miscellany小节是一些没法归到别处的选项。[2]在本小节，你会找到：

global-ignores
: When running the **svn status** command, Subversion lists unversioned files and directories along with the versioned ones, annotating them with a ? character (see 第 2.4.3.1 节 “查看你的修改概况”). Sometimes it can be annoying to see uninteresting, unversioned items—for example, object files that result from a program's compilation—in this display. The global-ignores

[2]就是一个大杂烩?

option is a list of whitespace-delimited globs that describe the names of files and directories that Subversion should not display unless they are versioned. The default value is `*.o *.lo *.la #*# .*.rej *.rej .*~ *~ .#* .DS_Store`.

就像**svn status**，**svn add**和**svn import**命令也会忽略匹配这个列表的文件，你可以用单个的`--no-ignore`命令行参数来覆盖这个选项。

For information on finer-grained control of ignored items, see 第 3.4 节 “忽略未版本控制的条目”.

`enable-auto-props`
这里指示Subversion自动对新加的或者导入的文件设置属性，缺省值是`no`，可以设置为`yes`来开启自动添加属性，这个文件的`auto-props`小节会说明哪些属性会被设置到哪些文件。

`log-encoding`
这个变量设置提交日志缺省的字符集，是`--encoding`选项(见第 9.1.1 节 “svn 选项”)的永久形式，Subversion版本库保存了一些UTF-8的日志信息，并且假定你的日志信息是用操作系统的本地编码，如果你提交的信息使用别的编码方式，你一定要指定不同的编码。

`use-commit-times`
Normally your working copy files have timestamps that reflect the last time they were touched by any process, whether your own editor or some **svn** subcommand. This is generally convenient for people developing software, because build systems often look at timestamps as a way of deciding which files need to be recompiled.

在其他情形，有时候如果工作拷贝的文件时间戳反映了上一次在版本库中更改的时间会非常好，**svn export**命令会一直放置这些“上次提交的时间戳”放到它创建的目录树。通过设置这个config参数为`yes`，**svn checkout**、**svn update**、 **svn switch**和**svn revert**命令也会为它们操作的文件设置上次提交的时间戳。

`mime-types-file`
This option, new to Subversion 1.5, specifies the path of a MIME types mapping file, such as the `mime.types` file provided by the Apache HTTP Server. Subversion uses this file to assign MIME types to newly added or imported files. See 第 3.2.4 节 “自动设置属性” and 第 3.3.1 节 “文件内容类型” for more about Subversion's detection and use of file content types.

`preserved-conflict-file-exts`
The value of this option is a space-delimited list of file extensions that Subversion should preserve when generating conflict filenames. By default, the list is empty. This option is new to Subversion 1.5.

When Subversion detects conflicting file content changes, it defers resolution of those conflicts to the user. To assist in the resolution, Subversion keeps pristine copies of the various competing versions of the file in the working copy. By default, those conflict files have names constructed by appending to the original filename a custom extension such as `.mine` or `.`*`REV`* (where *`REV`* is a revision number). A mild annoyance with this naming scheme is that on operating systems where a file's extension determines the default application used to open and edit that file, appending a custom extension prevents the file from being easily opened by its native application. For example, if the file `ReleaseNotes.pdf` was conflicted, the conflict files might be named

`ReleaseNotes.pdf.mine` or `ReleaseNotes.pdf.r4231`. While your system might be configured to use Adobe's Acrobat Reader to open files whose extensions are `.pdf`, there probably isn't an application configured on your system to open all files whose extensions are `.r4231`.

You can fix this annoyance by using this configuration option, though. For files with one of the specified extensions, Subversion will append to the conflict file names the custom extension just as before, but then also reappend the file's original extension. Using the previous example, and assuming that `pdf` is one of the extensions configured in this list thereof, the conflict files generated for `ReleaseNotes.pdf` would instead be named `ReleaseNotes.pdf.mine.pdf` and `ReleaseNotes.pdf.r4231.pdf`. Because each file ends in `.pdf`, the correct default application will be used to view them.

`交互式冲突`
: This is a Boolean option that specifies whether Subversion should try to resolve conflicts interactively. If its value is `yes` (which is the default value), Subversion will prompt the user for how to handle conflicts in the manner demonstrated in 第 2.4.5 节 “解决冲突(合并别人的修改)”. Otherwise, it will simply flag the conflict and continue its operation, postponing resolution to a later time.

`no-unlock`
: This Boolean option corresponds to **svn commit**'s `--no-unlock` option, which tells Subversion not to release locks on files you've just committed. If this runtime option is set to `yes`, Subversion will never release locks automatically, leaving you to run **svn unlock** explicitly. It defaults to `no`.

The `auto-props` section controls the Subversion client's ability to automatically set properties on files when they are added or imported. It contains any number of key-value pairs in the format `PATTERN = PROPNAME=VALUE[;PROPNAME=VALUE ...]`, where `PATTERN` is a file pattern that matches one or more filenames and the rest of the line is a semicolon-delimited set of property assignments. Multiple matches on a file will result in multiple propsets for that file; however, there is no guarantee that auto-props will be applied in the order in which they are listed in the config file, so you can't have one rule “override” another. You can find several examples of auto-props usage in the `config` file. Lastly, don't forget to set `enable-auto-props` to `yes` in the `miscellany` section if you want to enable auto-props.

7.2. 本地化

Localization is the act of making programs behave in a region-specific way. When a program formats numbers or dates in a way specific to your part of the world or prints messages (or accepts input) in your native language, the program is said to be *localized*. This section describes steps Subversion has made toward localization.

7.2.1. 理解区域设置

许多现代操作系统都有一个“当前地区”的概念—也就是本地化习惯服务的国家和地区。这些习惯—通常是被一些运行配置机制选择—影响程序展现数据的方式，也有接受用户输入的方式。

在类Unix的系统，你可以运行**locale**命令来检查本地关联的运行配置的选项值：

```
$ locale
LANG=
LC_COLLATE="C"
LC_CTYPE="C"
LC_MESSAGES="C"
LC_MONETARY="C"
LC_NUMERIC="C"
LC_TIME="C"
LC_ALL="C"
$
```

The output is a list of locale-related environment variables and their current values. In this example, the variables are all set to the default `C` locale, but users can set these variables to specific country/language code combinations. For example, if one were to set the `LC_TIME` variable to `fr_CA`, programs would know to present time and date information formatted according to a French-speaking Canadian's expectations. And if one were to set the `LC_MESSAGES` variable to `zh_TW`, programs would know to present human-readable messages in Traditional Chinese. Setting the `LC_ALL` variable has the effect of changing every locale variable to the same value. The value of `LANG` is used as a default value for any locale variable that is unset. To see the list of available locales on a Unix system, run the command **`locale -a`**.

在Windows，地区配置是通过“地区和语言选项”控制面板管理的，可以从已存在的地区查看选择，甚至可以自定义(会是个很讨厌的复杂事情)许多显示格式习惯。

7.2.2. Subversion 对区域设置的使用

Subversion客户端，**svn**通过两种方式支持当前的地区配置。首先，它会注意`LC_MESSAGES`的值，然后尝试使用特定的语言打印所有的信息，例如：

```
$ export LC_MESSAGES=de_DE
$ svn help cat
cat: Gibt den Inhalt der angegebenen Dateien oder URLs aus.
Aufruf: cat ZIEL[@REV]...
…
```

This behavior works identically on both Unix and Windows systems. Note, though, that while your operating system might have support for a certain locale, the Subversion client still may not be able to speak the particular language. In order to produce localized messages, human volunteers must provide translations for each language. The translations are written using the GNU gettext package, which results in translation modules that end with the `.mo` filename extension. For example, the German translation file is named `de.mo`. These translation files are installed somewhere on your system. On Unix, they typically live in `/usr/share/locale/`, while on Windows they're often found in the `\share\locale\` folder in Subversion's installation area. Once installed, a module is named after the program for which it provides translations. For example, the `de.mo` file may ultimately end up installed as `/usr/share/locale/de/LC_MESSAGES/subversion.mo`. By browsing the installed `.mo` files, you can see which languages the Subversion client is able to speak.

The second way in which the locale is honored involves how **svn** interprets your input. The repository stores all paths, filenames, and log messages in Unicode, encoded as UTF-8. In that sense, the repository is *internationalized*—that is, the repository is ready to accept input in any human language. This means, however, that the Subversion client is responsible for sending only UTF-8 filenames and log messages into the repository. To do this, it must convert the data from the native locale into UTF-8.

For example, suppose you create a file named `caffè.txt`, and then when committing the file, you write the log message as "Adesso il caffè è più forte." Both the filename and the log message contain non-ASCII characters, but because your locale is set to `it_IT`, the Subversion client knows to interpret them as Italian. It uses an Italian character set to convert the data to UTF-8 before sending it off to the repository.

注意当版本库要求UTF-8文件名和日志信息时，它不会注意到文件的内容，Subversion会把文件内容看作字节串，没有任何客户端和服务器会尝试理解或是编码这些内容。

字符集转换错误

当使用Subversion，你或许会碰到一个字符集转化关联的错误：

```
svn: Can't convert string from native encoding to 'UTF-8':
…
svn: Can't convert string from 'UTF-8' to native encoding:
…
```

Errors such as this typically occur when the Subversion client has received a UTF-8 string from the repository, but not all of the characters in that string can be represented using the encoding of the current locale. For example, if your locale is `en_US` but a collaborator has committed a Japanese filename, you're likely to see this error when you receive the file during an **svn update**.

The solution is either to set your locale to something that *can* represent the incoming UTF-8 data, or to change the filename or log message in the repository. (And don't forget to slap your collaborator's hand—projects should decide on common languages ahead of time so that all participants are using the same locale.)

7.3. 使用外置编辑器

The most obvious way to get data into Subversion is through the addition of files to version control, committing changes to those files, and so on. But other pieces of information besides merely versioned file data live in your Subversion repository. Some of these bits of information—commit log messages, lock comments, and some property values—tend to be textual in nature and are provided explicitly by users. Most of this information can be provided to the Subversion command-line client using the `--message` (`-m`) and `--file` (`-F`) options with the appropriate subcommands.

Each of these options has its pros and cons. For example, when performing a commit, `--file` (`-F`) works well if you've already prepared a text file that holds your commit log message. If you didn't, though, you can use `--message` (`-m`) to provide a log message on the command line. Unfortunately,

it can be tricky to compose anything more than a simple one-line message on the command line. Users want more flexibility—multiline, free-form log message editing on demand.

Subversion supports this by allowing you to specify an external text editor that it will launch as necessary to give you a more powerful input mechanism for this textual metadata. There are several ways to tell Subversion which editor you'd like use. Subversion checks the following things, in the order specified, when it wants to launch such an editor:

1. 命令行选项`--editor-cmd`

2. `SVN_EDITOR` environment variable

3. `editor-cmd` runtime configuration option

4. `VISUAL` environment variable

5. `EDITOR` environment variable

6. 也有可能Subversion会有一个内置的缺省值(官方编译版本不是如此)

The value of any of these options or variables is the beginning of a command line to be executed by the shell. Subversion appends to that command line a space and the pathname of a temporary file to be edited. So, to be used with Subversion, the configured or specified editor needs to support an invocation in which its last command-line parameter is a file to be edited, and it should be able to save the file in place and return a zero exit code to indicate success.

As noted, external editors can be used to provide commit log messages to any of the committing subcommands (such as **svn commit** or **import**, **svn mkdir** or **delete** when provided a URL target, etc.), and Subversion will try to launch the editor automatically if you don't specify either of the `--message` (`-m`) or `--file` (`-F`) options. The **svn propedit** command is built almost entirely around the use of an external editor. And beginning in version 1.5, Subversion will also use the configured external text editor when the user asks it to launch an editor during interactive conflict resolution. Oddly, there doesn't appear to be a way to use external editors to interactively provide lock comments.

7.4. 使用外置比较与合并工具

The interface between Subversion and external two- and three-way differencing tools harkens back to a time when Subversion's only contextual differencing capabilities were built around invocations of the GNU diffutils toolchain, specifically the **diff** and **diff3** utilities. To get the kind of behavior Subversion needed, it called these utilities with more than a handful of options and parameters, most of which were quite specific to the utilities. Some time later, Subversion grew its own internal differencing library, and as a failover mechanism, the `--diff-cmd` and `--diff3-cmd` options were added to the Subversion command-line client so that users could more easily indicate that they preferred to use the GNU diff and diff3 utilities instead of the newfangled internal diff library. If those options were used, Subversion would simply ignore the internal diff library, and fall back to running those external programs, lengthy argument lists and all. And that's where things remain today.

It didn't take long for folks to realize that having such easy configuration mechanisms for specifying that Subversion should use the external GNU diff and diff3 utilities located at a particular place on the

system could be applied toward the use of other differencing tools, too. After all, Subversion didn't actually verify that the things it was being told to run were members of the GNU diffutils toolchain. But the only configurable aspect of using those external tools is their location on the system—not the option set, parameter order, and so on. Subversion continues to throw all those GNU utility options at your external diff tool regardless of whether that program can understand those options. And that's where things get unintuitive for most users.

使用外置比较和合并工具的关键是使用包裹脚本将Subversion的输出转化为你的脚本程序可以理解的形式，然后将这些比较工具的输出转化为你的Subversion期望的格式—GNU工具可能使用的格式，下面的小节覆盖了那些期望格式的细节。

注意

The decision on when to fire off a contextual two- or three-way diff as part of a larger Subversion operation is made entirely by Subversion and is affected by, among other things, whether the files being operated on are human-readable as determined by their `svn:mime-type` property. This means, for example, that even if you had the niftiest Microsoft Word-aware differencing or merging tool in the universe, it would never be invoked by Subversion as long as your versioned Word documents had a configured MIME type that denoted that they were not human-readable (such as `application/msword`). For more about MIME type settings, see 第 3.3.1 节 “文件内容类型”

Subversion 1.5 introduces interactive resolution of conflicts (described in 第 2.4.5 节 “解决冲突(合并别人的修改)”), and one of the options provided to users is the ability to launch a third-party merge tool. If this action is taken, Subversion will consult the `merge-tool-cmd` runtime configuration option to find the name of an external merge tool and, upon finding one, will launch that tool with the appropriate input files. This differs from the configurable three-way differencing tool in a couple of ways. First, the differencing tool is always used to handle three-way differences, whereas the merge tool is employed only when three-way difference application has detected a conflict. Second, the interface is much cleaner—your configured merge tool need only accept as command-line parameters four path specifications: the base file, the “theirs” file (which contains upstream changes), the “mine” file (which contains local modifications), and the path of the file where the final resolved contents should be stored.

7.4.1. 外置 diff

Subversion calls external diff programs with parameters suitable for the GNU diff utility, and expects only that the external program will return with a successful error code. For most alternative diff programs, only the sixth and seventh arguments—the paths of the files that represent the left and right sides of the diff, respectively—are of interest. Note that Subversion runs the diff program once per modified file covered by the Subversion operation, so if your program runs in an asynchronous fashion (or is “backgrounded”), you might have several instances of it all running simultaneously. Finally, Subversion expects that your program return an error code of 1 if your program detected differences, or 0 if it did not—any other error code is considered a fatal error. [3]

例 7.2 “diffwrap.py” and 例 7.3 “diffwrap.bat” are templates for external diff tool wrappers in the Python and Windows batch scripting languages, respectively.

[3]GNU的diff手册这样说的：“返回0意味着没有区别，1是有有区别，其它值意味着出现问题。”

例 **7.2. diffwrap.py**

```
#!/usr/bin/env python
import sys
import os

# Configure your favorite diff program here.
DIFF = "/usr/local/bin/my-diff-tool"

# Subversion provides the paths we need as the last two parameters.
LEFT  = sys.argv[-2]
RIGHT = sys.argv[-1]

# Call the diff command (change the following line to make sense for
# your diff program).
cmd = [DIFF, '--left', LEFT, '--right', RIGHT]
os.execv(cmd[0], cmd)

# Return an errorcode of 0 if no differences were detected, 1 if some were
# Any other errorcode will be treated as fatal.
```

例 **7.3. diffwrap.bat**

```
@ECHO OFF

REM Configure your favorite diff program here.
SET DIFF="C:\Program Files\Funky Stuff\My Diff Tool.exe"

REM Subversion provides the paths we need as the last two parameters.
REM These are parameters 6 and 7 (unless you use svn diff -x, in
REM which case, all bets are off).
SET LEFT=%6
SET RIGHT=%7

REM Call the diff command (change the following line to make sense for
REM your diff program).
%DIFF% --left %LEFT% --right %RIGHT%

REM Return an errorcode of 0 if no differences were detected, 1 if some we
REM Any other errorcode will be treated as fatal.
```

7.4.2. 外置 diff3

Subversion calls external merge programs with parameters suitable for the GNU diff3 utility, expecting that the external program will return with a successful error code and that the full file contents that result from the completed merge operation are printed on the standard output stream (so that Subversion can

redirect them into the appropriate version-controlled file). For most alternative merge programs, only the ninth, tenth, and eleventh arguments, the paths of the files which represent the “mine,” “older,” and “yours” inputs, respectively, are of interest. Note that because Subversion depends on the output of your merge program, your wrapper script must not exit before that output has been delivered to Subversion. When it finally does exit, it should return an error code of 0 if the merge was successful, or 1 if unresolved conflicts remain in the output—any other error code is considered a fatal error.

例 7.4 “diff3wrap.py” and 例 7.5 “diff3wrap.bat” are templates for external merge tool wrappers in the Python and Windows batch scripting languages, respectively.

例 7.4. diff3wrap.py

```
#!/usr/bin/env python
import sys
import os

# Configure your favorite diff program here.
DIFF3 = "/usr/local/bin/my-merge-tool"

# Subversion provides the paths we need as the last three parameters.
MINE  = sys.argv[-3]
OLDER = sys.argv[-2]
YOURS = sys.argv[-1]

# Call the merge command (change the following line to make sense for
# your merge program).
cmd = [DIFF3, '--older', OLDER, '--mine', MINE, '--yours', YOURS]
os.execv(cmd[0], cmd)

# After performing the merge, this script needs to print the contents
# of the merged file to stdout.  Do that in whatever way you see fit.
# Return an errorcode of 0 on successful merge, 1 if unresolved conflicts
# remain in the result.  Any other errorcode will be treated as fatal.
```

例 **7.5. diff3wrap.bat**

```
@ECHO OFF

REM Configure your favorite diff3/merge program here.
SET DIFF3="C:\Program Files\Funky Stuff\My Merge Tool.exe"

REM Subversion provides the paths we need as the last three parameters.
REM These are parameters 9, 10, and 11.  But we have access to only
REM nine parameters at a time, so we shift our nine-parameter window
REM twice to let us get to what we need.
SHIFT
SHIFT
SET MINE=%7
SET OLDER=%8
SET YOURS=%9

REM Call the merge command (change the following line to make sense for
REM your merge program).
%DIFF3% --older %OLDER% --mine %MINE% --yours %YOURS%

REM After performing the merge, this script needs to print the contents
REM of the merged file to stdout.  Do that in whatever way you see fit.
REM Return an errorcode of 0 on successful merge, 1 if unresolved conflict
REM remain in the result.  Any other errorcode will be treated as fatal.
```

7.5. 总结

Sometimes there's a single right way to do things; sometimes there are many. Subversion's developers understand that while the majority of its exact behaviors are acceptable to most of its users, there are some corners of its functionality where such a universally pleasing approach doesn't exist. In those places, Subversion offers users the opportunity to tell it how *they* want it to behave.

In this chapter, we explored Subversion's runtime configuration system and other mechanisms by which users can control those configurable behaviors. If you are a developer, though, the next chapter will take you one step further. It describes how you can further customize your Subversion experience by writing your own software against Subversion's libraries.

第 8 章 嵌入 Subversion

Subversion has a modular design: it's implemented as a collection of libraries written in C. Each library has a well-defined purpose and application programming interface (API), and that interface is available not only for Subversion itself to use, but for any software that wishes to embed or otherwise programmatically control Subversion. Additionally, Subversion's API is available not only to other C programs, but also to programs written in higher-level languages such as Python, Perl, Java, and Ruby.

本章是为那些希望编写代码或其他语言绑定与Subversion交互的人准备的。如果你围绕Subversion功能编写健壮的脚本来简化你的生活，设法开发Subversion与其他软件的复杂集成，或者只是对Subversion不同库模块提供功能感兴趣，这一章是为你准备的。然而，如果你不能预见你会以此种程度参与Subversion，你可以放心的跳过本章，略过本章不会影响你对Subversion使用的体验。

8.1. 分层的库设计

Each of Subversion's core libraries can be said to exist in one of three main layers—the Repository layer, the Repository Access (RA) layer, or the Client layer (see 图 1 “Subversion 的架构” in the Preface). We will examine these layers shortly, but first, let's briefly summarize Subversion's various libraries. For the sake of consistency, we will refer to the libraries by their extensionless Unix library names (`libsvn_fs`, `libsvn_wc`, `mod_dav_svn`, etc.).

libsvn_client
: 客户端程序的主要接口

libsvn_delta
: 目录树和文本区别程序

libsvn_diff
: 上下文区别和合并例程

libsvn_fs
: Subversion文件系统库和模块加载器

libsvn_fs_base
: Berkeley DB文件系统后端

libsvn_fs_fs
: 本地文件系统(FSFS)后端

libsvn_ra
: 版本库访问通用组件和模块装载器

libsvn_ra_local
: 本地版本库访问模块

libsvn_ra_neon
: WebDAV版本库访问模块

libsvn_ra_serf
另一个(实验性的) WebDAV 版本库访问模块

libsvn_ra_svn
一个自定义版本库访问模块

libsvn_repos
版本库接口

libsvn_subr
各色各样的有用的子程序

libsvn_wc
工作拷贝管理库

mod_authz_svn
使用WebDAV访问Subversion版本库的Apache授权模块

mod_dav_svn
影射WebDAV操作为Subversion操作的Apache模块

单词“各色各样的(miscellaneous)”只在列表中出现过一次是一个好的迹象。Subversion开发团队非常注意将功能归入合适的层和库，或许模块化设计最大的好处就是从开发者的角度看减少了复杂性。作为一个开发者，你可以很快就描画出一副“大图像”，以便于你更精确地，也相对容易地找出某一功能所在的位置。

Another benefit of modularity is the ability to replace a given module with a whole new library that implements the same API without affecting the rest of the code base. In some sense, this happens within Subversion already. The `libsvn_ra_local`, `libsvn_ra_neon`, `libsvn_ra_serf`, and `libsvn_ra_svn` libraries each implement the same interface, all working as plug-ins to `libsvn_ra`. And all four communicate with the Repository layer—`libsvn_ra_local` connects to the repository directly; the other three do so over a network. The `libsvn_fs_base` and `libsvn_fs_fs` libraries are another pair of libraries that implement the same functionality in different ways—both are plug-ins to the common `libsvn_fs` library.

The client itself also highlights the benefits of modularity in the Subversion design. Subversion's `libsvn_client` library is a one-stop shop for most of the functionality necessary for designing a working Subversion client (see 第 8.1.3 节 “客户端层”). So while the Subversion distribution provides only the **svn** command-line client program, several third-party programs provide various forms of graphical client UIs. These GUIs use the same APIs that the stock command-line client does. This type of modularity has played a large role in the proliferation of available Subversion clients and IDE integrations and, by extension, to the tremendous adoption rate of Subversion itself.

8.1.1. 版本库层

When referring to Subversion's Repository layer, we're generally talking about two basic concepts—the versioned filesystem implementation (accessed via `libsvn_fs`, and supported by its `libsvn_fs_base` and `libsvn_fs_fs` plug-ins), and the repository logic that wraps it (as

implemented in `libsvn_repos`). These libraries provide the storage and reporting mechanisms for the various revisions of your version-controlled data. This layer is connected to the Client layer via the Repository Access layer, and is, from the perspective of the Subversion user, the stuff at the “other end of the line.”

The Subversion filesystem is not a kernel-level filesystem that one would install in an operating system (such as the Linux ext2 or NTFS), but instead is a virtual filesystem. Rather than storing “files” and “directories” as real files and directories (the kind you can navigate through using your favorite shell program), it uses one of two available abstract storage backends—either a Berkeley DB database environment or a flat-file representation. (To learn more about the two repository backends, see 第 5.2.3 节 “选择数据存储格式”.) There has even been considerable interest by the development community in giving future releases of Subversion the ability to use other backend database systems, perhaps through a mechanism such as Open Database Connectivity (ODBC). In fact, Google did something similar to this before launching the Google Code Project Hosting service: they announced in mid-2006 that members of its open source team had written a new proprietary Subversion filesystem plug-in that used Google's ultra-scalable Bigtable database for its storage.

The filesystem API exported by `libsvn_fs` contains the kinds of functionality you would expect from any other filesystem API—you can create and remove files and directories, copy and move them around, modify file contents, and so on. It also has features that are not quite as common, such as the ability to add, modify, and remove metadata (“properties”) on each file or directory. Furthermore, the Subversion filesystem is a versioning filesystem, which means that as you make changes to your directory tree, Subversion remembers what your tree looked like before those changes. And before the previous changes. And the previous ones. And so on, all the way back through versioning time to (and just beyond) the moment you first started adding things to the filesystem.

所有你对目录树的修改包含在Subversion事务的上下文中，下面描述了修改文件系统的例程：

1. 开始 Subversion 的提交事务。

2. 作出修改(添加、删除、属性修改等等。)。

3. 提交事务。

一旦你提交了你的事务，你的文件系统修改就会永久的作为历史保存起来，每个这样的周期会产生一个新的树，所有的修订版本都是永远可以访问的一个不变的快照。

事务的其它信息

The notion of a Subversion transaction can become easily confused with the transaction support provided by the underlying database itself, especially given the former's close proximity to the Berkeley DB database code in `libsvn_fs_base`. Both types of transaction exist to provide atomicity and isolation. In other words, transactions give you the ability to perform a set of actions in an all-or-nothing fashion—either all the actions in the set complete with success, or they all get treated as though *none* of them ever happened—and in a way that does not interfere with other processes acting on the data.

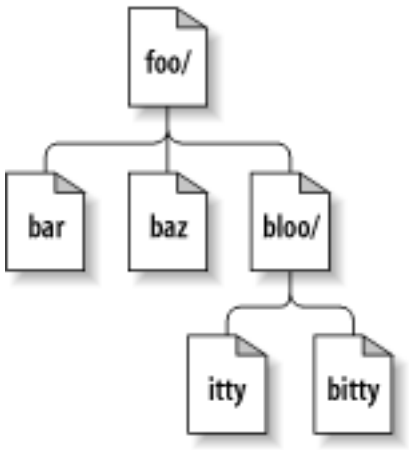
Database transactions generally encompass small operations related specifically to the modification of data in the database itself (such as changing the contents of a table row). Subversion transactions are larger in scope, encompassing higher-level operations such as making modifications to a set of files and directories that are intended to be stored as the next revision of the filesystem tree. If that isn't confusing enough, consider the fact that Subversion uses a database transaction during the creation of a Subversion transaction (so that if the creation of a Subversion transaction fails, the database will look as though we had never attempted that creation in the first place)!

很幸运的是用户的文件系统API，数据库提供的事务支持本身几乎完全从外表隐藏(也是一个完全模块化的模式所应该的)。只有当你开始研究文件系统本身的实现时，这些事情才可见(或者是开始感兴趣)。

Most of the functionality the filesystem interface provides deals with actions that occur on individual filesystem paths. That is, from outside the filesystem, the primary mechanism for describing and accessing the individual revisions of files and directories comes through the use of path strings such as `/foo/bar`, just as though you were addressing files and directories through your favorite shell program. You add new files and directories by passing their paths-to-be to the right API functions. You query for information about them by the same mechanism.

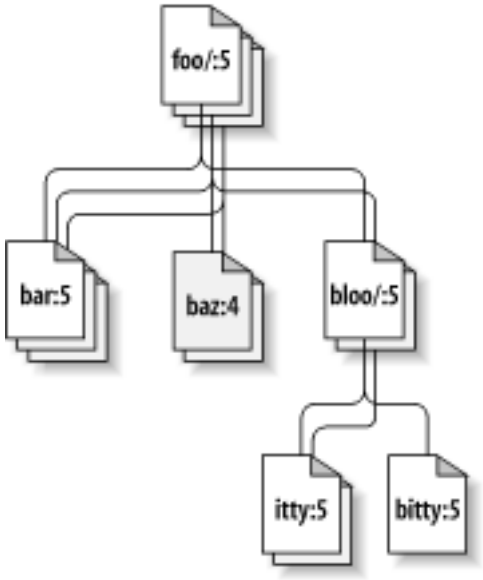
然而，不像大多数文件系统，一个单独的路径不足以在Subversion定位一个文件或目录，可以把目录树看作一个二维的系统，一个节点的兄弟代表了一种从左到右的动作，并且递减到子目录是一个向下的动作，图 8.1 “二维的文件和目录”展示了一个典型的树的形式。

图 **8.1.** 二维的文件和目录



The difference here is that the Subversion filesystem has a nifty third dimension that most filesystems do not have—Time! [1] In the filesystem interface, nearly every function that has a *path* argument also expects a *root* argument. This `svn_fs_root_t` argument describes either a revision or a Subversion transaction (which is simply a revision in the making) and provides that third dimension of context needed to understand the difference between `/foo/bar` in revision 32, and the same path as it exists in revision 98. 图 8.2 “版本时间 - 第三维！” shows revision history as an added dimension to the Subversion filesystem universe.

图 **8.2.** 版本时间 - 第三维！



像之前我们提到的，`libsvn_fs`的API感觉像是其它文件系统，只是有一个美妙的版本化能力。它设计为为所有对版本化的文件系统有兴趣的程序使用，不是巧合，Subversion本身也对这个功能很有兴趣。但是虽然文件系统API一定必须对基本的文件和目录版本化提供足够的支持，Subversion需要的更多—这是`libsvn_repos`到来的地方。

Subversion版本库库(`libsvn_repos`)建立在(逻辑上讲)`libsvn_fs`的API之上，不仅仅提供了版本化文件系统的功能，它没有包裹所有的文件系统功能—只有文件系统常规周期中的主要事件使用版本库接口包裹，如包括Subversion事务的创建和提交，修订版本属性的修改。这些特别的事件使用版本库库包裹是因为它们有一些关联的钩子。版本库钩子系统并没有与与版本化文件系统的紧密关联，所以它们存在于版本库的包裹库。

钩子机制需求是从文件系统代码的其它部分中抽象出单独的版本库库的一个原因，`libsvn_repos`的API提供了许多其他有用的工具，它们可以做到：

- 在Subversion版本库和版本库包括的文件系统的上创建、打开、销毁和执行恢复步骤。
- 描述两个文件系统树的区别。
- 关于所有(或者部分)修订版本中的文件系统中的一组文件的提交日志信息的查询
- 产生可读的文件系统“导出”—一个文件系统修订版本的完整展现。

[1]我们理解这一定会给科幻小说迷带来一个震撼，他们认为时间是第四维的，我们要为提出这样一个不同理论的断言而伤害了他们的作出道歉。

- 解析导出格式，加载导出的版本到一个不同的Subversion版本库。

伴随着Subversion的发展，版本库库会随着文件系统提供更多的功能和配置选项而不断成长。

8.1.2. 版本库访问层

If the Subversion Repository layer is at “the other end of the line,” the Repository Access (RA) layer is the line itself. Charged with marshaling data between the client libraries and the repository, this layer includes the `libsvn_ra` module loader library, the RA modules themselves (which currently includes `libsvn_ra_neon`, `libsvn_ra_local`, `libsvn_ra_serf`, and `libsvn_ra_svn`), and any additional libraries needed by one or more of those RA modules (such as the `mod_dav_svn` Apache module or `libsvn_ra_svn`'s server, **svnserve**).

Since Subversion uses URLs to identify its repository resources, the protocol portion of the URL scheme (usually `file://`, `http://`, `https://`, `svn://`, or `svn+ssh://`) is used to determine which RA module will handle the communications. Each module registers a list of the protocols it knows how to “speak” so that the RA loader can, at runtime, determine which module to use for the task at hand. You can determine which RA modules are available to the Subversion command-line client, and what protocols they claim to support, by running **`svn --version`**:

```
$ svn --version
svn, version 1.5.0 (r31699)
   compiled Jun 18 2008, 09:57:36

Copyright (C) 2000-2008 CollabNet.
Subversion is open source software, see http://subversion.tigris.org/
This product includes software developed by CollabNet (http://www.Collab.N

The following repository access (RA) modules are available:

* ra_neon : Module for accessing a repository via WebDAV protocol using Ne
  - handles 'http' scheme
  - handles 'https' scheme
* ra_svn : Module for accessing a repository using the svn network protoco
  - handles 'svn' scheme
* ra_local : Module for accessing a repository on local disk.
  - handles 'file' scheme
* ra_serf : Module for accessing a repository via WebDAV protocol using se
  - handles 'http' scheme
  - handles 'https' scheme

$
```

The public API exported by the RA layer contains functionality necessary for sending and receiving versioned data to and from the repository. And each of the available RA plug-ins is able to perform that task using a specific protocol—`libsvn_ra_dav` speaks HTTP/WebDAV (optionally using SSL encryption) with an Apache HTTP Server that is running the `mod_dav_svn` Subversion server module; `libsvn_ra_svn` speaks a custom network protocol with the **svnserve** program; and so on.

For those who wish to access a Subversion repository using still another protocol, that is precisely why the Repository Access layer is modularized! Developers can simply write a new library that implements the RA interface on one side and communicates with the repository on the other. Your new library can use existing network protocols or you can invent your own. You could use interprocess communication (IPC) calls, or—let's get crazy, shall we?—you could even implement an email-based protocol. Subversion supplies the APIs; you supply the creativity.

8.1.3. 客户端层

在客户端这一面，Subversion工作拷贝是所有动作发生的地方。大多数客户端库实现的功能是为了管理工作拷贝的目的实现的—满是文件子目录的目录是一个或多个版本库位置的可编辑的本地“影射”—从版本库访问层来回传递修改。

Subversion的工作拷贝库，`libsvn_wc`直接负责管理工作拷贝的数据，为了完成这一点，库会在工作拷贝的每个目录的特殊子目录中保存关于工作拷贝的管理性信息。这个子目录叫做`.svn`，出现在所有工作拷贝目录里，保存了各种记录了状态和用来在私有工作区工作的文件和目录。对那些熟悉CVS的用户，`.svn`子目录与CVS工作拷贝管理目录的作用类似，关于`.svn`管理区域的更多信息，见本章的第 8.2 节 “进入工作副本的管理区”。

The Subversion client library, `libsvn_client`, has the broadest responsibility; its job is to mingle the functionality of the working copy library with that of the Repository Access layer, and then to provide the highest-level API to any application that wishes to perform general revision control actions. For example, the function `svn_client_checkout()` takes a URL as an argument. It passes this URL to the RA layer and opens an authenticated session with a particular repository. It then asks the repository for a certain tree, and sends this tree into the working copy library, which then writes a full working copy to disk (`.svn` directories and all).

The client library is designed to be used by any application. While the Subversion source code includes a standard command-line client, it should be very easy to write any number of GUI clients on top of the client library. New GUIs (or any new client, really) for Subversion need not be clunky wrappers around the included command-line client—they have full access via the `libsvn_client` API to the same functionality, data, and callback mechanisms that the command-line client uses. In fact, the Subversion source code tree contains a small C program (which you can find at `tools/examples/minimal_client.c`) that exemplifies how to wield the Subversion API to create a simple client program.

直接绑定 - 关于正确性

为什么GUI程序要直接访问`libsvn_client`而不以命令行客户端的包裹运行？除了效率以外，这也关系到潜在的正确性问题。一个命令行客户端程序(如Subversion提供的)如果绑定了客户端库，需要将反馈和请求数据字节从C翻译为可读的输出，这种翻译是有损耗的，程序不能得到API所提供的所有信息，或者是得到紧凑的信息。

如果你已经包裹了这样一个命令行程序，第二个程序只能访问已经经过解释的(如我们提到的，不完全)信息，需要再次转化为它本身的展示格式。由于各层的包裹，原始数据的完整性越来越难以保证，结果很像对喜欢的录音带或录像带反复的拷贝(一个拷贝…)。

但是关于直接绑定API使用，而不是包裹程序，这是Subversion项目对其API兼容性的承诺。在小版本的变化(如从1.3到1.4)中API的不会有函数原形的改变，简单来说就是你不需要将你程序源代码升级，因为你只是升级到了一个新版本的Subversion。某些方法可能会被废弃，但依然工作，这给你了缓冲时间来最终适应新API。Subversion的命令行输出没有这种兼容性承诺，可能会在每个版本更改。

8.2. 进入工作副本的管理区

As we mentioned earlier, each directory of a Subversion working copy contains a special subdirectory called `.svn` that houses administrative data about that working copy directory. Subversion uses the information in `.svn` to keep track of things such as:

- Which repository location(s) are represented by the files and subdirectories in the working copy directory
- What revision of each of those files and directories is currently present in the working copy
- Any user-defined properties that might be attached to those files and directories
- Pristine (unedited) copies of the working copy files

Subversion工作拷贝管理区域的布局和内容主要是考虑的实现细节，不是被人来使用的。开发者被鼓励使用Subversion的API或工具来访问和处理工作拷贝数据，反对直接读写操作组成工作拷贝管理区域的文件。工作拷贝中管理数据采用的文件格式会不断改变—只是公共API成功的隐藏了这种改变。在本小节，我们将会探讨一些实现细节来安抚你们的焦虑。

8.2.1. 条目文件

Perhaps the single most important file in the `.svn` directory is the `entries` file. It contains the bulk of the administrative information about the versioned items in a working copy directory. This one file tracks the repository URLs, pristine revision, file checksums, pristine text and property timestamps, scheduling and conflict state information, last-known commit information (author, revision, timestamp), local copy history—practically everything that a Subversion client is interested in knowing about a versioned (or to-be-versioned) resource!

熟悉CVS管理目录的人可能会发现，Subversion的`.svn/entries`实现了CVS的`CVS/Entries`、`CVS/Root`和`CVS/Repository`的功能。

.svn/entries的格式曾经多次修改，最初是XML文件，现在使用自定义的—尽管依然是可读的文件格式。早期的Subversion需要频繁调试文件内容，所以选择了XML这种格式，随着Subversion的成熟，频繁调试的需求消失了，而产生了用户对性能的要求。当然，Subversion的工作拷贝库可以从一种格式自动升级到另一种格式—按照老格式读取，然后按照新格式写—避免了重新检出工作拷贝，但是也造成了不同版本Subversion程序访问同一份工作拷贝的复杂情形。

8.2.2. 原始副本和属性文件

As mentioned before, the `.svn` directory also holds the pristine “text-base” versions of files. You can find those in `.svn/text-base`. The benefits of these pristine copies are multiple—network-free checks for local modifications and difference reporting, network-free reversion of modified or missing files, more efficient transmission of changes to the server—but they come at the cost of having each versioned file stored at least twice on disk. These days, this seems to be a negligible penalty for most files. However, the situation gets uglier as the size of your versioned files grows. Some attention is being given to making the presence of the “text-base” an option. Ironically, though, it is as your versioned files' sizes get larger that the existence of the “text-base” becomes more crucial—who wants to transmit a huge file across a network just because she wants to commit a tiny change to it?

Similar in purpose to the “text-base” files are the property files and their pristine “prop-base” copies, located in `.svn/props` and `.svn/prop-base`, respectively. Since directories can have properties too, there are also `.svn/dir-props` and `.svn/dir-prop-base` files.

8.3. 使用 API

Developing applications against the Subversion library APIs is fairly straightforward. Subversion is primarily a set of C libraries, with header (`.h`) files that live in the `subversion/include` directory of the source tree. These headers are copied into your system locations (e.g., `/usr/local/include`) when you build and install Subversion itself from source. These headers represent the entirety of the functions and types meant to be accessible by users of the Subversion libraries. The Subversion developer community is meticulous about ensuring that the public API is well documented—refer directly to the header files for that documentation.

When examining the public header files, the first thing you might notice is that Subversion's datatypes and functions are namespace-protected. That is, every public Subversion symbol name begins with `svn_`, followed by a short code for the library in which the symbol is defined (such as `wc`, `client`, `fs`, etc.), followed by a single underscore (_), and then the rest of the symbol name. Semipublic functions (used among source files of a given library but not by code outside that library, and found inside the library directories themselves) differ from this naming scheme in that instead of a single underscore after the library code, they use a double underscore (__). Functions that are private to a given source file have no special prefixing and are declared `static`. Of course, a compiler isn't interested in these naming conventions, but they help to clarify the scope of a given function or datatype.

Another good source of information about programming against the Subversion APIs is the project's own hacking guidelines, which you can find at `http://subversion.tigris.org/hacking.html`. This document contains useful information, which, while aimed at developers and

would-be developers of Subversion itself, is equally applicable to folks developing against Subversion as a set of third-party libraries. [2]

8.3.1. Apache 可移植运行库

Along with Subversion's own datatypes, you will see many references to datatypes that begin with `apr_`—symbols from the Apache Portable Runtime (APR) library. APR is Apache's portability library, originally carved out of its server code as an attempt to separate the OS-specific bits from the OS-independent portions of the code. The result was a library that provides a generic API for performing operations that differ mildly—or wildly—from OS to OS. While the Apache HTTP Server was obviously the first user of the APR library, the Subversion developers immediately recognized the value of using APR as well. This means that there is practically no OS-specific code in Subversion itself. Also, it means that the Subversion client compiles and runs anywhere that the Apache HTTP Server does. Currently, this list includes all flavors of Unix, Win32, BeOS, OS/2, and Mac OS X.

In addition to providing consistent implementations of system calls that differ across operating systems, [3] APR gives Subversion immediate access to many custom datatypes, such as dynamic arrays and hash tables. Subversion uses these types extensively. But perhaps the most pervasive APR datatype, found in nearly every Subversion API prototype, is the `apr_pool_t`—the APR memory pool. Subversion uses pools internally for all its memory allocation needs (unless an external library requires a different memory management mechanism for data passed through its API), [4] and while a person coding against the Subversion APIs is not required to do the same, she *is* required to provide pools to the API functions that need them. This means that users of the Subversion API must also link against APR, must call `apr_initialize()` to initialize the APR subsystem, and then must create and manage pools for use with Subversion API calls, typically by using `svn_pool_create()`, `svn_pool_clear()`, and `svn_pool_destroy()`.

使用内存池编程

几乎每一个使用过C语言的开发者曾经感叹令人畏缩的内存管理，分配足够的内存，并且追踪内存的分配，在不需要时释放内存—这个任务会非常复杂。当然，如果没有正确地做到这一点会导致程序毁掉自己，或者更加严重一点，把电脑搞瘫。

另一方面高级语言使开发者完全摆脱了内存管理，
<placeholder-1></placeholder-1>
Java和Python之类的语言使用垃圾收集原理，在需要的时候分配对象内存，在不使用时进行清理。

APR提供了一种叫做池基础的中等的内存管理方法，允许开发者以一种低分辨率的方式控制内存—每块(或池“pool”)的内存，而不是每个对象。不是使用`malloc()`和其他按照对象分配内存的方式，你要求APR从内存创建一段内存池，当你结束使用在池中创建的对象，你销毁池，可以有效地取消其中的对象消耗的内存。通过池，你不需要跟踪每个对象的内存释放，你的程序只需要跟踪这些对象，将对象分配到池中，而池的生命周期(池的创建和删除之间的时间)满足所有对象的需要。

[2]当然，Subversion使用Subversion的API。
[3]Subversion使用尽可能多ANSI系统调用和数据类型。
[4]Neon和Berkeley DB就是这种库的例子。

8.3.2. URL 和路径需求

With remote version control operation as the whole point of Subversion's existence, it makes sense that some attention has been paid to internationalization (i18n) support. After all, while “remote” might mean “across the office,” it could just as well mean “across the globe.” To facilitate this, all of Subversion's public interfaces that accept path arguments expect those paths to be canonicalized—which is most easily accomplished by passing them through the `svn_path_canonicalize()` function—and encoded in UTF-8. This means, for example, that any new client binary that drives the `libsvn_client` interface needs to first convert paths from the locale-specific encoding to UTF-8 before passing those paths to the Subversion libraries, and then reconvert any resultant output paths from Subversion back into the locale's encoding before using those paths for non-Subversion purposes. Fortunately, Subversion provides a suite of functions (see `subversion/include/svn_utf.h`) that any program can use to do these conversions.

同样，Subversion的API需要所有的URL参数是正确的URI编码，所以，我们不会传递`file:///home/username/My File.txt`作为`My File.txt`的URL，而要传递`file:///home/username/My%20File.txt`。再次，Subversion提供了一些你可以使用的助手方法—`svn_path_uri_encode()`和`svn_path_uri_decode()`，分别用来URI的编码和解码。

8.3.3. 使用 C 和 C++ 以外的语言

If you are interested in using the Subversion libraries in conjunction with something other than a C program—say, a Python or Perl script—Subversion has some support for this via the Simplified Wrapper and Interface Generator (SWIG). The SWIG bindings for Subversion are located in `subversion/bindings/swig`. They are still maturing, but they are usable. These bindings allow you to call Subversion API functions indirectly, using wrappers that translate the datatypes native to your scripting language into the datatypes needed by Subversion's C libraries.

Significant efforts have been made toward creating functional SWIG-generated bindings for Python, Perl, and Ruby. To some extent, the work done preparing the SWIG interface files for these languages is reusable in efforts to generate bindings for other languages supported by SWIG (which include versions of C#, Guile, Java, MzScheme, OCaml, PHP, and Tcl, among others). However, some extra programming is required to compensate for complex APIs that SWIG needs some help translating between languages. For more information on SWIG itself, see the project's web site at `http://www.swig.org/`.

Subversion也有Java的语言绑定，javahl绑定(位于Subversion源目录树的`subversion/bindings/java`)不是基于SWIG的，而是javah和手写JNI的混合，JavaHL几乎覆盖Subversion客户端的API，目标是作为Java基础的Subversion客户端和集成IDE的实现。

Subversion的语言绑定缺乏Subversion核心模块的关注，但是通常可以作为一个产品信赖。大量脚本、应用、Subversion的GUI客户端和其他第三方工具现在已经成功地运用了Subversion语言绑定来完成Subversion的集成。

It's worth noting here that there are other options for interfacing with Subversion using other languages: alternative bindings for Subversion that aren't provided by the Subversion development community at all. You can find links to these alternative bindings on the Subversion project's links page (at `http://`

subversion.tigris.org/links.html), but there are a couple of popular ones we feel are especially noteworthy. First, Barry Scott's PySVN bindings (http://pysvn.tigris.org/) are a popular option for binding with Python. PySVN boasts of a more Pythonic interface than the more C-like APIs provided by Subversion's own Python bindings. And if you're looking for a pure Java implementation of Subversion, check out SVNKit (http://svnkit.com/), which is Subversion rewritten from the ground up in Java.

SVNKit 与 javahl

In 2005, a small company called TMate announced the 1.0.0 release of JavaSVN—a pure Java implementation of Subversion. Since then, the project has been renamed to SVNKit (available at http://svnkit.com/) and has seen great success as a provider of Subversion functionality to various Subversion clients, IDE integrations, and other third-party tools.

The SVNKit library is interesting in that, unlike the javahl library, it is not merely a wrapper around the official Subversion core libraries. In fact, it shares no code with Subversion at all. But while it is easy to confuse SVNKit with javahl, and easier still to not even realize which of these libraries you are using, folks should be aware that SVNKit differs from javahl in some significant ways. First, SVNKit is not developed as open source software and seems to have at any given time only a few developers working on it. Also, SVNKit's license is more restrictive than that of Subversion. Finally, by aiming to be a pure Java Subversion library, SVNKit is limited in which portions of Subversion can be reasonably cloned while still keeping up with Subversion's releases. This has already happened once—SVNKit cannot access BDB-backed Subversion repositories via the file:// protocol because there's no pure Java implementation of Berkeley DB that is file-format-compatible with the native implementation of that library.

That said, SVNKit has a well-established track record of reliability. And a pure Java solution is much more robust in the face of programming errors—a bug in SVNKit might raise a catchable Java Exception, but a bug in the Subversion core libraries as accessed via javahl can bring down your entire Java Runtime Environment. So, weigh the costs when choosing a Java-based Subversion implementation.

8.3.4. 代码样例

例 8.1 “使用版本库层”包含了一段C代码(C编写)描述了我们讨论的概念，它使用了版本库和文件系统接口(可以通过方法名svn_repos_和svn_fs_分辨)创建了一个添加目录的修订版本。你可以看到APR库的使用，为了内存分配而传递，这些代码也揭开了一些关于Subversion错误处理的晦涩事实—所有的Subversion错误必须需要明确的处理以防止内存泄露(在某些情况下，应用失败)。

例 8.1. 使用版本库层

```
  INT_ERR(svn_fs_youngest_rev(&youngest_rev, fs, pool));

  /* Begin a new transaction that is based on YOUNGEST_REV.  We are
   * less likely to have our commit rejected as conflicting if we
   * always try to make our changes against a copy of the latest snapshot
   * of the filesystem tree.
   */
  INT_ERR(svn_repos_fs_begin_txn_for_commit2(&txn, repos, youngest_rev,
                                             apr_hash_make(pool), pool));

  /* Now that we have started a new Subversion transaction, get a root
   * object that represents that transaction.
   */
  INT_ERR(svn_fs_txn_root(&txn_root, txn, pool));

  /* Create our new directory under the transaction root, at the path
   * NEW_DIRECTORY.
   */
  INT_ERR(svn_fs_make_dir(txn_root, new_directory, pool));

  /* Commit the transaction, creating a new revision of the filesystem
   * which includes our added directory path.
   */
  err = svn_repos_fs_commit_txn(&conflict_str, repos,
                                &youngest_rev, txn, pool);
  if (! err)
    {
      /* No error?  Excellent!  Print a brief report of our success.
       */
      printf("Directory '%s' was successfully added as new revision "
             "'%ld'.\n", new_directory, youngest_rev);
    }
  else if (err->apr_err == SVN_ERR_FS_CONFLICT)
    {
      /* Uh-oh.  Our commit failed as the result of a conflict
       * (someone else seems to have made changes to the same area
       * of the filesystem that we tried to modify).  Print an error
       * message.
       */
      printf("A conflict occurred at path '%s' while attempting "
             "to add directory '%s' to the repository at '%s'.\n",
             conflict_str, new_directory, repos_path);
    }
  else
    {
      /* Some other error has occurred.  Print an error message.
       */
      printf("An error occurred while attempting to add directory '%s' "
             "to the repository at '%s'.\n",
             new_directory, repos_path);
    }

  INT_ERR(err);
}
```

Note that in 例 8.1 “使用版本库层”, the code could just as easily have committed the transaction using `svn_fs_commit_txn()`. But the filesystem API knows nothing about the repository library's hook mechanism. If you want your Subversion repository to automatically perform some set of non-Subversion tasks every time you commit a transaction (e.g., sending an email that describes all the changes made in that transaction to your developer mailing list), you need to use the `libsvn_repos`-wrapped version of that function, which adds the hook triggering functionality —in this case, `svn_repos_fs_commit_txn()`. (For more information regarding Subversion's repository hooks, see 第 5.3.2 节 “实现版本库钩子”.)

现在我们转换一下语言，例 8.2 “使用 Python 处理版本库层”是使用Subversion SWIG的Python绑定实现了通过遍历版本库获取最新版本库修订版本，并且打印遍历访问路径的功能。

例 8.2. 使用 Python 处理版本库层

```
    "Recursively crawl DIRECTORY under ROOT in the filesystem, and retu
    a list of all the paths at or below DIRECTORY."""

    # Print the name of this directory.
    print directory + "/"

    # Get the directory entries for DIRECTORY.
    entries = svn.fs.svn_fs_dir_entries(root, directory)

    # Loop over the entries.
    names = entries.keys()
    for name in names:
        # Calculate the entry's full path.
        full_path = directory + '/' + name

        # If the entry is a directory, recurse.  The recursion will return
        # a list with the entry and all its children, which we will add to
        # our running list of paths.
        if svn.fs.svn_fs_is_dir(root, full_path):
            crawl_filesystem_dir(root, full_path)
        else:
            # Else it's a file, so print its path here.
            print full_path

def crawl_youngest(repos_path):
    """Open the repository at REPOS_PATH, and recursively crawl its
    youngest revision."""

    # Open the repository at REPOS_PATH, and get a reference to its
    # versioning filesystem.
    repos_obj = svn.repos.svn_repos_open(repos_path)
    fs_obj = svn.repos.svn_repos_fs(repos_obj)

    # Query the current youngest revision.
    youngest_rev = svn.fs.svn_fs_youngest_rev(fs_obj)

    # Open a root object representing the youngest (HEAD) revision.
    root_obj = svn.fs.svn_fs_revision_root(fs_obj, youngest_rev)

    # Do the recursive crawl.
    crawl_filesystem_dir(root_obj, "")

if __name__ == "__main__":
    # Check for sane usage.
    if len(sys.argv) != 2:
        sys.stderr.write("Usage: %s REPOS_PATH\n"
                         % (os.path.basename(sys.argv[0])))
        sys.exit(1)

    # Canonicalize the repository path.
    repos_path = svn.core.svn_path_canonicalize(sys.argv[1])

    # Do the real work.
    crawl_youngest(repos_path)
```

同样的C程序需要处理APR内存池系统，但是Python自己处理内存，Subversion的Python绑定也遵循这种习惯。在C语言中，为表示路径和条目的hash需要处理自定义的数据类型(例如APR提供的库)，但是Python有hash(叫做“dictionaries”)，并且是内置数据类型，而且还提供了一系列操作这些类型的函数，所以SWIG(通过Subversion的语言绑定层的自定义帮助)要小心的将这些自定义数据类型映射到目标语言的数据类型，这为目标语言的用户提供了一个更加直观的接口。

The Subversion Python bindings can be used for working copy operations, too. In the previous section of this chapter, we mentioned the `libsvn_client` interface and how it exists for the sole purpose of simplifying the process of writing a Subversion client. 例 8.3 “一个 Python 状态爬虫” is a brief example of how that library can be accessed via the SWIG Python bindings to re-create a scaled-down version of the **svn status** command.

例 **8.3.** 一个 **Python** 状态爬虫

```
def do_status(wc_path, verbose):
    # Build a client context baton.
    ctx = svn.client.svn_client_ctx_t()
    def _status_callback(path, status):
        """A callback function for svn_client_status."""
        # Print the path, minus the bit that overlaps with the root of
        # the status crawl
        text_status = generate_status_code(status.text_status)
        prop_status = generate_status_code(status.prop_status)
        print '%s%s  %s' % (text_status, prop_status, path)

    # Do the status crawl, using _status_callback() as our callback functi
    revision = svn.core.svn_opt_revision_t()
    revision.type = svn.core.svn_opt_revision_head
    svn.client.svn_client_status2(wc_path, revision, _status_callback,
                                  svn.core.svn_depth_infinity, verbose,
                                  0, 0, 1, ctx)

def usage_and_exit(errorcode):
    """Print usage message, and exit with ERRORCODE."""
    stream = errorcode and sys.stderr or sys.stdout
    stream.write("""Usage: %s OPTIONS WC-PATH
Options:
  --help, -h    : Show this usage message
  --verbose, -v : Show all statuses, even uninteresting ones
""" % (os.path.basename(sys.argv[0])))
    sys.exit(errorcode)

if __name__ == '__main__':
    # Parse command-line options.
    try:
        opts, args = getopt.getopt(sys.argv[1:], "hv", ["help", "verbose"]
    except getopt.GetoptError:
        usage_and_exit(1)
    verbose = 0
    for opt, arg in opts:
        if opt in ("-h", "--help"):
            usage_and_exit(0)
        if opt in ("-v", "--verbose"):
            verbose = 1
    if len(args) != 1:
        usage_and_exit(2)

    # Canonicalize the repository path.
    wc_path = svn.core.svn_path_canonicalize(args[0])

    # Do the real work.
    try:
        do_status(wc_path, verbose)
    except svn.core.SubversionException, e:
        sys.stderr.write("Error (%d): %s\n" % (e.apr_err, e.message))
        sys.exit(1)
```

As was the case in 例 8.2 “使用 Python 处理版本库层”, this program is pool-free and uses, for the most part, normal Python datatypes. The call to `svn_client_ctx_t()` is deceiving because the public Subversion API has no such function—this just happens to be a case where SWIG's automatic language generation bleeds through a little bit (the function is a sort of factory function for Python's version of the corresponding complex C structure). Also note that the path passed to this program (like the last one) gets run through `svn_path_canonicalize()`, because to *not* do so runs the risk of triggering the underlying Subversion C library's assertions about such things, which translates into rather immediate and unceremonious program abortion.

8.4. 总结

One of Subversion's greatest features isn't something you get from running its command-line client or other tools. It's the fact that Subversion was designed modularly and provides a stable, public API so that others—like yourself, perhaps—can write custom software that drives Subversion's core logic.

In this chapter, we took a closer look at Subversion's architecture, examining its logical layers and describing that public API, the very same API that Subversion's own layers use to communicate with each other. Many developers have found interesting uses for the Subversion API, from simple repository hook scripts, to integrations between Subversion and some other application, to completely different version control systems. What unique itch will *you* scratch with it?

第 9 章 Subversion 完全参考

本章是使用Subversion的一个完全手册，包括了命令行客户端(**svn**)和它的所有子命令，也有版本库管理程序(**svnadmin**和**svnlook**)和它们各自的子命令。

9.1. Subversion 命令行客户端：svn

To use the command-line client, type **svn**, the subcommand you wish to use, [1] and any options or targets that you wish to operate on—the subcommand and the options need not appear in a specific order. For example, all of the following are valid ways to use **svn status**:

```
$ svn -v status
$ svn status -v
$ svn status -v myfile
```

你可以在第 2 章 基本使用发现更多使用客户端命令的例子，以及第 3.2 节 “属性”中的管理属性的命令。

9.1.1. svn 选项

While Subversion has different options for its subcommands, all options exist in a single namespace—that is, each option is guaranteed to mean the same thing regardless of the subcommand you use it with. For example, `--verbose` (`-v`) always means “verbose output,” regardless of the subcommand you use it with.

The **svn** command-line client usually exits quickly with an error if you pass it an option which does not apply to the specified subcommand. But as of Subversion 1.5, several of the options which apply to all—or nearly all—of the subcommands have been deemed acceptable by all subcommands, even if they have no effect on some of them. They appear grouped together in the command-line client's usage messages as global options. This was done to assist folks who write scripts which wrap the command-line client. These global options are as follows:

`--config-dir` *DIR*
: 指导Subversion从指定目录而不是默认位置(用户主目录的`.subversion`)读取配置信息。

`--no-auth-cache`
: Prevents caching of authentication information (e.g., username and password) in the Subversion runtime configuration directories.

`--non-interactive`
: Disables all interactive prompting. Some examples of interactive prompting include requests for authentication credentials and conflict resolution decisions. This is useful if you're running Subversion inside an automated script and it's more appropriate to have Subversion fail than to prompt for more information.

[1]是的，使用`--version`选项不需要子命令，几分钟后我们会到达那个部分。

`--password` *PASSWD*
Specifies the password to use when authenticating against a Subversion server. If not provided, or if incorrect, Subversion will prompt you for this information as needed.

`--username` *NAME*
Specifies the username to use when authenticating against a Subversion server. If not provided, or if incorrect, Subversion will prompt you for this information as needed.

The rest of the options apply and are accepted by only a subset of the subcommand. They are as follows:

`--accept` *ACTION*
Specifies an action for automatic conflict resolution. Possible actions are `postpone`, `base`, `mine-full`, `theirs-full`, `edit`, and `launch`.

`--auto-props`
开启auto-props，覆盖config文件中的enable-auto-props指示。

`--change` (`-c`) *ARG*
Used as a means to refer to a specific “change” (a.k.a. a revision). This option is syntactic sugar for “-r ARG-1:ARG”.

`--changelist` *ARG*
Instructs Subversion to operate only on members of the changelist named *ARG*. You can use this option multiple times to specify sets of changelists.

`--cl` *ARG*
An alias for the `--changelist` option.

`--depth` *ARG*
Instructs Subversion to limit the scope of an operation to a particular tree depth. *ARG* is one of `empty`, `files`, `immediates`, or `infinity`.

`--diff-cmd` *CMD*
指定用来表示文件区别的外部程序，当**svn diff**调用时，会使用Subversion的内置区别引擎，默认会提供统一区别输出，如果你希望使用一个外置区别程序，使用`--diff-cmd`。你可以通过`--extensions`(本小节后面有更多介绍)把选项传递到区别程序。

`--diff3-cmd` *CMD*
指定一个外置程序用来合并文件。

`--dry-run`
检验运行一个命令的效果，但没有实际的修改—可以用在磁盘和版本库。

`--editor-cmd` *CMD*
指定一个外部程序来编辑日志信息或是属性值。如何设定缺省编辑器见第 7.1.3.2 节 “配置”的editor-cmd小节。

`--encoding` *ENC*
告诉Subversion你的提交日志信息是通过提供的字符集编码的，缺省时是你的操作系统的本地编码，如果你的提交信息使用其它编码，你一定要指定这个值。

`--extensions` (`-x`) *ARGS*
Specifies an argument or arguments that Subversion should pass to an external diff command. This option is valid only when used with the **svn diff** or **svn merge** commands, with the `--diff-cmd` option. If you wish to pass multiple arguments, you must enclose all of them in quotes (e.g., **svn diff --diff-cmd /usr/bin/diff -x "-b -E"**).

`--file` (`-F`) *FILENAME*
为特定子命令使用命名文件的的内容，尽管不同的子命令对这些内容做不同的事情。例如，**svn commit**使用内容作为提交日志，而**svn propset**使用它作为属性值。

`--force`
Forces a particular command or operation to run. Subversion will prevent you from performing some operations in normal usage, but you can pass the force option to tell Subversion “I know what I'm doing as well as the possible repercussions of doing it, so let me at 'em.” This option is the programmatic equivalent of doing your own electrical work with the power on—if you don't know what you're doing, you're likely to get a nasty shock.

`--force-log`
Forces a suspicious parameter passed to the `--message` (`-m`) or `--file` (`-F`) option to be accepted as valid. By default, Subversion will produce an error if parameters to these options look like they might instead be targets of the subcommand. For example, if you pass a versioned file's path to the `--file` (`-F`) option, Subversion will assume you've made a mistake, that the path was instead intended as the target of the operation, and that you simply failed to provide some other—unversioned—file as the source of your log message. To assert your intent and override these types of errors, pass the `--force-log` option to subcommands that accept log messages.

`--help` (`-h`或`-?`)
如果同一个或多个子命令一起使用，会显示每个子命令内置的帮助文本，如果单独使用，它会显示常规的客户端帮助文本。

`--ignore-ancestry`
告诉Subversion在计算区别(只依赖于路径内容)时忽略祖先。

`--ignore-externals`
Tells Subversion to ignore externals definitions and the external working copies managed by them.

`--incremental`
打印适合串联的输出格式。

`--keep-changelists`
Tells Subversion not to delete changelists after committing.

`--keep-local`
Keeps the local copy of a file or directory (used with the **svn delete** command).

`--limit` (`-l`) *NUM*
Shows only the first *NUM* log messages.

`--message` (`-m`) *MESSAGE*
表示你会在命令行中指定日志信息，紧跟这个开关，例如：

```
$ svn commit -m "They don't make Sunday."
```

`--new` *ARG*
使用*ARG*作为新的目标(结合**svn diff**使用)。

`--no-auto-props`
关闭auto-props，覆盖`config`文件中的`enable-auto-props`指示。

`--no-diff-deleted`
防止Subversion打印删除文件的区别信息，缺省的行为方式是当你删除了一个文件后运行**svn diff**打印的区别与删除文件所有的内容得到的结果一样。

`--no-ignore`
在状态列表中显示`global-ignores`配置选项或者是`svn:ignore`属性忽略的文件。见第 7.1.3.2 节 “配置”和第 3.4 节 “忽略未版本控制的条目”查看详情。

`--no-unlock`
Tells Subversion not to automatically unlock files (the default commit behavior is to unlock all files listed as part of the commit). See 第 3.7 节 “锁定” for more information.

`--non-recursive` (`-N`)
废弃的。防止子命令迭代到子目录，大多数子命令缺省是迭代的，但是一些子命令—通常是那些潜在的删除或者是取消本地修改的命令—千万不要。

`--notice-ancestry`
Pays attention to ancestry when calculating differences.

`--old` *ARG*
使用*ARG*作为旧的目标(结合**svn diff**使用)。

`--parents`
Creates and adds nonexistent or nonversioned parent subdirectories to the working copy or repository as part of an operation. This is useful for automatically creating multiple subdirectories where none currently exist. If performed on a URL, all the directories will be created in a single commit.

`--quiet` (`-q`)
请求客户端在执行操作时只显示重要信息。

`--record-only`
Marks revisions as merged (for use with `--revision`).

`--recursive` (`-R`)
Makes a subcommand recurse into subdirectories. Most subcommands recurse by default.

`--reintegrate`
Used with the **svn merge** subcommand, merges all of the source URL's changes into the working copy. See 第 4.3.2 节 “保持分支同步” for details.

`--relocate` *目的路径 [PATH...]*
svn switch子命令中使用，用来修改你的工作拷贝所引用的版本库位置。当版本库的位置修改了，而你有一个工作拷贝，希望继续使用时非常有用。见**svn switch**的例子。

`--remove` *ARG*
Disassociates *ARG* from a changelist

`--revision` (`-r`) *REV*
指出你将为特定操作提供一个修订版本(或修订版本的范围)，你可以提供修订版本号，修订版本关键字或日期(在华括号中)作为修订版本开关的参数。如果你希望提供一个修订版本范围，你可以提供用冒号隔开的两个修订版本，举个例子：

```
$ svn log -r 1729
$ svn log -r 1729:HEAD
$ svn log -r 1729:1744
$ svn log -r {2001-12-04}:{2002-02-17}
$ svn log -r 1729:{2002-02-17}
```

见第 3.1.1 节 “修订版本关键字”查看更多信息。

`--revprop`
操作针对修订版本属性，而不是Subversion文件或目录的属性。这个选项需要你传递`--revision`(`-r`)参数。

`--set-depth` *ARG*
Sets the sticky depth on a directory in a working copy to one of `empty`, `files`, `immediates`, or `infinity`.

`--show-revs` *ARG*
Used to make **svn mergeinfo** display either `merged` or `eligible` revisions.

`--show-updates` (`-u`)
Causes the client to display information about which files in your working copy are out of date. This doesn't actually update any of your files—it just shows you which files will be updated if you then use **svn update**.

`--stop-on-copy`
导致Subversion子命令在传递历史时会在版本化资源拷贝时停止收集历史信息—也就是历史中资源从另一个位置拷贝过来的位置—如果遇到。

`--strict`
导致Subversion使用严格的语法，就是明确使用特定而不是含糊的子命令(也就是，**svn propget**)。

`--targets` *FILENAME*
告诉Subversion从你提供的文件中得到希望操作的文件列表，而不是在命令行列出所有的文件。

`--use-merge-history` (`-g`)
Uses or displays additional information from merge history.

`--verbose` (`-v`)
请求客户端在运行子命令打印尽量多的信息，会导致Subversion打印额外的字段，每个文件的细节信息或者是关于动作的附加信息。

`--version`
Prints the client version info. This information includes not only the version number of the client, but also a listing of all repository access modules that the client can use to access a Subversion repository. With `--quiet` (`-q`) it prints only the version number in a compact form.

`--with-all-revprops`
Used with the `--xml` option to **svn log**, will retrieve and display all revision properties in the log output.

`--with-revprop` *ARG*
When used with any command that writes to the repository, sets the revision property, using the *NAME=VALUE* format, *NAME* to *VALUE*. When used with **svn log** in `--xml` mode, this displays the value of *ARG* in the log output.

`--xml`
使用XML格式打印输出。

9.1.2. svn 子命令

Here are the various subcommands for the **svn** program. For the sake of brevity, we omit the global options (described in 第 9.1.1 节 “svn 选项”) from the subcommand descriptions which follow.

名称

svn add — 添加文件、目录或符号链。

概要

```
svn add PATH...
```

描述

文件、目录或符号链到你的工作拷贝并且预定添加到版本库。它们会在下次提交上传并添加到版本库，如果你在提交之前改变了主意，你可以使用**svn revert**取消预定。

别名

无

改变

工作拷贝2

访问版本库

否

选项

```
--auto-props
--depth ARG
--force
--no-auto-props
--no-ignore
--parents
--quiet (-q)
--targets FILENAME
```

例子

添加一个文件到工作拷贝：

```
$ svn add foo.c
A         foo.c
```

当添加一个目录，**svn add**缺省的行为方式是递归的：

```
$ svn add testdir
A         testdir
A         testdir/a
A         testdir/b
A         testdir/c
A         testdir/d
```

你可以只添加一个目录而不包括其内容：

```
$ svn add --depth=empty otherdir
A         otherdir
```

Normally, the command **svn add *** will skip over any directories that are already under version control. Sometimes, however, you may want to add every unversioned object in your working copy, including those hiding deeper. Passing the `--force` option makes **svn add** recurse into versioned directories:

```
$ svn add * --force
A         foo.c
A         somedir/bar.c
A  (bin)  otherdir/docs/baz.doc
…
```

名称

svn blame — 显示特定文件和URL内嵌的作者和修订版本信息。

概要

```
svn blame TARGET[@REV]...
```

描述

显示特定文件和URL内嵌的作者和修订版本信息。每一行文本在开头都放了最后修改的作者(用户名)和修订版本号。

别名

praise, **annotate**, **ann**

改变

无2

访问版本库

是

选项

```
--extensions (-x) ARG
--force
--incremental
--revision (-r) ARG
--use-merge-history (-g)
--verbose (-v)
--xml
```

例子

If you want to see blame-annotated source for `readme.txt` in your test repository:

```
$ svn blame http://svn.red-bean.com/repos/test/readme.txt
     3      sally This is a README file.
     5      harry You should read this.
```

Even if **svn blame** says that Harry last modified `readme.txt` in revision 5, you'll have to examine exactly what the revision changed to be sure that Harry changed the *context* of the line—he may have adjusted just the whitespace.

If you use the `--xml` option, you can get XML output describing the blame annotations, but not the contents of the lines themselves:

```
$ svn blame --xml http://svn.red-bean.com/repos/test/readme.txt
```

```
<?xml version="1.0"?>
<blame>
<target
   path="sandwich.txt">
<entry
   line-number="1">
<commit
   revision="3">
<author>sally</author>
<date>2008-05-25T19:12:31.428953Z</date>
</commit>
</entry>
<entry
   line-number="2">
<commit
   revision="5">
<author>harry</author>
<date>2008-05-29T03:26:12.293121Z</date>
</commit>
</entry>
</target>
</blame>
```

名称

svn cat — 输出特定文件或URL的内容。

概要

```
svn cat TARGET[@REV]...
```

描述

Output the contents of the specified files or URLs. For listing the contents of directories, see **svn list** later in this chapter.

别名

无

改变

无2

访问版本库

是

选项

```
--revision (-r) REV
```

例子

如果你希望不检出而察看版本库的readme.txt的内容：

```
$ svn cat http://svn.red-bean.com/repos/test/readme.txt
This is a README file.
You should read this.
```

提示

If your working copy is out of date (or you have local modifications) and you want to see the `HEAD` revision of a file in your working copy, **svn cat -r HEAD *FILENAME*** will automatically fetch the `HEAD` revision of the specified path:

```
$ cat foo.c
This file is in my local working copy
and has changes that I've made.

$ svn cat -r HEAD foo.c
Latest revision fresh from the repository!
```

名称

svn changelist — Associate (or deassociate) local paths with a changelist.

概要

```
changelist CLNAME TARGET...

changelist --remove TARGET...
```

描述

Used for dividing files in a working copy into a changelist (logical named grouping) in order to allow users to easily work on multiple file collections within a single working copy.

别名

cl

改变

工作拷贝2

访问版本库

否

选项

```
--changelist ARG
--depth ARG
--quiet (-q)
--recursive (-R)
--remove
--targets ARG
```

例子

Edit three files, add them to a changelist, then commit only files in that changelist:

```
$ svn cl issue1729 foo.c bar.c baz.c
Path 'foo.c' is now a member of changelist 'issue1729'.
Path 'bar.c' is now a member of changelist 'issue1729'.
Path 'baz.c' is now a member of changelist 'issue1729'.

$ svn status
A      someotherfile.c
A      test/sometest.c

--- Changelist 'issue1729':
A      foo.c
A      bar.c
```

```
A      baz.c

$ svn commit --changelist issue1729 -m "Fixing Issue 1729."
Adding         bar.c
Adding         baz.c
Adding         foo.c
Transmitting file data ...
Committed revision 2.

$ svn status
A      someotherfile.c
A      test/sometest.c
```

注意，只有在变更列表*issue1729*中的文件被提交了。

名称

svn checkout — 从版本库取出一个工作拷贝。

概要

```
svn checkout URL[@REV]... [PATH]
```

描述

从版本库取出一个工作拷贝，如果省略*PATH*，URL的基名称会作为目标，如果给定多个URL，每一个都会检出到*PATH*的子目录，使用URL基名称的子目录名称。

别名

co

改变

Creates a working copy

访问版本库

是

选项

```
--depth ARG
--force
--ignore-externals
--quiet (-q)
--revision (-r) REV
```

例子

取出一个工作拷贝到mine目录：

```
$ svn checkout file:///var/svn/repos/test mine
A  mine/a
A  mine/b
A  mine/c
A  mine/d
Checked out revision 20.
$ ls
mine
```

检出两个目录到两个单独的工作拷贝：

```
$ svn checkout file:///var/svn/repos/test  file:///var/svn/repos/quiz
A  test/a
A  test/b
```

```
A  test/c
A  test/d
Checked out revision 20.
A  quiz/l
A  quiz/m
Checked out revision 13.
$ ls
quiz  test
```

检出两个目录到两个单独的工作拷贝，但是将两个目录都放到`working-copies`:

```
$ svn checkout file:///var/svn/repos/test  file:///var/svn/repos/quiz work
A  working-copies/test/a
A  working-copies/test/b
A  working-copies/test/c
A  working-copies/test/d
Checked out revision 20.
A  working-copies/quiz/l
A  working-copies/quiz/m
Checked out revision 13.
$ ls
working-copies
```

如果你打断一个检出(或其它打断检出的事情，如连接失败。)，你可以使用同样的命令重新开始或者是更新不完整的工作拷贝：

```
$ svn checkout file:///var/svn/repos/test mine
A  mine/a
A  mine/b
^C
svn: The operation was interrupted
svn: caught SIGINT

$ svn checkout file:///var/svn/repos/test mine
A  mine/c
^C
svn: The operation was interrupted
svn: caught SIGINT

$ svn update mine
A  mine/d
Updated to revision 20.
```

If you wish to check out some revision other than the most recent one, you can do so by providing the `--revision` (`-r`) option to the **svn checkout** command:

```
$ svn checkout -r 2 file:///var/svn/repos/test mine
A  mine/a
Checked out revision 2.
```

名称

svn cleanup — Recursively clean up the working copy

概要

```
svn cleanup [PATH...]
```

描述

递归清理工作拷贝，删除未完成的工作拷贝锁定，并恢复未完成的操作。如果你得到一个“工作拷贝已锁定”的错误，运行这个命令可以删除无效的锁定，让你的工作拷贝再次回到可用的状态。

如果，因为一些原因，运行外置的区别程序(例如，用户输入或是网络错误)有时候会导致一个**svn update**失败，使用`--diff3-cmd`选项可以完全清除你的外置区别程序所作的合并，你也可以使用`--config-dir`指定任何配置目录，但是你应该不会经常使用这些选项。

别名

无

改变

工作拷贝2

访问版本库

否

选项

```
--diff3-cmd CMD
```

例子

Well, there's not much to the examples here, as **svn cleanup** generates no output. If you pass no *PATH*, then “.” is used:

```
$ svn cleanup

$ svn cleanup /var/svn/working-copy
```

名称

svn commit — 将修改从工作拷贝发送到版本库。

概要

```
svn commit [PATH...]
```

描述

Send changes from your working copy to the repository. If you do not supply a log message with your commit by using either the `--file` or `--message` option, **svn** will launch your editor for you to compose a commit message. See the `editor-cmd` list entry in 第 7.1.3.2 节 “配置”.

svn commit will send any lock tokens that it finds and will release locks on all *PATH*s committed (recursively) unless `--no-unlock` is passed.

提示

如果你开始一个提交并且Subversion启动了你的编辑器来编辑提交信息，你仍可以退出而不会提交你的修改，如果你希望取消你的提交，只需要退出编辑器而不保存你的提交信息，Subversion会提示你是选择取消提交、空信息继续还是重新编辑信息。

别名

ci (short for “check in”; not **co**, which is an alias for the **checkout** subcommand)

改变

工作拷贝；版本库

访问版本库

是

选项

```
--changelist ARG
--depth ARG
--editor-cmd ARG
--encoding ENC
--file (-F) FILE
--force-log
--keep-changelists
--message (-m) TEXT
--no-unlock
--quiet (-q)
--targets FILENAME
--with-revprop ARG
```

例子

使用命令行提交一个包含日志信息的文件修改，当前目录(“.”)是没有说明的目标路径：

```
$ svn commit -m "added howto section."
Sending        a
Transmitting file data .
Committed revision 3.
```

Commit a modification to the file `foo.c` (explicitly specified on the command line) with the commit message in a file named `msg`:

```
$ svn commit -F msg foo.c
Sending        foo.c
Transmitting file data .
Committed revision 5.
```

如果你希望使用在`--file`选项中使用在版本控制之下的文件作为参数，你需要使用`--force-log`选项：

```
$ svn commit --file file_under_vc.txt foo.c
svn: The log message file is under version control
svn: Log message file is a versioned file; use '--force-log' to override

$ svn commit --force-log --file file_under_vc.txt foo.c
Sending        foo.c
Transmitting file data .
Committed revision 6.
```

提交一个已经预定要删除的文件：

```
$ svn commit -m "removed file 'c'."
Deleting       c

Committed revision 7.
```

名称

svn copy — 拷贝工作拷贝的一个文件或目录到版本库。

概要

```
svn copy SRC[@REV]... DST
```

描述

Copy one or more files in a working copy or in the repository. When copying multiple sources, they will be added as children of *DST*, which must be a directory. *SRC* and *DST* can each be either a working copy (WC) path or URL:

WC → WC
　拷贝并且预定一个添加的项目(包含历史)。

WC → URL
　将WC或URL的拷贝立即提交。

URL → WC
　检出URL到WC，并且加入到添加计划。

URL → URL
　完全的服务器端拷贝，通常用在分支和标签。

When copying multiple sources, they will be added as children of *DST*, which must be a directory.

If no peg revision (i.e., *@REV*) is supplied, by default the BASE revision will be used for files copied from the working copy, while the HEAD revision will be used for files copied from a URL.

注意

你只可以在单个版本库中拷贝文件，Subversion还不支持跨版本库的拷贝。

别名

cp
改变

Repository if destination is a URL; working copy if destination is a WC path
访问版本库

如果目标是版本库，或者需要查看修订版本号，则会访问版本库。
选项

```
--editor-cmd EDITOR
--encoding ENC
--file (-F) FILE
--force-log
```

```
--message (-m) TEXT
--parents
--quiet (-q)
--revision (-r) REV
--with-revprop ARG
```

例子

Copy an item within your working copy (this schedules the copy—nothing goes into the repository until you commit):

```
$ svn copy foo.txt bar.txt
A         bar.txt
$ svn status
A  +   bar.txt
```

拷贝工作拷贝的一个文件或目录到版本库：

```
$ svn cp bat.c baz.c qux.c src
A         src/bat.c
A         src/baz.c
A         src/qux.c
```

Copy revision 8 of `bat.c` into your working copy under a different name:

```
$ svn cp -r 8 bat.c ya-old-bat.c
A         ya-old-bat.c
```

Copy an item in your working copy to a URL in the repository (this is an immediate commit, so you must supply a commit message):

```
$ svn copy near.txt file:///var/svn/repos/test/far-away.txt -m "Remote cop

Committed revision 8.
```

Copy an item from the repository to your working copy (this just schedules the copy—nothing goes into the repository until you commit):

```
$ svn copy file:///var/svn/repos/test/far-away -r 6 near-here
A         near-here
```

提示

这是恢复死掉文件的推荐方式！

最后，是在URL之间拷贝：

```
$ svn copy file:///var/svn/repos/test/far-away \
           file:///var/svn/repos/test/over-there -m "remote copy."

Committed revision 9.


$ svn copy file:///var/svn/repos/test/trunk \
           file:///var/svn/repos/test/tags/0.6.32-prerelease -m "tag tree"

Committed revision 12.
```

提示

This is the easiest way to "tag" a revision in your repository—just **svn copy** that revision (usually `HEAD`) into your `tags` directory.

不要担心忘记作标签—你可以在以后任何时候给一个旧版本作标签：

```
$ svn copy -r 11 file:///var/svn/repos/test/trunk \
           file:///var/svn/repos/test/tags/0.6.32-prerelease \
           -m "Forgot to tag at rev 11"

Committed revision 13.
```

名称

svn delete — 从工作拷贝或版本库删除一个项目。

概要

```
svn delete PATH...

svn delete URL...
```

描述

*PATH*指定的项目会在下次提交删除，除非给定--keep-local选项，否则文件(和没有提交的目录)会立即从版本库删除，这个命令不会删除任何未版本化或已经修改的项目；使用--force选项可以覆盖这种行为方式。

URL指定的项目会在直接提交中从版本库删除，多个URL的提交是原子操作。

别名

del, **remove**, **rm**

改变

如果操作对象是文件则是工作拷贝变化；如果对象是URL则会影响版本库。

访问版本库

对URL操作时访问

选项

```
--editor-cmd EDITOR
--encoding ENC
--file (-F) FILE
--force
--force-log
--keep-local
--message (-m) TEXT
--quiet (-q)
--targets FILENAME
--with-revprop ARG
```

例子

使用**svn**从工作拷贝删除文件只是预定要删除，当你提交，文件才会从版本库删除。

```
$ svn delete myfile
D         myfile

$ svn commit -m "Deleted file 'myfile'."
Deleting       myfile
```

```
Transmitting file data .
Committed revision 14.
```

然而直接删除一个URL，你需要提供一个日志信息：

```
$ svn delete -m "Deleting file 'yourfile'" \
             file:///var/svn/repos/test/yourfile

Committed revision 15.
```

如下是强制删除本地已修改文件的例子：

```
$ svn delete over-there
svn: Attempting restricted operation for modified resource
svn: Use --force to override this restriction
svn: 'over-there' has local modifications

$ svn delete --force over-there
D         over-there
```

名称

svn diff — This displays the differences between two revisions or paths.

概要

```
diff [-c M | -r N[:M]] [TARGET[@REV]...]

diff [-r N[:M]] --old=OLD-TGT[@OLDREV] [--new=NEW-TGT[@NEWREV]] [PATH...]

diff OLD-URL[@OLDREV] NEW-URL[@NEWREV]
```

描述

- 显示两条路径的区别，你可以通过下面的方式使用**svn diff**:
- Use just **svn diff** to display local modifications in a working copy.
- 显示*TARGET*在*REV*的样子时两个修订版本之间所作的修改，*TARGET*可以是任何工作拷贝路径或任何*URL*，如果*TARGET*是工作拷贝路径，则*N*缺省是BASE，而*M*是工作拷贝；如果是*URL*，则必须指定*N*，而*M*缺省是HEAD。“–c M”选项与“–r N:M”等价，其中N = M-1。使用“–c -M”则相反：“–r M:N”的意思是N = M-1。
- Display the differences between *OLD-TGT* as it was seen in *OLDREV* and *NEW-TGT* as it was seen in *NEWREV*. *PATH*s, if given, are relative to *OLD-TGT* and *NEW-TGT* and restrict the output to differences for those paths. *OLD-TGT* and *NEW-TGT* may be working copy paths or *URL[@REV]*. *NEW-TGT* defaults to *OLD-TGT* if not specified. `-r N` makes *OLDREV* default to N; `-r N:M` makes *OLDREV* default to *N* and *NEWREV* default to *M*.

svn diff OLD-URL[@OLDREV] NEW-URL[@NEWREV] is shorthand for **svn diff --old=OLD-URL[@OLDREV] --new=NEW-URL[@NEWREV].**

svn diff -r N:M URL is shorthand for **svn diff -r N:M --old=URL --new=URL**.

svn diff [-r N[:M]] URL1[@N] URL2[@M] is shorthand for **svn diff [-r N[:M]] --old=URL1 --new=URL2**.

If *TARGET* is a URL, then revs N and M can be given either via the `--revision` option or by using the “@” notation as described earlier.

If *TARGET* is a working copy path, the default behavior (when no `--revision` option is provided) is to display the differences between the base and working copies of *TARGET*. If a `--revision` option is specified in this scenario, though, it means:

`--revision N:M`
服务器比较*TARGET@N*和*TARGETTARGET@M*。

`--revision N`
The client compares *TARGET@N* against the working copy.

If the alternate syntax is used, the server compares *URL1* and *URL2* at revisions *N* and *M*, respectively. If either *N* or *M* is omitted, a value of HEAD is assumed.

By default, **svn diff** ignores the ancestry of files and merely compares the contents of the two files being compared. If you use `--notice-ancestry`, the ancestry of the paths in question will be taken into consideration when comparing revisions (i.e., if you run **svn diff** on two files with identical contents but different ancestry, you will see the entire contents of the file as having been removed and added again).

别名

di

改变

无2

访问版本库

获得工作拷贝非BASE修订版本的区别时会

选项

```
--change (-c) ARG
--changelist ARG
--depth ARG
--diff-cmd CMD
--extensions (-x) "ARGS"
--force
--new ARG
--no-diff-deleted
--notice-ancestry
--old ARG
--revision (-r) ARG
--summarize
--xml
```

例子

比较BASE和你的工作拷贝(**svn diff**最经常的用法)：

```
$ svn diff COMMITTERS
Index: COMMITTERS
===================================================================
--- COMMITTERS (revision 4404)
+++ COMMITTERS (working copy)
```

See what changed in the file `COMMITTERS` revision 9115:

```
$ svn diff -c 9115 COMMITTERS
Index: COMMITTERS
===================================================================
--- COMMITTERS (revision 3900)
+++ COMMITTERS (working copy)
```

察看你的工作拷贝对旧的修订版本的修改：

```
$ svn diff -r 3900 COMMITTERS
Index: COMMITTERS
===================================================================
--- COMMITTERS (revision 3900)
+++ COMMITTERS (working copy)
```

使用“@”语法与修订版本3000和35000比较：

```
$ svn diff http://svn.collab.net/repos/svn/trunk/COMMITTERS@3000 \
           http://svn.collab.net/repos/svn/trunk/COMMITTERS@3500
Index: COMMITTERS
===================================================================
--- COMMITTERS (revision 3000)
+++ COMMITTERS (revision 3500)
…
```

使用范围符号来比较修订版本3000和3500(在这种情况下只能传递一个URL)：

```
$ svn diff -r 3000:3500 http://svn.collab.net/repos/svn/trunk/COMMITTERS
Index: COMMITTERS
===================================================================
--- COMMITTERS (revision 3000)
+++ COMMITTERS (revision 3500)
```

使用范围符号比较修订版本3000和3500trunk中的所有文件：

```
$ svn diff -r 3000:3500 http://svn.collab.net/repos/svn/trunk
```

使用范围符号比较修订版本3000和3500trunk中的三个文件：

```
$ svn diff -r 3000:3500 --old http://svn.collab.net/repos/svn/trunk \
           COMMITTERS README HACKING
```

如果你有工作拷贝，你不必输入这么长的URL：

```
$ svn diff -r 3000:3500 COMMITTERS
Index: COMMITTERS
===================================================================
--- COMMITTERS (revision 3000)
+++ COMMITTERS (revision 3500)
```

使用--diff-cmd*CMD*-x来指定外部区别程序

```
$ svn diff --diff-cmd /usr/bin/diff -x "-i -b" COMMITTERS
Index: COMMITTERS
===================================================================
0a1,2
> This is a test
>
```

Lastly, you can use the `--xml` option along with the `--summarize` option to view XML describing the changes that occurred between revisions, but not the contents of the diff itself:

```
$ svn diff --summarize --xml http://svn.red-bean.com/repos/test@r2 \
           http://svn.red-bean.com/repos/test
<?xml version="1.0"?>
<diff>
<paths>
<path
   props="none"
   kind="file"
   item="modified"&gt;http://svn.red-bean.com/repos/test/sandwich.txt&lt;/
<path
   props="none"
   kind="file"
   item="deleted"&gt;http://svn.red-bean.com/repos/test/burrito.txt&lt;/pa
<path
   props="none"
   kind="dir"
   item="added"&gt;http://svn.red-bean.com/repos/test/snacks&lt;/path&gt;
</paths>
</diff>
```

名称

svn export — 导出一个干净的目录树。

概要

```
svn export [-r REV] URL[@PEGREV] [PATH]

svn export [-r REV] PATH1[@PEGREV] [PATH2]
```

描述

The first form exports a clean directory tree from the repository specified by *URL*—at revision *REV* if it is given; otherwise, at HEAD, into *PATH*. If *PATH* is omitted, the last component of the *URL* is used for the local directory name.

从工作拷贝导出干净目录树的第二种形式是指定*PATH1*到*PATH2*，所有的本地修改将会保留，但是不再版本控制下的文件不会拷贝。

别名

无

改变

本地磁盘

访问版本库

只有当从URL导出时会访问

选项

```
--depth ARG
--force
--ignore-externals
--native-eol EOL
--quiet (-q)
--revision (-r) REV
```

例子

从你的工作拷贝导出(不会打印每一个文件和目录)：

```
$ svn export a-wc my-export
Export complete.
```

从版本库导出目录(打印所有的文件和目录)：

```
$ svn export file:///var/svn/repos my-export
A  my-export/test
```

```
A  my-export/quiz
…
Exported revision 15.
```

When rolling operating-system-specific release packages, it can be useful to export a tree that uses a specific EOL character for line endings. The `--native-eol` option will do this, but it affects only files that have `svn:eol-style = native` properties attached to them. For example, to export a tree with all CRLF line endings (possibly for a Windows `.zip` file distribution):

```
$ svn export file:///var/svn/repos my-export --native-eol CRLF
A  my-export/test
A  my-export/quiz
…
Exported revision 15.
```

你可以为`--native-eol`选项指定`LR`、`CR`或`CRLF`作为行结束符。

名称

svn help — 求助!

概要

```
svn help [SUBCOMMAND...]
```

描述

当手边没有这本书时，这是你使用Subversion最好的朋友!

别名

?, **h**

使用`-?`、`-h`和`--help`选项与使用**help**子命令效果相同。

改变

无2

访问版本库

否

选项

名称

svn import — 递归提交一个路径的拷贝到版本库。

概要

```
svn import [PATH] URL
```

描述

递归提交一个*PATH*到*URL*。如果省略*PATH*，默认是“.”。版本库中对应的父目录必须已经创建。未版本化的文件和管道会被忽略，即使设置了--force选项。

别名

无

改变

版本库

访问版本库

是

选项

```
--auto-props
--depth ARG
--editor-cmd EDITOR
--encoding ENC
--file (-F) FILE
--force
--force-log
--message (-m) TEXT
--no-auto-props
--no-ignore
--quiet (-q)
--with-revprop ARG
```

例子

这将本地目录myproj导入到版本库的trunk/misc，trunk/misc在导入之前不需要存在—**svn import**会递归的为你创建目录。

```
$ svn import -m "New import" myproj \
             http://svn.red-bean.com/repos/trunk/misc
Adding         myproj/sample.txt
…
Transmitting file data .........
Committed revision 16.
```

需要知道这样不会在版本库创建目录myproj，如果你希望这样，请在URL后添加myproj：

```
$ svn import -m "New import" myproj \
             http://svn.red-bean.com/repos/trunk/misc/myproj
Adding         myproj/sample.txt
…
Transmitting file data .........
Committed revision 16.
```

在导入数据之后，你会发现原先的目录树并没有纳入版本控制，为了开始工作，你还是要运行**svn checkout**得到一个干净的目录树工作拷贝。

名称

svn info — 显示本地或远程条目的信息。

概要

```
svn info [TARGET[@REV]...]
```

描述

打印你的工作拷贝路径和URL的信息，包括：

- 路经
- 名称
- URL
- 版本库的根
- 版本库的UUID
- Revision
- 节点类型
- 最后修改的作者
- 最后修改的修订版本
- 最后修改的日期
- 锁定令牌
- 锁定拥有者
- 锁定创建时间(date)
- Lock失效时间(date)

Additional kinds of information available only for working copy paths are:

- Schedule
- 拷贝自的URL
- 拷贝自的修订版本
- 数据最后更新
- 属性最后更新
- Checksum

- Conflict previous base file
- Conflict previous working file
- Conflict current base file
- Conflict properties file

别名

无
改变

无2
访问版本库

对URL操作时访问
选项

```
--changelist ARG
--depth ARG
--incremental
--recursive (-R)
--revision (-r) REV
--targets FILENAME
--xml
```

例子

svn info会展示工作拷贝所有项目的所有有用信息，它会显示文件的信息：

```
$ svn info foo.c
Path: foo.c
Name: foo.c
URL: http://svn.red-bean.com/repos/test/foo.c
Repository Root: http://svn.red-bean.com/repos/test
Repository UUID: 5e7d134a-54fb-0310-bd04-b611643e5c25
Revision: 4417
Node Kind: file
Schedule: normal
Last Changed Author: sally
Last Changed Rev: 20
Last Changed Date: 2003-01-13 16:43:13 -0600 (Mon, 13 Jan 2003)
Text Last Updated: 2003-01-16 21:18:16 -0600 (Thu, 16 Jan 2003)
Properties Last Updated: 2003-01-13 21:50:19 -0600 (Mon, 13 Jan 2003)
Checksum: /3L38YwzhT93BWvgpdF6Zw==
```

它也会展示目录的信息：

```
$ svn info vendors
Path: vendors
URL: http://svn.red-bean.com/repos/test/vendors
Repository Root: http://svn.red-bean.com/repos/test
Repository UUID: 5e7d134a-54fb-0310-bd04-b611643e5c25
Revision: 19
Node Kind: directory
Schedule: normal
Last Changed Author: harry
Last Changed Rev: 19
Last Changed Date: 2003-01-16 23:21:19 -0600 (Thu, 16 Jan 2003)
```

svn info也可以针对URL操作(另外，可以注意一下例子中的`readme.doc`文件已经被锁定，所以也会显示锁定信息):

```
$ svn info http://svn.red-bean.com/repos/test/readme.doc
Path: readme.doc
Name: readme.doc
URL: http://svn.red-bean.com/repos/test/readme.doc
Repository Root: http://svn.red-bean.com/repos/test
Repository UUID: 5e7d134a-54fb-0310-bd04-b611643e5c25
Revision: 1
Node Kind: file
Schedule: normal
Last Changed Author: sally
Last Changed Rev: 42
Last Changed Date: 2003-01-14 23:21:19 -0600 (Tue, 14 Jan 2003)
Text Last Updated: 2003-01-14 23:21:19 -0600 (Tue, 14 Jan 2003)
Checksum: d41d8cd98f00b204e9800998ecf8427e
Lock Token: opaquelocktoken:14011d4b-54fb-0310-8541-dbd16bd471b2
Lock Owner: harry
Lock Created: 2003-01-15 17:35:12 -0600 (Wed, 15 Jan 2003)
```

Lastly, **svn info** output is available in XML format by passing the`--xml` option:

```
$ svn info --xml http://svn.red-bean.com/repos/test
<?xml version="1.0"?>
<info>
<entry
   kind="dir"
   path="."
   revision="1">
<url>http://svn.red-bean.com/repos/test</url>
<repository>
<root>http://svn.red-bean.com/repos/test</root>
<uuid>5e7d134a-54fb-0310-bd04-b611643e5c25</uuid>
```

```
</repository>
<wc-info>
<schedule>normal</schedule>
<depth>infinity</depth>
</wc-info>
<commit
   revision="1">
<author>sally</author>
<date>2003-01-15T23:35:12.847647Z</date>
</commit>
</entry>
</info>
```

名称

svn list — 列出版本库目录的条目。

概要

```
svn list [TARGET[@REV]...]
```

描述

列出每一个*TARGET*文件和*TARGET*目录的内容，如果*TARGET*是工作拷贝路径，会使用对应的版本库URL。

缺省的*TARGET*是“.”，意味着当前工作拷贝的版本库URL。

如果一个客户端连接到**svnserve**进程，如下事情会发生：

- 最后一次提交的修订版本号
- 最后一次提交的作者
- If locked, the letter “O” (see the preceding section on svn info for details).
- 大小(单位字节)
- 最后提交的日期时间

使用选项`--xml`，输出是XML格式(如果没有指定`--incremental`，会包括一个头和一个围绕的元素)。会展示所有的信息；不接受`--verbose`选项。

别名

ls

改变

无2

访问版本库

是

选项

```
--depth ARG
--incremental
--recursive (-R)
--revision (-r) REV
--verbose (-v)
--xml
```

例子

如果你希望在没有下载工作拷贝时查看版本库有哪些文件，**svn list**会非常有用：

```
$ svn list http://svn.red-bean.com/repos/test/support
README.txt
INSTALL
examples/
…
```

你也可以传递--verbose选项来得到额外信息，非常类似UNIX的**ls -l**命令：

```
$ svn list --verbose file:///var/svn/repos
     16 sally         28361 Jan 16 23:18 README.txt
     27 sally             0 Jan 18 15:27 INSTALL
     24 harry               Jan 18 11:27 examples/
```

You can also get **svn list** output in XML format with the --xml option:

```
$ svn list --xml http://svn.red-bean.com/repos/test
<?xml version="1.0"?>
<lists>
<list
   path="http://svn.red-bean.com/repos/test">
<entry
   kind="dir">
<name>examples</name>
<size>0</size>
<commit
   revision="24">
<author>harry</author>
<date>2008-01-18T06:35:53.048870Z</date>
</commit>
</entry>
...
</list>
</lists>
```

更多细节见第 2.5.3.2 节 “svn list”。

名称

svn lock — Lock working copy paths or URLs in the repository so that no other user can commit changes to them.

概要

Synopsis

描述

svn lock TARGET...

别名

无

改变

工作拷贝，版本库

访问版本库

是

选项

```
--encoding ENC
--file (-F) FILE
--force
--force-log
--message (-m) TEXT
--targets FILENAME
```

例子

在工作拷贝锁定两个文件：

```
$ svn lock tree.jpg house.jpg
'tree.jpg' locked by user 'harry'.
'house.jpg' locked by user 'harry'.
```

锁定工作拷贝的一个被其它用户锁定的文件：

```
$ svn lock tree.jpg
svn: warning: Path '/tree.jpg is already locked by user 'sally in \
     filesystem '/var/svn/repos/db'

$ svn lock --force tree.jpg
'tree.jpg' locked by user 'harry'.
```

没有工作拷贝的情况下锁定文件：

```
$ svn lock http://svn.red-bean.com/repos/test/tree.jpg
'tree.jpg' locked by user 'harry'.
```

更多细节见第 3.7 节 “锁定”。

名称

svn log — 显示提交日志信息。

概要

```
svn log [PATH]

svn log URL[@REV] [PATH...]
```

描述

Shows log messages from the repository. If no arguments are supplied, **svn log** shows the log messages for all files and directories inside (and including) the current working directory of your working copy. You can refine the results by specifying a path, one or more revisions, or any combination of the two. The default revision range for a local path is `BASE:1`.

If you specify a URL alone, it prints log messages for everything the URL contains. If you add paths past the URL, only messages for those paths under that URL will be printed. The default revision range for a URL is `HEAD:1`.

svn log使用`--verbose`选项也会打印所有影响路径的日志信息，使用`--quiet`选项不会打印日志信息正文本身(这与`--verbose`协调一致)。

每个日志信息只会打印一次，即使是那些明确请求不止一次的路径，日志会跟随在拷贝过程中，使用`--stop-on-copy`可以关闭这个特性，可以用来监测分支点。

别名

无

改变

无2

访问版本库

是

选项

```
--change (-c) ARG
--incremental
--limit (-l) NUM
--quiet (-q)
--revision (-r) REV
--stop-on-copy
--targets FILENAME
--use-merge-history (-g)
--verbose (-v)
--with-all-revprops
--with-revprop ARG
--xml
```

例子

You can see the log messages for all the paths that changed in your working copy by running **svn log** from the top:

```
$ svn log
------------------------------------------------------------------------
r20 | harry | 2003-01-17 22:56:19 -0600 (Fri, 17 Jan 2003) | 1 line

Tweak.
------------------------------------------------------------------------
r17 | sally | 2003-01-16 23:21:19 -0600 (Thu, 16 Jan 2003) | 2 lines
…
```

检验一个特定文件所有的日志信息：

```
$ svn log foo.c
------------------------------------------------------------------------
r32 | sally | 2003-01-13 00:43:13 -0600 (Mon, 13 Jan 2003) | 1 line

Added defines.
------------------------------------------------------------------------
r28 | sally | 2003-01-07 21:48:33 -0600 (Tue, 07 Jan 2003) | 3 lines
…
```

如果你手边没有工作拷贝，你可以查看一个URL的日志：

```
$ svn log http://svn.red-bean.com/repos/test/foo.c
------------------------------------------------------------------------
r32 | sally | 2003-01-13 00:43:13 -0600 (Mon, 13 Jan 2003) | 1 line

Added defines.
------------------------------------------------------------------------
r28 | sally | 2003-01-07 21:48:33 -0600 (Tue, 07 Jan 2003) | 3 lines
…
```

如果你希望查看某个URL下面不同的多个路径，你可以使用`URL [PATH...]`语法。

```
$ svn log http://svn.red-bean.com/repos/test/ foo.c bar.c
------------------------------------------------------------------------
r32 | sally | 2003-01-13 00:43:13 -0600 (Mon, 13 Jan 2003) | 1 line

Added defines.
------------------------------------------------------------------------
r31 | harry | 2003-01-10 12:25:08 -0600 (Fri, 10 Jan 2003) | 1 line
```

```
Added new file bar.c
------------------------------------------------------------------------
r28 | sally | 2003-01-07 21:48:33 -0600 (Tue, 07 Jan 2003) | 3 lines
…
```

The `--verbose` option causes **svn log** to include information about the paths that were changed in each displayed revision. These paths appear, one path per line of output, with action codes that indicate what type of change was made to the path.

```
$ svn log -v http://svn.red-bean.com/repos/test/ foo.c bar.c
------------------------------------------------------------------------
r32 | sally | 2003-01-13 00:43:13 -0600 (Mon, 13 Jan 2003) | 1 line
Changed paths:
   M /foo.c

Added defines.
------------------------------------------------------------------------
r31 | harry | 2003-01-10 12:25:08 -0600 (Fri, 10 Jan 2003) | 1 line
Changed paths:
   A /bar.c

Added new file bar.c
------------------------------------------------------------------------
r28 | sally | 2003-01-07 21:48:33 -0600 (Tue, 07 Jan 2003) | 3 lines
…
```

svn log uses just a handful of action codes, and they are similar to the ones the **svn update** command uses:

`A`
: The item was added.

`D`
: The item was deleted.

`M`
: 条目属性改变了，注意开头的空格。

`R`
: The item was replaced by a different one at the same location.

In addition to the action codes which precede the changed paths, **svn log** with the `--verbose` option will note whether a path was added or replaced as the result of a copy operation. It does so by printing `(from` *`COPY-FROM-PATH`*`:`*`COPY-FROM-REV`*`)` after such paths.

当你想连接多个对日志命令的调用结果，你会希望使用`--incremental`选项。**svn log**通常会在日志信息的开头和每一小段间打印一行虚线，如果你对一段修订版本运行**svn log**，你会得到下面的结果：

```
$ svn log -r 14:15
------------------------------------------------------------------------
r14 | ...

------------------------------------------------------------------------
r15 | ...

------------------------------------------------------------------------
```

然而，如果你希望收集两个不连续的日志信息到一个文件，你会这样做：

```
$ svn log -r 14 > mylog
$ svn log -r 19 >> mylog
$ svn log -r 27 >> mylog
$ cat mylog
------------------------------------------------------------------------
r14 | ...

------------------------------------------------------------------------
------------------------------------------------------------------------
r19 | ...

------------------------------------------------------------------------
------------------------------------------------------------------------
r27 | ...

------------------------------------------------------------------------
```

你可以使用incremental选项来避免两行虚线带来的混乱：

```
$ svn log --incremental -r 14 > mylog
$ svn log --incremental -r 19 >> mylog
$ svn log --incremental -r 27 >> mylog
$ cat mylog
------------------------------------------------------------------------
r14 | ...

------------------------------------------------------------------------
r19 | ...

------------------------------------------------------------------------
r27 | ...
```

--incremental选项为--xml提供了一个相似的输出控制。

```
$ svn log --xml --incremental -r 1 sandwich.txt
```

```
<logentry
   revision="1">
<author>harry</author>
<date>2008-06-03T06:35:53.048870Z</date>
<msg>Initial Import.</msg>
</logentry>
```

提示

Sometimes when you run **svn log** on a specific path and a specific revision, you see no log information output at all, as in the following:

```
$ svn log -r 20 http://svn.red-bean.com/untouched.txt
------------------------------------------------------------------------
```

That just means the path wasn't modified in that revision. To get log information for that revision, either run the log operation against the repository's root URL, or specify a path that you happen to know was changed in that revision:

```
$ svn log -r 20 touched.txt
------------------------------------------------------------------------
r20 | sally | 2003-01-17 22:56:19 -0600 (Fri, 17 Jan 2003) | 1 lin

Made a change.
------------------------------------------------------------------------
```

名称

svn merge — 应用两组源文件的差别到工作拷贝路径。

概要

```
svn merge sourceURL1[@N] sourceURL2[@M] [WCPATH]

svn merge sourceWCPATH1@N sourceWCPATH2@M [WCPATH]

svn merge [[-c M]... | [-r N:M]...] [SOURCE[@REV] [WCPATH]]
```

描述

In the first form, the source URLs are specified at revisions *N* and *M*. These are the two sources to be compared. The revisions default to `HEAD` if omitted.

In the second form, the URLs corresponding to the source working copy paths define the sources to be compared. The revisions must be specified.

在第三种形式里，*SOURCE*可以是URL或工作拷贝路径(会使用对应的URL)，如果没有指定，*SOURCE*将会与*WCPATH*相同。修订版本*REV*的*SOURCE*会比较存在的*N*和。如果没有指定*REV*，则假定是*HEAD*。

`-c M` is equivalent to `-r <M-1>:M`, and `-c -M` does the reverse: `-r M:<M-1>`. If no revision ranges are specified, the default range of `1:HEAD` is used. Multiple `-c` and/or `-r` instances may be specified, and mixing of forward and reverse ranges is allowed—the ranges are internally compacted to their minimum representation before merging begins (which may result in no-op).

*WCPATH*是接收变化的工作拷贝路径，如果省略*WCPATH*，会假定缺省值“.”，除非源有相同基本名称与“.”中的某一文件名字匹配：在这种情况下，区别会应用到那个文件。

Subversion will internally track metadata about the merge operation only if the two sources are ancestrally related—if the first source is an ancestor of the second or vice versa. This is guaranteed to be the case when using the third form. Unlike **svn diff**, the merge command takes the ancestry of a file into consideration when performing a merge operation. This is very important when you're merging changes from one branch into another and you've renamed a file on one branch but not the other.

别名

无

改变

工作拷贝2

访问版本库

只有在对URL操作时会

选项

```
--accept ARG
```

```
--change (-c) REV
--depth ARG
--diff3-cmd CMD
--dry-run
--extensions (-x) ARG
--force
--ignore-ancestry
--quiet (-q)
--record-only
--reintegrate
--revision (-r) REV
```

例子

Merge a branch back into the trunk (assuming that you have an up-to-date working copy of the trunk):

```
$ svn merge --reintegrate \
            http://svn.example.com/repos/calc/branches/my-calc-branch
--- Merging differences between repository URLs into '.':
U    button.c
U    integer.c
U    Makefile
 U   .

$ # build, test, verify, ...

$ svn commit -m "Merge my-calc-branch back into trunk!"
Sending        .
Sending        button.c
Sending        integer.c
Sending        Makefile
Transmitting file data ..
Committed revision 391.
```

合并一个单独文件的修改：

```
$ cd myproj
$ svn merge -r 30:31 thhgttg.txt
U  thhgttg.txt
```

名称

svn mergeinfo — 查询合并相关信息，见第 4.3.3 节 “合并信息和预览” 。

概要

```
svn mergeinfo SOURCE_URL[@REV] [TARGET[@REV]...]
```

描述

Query information related to merges (or potential merges) between *SOURCE-URL* and *TARGET*. If the `--show-revs` option is not provided, display revisions which have been merged from *SOURCE-URL* to *TARGET*. Otherwise, display either `merged` or `eligible` revisions as specified by the `--show-revs` option.

别名

无

改变

无2

访问版本库

是

选项

```
--revision (-r) REV
```

例子

Find out which changesets your trunk directory has already received as well as what changesets it's still eligible to receive:

```
$ svn mergeinfo branches/test
Path: branches/test
  Source path: /trunk
    Merged ranges: r2:13
    Eligible ranges: r13:15
```

名称

svn mkdir — 创建一个纳入版本控制的新目录。

概要

```
svn mkdir PATH...

svn mkdir URL...
```

描述

Create a directory with a name given by the final component of the *PATH* or *URL*. A directory specified by a working copy *PATH* is scheduled for addition in the working copy. A directory specified by a URL is created in the repository via an immediate commit. Multiple directory URLs are committed atomically. In both cases, all the intermediate directories must already exist unless the `--parents` option is used.

别名

无

改变

工作拷贝；如果是对URL操作则会影响版本库

访问版本库

只有在对URl操作时会

选项

```
--editor-cmd EDITOR
--encoding ENC
--file (-F) FILE
--force-log
--message (-m) TEXT
--parents
--quiet (-q)
--with-revprop ARG
```

例子

在工作拷贝创建一个目录：

```
$ svn mkdir newdir
A         newdir
```

Create one in the repository (this is an instant commit, so a log message is required):

```
$ svn mkdir -m "Making a new dir." http://svn.red-bean.com/repos/newdir
```

```
Committed revision 26.
```

名称

svn move — 移动一个文件或目录。

概要

```
svn move SRC... DST
```

描述

这个命令移动文件或目录到你的工作拷贝或者是版本库。

提示

这个命令同**svn copy**加一个**svn delete**等同。

When moving multiple sources, they will be added as children of *DST*, which must be a directory.

注意

Subversion does not support moving between working copies and URLs. In addition, you can only move files within a single repository—Subversion does not support cross-repository moving. Subversion supports the following types of moves within a single repository:

WC → WC
　移动和预订一个文件或目录将要添加(包含历史)。

URL → URL
　完全服务器端的重命名。

别名

mv, **rename**, **ren**
改变

工作拷贝；如果是对URL操作则会影响版本库
访问版本库

只有在对URl操作时会
选项

```
--editor-cmd EDITOR
--encoding ENC
--file (-F) FILE
--force
--force-log
--message (-m) TEXT
--parents
```

```
--quiet (-q)
--revision (-r) REV
--with-revprop ARG
```

例子

移动工作拷bede一个文件：

```
$ svn move foo.c bar.c
A         bar.c
D         foo.c
```

移动工作拷贝的一些文件到子目录：

```
$ svn move baz.c bat.c qux.c src
A         src/baz.c
D         baz.c
A         src/bat.c
D         bat.c
A         src/qux.c
D         qux.c
```

Move a file in the repository (this is an immediate commit, so it requires a commit message):

```
$ svn move -m "Move a file" http://svn.red-bean.com/repos/foo.c \
                            http://svn.red-bean.com/repos/bar.c

Committed revision 27.
```

名称

svn propdel — 删除一个项目的一个属性。

概要

```
svn propdel PROPNAME [PATH...]

svn propdel PROPNAME --revprop -r REV [TARGET]
```

描述

This removes properties from files, directories, or revisions. The first form removes versioned properties in your working copy, and the second removes unversioned remote properties on a repository revision (*TARGET* determines only which repository to access).

别名

pdel, **pd**

改变

工作拷贝；对URL操作时是版本库

访问版本库

只有在对URl操作时会

选项

```
--changelist ARG
--depth ARG
--quiet (-q)
--recursive (-R)
--revision (-r) REV
--revprop
```

例子

删除你的工作拷贝中一个文件的一个属性

```
$ svn propdel svn:mime-type some-script
property 'svn:mime-type' deleted from 'some-script'.
```

删除一个修订版本的属性：

```
$ svn propdel --revprop -r 26 release-date
property 'release-date' deleted from repository revision '26'
```

名称

svn propedit — Edit the property of one or more items under version control. See svn propset later in this chapter.

概要

```
svn propedit PROPNAME TARGET...

svn propedit PROPNAME --revprop -r REV [TARGET]
```

描述

Edit one or more properties using your favorite editor. The first form edits versioned properties in your working copy, and the second edits unversioned remote properties on a repository revision (*TARGET* determines only which repository to access).

别名

pedit, **pe**

改变

工作拷贝；对URL操作时是版本库

访问版本库

只有在对URl操作时会

选项

```
--editor-cmd EDITOR
--encoding ENC
--file (-F) ARG
--force
--force-log
--message (-m) ARG
--revision (-r) REV
--revprop
--with-revprop ARG
```

例子

svn propedit对修改多个值的属性非常简单：

```
$ svn propedit svn:keywords  foo.c
    <svn will launch your favorite editor here, with a buffer open
    containing the current contents of the svn:keywords property.  You
    can add multiple values to a property easily here by entering one
    value per line.>
Set new value for property 'svn:keywords' on 'foo.c'
```

名称

svn propget — 打印一个属性的值。

概要

```
svn propget PROPNAME [TARGET[@REV]...]

svn propget PROPNAME --revprop -r REV [URL]
```

描述

Print the value of a property on files, directories, or revisions. The first form prints the versioned property of an item or items in your working copy, and the second prints unversioned remote properties on a repository revision. See 第 3.2 节 “属性” for more information on properties.

别名

pget, **pg**

改变

工作拷贝；对URL操作时是版本库

访问版本库

只有在对URl操作时会

选项

```
--changelist ARG
--depth ARG
--recursive (-R)
--revision (-r) REV
--revprop
--strict
--xml
```

例子

检查工作拷贝的一个文件的一个属性：

```
$ svn propget svn:keywords foo.c
Author
Date
Rev
```

对于修订版本属性相同：

```
$ svn propget svn:log --revprop -r 20
Began journal.
```

Lastly, you can get **svn propget** output in XML format with the `--xml` option:

```
$ svn propget --xml svn:ignore .
<?xml version="1.0"?>
<properties>
<target
   path="">
<property
   name="svn:ignore">*.o
</property>
</target>
</properties>
```

名称

svn proplist — 列出所有的属性。

概要

```
svn proplist [TARGET[@REV]...]

svn proplist --revprop -r REV [TARGET]
```

描述

List all properties on files, directories, or revisions. The first form lists versioned properties in your working copy, and the second lists unversioned remote properties on a repository revision (*TARGET* determines only which repository to access).

别名

plist, **pl**

改变

工作拷贝；对URL操作时是版本库

访问版本库

只有在对URl操作时会

选项

```
--changelist ARG
--depth ARG
--quiet (-q)
--recursive (-R)
--revision (-r) REV
--revprop
--verbose (-v)
--xml
```

例子

You can use **proplist** to see the properties on an item in your working copy:

```
$ svn proplist foo.c
Properties on 'foo.c':
  svn:mime-type
  svn:keywords
  owner
```

But with the `--verbose` flag, **svn proplist** is extremely handy as it also shows you the values for the properties:

```
$ svn proplist --verbose foo.c
Properties on 'foo.c':
  svn:mime-type : text/plain
  svn:keywords : Author Date Rev
  owner : sally
```

Lastly, you can get **svn proplist** output in xml format with the `--xml` option:

```
$ svn proplist --xml
<?xml version="1.0"?>
<properties>
<target
   path=".">
<property
   name="svn:ignore"/>
</target>
</properties>
```

名称

svn propset — Set *PROPNAME* to *PROPVAL* on files, directories, or revisions.

概要

```
svn propset PROPNAME [PROPVAL | -F VALFILE] PATH...

svn propset PROPNAME --revprop -r REV [PROPVAL | -F VALFILE] [TARGET]
```

描述

设置文件、目录或者修订版本的属性*PROPNAME*为*PROPVAL*。第一个例子在工作拷贝创建了一个版本化的本地属性修改，第二个例子创建了一个未版本化的远程的对版本库修订版本的属性修改(*TARGET*只是用来确定访问哪个版本库)。

提示

Subversion has a number of “special” properties that affect its behavior. See 第 9.10 节 “Subversion 属性” later in this chapter for more on these properties.

别名

pset, *ps*

改变

工作拷贝；对URL操作时是版本库

访问版本库

只有在对URl操作时会

选项

```
--changelist ARG
--depth ARG
--encoding ENC
--file (-F) FILE
--force
--quiet (-q)
--recursive (-R)
--revision (-r) REV
--revprop
--targets FILENAME
```

例子

Set the MIME type for a file:

```
$ svn propset svn:mime-type image/jpeg foo.jpg
```

```
property 'svn:mime-type' set on 'foo.jpg'
```

在UNIX系统，如果你希望一个文件设置执行权限：

```
$ svn propset svn:executable ON somescript
property 'svn:executable' set on 'somescript'
```

或许为了合作者的利益你有一个内部的属性设置：

```
$ svn propset owner sally foo.c
property 'owner' set on 'foo.c'
```

如果你在特定修订版本的日志信息里有一些错误，并且希望修改，可以使用`--revprop`设置`svn:log`为新的日志信息：

```
$ svn propset --revprop -r 25 svn:log "Journaled about trip to New York."
property 'svn:log' set on repository revision '25'
```

或者，你没有工作拷贝，你可以提供一个URL：

```
$ svn propset --revprop -r 26 svn:log "Document nap." \
              http://svn.red-bean.com/repos
property 'svn:log' set on repository revision '25'
```

Lastly, you can tell **propset** to take its input from a file. You could even use this to set the contents of a property to something binary:

```
$ svn propset owner-pic -F sally.jpg moo.c
property 'owner-pic' set on 'moo.c'
```

注意

缺省，你不可以在Subversion版本库修改修订版本属性，你的版本库管理员必须显示的通过创建一个名字为`pre-revprop-change`的钩子来允许修订版本属性修改，关于钩子脚本的详情请见第 5.3.2 节 “实现版本库钩子”。

名称

svn resolved — 解决工作拷贝文件或目录的冲突。

概要

```
svn resolve PATH...
```

描述

Resolve “conflicted” state on working copy files or directories. This routine does not semantically resolve conflict markers; however, it replaces *PATH* with the version specified by the `--accept` argument and then removes conflict-related artifact files. This allows *PATH* to be committed again—that is, it tells Subversion that the conflicts have been “resolved.”. You can pass the following arguments to the `--accept` command depending on your desired resolution:

`base`
这是你的做更新操作以前的BASE版本文件，就是你在上次更新之后未作更改的版本。

`working`
Assuming that you've manually handled the conflict resolution, choose the version of the file as it currently stands in your working copy.

`mine-full`
Resolve all conflicted files with copies of the files as they stood immediately before you ran **svn update**.

`theirs-full`
Resolve all conflicted files with copies of the files that were fetched from the server when you ran **svn update**.

关于解决冲突的深入介绍可以看第 2.4.5 节 “解决冲突(合并别人的修改)” 。

别名

无

改变

工作拷贝2

访问版本库

否

选项

```
--accept ARG
--depth ARG
--quiet (-q)
--recursive (-R)
--targets FILENAME
```

例子

Here's an example where, after a postponed conflict resolution during update, **svn resolve** replaces the all conflicts in file `foo.c` with your edits:

```
$ svn up
Conflict discovered in 'foo.c'.
Select: (p) postpone, (df) diff-full, (e) edit,
        (h) help for more options: p
C    foo.c
Updated to revision 5.

$ svn resolve --accept mine-full foo.c
Resolved conflicted state of 'foo.c'
```

名称

svn resolved — 废弃的。删除工作拷贝文件或目录的废弃的状态。

概要

```
svn resolved PATH...
```

描述

This command has been deprecated in favor of running **svn resolve --accept working *PATH***. See svn resolved in the preceding section for details.

删除工作拷贝文件或目录的“conflicted”状态。这个程序不是语义上的改变冲突标志，它只是删除冲突相关的人造文件，从而重新允许*PATH*提交；也就是说，它告诉Subversion冲突已经“解决了”。关于解决冲突更深入的考虑可以查看第 2.4.5 节 “解决冲突(合并别人的修改)”。

别名

无

改变

工作拷贝2

访问版本库

否

选项

```
--depth ARG
--quiet (-q)
--recursive (-R)
--targets FILENAME
```

例子

如果你在更新时得到冲突，你的工作拷贝会产生三个新的文件：

```
$ svn update
C  foo.c
Updated to revision 31.
$ ls
foo.c
foo.c.mine
foo.c.r30
foo.c.r31
```

当你解决了`foo.c`的冲突，并且准备提交，运行**svn resolved**让你的工作拷贝知道你已经完成了所有事情。

警告

你可以仅仅删除冲突的文件并且提交，但是**svn resolved**除了删除冲突文件，还修正了一些记录在工作拷贝管理区域的记录数据，所以我们推荐你使用这个命令。

名称

svn revert — 取消所有的本地编辑。

概要

```
svn revert PATH...
```

描述

Reverts any local changes to a file or directory and resolves any conflicted states. **svn revert** will revert not only the contents of an item in your working copy, but also any property changes. Finally, you can use it to undo any scheduling operations that you may have performed (e.g., files scheduled for addition or deletion can be “unscheduled”).

别名

无

改变

工作拷贝2

访问版本库

否

选项

```
--changelist ARG
--depth ARG
--quiet (-q)
--recursive (-R)
--targets FILENAME
```

例子

丢弃对一个文件的修改：

```
$ svn revert foo.c
Reverted foo.c
```

如果你希望恢复一整个目录的文件，可以使用`--depth=infinity`选项：

```
$ svn revert --depth=infinity .
Reverted newdir/afile
Reverted foo.c
Reverted bar.txt
```

最后，你可以取消预定的操作：

```
$ svn add mistake.txt whoops
A         mistake.txt
A         whoops
A         whoops/oopsie.c

$ svn revert mistake.txt whoops
Reverted mistake.txt
Reverted whoops

$ svn status
?      mistake.txt
?      whoops
```

警告

svn revert本身有固有的危险，因为它的目的是放弃数据—未提交的修改。一旦你选择了恢复，Subversion没有方法找回未提交的修改。

如果你没有给**svn revert**提供了目标，它不会做任何事情—为了保护你不小心失去对工作拷贝的修改，**svn revert**需要你提供至少一个目标。

名称

svn status — 打印工作拷贝文件和目录的状态。

概要

```
svn status [PATH...]
```

描述

打印工作拷贝文件和目录的状态。如果没有参数，只会打印本地修改的项目(不会访问版本库)，使用`--show-updates`选项，会添加工作修订版本和服务器过期信息。使用`--verbose`会打印每个项目的完全修订版本信息，使用`--quiet`，会打印本地修改条目的总结信息。

输出的前六列都是一个字符宽，每一列给出了工作拷贝项目的每一方面的信息。

The first column indicates that an item was added, deleted, or otherwise changed:

`' '`
没有修改。

`'A'`
预定要添加的项目。

`'D'`
预定要删除的项目。

`'M'`
项目已经修改了。

`'R'`
项目在工作拷贝中已经被替换了。这意味着文件预定要删除，然后有一个同样名称的文件要在同一个位置替换它。

`'C'`
项目的内容(相对于属性)与更新得到的数据冲突了。

`'X'`
项目与外部定义相关。

`'I'`
项目被忽略(例如使用`svn:ignore`属性)。

`'?'`
项目不在版本控制之下。

`'!'`
Item is missing (e.g., you moved or deleted it without using **svn**). This also indicates that a directory is incomplete (a checkout or update was interrupted).

'~'
Item is versioned as one kind of object (file, directory, link), but has been replaced by a different kind of object.

The second column tells the status of a file's or directory's properties:

' '
没有修改。

'M'
这个项目的属性已经修改。

'C'
这个项目的属性与从版本库得到的更新有冲突。

The third column is populated only if the working copy directory is locked (see 第 2.6 节 “有时你只需要清理”):

' '
项目没有锁定。

'L'
项目已经锁定。

The fourth column is populated only if the item is scheduled for addition-with-history:

' '
没有历史预定要提交。

'+'
历史预定要伴随提交。

The fifth column is populated only if the item is switched relative to its parent (see 第 4.5 节 “使用分支”):

' '
项目是它的父目录的孩子。

'S'
项目已经转换。

The sixth column is populated with lock information:

' '
当使用--show-updates，文件没有锁定。如果不使用--show-updates，这意味着文件在工作拷贝被锁定。

K
文件锁定在工作拷贝。

O
文件被另一个用户或另一个工作拷贝锁定，只有在使用--show-updates时显示。

T
文件锁定在工作拷贝，但是锁定被“窃取”而不可用。文件当前锁定在版本库，只有在使用--show-updates时显示。

B
文件锁定在工作拷贝，但是锁定被“破坏”而不可用。文件当前锁定在版本库，只有在使用--show-updates时显示。

The out-of-date information appears in the seventh column (only if you pass the --show-updates option):

' '
这个项目在工作拷贝是最新的。

'*'
在服务器这个项目有了新的修订版本。

The remaining fields are variable width and delimited by spaces. The working revision is the next field if the --show-updates or --verbose option is passed.

如果传递--verbose选项，最后提交的修订版本和最后的提交作者会在后面显示。

工作拷贝路径永远是最后一个字段，所以它可以包括空格。
别名

stat, **st**
改变

无2
访问版本库

只有使用--show-updates时会访问
选项

```
--changelist ARG
--depth ARG
--ignore-externals
--incremental
--no-ignore
--quiet (-q)
--show-updates (-u)
--verbose (-v)
--xml
```

例子

这是查看你在工作拷贝所做的修改的最简单的方法。

```
$ svn status wc
```

```
 M     wc/bar.c
A  +   wc/qax.c
```

If you want to find out what files in your working copy are out of date, pass the `--show-updates` option (this will *not* make any changes to your working copy). Here you can see that `wc/foo.c` has changed in the repository since we last updated our working copy:

```
$ svn status --show-updates wc
 M           965    wc/bar.c
       *     965    wc/foo.c
A  +         965    wc/qax.c
Status against revision:    981
```

注意

`--show-updates`*only* places an asterisk next to items that are out of date (i.e., items that will be updated from the repository if you later use **svn update**). `--show-updates` does *not* cause the status listing to reflect the repository's version of the item (although you can see the revision number in the repository by passing the `--verbose` option).

The most information you can get out of the status subcommand is as follows:

```
$ svn status --show-updates --verbose wc
 M           965       938 sally        wc/bar.c
       *     965       922 harry        wc/foo.c
A  +         965       687 harry        wc/qax.c
             965       687 harry        wc/zig.c
Head revision:   981
```

Lastly, you can get **svn status** output in XML format with the `--xml` option:

```
$ svn status --xml wc
<?xml version="1.0"?>
<status>
<target
   path="wc">
<entry
   path="qax.c">
<wc-status
   props="none"
   item="added"
   revision="0">
</wc-status>
</entry>
<entry
   path="bar.c">
<wc-status
```

```
   props="normal"
   item="modified"
   revision="965">
<commit
   revision="965">
<author>sally</author>
<date>2008-05-28T06:35:53.048870Z</date>
</commit>
</wc-status>
</entry>
</target>
</status>
```

关于**svn status**的更多例子可以见第 2.4.3.1 节 “查看你的修改概况”。

名称

svn switch — 把工作拷贝更新到别的URL。

概要

```
svn switch URL[@PEGREV] [PATH]

switch --relocate FROM TO [PATH...]
```

描述

这个子命令(没有--relocate选项)更新你的工作拷贝来反映新的URL—通常是一个与你的工作拷贝分享共同祖先的URL，尽管这不是必需的。这是Subversion移动工作拷贝到分支的方式。更深入的了解请见第 4.5 节 “使用分支”。

If --force is used, unversioned obstructing paths in the working copy do not automatically cause a failure if the switch attempts to add the same path. If the obstructing path is the same type (file or directory) as the corresponding path in the repository, it becomes versioned but its contents are left untouched in the working copy. This means that an obstructing directory's unversioned children may also obstruct and become versioned. For files, any content differences between the obstruction and the repository are treated like a local modification to the working copy. All properties from the repository are applied to the obstructing path.

As with most subcommands, you can limit the scope of the switch operation to a particular tree depth using the --depth option. Alternatively, you can use the --set-depth option to set a new “sticky” working copy depth on the switch target. Currently, the depth of a working copy directory can only be increased (telescoped more deeply); you cannot make a directory more shallow.

--relocate选项导致**svn switch**做不同的事情：它更新你的工作拷贝指向到同一个版本库目录，但是不同的URL(通常因为管理员将版本库转移了服务器，或到了同一个服务器的另一个URL)。

别名

sw

改变

工作拷贝2

访问版本库

是

选项

```
--accept ARG
--depth ARG
--diff3-cmd CMD
--force
--ignore-externals
--quiet (-q)
```

```
--relocate
--revision (-r) REV
--set-depth ARG
```

例子

如果你目前所在目录vendors分支到vendors-with-fix，你希望转移到那个分支：

```
$ svn switch http://svn.red-bean.com/repos/branches/vendors-with-fix .
U  myproj/foo.txt
U  myproj/bar.txt
U  myproj/baz.c
U  myproj/qux.c
Updated to revision 31.
```

To switch back, just provide the URL to the location in the repository from which you originally checked out your working copy:

```
$ svn switch http://svn.red-bean.com/repos/trunk/vendors .
U  myproj/foo.txt
U  myproj/bar.txt
U  myproj/baz.c
U  myproj/qux.c
Updated to revision 31.
```

提示

You can switch just part of your working copy to a branch if you don't want to switch your entire working copy.

有时候管理员会修改版本库的“基本位置”—换句话说，版本库的内容并不改变，但是访问根的主URL变了。举个例子，主机名变了、URL模式变了或者是URL中的任何一部分改变了。我们不选择重新检出一个工作拷贝，你可以使用**svn switch**来重写版本库所有URL的开头。如果使用带--relocate选项的**svn switch**来做这种替换，Subversion会访问版本库来验证重定位请求(当然是查看新URL的版本库)，然后重写元数据。此类switch操作不会修改文件内容—是一个只修改元数据的工作拷贝修改。

```
$ svn checkout file:///var/svn/repos test
A  test/a
A  test/b
…

$ mv repos newlocation
$ cd test/

$ svn update
svn: Unable to open an ra_local session to URL
```

```
svn: Unable to open repository 'file:///var/svn/repos'

$ svn switch --relocate file:///var/svn/repos file:///tmp/newlocation .
$ svn update
At revision 3.
```

警告

小心使用`--relocate`选项，如果你输入了错误的选项，你会在工作拷贝创建无意义的URL，会导致整个工作区不可用并且难于修复。理解何时应该使用`--relocate`也是非常重要的，下面是一些规则：

- If the working copy needs to reflect a new directory *within* the repository, use just **svn switch**.
- If the working copy still reflects the same repository directory, but the location of the repository itself has changed, use **svn switch** with the `--relocate` option.

名称

svn unlock — 解锁工作拷贝路径或URL。

概要

```
svn unlock TARGET...
```

描述

Unlock each *TARGET*. If any *TARGET* is locked by another user or no valid lock token exists in the working copy, print a warning and continue unlocking the rest of the *TARGET*s. Use `--force` to break a lock belonging to another user or working copy.

别名

无

改变

工作拷贝，版本库

访问版本库

是

选项

```
--force
--targets FILENAME
```

例子

解锁工作拷贝中的两个文件：

```
$ svn unlock tree.jpg house.jpg
'tree.jpg' unlocked.
'house.jpg' unlocked.
```

解锁工作拷贝的一个被其他用户锁定的文件：

```
$ svn unlock tree.jpg
svn: 'tree.jpg' is not locked in this working copy
$ svn unlock --force tree.jpg
'tree.jpg' unlocked.
```

没有工作拷贝时解锁一个文件：

```
$ svn unlock http://svn.red-bean.com/repos/test/tree.jpg
'tree.jpg unlocked.
```

更多细节见第 3.7 节 “锁定” 。

名称

svn update — 更新你的工作拷贝。

概要

```
svn update [PATH...]
```

描述

svn update会把版本库的修改带到工作拷贝，如果没有给定修订版本，它会把你的工作拷贝更新到HEAD修订版本，否则，它会把工作拷贝更新到你用--revision指定的修订版本。为了保持同步，**svn update**也会删除所有在工作拷贝发现的无效锁定(见第 2.6 节 “有时你只需要清理”)。

对于每一个更新的项目开头都有一个表示所做动作的字符，这些字符有下面的意思：

A
添加

B
Broken lock (third column only)

D
删除

U
更新

C
冲突

G
合并

E
Existed

A character in the first column signifies an update to the actual file, whereas updates to the file's properties are shown in the second column. Lock information is printed in the third column.

As with most subcommands, you can limit the scope of the update operation to a particular tree depth using the --depth option. Alternatively, you can use the --set-depth option to set a new “sticky” working copy depth on the update target. Currently, the depth of a working copy directory can only be increased (telescoped more deeply); you cannot make a directory more shallow.

别名

up

改变

工作拷贝2

访问版本库

是
选项

```
--accept ARG
--changelist
--depth ARG
--diff3-cmd CMD
--editor-cmd ARG
--force
--ignore-externals
--quiet (-q)
--revision (-r) REV
--set-depth ARG
```

例子

获取你上次更新之后版本库的修改：

```
$ svn update
A  newdir/toggle.c
A  newdir/disclose.c
A  newdir/launch.c
D  newdir/README
Updated to revision 32.
```

你也可以将工作拷贝更新到旧的修订版本(Subversion没有CVS的“sticky”文件的概念；见附录 B, *CVS* 用户的 *Subversion* 指南)：

```
$ svn update -r30
A  newdir/README
D  newdir/toggle.c
D  newdir/disclose.c
D  newdir/launch.c
U  foo.c
Updated to revision 30.
```

提示

如果你希望检查单个文件的旧的修订版本，你会希望使用**svn cat**。

9.2. svnadmin

svnadmin is the administrative tool for monitoring and repairing your Subversion repository. For detailed information on repository administration, see the maintenance section for 第 5.4.1.1 节 “svnadmin”.

因为**svnadmin**直接访问版本库(因此只可以在存放版本库的机器上使用)，它通过路径访问版本库，而不是URL。

9.2.1. svnadmin 选项

Options in **svmadin** are global, just as they are in **svn**:

`--bdb-log-keep`
(Berkeley DB-specific.) Disable automatic log removal of database logfiles. Having these logfiles around can be convenient if you need to restore from a catastrophic repository failure.

`--bdb-txn-nosync`
(Berkeley DB特定)在提交数据库事务时关闭fsync。可以在**svnadmin create**命令创建Berkeley DB后端时开启`DB_TXN_NOSYNC`(可以改进速度，但是有相关的风险)。

`--bypass-hooks`
绕过版本库钩子系统。

`--clean-logs`
删除不使用的Berkeley DB日志。

`--force-uuid`
缺省情况下，当版本库加载已经包含修订版本的数据时**svnadmin**会忽略流中的`UUID`，这个选项会导致版本库的`UUID`设置为流的`UUID`。

`--ignore-uuid`
缺省情况下，当加载空版本库时，**svnadmin**会使用来自流中的`UUID`，这个选项会导致忽略UUID(如果你的配置文件已经设置了`--force-uuid`，将会用于将其覆盖)。

`--incremental`
导出一个修订版本针对前一个修订版本的区别，而不是通常的完全结果。

`--parent-dir` *`DIR`*
当加载一个转储文件时，根路径为*`DIR`*而不是`/`。

`--pre-1.4-compatible`
When creating a new repository, use a format that is compatible with versions of Subversion earlier than Subversion 1.4.

`--pre-1.5-compatible`
When creating a new repository, use a format that is compatible with versions of Subversion earlier than Subversion 1.5.

`--revision` (`-r`) *`ARG`*
指定一个操作的修订版本。

`--quiet`
不显示通常的过程—只显示错误。

`--use-post-commit-hook`
When loading a dump file, runs the repository's `post-commit` hook after finalizing each newly loaded revision.

`--use-post-revprop-change-hook`
When changing a revision property, runs the repository's `post-revprop-change` hook after changing the revision property.

`--use-pre-commit-hook`
When loading a dump file, runs the repository's `pre-commit` hook before finalizing each newly loaded revision. If the hook fails, aborts the commit and terminates the load process.

`--use-pre-revprop-change-hook`
When changing a revision property, runs the repository's `pre-revprop-change` hook before changing the revision property. If the hook fails, aborts the modification and terminates.

9.2.2. svnadmin 子命令

Here are the various subcommands for the **svnadmin** program.

名称

svnadmin crashtest — Simulate a process that crashes.

概要

```
svnadmin crashtest REPOS_PATH
```

描述

Open the repository at *REPOS_PATH*, then abort, thus simulating a process that crashes while holding an open repository handle. This is used for testing automatic repository recovery (a new feature in Berkeley DB 4.4). It's unlikely that you'll need to run this command.

选项

无

例子

```
$ svnadmin crashtest /var/svn/repos
Aborted
```

Exciting, isn't it?

名称

svnadmin create — 创建一个新的空的版本库。

概要

```
svnadmin create REPOS_PATH
```

描述

Create a new, empty repository at the path provided. If the provided directory does not exist, it will be created for you. [1] As of Subversion 1.2, **svnadmin** creates new repositories with the `FSFS` filesystem backend by default.

While **svnadmin create** will create the base directory for a new repository, it will not create intermediate directories. For example, if you have an empty directory named `/var/svn`, creating `/var/svn/repos` will work, while attempting to create `/var/svn/subdirectory/repos` will fail with an error.

选项

```
--bdb-log-keep
--bdb-txn-nosync
--config-dir DIR
--fs-type TYPE
--pre-1.4-compatible
--pre-1.5-compatible
```

例子

Creating a new repository is this easy:

```
$ svnadmin create /var/svn/repos
```

在Subversion 1.0，一定会创建一个Berkeley DB版本库，在Subversion 1.1，Berkeley DB版本库是缺省类型，但是一个FSFS版本库也是可以创建，使用`--fs-type`选项：

```
$ svnadmin create /var/svn/repos --fs-type fsfs
```

[1] 记住**svnadmin**只工作在本地路径，而不是*URL*。

名称

svnadmin deltify — 修订版本范围的路径的增量变化。

概要

```
svnadmin deltify [-r LOWER[:UPPER]] REPOS_PATH
```

描述

svnadmin deltify因为历史原因而存在，这个命令已经废弃，不再需要。

它开始于当Subversion提供了管理员控制版本库压缩策略的能力，结果是复杂工作得到了非常小的收益，所以这个“特性”被废弃了。

选项

```
--quiet (-q)
--revision (-r) REV
```

名称

svnadmin dump — Dump the contents of the filesystem to `stdout`.

概要

```
svnadmin dump REPOS_PATH [-r LOWER[:UPPER]] [--incremental]
```

描述

Dump the contents of the filesystem to `stdout` in a “dump file” portable format, sending feedback to `stderr`. Dump revisions *`LOWER`* rev through *`UPPER`* rev. If no revisions are given, dump all revision trees. If only *`LOWER`* is given, dump that one revision tree. See 第 5.4.5 节 “版本库数据的移植” for a practical use.

By default, the Subversion dump stream contains a single revision (the first revision in the requested revision range) in which every file and directory in the repository in that revision is presented as though that whole tree was added at once, followed by other revisions (the remainder of the revisions in the requested range), which contain only the files and directories that were modified in those revisions. For a modified file, the complete full-text representation of its contents, as well as all of its properties, are presented in the dump file; for a directory, all of its properties are presented.

Two useful options modify the dump file generator's behavior. The first is the `--incremental` option, which simply causes that first revision in the dump stream to contain only the files and directories modified in that revision, instead of being presented as the addition of a new tree, and in exactly the same way that every other revision in the dump file is presented. This is useful for generating a relatively small dump file to be loaded into another repository that already has the files and directories that exist in the original repository.

The second useful option is `--deltas`. This option causes **svnadmin dump** to, instead of emitting full-text representations of file contents and property lists, emit only deltas of those items against their previous versions. This reduces (in some cases, drastically) the size of the dump file that **svnadmin dump** creates. There are, however, disadvantages to using this option—deltified dump files are more CPU-intensive to create, cannot be operated on by **svndumpfilter**, and tend not to compress as well as their nondeltified counterparts when using third-party tools such as **gzip** and **bzip2**.

选项

```
--deltas
--incremental
--quiet (-q)
--revision (-r) REV
```

例子

转储整个版本库：

```
$ svnadmin dump /var/svn/repos > full.dump
* Dumped revision 0.
```

```
* Dumped revision 1.
* Dumped revision 2.
…
```

从版本库增量转储一个单独的事务：

```
$ svnadmin dump /var/svn/repos -r 21 --incremental > incr.dump
* Dumped revision 21.
```

名称

svnadmin help — 求助!

概要

```
svnadmin help [SUBCOMMAND...]
```

描述

This subcommand is useful when you're trapped on a desert island with neither a Net connection nor a copy of this book.

别名

?, **h**

名称

svnadmin hotcopy — 制作一个版本库的热备份。

概要

```
svnadmin hotcopy REPOS_PATH NEW_REPOS_PATH
```

描述

This subcommand makes a full “hot” backup of your repository, including all hooks, configuration files, and, of course, database files. If you pass the `--clean-logs` option, **svnadmin** will perform a hot copy of your repository, and then remove unused Berkeley DB logs from the original repository. You can run this command at any time and make a safe copy of the repository, regardless of whether other processes are using the repository.

选项

`--clean-logs`

警告

就像第 5.2.3.1 节 “Berkeley DB”描述的，热拷贝的Berkeley DB版本库不能跨操作系统移植，也不能在不同“字节续”的主机上工作。

名称

svnadmin list-dblogs — Ask Berkeley DB which logfiles exist for a given Subversion repository (applies only to repositories using the `bdb` backend).

概要

```
svnadmin list-dblogs REPOS_PATH
```

描述

Berkeley DB creates logs of all changes to the repository, which allow it to recover in the face of catastrophe. Unless you enable `DB_LOG_AUTOREMOVE`, the logfiles accumulate, although most are no longer used and can be deleted to reclaim disk space. See 第 5.4.3 节 “管理磁盘空间” for more information.

名称

svnadmin list-unused-dblogs — Ask Berkeley DB which logfiles can be safely deleted (applies only to repositories using the `bdb` backend).

概要

```
svnadmin list-unused-dblogs REPOS_PATH
```

描述

Berkeley DB creates logs of all changes to the repository, which allow it to recover in the face of catastrophe. Unless you enable `DB_LOG_AUTOREMOVE`, the logfiles accumulate, although most are no longer used and can be deleted to reclaim disk space. See 第 5.4.3 节 “管理磁盘空间” for more information.

例子

Remove all unused logfiles from the repository:

```
$ svnadmin list-unused-dblogs /var/svn/repos
/var/svn/repos/log.0000000031
/var/svn/repos/log.0000000032
/var/svn/repos/log.0000000033

$ svnadmin list-unused-dblogs /var/svn/repos | xargs rm
## disk space reclaimed!
```

名称

svnadmin load — Read a repository dump stream from `stdin`.

概要

```
svnadmin load REPOS_PATH
```

描述

Read a repository dump stream from `stdin`, committing new revisions into the repository's filesystem. Send progress feedback to `stdout`.

选项

```
--force-uuid
--ignore-uuid
--parent-dir
--quiet (-q)
--use-post-commit-hook
--use-pre-commit-hook
```

例子

这里显示了加载一个备份文件到版本库(当然，使用**svnadmin dump**)：

```
$ svnadmin load /var/svn/restored < repos-backup
<<< Started new txn, based on original revision 1
     * adding path : test ... done.
     * adding path : test/a ... done.
…
```

或者你希望加载到一个子目录：

```
$ svnadmin load --parent-dir new/subdir/for/project \
                /var/svn/restored < repos-backup
<<< Started new txn, based on original revision 1
     * adding path : test ... done.
     * adding path : test/a ... done.
…
```

名称

svnadmin lslocks — 打印所有锁定的描述。

概要

```
svnadmin lslocks REPOS_PATH [PATH-IN-REPOS]
```

描述

Print descriptions of all locks in repository *REPOS_PATH* underneath the path *PATH-IN-REPOS*. If *PATH-IN-REPOS* is not provided, it defaults to the root directory of the repository.

选项

无

例子

显示了版本库/var/svn/repos中一个锁定的文件：

```
$ svnadmin lslocks /var/svn/repos
Path: /tree.jpg
UUID Token: opaquelocktoken:ab00ddf0-6afb-0310-9cd0-dda813329753
Owner: harry
Created: 2005-07-08 17:27:36 -0500 (Fri, 08 Jul 2005)
Expires:
Comment (1 line):
Rework the uppermost branches on the bald cypress in the foreground.
```

名称

svnadmin lstxns — 打印所有未提交的事物名称。

概要

```
svnadmin lstxns REPOS_PATH
```

描述

打印所有未提交的事物名称。关于未提交事物是怎样创建和如何使用的信息见第 5.4.3.2 节“删除终止的事务”。

例子

List all outstanding transactions in a repository:

```
$ svnadmin lstxns /var/svn/repos/
1w
1x
```

名称

svnadmin recover — Bring a repository database back into a consistent state (applies only to repositories using the `bdb` backend). In addition, if `repos/conf/passwd` does not exist, it will create a default passwordfile .

概要

```
svnadmin recover REPOS_PATH
```

描述

在你得到的错误说明你需要恢复版本库时运行这个命令。

选项

```
--wait
```

例子

恢复挂起的版本库：

```
$ svnadmin recover /var/svn/repos/
Repository lock acquired.
Please wait; recovering the repository may take some time...

Recovery completed.
The latest repos revision is 34.
```

Recovering the database requires an exclusive lock on the repository. (This is a “database lock”; see the sidebar 锁定的三种含义.) If another process is accessing the repository, then **svnadmin recover** will error:

```
$ svnadmin recover /var/svn/repos
svn: Failed to get exclusive repository access; perhaps another process
such as httpd, svnserve or svn has it open?

$
```

`--wait`选项可以导致**svnadmin recover**一直等待其它进程断开连接：

```
$ svnadmin recover /var/svn/repos --wait
Waiting on repository lock; perhaps another process has it open?

### time goes by…

Repository lock acquired.
```

```
Please wait; recovering the repository may take some time...

Recovery completed.
The latest repos revision is 34.
```

名称

svnadmin rmlocks — 无条件的删除版本库的一个或多个锁定。

概要

```
svnadmin rmlocks REPOS_PATH LOCKED_PATH...
```

描述

Remove one or more locks from each *LOCKED_PATH*.

选项

无

例子

这删除了版本库/var/svn/repos里tree.jpg和house.jpg文件上的锁定:

```
$ svnadmin rmlocks /var/svn/repos tree.jpg house.jpg
Removed lock on '/tree.jpg.
Removed lock on '/house.jpg.
```

名称

svnadmin rmtxns — 从版本库删除事物。

概要

```
svnadmin rmtxns REPOS_PATH TXN_NAME...
```

描述

删除版本库的事物，更多细节在第 5.4.3.2 节 “删除终止的事务”。

选项

```
--quiet (-q)
```

例子

删除命名的事物：

```
$ svnadmin rmtxns /var/svn/repos/ 1w 1x
```

很幸运，**lstxns**的输出作为**rmtxns**输入工作良好：

```
$ svnadmin rmtxns /var/svn/repos/  `svnadmin lstxns /var/svn/repos/`
```

从版本库删除所有未提交的事务。

名称

svnadmin setlog — 设置某个修订版本的日志信息。

概要

```
svnadmin setlog REPOS_PATH -r REVISION FILE
```

描述

设置修订版本*REVISION*的日志信息为*FILE*的内容。

This is similar to using **svn propset** with the `--revprop` option to set the `svn:log` property on a revision, except that you can also use the option `--bypass-hooks` to avoid running any pre- or post-commit hooks, which is useful if the modification of revision properties has not been enabled in the `pre-revprop-change` hook.

警告

修订版本属性不在版本控制之下的，所以这个命令会永久覆盖前一个日志信息。

选项

```
--bypass-hooks
--revision (-r) REV
```

例子

设置修订版本19的日志信息为文件`msg`的内容：

```
$ svnadmin setlog /var/svn/repos/ -r 19 msg
```

名称

svnadmin setrevprop — Set a property on a revision.

概要

```
svnadmin setrevprop REPOS_PATH -r REVISION NAME FILE
```

描述

Set the property *NAME* on revision *REVISION* to the contents of *FILE*. Use `--use-pre-revprop-change-hook` or `--use-post-revprop-change-hook` to trigger the revision property-related hooks (e.g., if you want an email notification sent from your `post-revprop-change-hook`).

选项

```
--revision (-r) ARG
--use-post-revprop-change-hook
--use-pre-revprop-change-hook
```

例子

The following sets the revision property `repository-photo` to the contents of the file `sandwich.png`:

```
$svnadmin setrevprop /var/svn/repos -r 0 repository-photo sandwich.png
```

As you can see, **svnadmin setrevprop** has no output upon success.

名称

svnadmin setuuid — 重置版本库的UUID。

概要

```
svnadmin setuuid REPOS_PATH [NEW_UUID]
```

描述

Reset the repository UUID for the repository located at *REPOS_PATH*. If *NEW_UUID* is provided, use that as the new repository UUID; otherwise, generate a brand-new UUID for the repository.

选项

无

例子

If you've **svnsync**ed `/var/svn/repos` to `/var/svn/repos-new` and intend to use `repos-new` as your canonical repository, you may want to change the UUID for `repos-new` to the UUID of `repos` so that your users don't have to check out a new working copy to accommodate the change:

```
$ svnadmin setuuid /var/svn/repos-new 2109a8dd-854f-0410-ad31-d604008985ab
```

As you can see, **svnadmin setuuid** has no output upon success.

名称

svnadmin upgrade — Upgrade a repository to the latest supported schema version.

概要

```
svnadmin upgrade REPOS_PATH
```

描述

Upgrade the repository located at *`REPOS_PATH`* to the latest supported schema version.

This functionality is provided as a convenience for repository administrators who wish to make use of new Subversion functionality without having to undertake a potentially costly full repository dump and load operation. As such, the upgrade performs only the minimum amount of work needed to accomplish this while still maintaining the integrity of the repository. While a dump and subsequent load guarantee the most optimized repository state, **svnadmin upgrade** does not.

警告

You should *always* back up your repository before upgrading.

选项

无

例子

Upgrade the repository at path `/var/repos/svn`:

```
$ svnadmin upgrade /var/repos/svn
Repository lock acquired.
Please wait; upgrading the repository may take some time...

Upgrade completed.
```

名称

svnadmin verify — 验证版本库保存的数据。

概要

```
svnadmin verify REPOS_PATH
```

描述

Run this command if you wish to verify the integrity of your repository. This basically iterates through all revisions in the repository by internally dumping all revisions and discarding the output—it's a good idea to run this on a regular basis to guard against latent hard disk failures and “bitrot.” If this command fails—which it will do at the first sign of a problem—that means your repository has at least one corrupted revision, and you should restore the corrupted revision from a backup (you did make a backup, didn't you?).

选项

```
--quiet (-q)
--revision (-r) ARG
```

例子

检验挂起的版本库：

```
$ svnadmin verify /var/svn/repos/
* Verified revision 1729.
```

9.3. svnlook

svnlook是检验Subversion版本库不同方面的命令行工具，它不会对版本库有任何修改—它只是用来“看”—**svnlook**通常被版本库钩子使用，但是版本库管理也会发现它在诊断目的上也非常有用。

Since **svnlook** works via direct repository access (and thus can be used only on the machine that holds the repository), it refers to the repository with a path, not a URL.

如果没有指定修订版本或事物，**svnlook**缺省的是版本库最年轻的(最新的)修订版本。

9.3.1. svnlook 选项

Options in **svnlook** are global, just as they are in **svn** and **svnadmin**; however, most options apply to only one subcommand since the functionality of **svnlook** is (intentionally) limited in scope:

```
--copy-info
```

Causes **svnlook changed** to show detailed copy source information.

`--no-diff-deleted`
防止**svnlook diff**打印删除文件的区别，缺省的行为方式是当一个文件在一次事物/修订版本中删除后，得到的结果与保留这个文件的内容变成空相同。

`--no-diff-added`
防止**svnlook diff**打印添加文件的区别。缺省的行为方式是，当添加一个文件时，**svn diff**打印的信息和比较一个空白文件相同。

`--revision (-r)`
Specifies a particular revision number that you wish to examine.

`--revprop`
操作针对修订版本属性，而不是Subversion文件或目录的属性。这个选项需要你传递`--revision(-r)`参数。

`--transaction (-t)`
Specifies a particular transaction ID that you wish to examine.

`--show-ids`
Shows the filesystem node revision IDs for each path in the filesystem tree.

9.3.2. svnlook 子命令

Here are the various subcommands for the **svnlook** program.

名称

svnlook author — 打印作者。

概要

```
svnlook author REPOS_PATH
```

描述

打印版本库一个修订版本或者事物的作者。

选项

```
--revision (-r) REV
--transaction (-t) TXN
```

例子

svnlook author垂手可得，但是并不令人激动：

```
$ svnlook author -r 40 /var/svn/repos
sally
```

名称

svnlook cat — 打印一个文件的内容。

概要

```
svnlook cat REPOS_PATH PATH_IN_REPOS
```

描述

打印一个文件的内容。

选项

```
--revision (-r) REV
--transaction (-t) TXN
```

例子

这会显示事物`ax8`中一个文件的内容，位于`/trunk/README`:

```
$ svnlook cat -t ax8 /var/svn/repos /trunk/README

               Subversion, a version control system.
               =====================================

$LastChangedDate: 2003-07-17 10:45:25 -0500 (Thu, 17 Jul 2003) $

Contents:

     I. A FEW POINTERS
    II. DOCUMENTATION
   III. PARTICIPATING IN THE SUBVERSION COMMUNITY
…
```

名称

svnlook changed — 打印修改的路径。

概要

```
svnlook changed REPOS_PATH
```

描述

打印在特定修订版本或事物修改的路径，也是在前两列使用“svn update样式的”状态字符：

'A '
  条目添加到版本库

'D '
  条目从版本库删除

'U '
  文件内容改变了

'_U'
  Properties of item changed; note the leading underscore

'UU'
  文件内容和属性修改了

文件和目录可以区分，目录路径后面会显示字符“/”。

选项

```
--copy-info
--revision (-r) REV
--transaction (-t) TXN
```

例子

这里显示了在测试版本库中修订版本39改变的文件和目录，注意修改的第一个项目是一个目录，证据就是结尾的/：

```
$ svnlook changed -r 39 /var/svn/repos
A   trunk/vendors/deli/
A   trunk/vendors/deli/chips.txt
A   trunk/vendors/deli/sandwich.txt
A   trunk/vendors/deli/pickle.txt
U   trunk/vendors/baker/bagel.txt
_U  trunk/vendors/baker/croissant.txt
UU  trunk/vendors/baker/pretzel.txt
D   trunk/vendors/baker/baguette.txt
```

如下是显示文件重命名修订版本的例子：

```
$ svnlook changed -r 64 /var/svn/repos
A   trunk/vendors/baker/toast.txt
D   trunk/vendors/baker/bread.txt
```

Unfortunately, nothing in the preceding output reveals the connection between the deleted and added files. Use the `--copy-info` option to make this relationship more apparent:

```
$ svnlook changed -r 64 --copy-info /var/svn/repos
A + trunk/vendors/baker/toast.txt
    (from trunk/vendors/baker/bread.txt:r63)
D   trunk/vendors/baker/bread.txt
```

名称

svnlook date — 打印时间戳。

概要

```
svnlook date REPOS_PATH
```

描述

打印版本库一个修订版本或事物的时间戳。

选项

```
--revision (-r) REV
--transaction (-t) TXN
```

例子

显示测试版本库修订版本40的日期：

```
$ svnlook date -r 40 /var/svn/repos/
2003-02-22 17:44:49 -0600 (Sat, 22 Feb 2003)
```

名称

svnlook diff — 打印修改的文件和属性的区别。

概要

```
svnlook diff REPOS_PATH
```

描述

打印版本库中GNU样式的文件和属性修改区别。

选项

```
--diff-copy-from
--no-diff-added
--no-diff-deleted
--revision (-r) REV
--transaction (-t) TXN
```

例子

这显示了一个新添加的(空的)文件，一个删除的文件和一个拷贝的文件：

```
$ svnlook diff -r 40 /var/svn/repos/
Copied: egg.txt (from rev 39, trunk/vendors/deli/pickle.txt)

Added: trunk/vendors/deli/soda.txt
===================================================================

Modified: trunk/vendors/deli/sandwich.txt
===================================================================
--- trunk/vendors/deli/sandwich.txt (original)
+++ trunk/vendors/deli/sandwich.txt 2003-02-22 17:45:04.000000000 -0600
@@ -0,0 +1 @@
+Don't forget the mayo!

Modified: trunk/vendors/deli/logo.jpg
===================================================================
(Binary files differ)

Deleted: trunk/vendors/deli/chips.txt
===================================================================

Deleted: trunk/vendors/deli/pickle.txt
===================================================================
```

If a file has a nontextual `svn:mime-type` property, the differences are not explicitly shown.

名称

svnlook dirs-changed — 打印本身修改的目录。

概要

```
svnlook dirs-changed REPOS_PATH
```

描述

打印本身修改(属性编辑)或子文件修改的目录。

选项

```
--revision (-r) REV
--transaction (-t) TXN
```

例子

这显示了在我们的实例版本库中在修订版本40修改的目录：

```
$ svnlook dirs-changed -r 40 /var/svn/repos
trunk/vendors/deli/
```

名称

svnlook help — 求助!

概要

也可以是`svnlook -h`和`svnlook –?`。

描述

Displays the help message for **svnlook**. This command, like its brother, **svn help**, is also your friend, even though you never call it anymore and forgot to invite it to your last party.

选项

无

别名

?, **h**

名称

svnlook history — 打印版本库(如果没有路径，则是根目录)某一个路径的历史。

概要

```
svnlook history REPOS_PATH
            [PATH_IN_REPOS]
```

描述

打印版本库(如果没有路径，则是根目录)某一个路径的历史。

选项

```
--limit (-l) ARG
--revision (-r) REV
--show-ids
```

例子

This shows the history output for the path `/branches/bookstore` as of revision 13 in our sample repository:

```
$ svnlook history -r 13 /var/svn/repos /branches/bookstore --show-ids
REVISION   PATH <ID>
--------   ---------
      13   /branches/bookstore <1.1.r13/390>
      12   /branches/bookstore <1.1.r12/413>
      11   /branches/bookstore <1.1.r11/0>
       9   /trunk <1.0.r9/551>
       8   /trunk <1.0.r8/131357096>
       7   /trunk <1.0.r7/294>
       6   /trunk <1.0.r6/353>
       5   /trunk <1.0.r5/349>
       4   /trunk <1.0.r4/332>
       3   /trunk <1.0.r3/335>
       2   /trunk <1.0.r2/295>
       1   /trunk <1.0.r1/532>
```

名称

svnlook info — 打印作者、时间戳、日志信息大小和日志信息。

概要

```
svnlook info REPOS_PATH
```

描述

打印作者、时间戳、日志信息大小(字节)和日志信息，然后是一个换行符。

选项

```
--revision (-r) REV
--transaction (-t) TXN
```

例子

显示了你的实例版本库在修订版本40的信息输出：

```
$ svnlook info -r 40 /var/svn/repos
sally
2003-02-22 17:44:49 -0600 (Sat, 22 Feb 2003)
16
Rearrange lunch.
```

名称

svnlook lock — 如果版本库路径已经被锁定，描述它。

概要

```
svnlook lock REPOS_PATH PATH_IN_REPOS
```

描述

打印*PATH_IN_REPOS*锁定的所有信息，如果*PATH_IN_REPOS*没有锁定，则不打印任何内容。

选项

无

例子

这描述了文件tree.jpg的锁定。

```
$ svnlook lock /var/svn/repos tree.jpg
UUID Token: opaquelocktoken:ab00ddf0-6afb-0310-9cd0-dda813329753
Owner: harry
Created: 2005-07-08 17:27:36 -0500 (Fri, 08 Jul 2005)
Expires:
Comment (1 line):
Rework the uppermost branches on the bald cypress in the foreground.
```

名称

svnlook log — 日志信息本身， 后接换行。

概要

```
svnlook log REPOS_PATH
```

描述

打印日志信息。

选项

```
--revision (-r) REV
--transaction (-t) TXN
```

例子

这显示了实例版本库在修订版本40的日志输出：

```
$ svnlook log /var/svn/repos/
Rearrange lunch.
```

名称

svnlook propget — 打印版本库中一个路径一个属性的原始值。

概要

```
svnlook propget REPOS_PATH PROPNAME [PATH_IN_REPOS]
```

描述

列出版本库中一个路径一个属性的值。

别名

pg, **pget**

选项

```
--revision (-r) REV
--revprop
--transaction (-t) TXN
```

例子

这显示了HEAD修订版本中文件/trunk/sandwich的“seasonings”属性的值：

```
$ svnlook pg /var/svn/repos seasonings /trunk/sandwich
mustard
```

名称

svnlook proplist — 打印版本化的文件和目录的属性名称和值。

概要

```
svnlook proplist REPOS_PATH [PATH_IN_REPOS]
```

描述

列出版本库中一个路径的属性，使用--verbose选项也会显示所有的属性值。

别名

pl, **plist**

选项

```
--revision (-r) REV
--revprop
--transaction (-t) TXN
--verbose (-v)
```

例子

这显示了HEAD修订版本中/trunk/README的属性名称：

```
$ svnlook proplist /var/svn/repos /trunk/README
  original-author
  svn:mime-type
```

This is the same command as in the preceding example, but this time showing the property values as well:

```
$ svnlook --verbose proplist /var/svn/repos /trunk/README
  original-author : harry
  svn:mime-type : text/plain
```

名称

svnlook tree — 打印树。

概要

```
svnlook tree REPOS_PATH [PATH_IN_REPOS]
```

描述

打印树，从*PATH_IN_REPOS*(如果提供，会作为树的根)开始，可以选择显示节点修订版本ID。

选项

```
--full-paths
--non-recursive (-N)
--revision (-r) REV
--show-ids
--transaction (-t) TXN
```

例子

This shows the tree output (with nodeIDs) for revision 13 in our sample repository:

```
$ svnlook tree -r 13 /var/svn/repos --show-ids
/ <0.0.r13/811>
 trunk/ <1.0.r9/551>
  button.c <2.0.r9/238>
  Makefile <3.0.r7/41>
  integer.c <4.0.r6/98>
 branches/ <5.0.r13/593>
  bookstore/ <1.1.r13/390>
   button.c <2.1.r12/85>
   Makefile <3.0.r7/41>
   integer.c <4.1.r13/109>
```

名称

svnlook uuid — 打印版本库的UUID。

概要

```
svnlook uuid REPOS_PATH
```

描述

打印版本库的UUID，UUID是版本库的*u*niversal *u*nique*id*entifier(全局唯一标示)，Subversion客户端可以使用这个标示区分不同的版本库。

选项

无

例子

```
$ svnlook uuid /var/svn/repos
e7fe1b91-8cd5-0310-98dd-2f12e793c5e8
```

名称

svnlook youngest — 显示最年轻的修订版本号。

概要

```
svnlook youngest REPOS_PATH
```

描述

打印一个版本库最年轻的修订版本号。

选项

无

例子

这显示了在实例版本库显示最年轻的修订版本：

```
$ svnlook youngest /var/svn/repos/
42
```

9.4. svnsync

svnsync是Subversion的远程版本库镜像工具，它允许你把一个版本库的内容录入到另一个。

在任何镜像场景中，有两个版本库：源版本库，镜像(或“sink”)版本库，源版本库就是**svnsync**获取修订版本的库，镜像版本库是源版本库修订版本的目标，两个版本库可以是在本地或远程—它们只是通过URL跟踪。

svnsync进程只需要对源版本库有读权限；它不会尝试修改它。但是很明显，**svnsync**可以读写访问镜像版本库。

警告

svnsync对于不能作为镜像操作一部分的修改非常敏感，为了防止发生这个情况，最好保证**svnsync**是唯一可以修改镜像版本库的进程。

9.4.1. svnsync 选项

Options in **svnsync** are global, just as they are in **svn** and **svnadmin**:

`--config-dir` *DIR*
指导Subversion从指定目录而不是默认位置(用户主目录的`.subversion`)读取配置信息。

`--no-auth-cache`
Prevents caching of authentication information (e.g., username and password) in the Subversion runtime configuration directories.

`--non-interactive`
In the case of an authentication failure or insufficient credentials, prevents prompting for credentials (e.g., username or password). This is useful if you're running Subversion inside an automated script and it's more appropriate to have Subversion fail than to prompt for more information.

`--quiet` (`-q`)
请求客户端在执行操作时只显示重要信息。

`--source-password` *`PASSWD`*
Specifies the password for the Subversion server from which you are syncing. If not provided, or if incorrect, Subversion will prompt you for this information as needed.

`--source-username` *`NAME`*
Specifies the username for the Subversion server from which you are syncing. If not provided, or if incorrect, Subversion will prompt you for this information as needed.

`--sync-password` *`PASSWD`*
Specifies the password for the Subversion server to which you are syncing. If not provided, or if incorrect, Subversion will prompt you for this information as needed.

`--sync-username` *`NAME`*
Specifies the username for the Subversion server to which you are syncing. If not provided, or if incorrect, Subversion will prompt you for this information as needed.

9.4.2. svnsync 子命令

Here are the various subcommands for the **svnsync** program.

名称

svnsync copy-revprops — Copy all revision properties for a particular revision (or range of revisions) from the source repository to the mirror repository.

概要

```
svnsync copy-revprops DEST_URL [REV[:REV2]
```

描述

Because Subversion revision properties can be changed at any time, it's possible that the properties for some revision might be changed after that revision has already been synchronized to another repository. Because the **svnsync synchronize** command operates only on the range of revisions that have not yet been synchronized, it won't notice a revision property change outside that range. Left as is, this causes a deviation in the values of that revision's properties between the source and mirror repositories. **svnsync copy-revprops** is the answer to this problem. Use it to resynchronize the revision properties for a particular revision or range of revisions.

别名

无

选项

```
--config-dir DIR
--no-auth-cache
--non-interactive
--quiet (-q)
--source-password ARG
--source-username ARG
--sync-password ARG
--sync-username ARG
```

例子

为单个修订版本重新同步修订版本属性：

```
$ svnsync copy-revprops file:///var/svn/repos-mirror 6
Copied properties for revision 6.
$
```

名称

svnsync help — 求助!

概要

```
svnsync help
```

描述

This subcommand is useful when you're trapped in a foreign prison with neither a Net connection nor a copy of this book, but you do have a local Wi-Fi network running and you'd like to sync a copy of your repository over to the backup server that Ira The Knife is running over in cell block D.

别名

无

选项

无

名称

svnsync initialize — Initialize a mirror repository for synchronization from the source repository.

概要

```
svnsync initialize MIRROR_URL SOURCE_URL
```

描述

svnsync initialize检验版本库是否满足了新镜像版本库的需求—它必须没有存在的版本历史，并允许修订版本修改—记录镜像版本库与源版本库关联的初始管理信息，这是对即将镜像的版本库的第一个**svnsync**操作。

别名

init

选项

```
--config-dir DIR
--no-auth-cache
--non-interactive
--quiet (-q)
--source-password ARG
--source-username ARG
--sync-password ARG
--sync-username ARG
```

例子

因为无法修改修订版本属性而初始化镜像版本库失败：

```
$ svnsync initialize file:///var/svn/repos-mirror http://svn.example.com/r
svnsync: Repository has not been enabled to accept revision propchanges;
ask the administrator to create a pre-revprop-change hook
$
```

以镜像初始化版本库，包含已创建允许所有修订版本属性修改的`pre-revprop-change`钩子：

```
$ svnsync initialize file:///var/svn/repos-mirror http://svn.example.com/r
Copied properties for revision 0.
$
```

名称

svnsync synchronize — 将所有未完成的修订版本从源版本库转移到镜像版本库。

概要

```
svnsync synchronize DEST_URL
```

描述

The **svnsync synchronize** command does all the heavy lifting of a repository mirroring operation. After consulting with the mirror repository to see which revisions have already been copied into it, it then begins to copy any not-yet-mirrored revisions from the source repository.

svnsync synchronize can be gracefully canceled and restarted.

As of Subversion 1.5, you can limit **svnsync** to a subdirectory of the source repository by specifying the subdirectory as part of the *SOURCE_URL*.

别名

sync

选项

```
--config-dir DIR
--no-auth-cache
--non-interactive
--quiet (-q)
--source-password ARG
--source-username ARG
--sync-password ARG
--sync-username ARG
```

例子

从源版本库拷贝未同步修订版本到镜像版本库：

```
$ svnsync synchronize file:///var/svn/repos-mirror
Committed revision 1.
Copied properties for revision 1.
Committed revision 2.
Copied properties for revision 2.
Committed revision 3.
Copied properties for revision 3.
…
Committed revision 45.
Copied properties for revision 45.
Committed revision 46.
Copied properties for revision 46.
```

```
Committed revision 47.
Copied properties for revision 47.
$
```

9.5. svnserve

当对远程源版本库使用**svnsync**时，使用Subversion的自定义网络协议。

svnserve允许Subversion版本库使用svn网络协议，你可以作为独立服务器进程运行svnserve，或者是使用其它进程，如**inetd**、**xinetd**(也是svn://)或使用svn+ssh://访问方法的**sshd**为你启动进程。

一旦客户端已经选择了一个版本库来传递它的URL，**svnserve**会读取版本库目录的conf/svnserve.conf文件，来检测版本库特定的设置，如使用哪个认证数据库和应用怎样的授权策略。关于svnserve.conf文件的详情见第 6.3 节 “svnserve - 定制的服务器 ”。

9.5.1. svnserve 选项

不象前面描述的命令，**svnserve**没有子命令—**svnserve**完全通过选项控制。

--daemon (-d)
导致**svnserve**以守护进程方式运行，**svnserve**维护本身并且接受和服务svn端口(缺省3690)的TCP/IP连接。

--foreground
When used together with -d, causes **svnserve** to stay in the foreground. This is mainly useful for debugging.

--inetd (-i)
Causes **svnserve** to use the stdin and stdout file descriptors, as is appropriate for a daemon running out of **inetd**.

--help (-h)
显示有用的摘要和选项。

--listen-host=*HOST*
svnserve监听的*HOST*，可能是一个主机名或是一个IP地址。

--listen-once (-X)
导致**svnserve**在svn端口接受一个连接，并为之服务，完成后退出。这个选项主要用来调试。

--listen-port=*PORT*
Causes **svnserve** to listen on *PORT* when run in daemon mode. (FreeBSD daemons listen only on tcp6 by default—this option tells them to also listen on tcp4.)

--pid-file *FILENAME*
Causes **svnserve** to write its process ID to *FILENAME*, which must be writable by the user under which **svnserve** is running.

`--root=ROOT` (`-r=ROOT`)
设置**svnserve**服务的版本库的虚拟根，客户端提供的URL中显示的路径会解释为这个根的相对路径，不会允许离开这个根。

`--threads` (`-T`)
当以守护进程模式运行，导致**svnserve**为每个连接产生一个线程而不是一个进程(例如在Windows中运行时)，**svnserve**进程本身在启动后会一直在后台。

`--tunnel` (`-t`)
Causes **svnserve** to run in tunnel mode, which is just like the **inetd** mode of operation (both modes serve one connection over `stdin`/`stdout`, and then exit), except that the connection is considered to be preauthenticated with the username of the current UID. This flag is automatically passed for you by the client when running over a tunnel agent such as **ssh**. That means there's rarely any need for *you* to pass this option to **svnserve**. So, if you find yourself typing `svnserve --tunnel` on the command line and wondering what to do next, see 第 6.3.4 节 “穿越 SSH 隧道”.

`--tunnel-user NAME`
Used in conjunction with the `--tunnel` option, tells **svnserve** to assume that *NAME* is the authenticated user, rather than the UID of the **svnserve** process. This is useful for users wishing to share a single system account over SSH, but to maintain separate commit identities.

`--version`
显示版本信息，版本库后端存在和可用的模块列表，然后退出。

9.6. svndumpfilter

svndumpfilter is a command-line utility for removing history from a Subversion dump file by either excluding or including paths beginning with one or more named prefixes. For details, see 第 5.4.1.3 节 “svndumpfilter”.

9.6.1. svndumpfilter 选项

Options in **svndumpfilter** are global, just as they are in **svn** and **svnadmin**:

`--drop-empty-revs`
If filtering causes any revision to be empty (i.e., causes no change to the repository), removes these revisions from the final dump file.

`--renumber-revs`
Renumbers revisions that remain after filtering.

`--skip-missing-merge-sources`
Skips merge sources that have been removed as part of the filtering. Without this option, **svndumpfilter** will exit with an error if the merge source for a retained path is removed by filtering.

`--preserve-revprops`
If all nodes in a revision are removed by filtering and `--drop-empty-revs` is not passed, the default behavior of **svndumpfilter** is to remove all revision properties except for the date and the log message (which will merely indicate that the revision is empty). Passing this option will

preserve existing revision properties (which may or may not make sense since the related content is no longer present in the dump file).

`--quiet`
Does not display filtering statistics.

9.6.2. svndumpfilter 子命令

Here are the various subcommands for the **svndumpfilter** program.

名称

svndumpfilter exclude — Filter out nodes with given prefixes from the dump stream.

概要

```
svndumpfilter exclude PATH_PREFIX...
```

描述

This can be used to exclude nodes that begin with one or more *PATH_PREFIX*es from a filtered dump file.

选项

```
--drop-empty-revs
--preserve-revprops
--quiet
--renumber-revs
--skip-missing-merge-sources
```

例子

If we have a dump file from a repository with a number of different picnic-related directories in it, but we want to keep everything *except* the `sandwiches` part of the repository, we'll exclude only that path:

```
$ svndumpfilter exclude sandwiches < dumpfile > filtered-dumpfile
Excluding prefixes:
   '/sandwiches'

Revision 0 committed as 0.
Revision 1 committed as 1.
Revision 2 committed as 2.
Revision 3 committed as 3.
Revision 4 committed as 4.

Dropped 1 node(s):
   '/sandwiches'
```

名称

svndumpfilter include — Filter out nodes without given prefixes from dump stream.

概要

```
svndumpfilter include PATH_PREFIX...
```

描述

Can be used to include nodes that begin with one or more *PATH_PREFIX*es in a filtered dump file (thus excluding all other paths).

选项

```
--drop-empty-revs
--preserve-revprops
--quiet
--renumber-revs
--skip-missing-merge-sources
```

例子

If we have a dump file from a repository with a number of different picnic-related directories in it, but want to keep only the `sandwiches` part of the repository, we'll include only that path:

```
$ svndumpfilter include sandwiches < dumpfile > filtered-dumpfile
Including prefixes:
   '/sandwiches'

Revision 0 committed as 0.
Revision 1 committed as 1.
Revision 2 committed as 2.
Revision 3 committed as 3.
Revision 4 committed as 4.

Dropped 3 node(s):
   '/drinks'
   '/snacks'
   '/supplies'
```

名称

svndumpfilter help — Help!.

概要

```
svndumpfilter help [SUBCOMMAND...]
```

描述

Displays the help message for **svndumpfilter**. Unlike other help commands documented in this chapter, there is no witty commentary for this help command. The authors of this book deeply regret the omission.

选项

无

9.7. svnversion

名称

svnversion — 总结工作拷贝的本地修订版本。

概要

```
svnversion [OPTIONS] [WC_PATH [TRAIL_URL]]
```

描述

svnversion是用来总结工作拷贝修订版本混合的程序，结果修订版本号或范围会写到标准输出。

通常在构建过程中利用其输出定义程序的版本号码。

TRAIL_URL, if present, is the trailing portion of the URL used to determine whether *WC_PATH* itself is switched (detection of switches within *WC_PATH* does not rely on *TRAIL_URL*).

当没有定义*WC_PATH*，会使用当前路径作为工作拷贝路径，如果没有显式定义*WC_PATH*，*TRAIL_URL*将无法定义。

选项

Like **svnserve**, **svnversion** has no subcommands—only options:

`--no-newline` (`-n`)
Omits the usual trailing newline from the output.

`--committed` (`-c`)
Uses the last-changed revisions rather than the current (i.e., highest locally available) revisions.

`--help` (`-h`)
Prints a help summary.

`--version`
Prints the version of **svnversion** and exit with no error.

例子

If the working copy is all at the same revision (e.g., immediately after an update), then that revision is printed out:

```
$ svnversion
4168
```

You can add *TRAIL_URL* to make sure the working copy is not switched from what you expect. Note that the *WC_PATH* is required in this command:

```
$ svnversion . /var/svn/trunk
4168
```

对于混合修订版本的工作拷贝，修订版本的范围会被打印：

```
$ svnversion
4123:4168
```

如果工作拷贝包含修改，后面会紧跟一个"M"：

```
$ svnversion
4168M
```

如果工作拷贝已经跳转，后面会有一个"S"：

```
$ svnversion
4168S
```

因此，这里是一个混合修订版本，跳转的工作拷贝包含了一些本地修改：

```
$ svnversion
4212:4168MS
```

如果从一个目录而不是工作拷贝调用，**svnversion**假定它是一个导出的工作拷贝并且打印"exported"：

```
$ svnversion
exported
```

9.8. mod_dav_svn

名称

mod_dav_svn Configuration Directives — Apache通过Apache HTTP服务器用来维护Subversion版本库配置指示。

描述

This section briefly describes each Subversion Apache configuration directive. For an in-depth description of configuring Apache with Subversion, see 第 6.4 节 “httpd - Apache 的 HTTP 服务器”.)

指示

These are the `httpd.conf` directives that apply to **mod_dav_svn**:

`DAV svn`
: Must be included in any `Directory` or `Location` block for a Subversion repository. It tells **httpd** to use the Subversion backend for `mod_dav` to handle all requests.

`SVNAllowBulkUpdates On|Off`
: Toggles support for all-inclusive responses to update-style `REPORT` requests. Subversion clients use `REPORT` requests to get information about directory tree checkouts and updates from **mod_dav_svn**. They can ask the server to send that information in one of two ways: with the entirety of the tree's information in one massive response, or with a *skelta* (a skeletal representation of a tree delta) which contains just enough information for the client to know what *additional* data to request from the server. When this directive is included with a value of `Off`, **mod_dav_svn** will only ever respond to these `REPORT` requests with skelta responses, regardless of the type of responses requested by the client.

  Most folks won't need to use this directive at all. It primarily exists for administrators who wish—for security or auditing reasons—to force Subversion clients to fetch individually all the files and directories needed for updates and checkouts, thus leaving an audit trail of `GET` and `PROPFIND` requests in Apache's logs. The default value of this directive is `On`.

`SVNAutoversioning On|Off`
: When its value is `On`, allows write requests from WebDAV clients to result in automatic commits. A generic log message is auto-generated and attached to each revision. If you enable autoversioning, you'll likely want to set `ModMimeUsePathInfo On` so that `mod_mime` can set `svn:mime-type` to the correct MIME type automatically (as best as `mod_mime` is able to, of course). For more information, see 附录 C, *WebDAV* 和自动版本. The default value of this directive is `Off`.

`SVNPath` *`directory-path`*
: Specifies the location in the filesystem for a Subversion repository's files. In a configuration block for a Subversion repository, either this directive or `SVNParentPath` must be present, but not both.

`SVNSpecialURI` *`component`*
: Specifies the URI component (namespace) for special Subversion resources. The default is `!svn`, and most administrators will never use this directive. Set this only if there is a pressing need to have a file named `!svn` in your repository. If you change this on a server already in use, it will break all of the outstanding working copies, and your users will hunt you down with pitchforks and flaming torches.

`SVNReposName` *名称*
指定Subversion版本库在`HTTP GET`请求中使用的名字，这个值会作为所有目录列表(当你用web浏览器察看Subversion版本库时会看到)的标题，这个指示是可选的。

`SVNIndexXSLT` *`directory-path`*
目录列表所使用的XSL转化的URI，这个指示可选。

`SVNParentPath` *`directory-path`*
指定子目录会是版本库的父目录在文件系统的位置，在一个Subversion版本库的配置块里，必须提供这个指示或`SVNPath`，但不能同时存在。

`SVNPathAuthz On|Off|short_circuit`
Controls path-based authorization by enabling subrequests (`On`), disabling subrequests (`Off`; see 第 6.4.4.3 节 “禁用基于路径的检查”), or querying **mod_authz_svn** directly (`short_circuit`). The default value of this directive is `On`.

`SVNListParentPath On|Off`
When set to `On`, allows a `GET` of `SVNParentPath`, which results in a listing of all repositories under that path. The default setting is `Off`.

`SVNMasterURI` *`url`*
Specifies a URI to the master Subversion repository (used for a write-through proxy).

`SVNActivitiesDB` *`directory-path`*
Specifies the location in the filesystem where the activities database should be stored. By default, **mod_dav_svn** creates and uses a directory in the repository called `dav/activities.d`. The path specified with this option must be an absolute path.

If specified for an `SVNParentPath` area, **mod_dav_svn** appends the basename of the repository to the path specified here. For example:

```
<Location /svn>
  DAV svn

  # any "/svn/foo" URL will map to a repository in
  # /net/svn.nfs/repositories/foo
  SVNParentPath         "/net/svn.nfs/repositories"

  # any "/svn/foo" URL will map to an activities db in
  #  /var/db/svn/activities/foo
  SVNActivitiesDB       "/var/db/svn/activities"
</Location>
```

高级日志

This is a list of Subversion action log messages produced by Apache's high-level logging mechanism, followed by an example of the log message. See 第 6.4.5.2 节 “Apache 日志” for details on logging.

Checkout or export
```
checkout-or-export /path r62 depth=infinity
```

Commit
```
commit harry r100
```

Diffs
```
diff /path r15:20 depth=infinity ignore-ancestry

diff /path1@15 /path2@20 depth=infinity ignore-ancestry
```

Fetch a directory
```
get-dir /trunk r17 text
```

Fetch a file
```
get-file /path r20 props
```

Fetch a file revision
```
get-file-revs /path r12:15 include-merged-revisions
```

Fetch merge information
```
get-mergeinfo (/path1 /path2)
```

Lock
```
lock /path steal
```

Log
```
log    (/path1,/path2,/path3)    r20:90    discover-changed-paths
revprops=()
```

Replay revisions (svnsync)
```
replay /path r19
```

修订版本属性修改
```
change-rev-prop r50 propertyname
```

修订版本属性列表
```
rev-proplist r34
```

状态
```
status /path r62 depth=infinity
```

Switch
```
switch /pathA /pathB@50 depth=infinity
```

Unlock
```
unlock /path break
```

更新
```
update /path r17 send-copyfrom-args
```

9.9. mod_authz_svn

名称

mod_authz_svn Configuration Directives — Apache configuration directives for configuring path-based authorization for Subversion repositories served through the Apache HTTP Server.

描述

This section briefly describes each Apache configuration directive offered by **mod_authz_svn**. For an in-depth description of using path-based authorization in Subversion, see 第 6.5 节 “基于路径的授权”.)

指示

These are the `httpd.conf` directives that apply to **mod_authz_svn**:

`AuthzSVNAccessFile` *`file-path`*
: Consult *`file-path`* for access rules describing the permissions for paths in Subversion repository.

`AuthzSVNAnonymous On|Off`
: Set to `Off` to disable two special-case behaviours of this module: interaction with the `Satisfy Any` directive and enforcement of the authorization policy even when no `Require` directives are present. The default value of this directive is `On`.

`AuthzSVNAuthoritative On|Off`
: Set to `Off` to allow access control to be passed along to lower modules. The default value of this directive is `On`.

`AuthzSVNNoAuthWhenAnonymousAllowed On|Off`
: Set to `On` to suppress authentication and authorization for requests which anonymous users are allowed to perform. The default value of this directive is `On`.

9.10. Subversion 属性

Subversion allows users to invent arbitrarily named versioned properties on files and directories, as well as unversioned properties on revisions. The only restriction is on properties whose names begin with `svn:` (those are reserved for Subversion's own use). While these properties may be set by users to control Subversion's behavior, users may not invent new `svn:` properties.

9.10.1. 版本控制的属性

These are the versioned properties that Subversion reserves for its own use:

`svn:executable`
: 如果出现在一个文件上，客户端会将此文件在Unix工作拷贝中设置为可执行，见第 3.3.2 节 “文件的可执行性”。

`svn:mime-type`
: If present on a file, the value indicates the file's MIME type. This allows the client to decide whether line-based contextual merging is safe to perform during updates, and can also affect how the file behaves when fetched via a web browser. See 第 3.3.1 节 “文件内容类型”.

`svn:ignore`
如果出现在目录上，这是一组**svn status**和其它命令可以忽略的未版本化文件的名称模式，见第 3.4 节 “忽略未版本控制的条目”。

`svn:keywords`
如果出现在一个文件上，这个值告诉客户端如何扩展文件的特定关键字，见第 3.5 节 “关键字替换”。

`svn:eol-style`
If present on a file, the value tells the client how to manipulate the file's line-endings in the working copy and in exported trees. See 第 3.3.3 节 “行结束字符序列” and svn export earlier in this chapter.

`svn:externals`
如果出现在一个目录上，则这个值就是客户端必须要检出的路径和URL列表。见第 3.8 节 “外部定义”。

`svn:special`
如果出现在一个文件上，表示了那个文件不是一个普通的文件，而是一个符号链或者是其他特殊的对象[1]。

`svn:needs-lock`
如果出现在一个文件上，告诉客户端在工作拷贝将文件置为只读，可以提醒我们在修改以前必须解锁。见第 3.7.4 节 “锁定交流”。

`svn:mergeinfo`
Used by Subversion to track merge data. See 第 4.3.3 节 “合并信息和预览” for details, but you should never edit this property unless you *really* know what you're doing.

9.10.2. 未版本控制的属性

These are the unversioned properties tht Subversion reserves for its own use:

`svn:author`
If present, contains the authenticated username of the person who created the revision. (If not present, the revision was committed anonymously.)

`svn:date`
保存了ISO 8601格式的修订版本创建UTC时间，这个值来自服务器主机时钟，不是客户端的。

`svn:log`
保存了描述修订版本的日志信息。

`svn:autoversioned`
如果出现，则修订版本是通过自动版本化特性创建，见第 C.2 节 “自动版本化”。

[1]此时，符号链是唯一的“特别”对象，但是以后，也许Subversion会有更多的特别对象。

9.11. 版本库钩子

These are the repository hooks that Subversion provides:

名称

start-commit — 开始提交的通知

描述

The start-commit hook is run before the commit transaction is even created. It is typically used to decide whether the user has commit privileges at all.

If the start-commit hook program returns a nonzero exit value, the commit is stopped before the commit transaction is even created, and anything printed to `stderr` is marshalled back to the client.

输入参数

传递给你钩子程序的命令行参数，按照顺序是：

1. 版本库路径
2. 认证过的尝试提交的用户名
3. Colon-separated list of capabilities that a client passes to the server, including `depth`, `mergeinfo`, and `log-revprops` (new in Subversion 1.5).

普通用户

Access control (e.g., temporarily lock out commits for some reason).

A means to allow access only from clients that have certain capabilities.

名称

pre-commit — 在提交结束之前提醒。

描述

The `pre-commit` hook is run just before a commit transaction is promoted to a new revision. Typically, this hook is used to protect against commits that are disallowed due to content or location (e.g., your site might require that all commits to a certain branch include a ticket number from the bug tracker, or that the incoming log message is nonempty).

If the `pre-commit` hook program returns a nonzero exit value, the commit is aborted, the commit transaction is removed, and anything printed to `stderr` is marshalled back to the client.

输入参数

传递给你钩子程序的命令行参数，按照顺序是：

1. 版本库路径

2. 提交事务的名称

普通用户

修改确认和控制

名称

post-commit — 成功提交的通知。

描述

The `post-commit` hook is run after the transaction is committed and a new revision is created. Most people use this hook to send out descriptive emails about the commit or to notify some other tool (such as an issue tracker) that a commit has happened. Some configurations also use this hook to trigger backup processes.

If the `post-commit` hook returns a nonzero exit status, the commit *will not* be aborted since it has already completed. However, anything that the hook printed to `stderr` will be marshalled back to the client, making it easier to diagnose hook failures.

输入参数

传递给你钩子程序的命令行参数，按照顺序是：

1. 版本库路径

2. 提交创建的修订版本号

普通用户

提交通知；工具集成

名称

pre-revprop-change — 修订版本属性修改的通知。

描述

`pre-revprop-change`钩子在修订版本属性修改之前，正常提交范围之外被执行，不象其他钩子，这个钩子默认是拒绝所有的属性修改，钩子必须实际存在并且返回一个零值，这样属性修改才能实现。

If the `pre-revprop-change` hook doesn't exist, isn't executable, or returns a nonzero exit value, no change to the property will be made, and anything printed to `stderr` is marshalled back to the client.

输入参数

传递给你钩子程序的命令行参数，按照顺序是：

1. 版本库路径

2. 要修改属性的修订版本

3. Authenticated username attempting the property change

4. 属性名称已修改

5. 变更描述：`A`(添加的)，`D`(删除的)或`M`(修改的)

Additionally, Subversion passes the intended new value of the property to the hook program via standard input.

普通用户

访问控制；变更确认和控制

名称

post-revprop-change — 修订版本属性修改成功的通知

描述

The `post-revprop-change` hook is run immediately after the modification of a revision property when performed outside the scope of a normal commit. As you can derive from the description of its counterpart, the `pre-revprop-change` hook, this hook will not run at all unless the `pre-revprop-change` hook is implemented. It is typically used to send email notification of the property change.

If the `post-revprop-change` hook returns a nonzero exit status, the change *will not* be aborted since it has already completed. However, anything that the hook printed to `stderr` will be marshalled back to the client, making it easier to diagnose hook failures.

输入参数

传递给你钩子程序的命令行参数，按照顺序是：

1. 版本库路径

2. 已经修改属性的修订版本

3. 做出修改的认证用户名

4. 属性名称已修改

5. 变更描述：`A`(添加的)，`D`(删除的)或`M`(修改的)

此外，Subversion通过标准输入将属性的前一个值传递给钩子。

普通用户

Property change notification

名称

pre-lock — 路径锁定尝试的通知。

描述

The `pre-lock` hook runs whenever someone attempts to lock a path. It can be used to prevent locks altogether or to create a more complex policy specifying exactly which users are allowed to lock particular paths. If the hook notices a preexisting lock, it can also decide whether a user is allowed to “steal” the existing lock.

If the `pre-lock` hook program returns a nonzero exit value, the lock action is aborted and anything printed to `stderr` is marshalled back to the client.

输入参数

传递给你钩子程序的命令行参数，按照顺序是：

1. 版本库路径

2. 将要锁定的版本化路径

3. 尝试锁定的认证用户名

普通用户

访问控制

名称

post-lock — 成功锁定路径的通知。

描述

The `post-lock` hook runs after one or more paths have been locked. It is typically used to send email notification of the lock event.

If the `post-lock` hook returns a nonzero exit status, the lock *will not* be aborted since it has already completed. However, anything that the hook printed to `stderr` will be marshalled back to the client, making it easier to diagnose hook failures.

输入参数

传递给你钩子程序的命令行参数，按照顺序是：

1. 版本库路径

2. 锁定路径的认证用户名

此外，锁定路径通过标准输入传递给钩子程序，每行一个路径。

普通用户

锁定通知

名称

pre-unlock — 路径解锁尝试的通知。

描述

`pre-unlock`钩子在某人企图删除一个文件上的钩子时发生，可以用来制定哪些用户可以解除文件锁定的策略。制定破坏锁定的策略非常重要，如果一个用户A锁定了一个文件，允许用户B打开这个锁？如果这个锁已经一周了呢？这种事情可以通过钩子决定并执行。

If the `pre-unlock` hook program returns a nonzero exit value, the unlock action is aborted and anything printed to `stderr` is marshalled back to the client.

输入参数

传递给你钩子程序的命令行参数，按照顺序是：

1. 版本库路径

2. 将要锁定的版本化路径

3. 尝试锁定的认证用户名

普通用户

访问控制

名称

post-unlock — 路径成功解锁的通知。

描述

The `post-unlock` hook runs after one or more paths have been unlocked. It is typically used to send email notification of the unlock event.

If the `post-unlock` hook returns a nonzero exit status, the unlock *will not* be aborted since it has already completed. However, anything that the hook printed to `stderr` will be marshalled back to the client, making it easier to diagnose hook failures.

输入参数

传递给你钩子程序的命令行参数，按照顺序是：

1. 版本库路径

2. 路径解锁的认证用户名

此外，解锁路径通过标准输入传递给钩子程序，每行一个路径。

普通用户

解锁通知

附录 A. Subversion 快速入门指南

If you're eager to get Subversion up and running (and you enjoy learning by experimentation), this appendix will show you how to create a repository, import code, and then check it back out again as a working copy. Along the way, we give links to the relevant chapters of this book.

警告

If you're new to the entire concept of version control or to the “copy-modify-merge” model used by both CVS and Subversion, you should read 第 1 章 基本概念 before going any further.

A.1. 安装 Subversion

Subversion is built on a portability layer called APR—the Apache Portable Runtime library. The APR library provides all the interfaces that Subversion needs to function on different operating systems: disk access, network access, memory management, and so on. While Subversion is able to use Apache as one of its network server programs, its dependence on APR *does not* mean that Apache is a required component. APR is a standalone library usable by any application. It does mean, however, that like Apache, Subversion clients and servers run on any operating system that the Apache **httpd** server runs on: Windows, Linux, all flavors of BSD, Mac OS X, NetWare, and others.

最简单的安装Subversion的方法就是下载与你的操作系统对应的二进制程序包。在Subversion的网站(`http://subversion.tigris.org`)上通常可以找到由志愿者提供下载的程序包。在这个网站上，会提供微软操作系统上的图形化应用程序安装包。而对于类Unix系统，则可以使用其自身的程序包系统(PRMs、DEBs、ports tree等等)来获取Subversion。

Alternatively, you can build Subversion directly from source code, though it's not always an easy task. (If you're not experienced at building open source software packages, you're probably better off downloading a binary distribution instead!) From the Subversion web site, download the latest source code release. After unpacking it, follow the instructions in the `INSTALL` file to build it. Note that a released source package may not contain everything you need to build a command-line client capable of talking to a remote repository. Starting with Subversion 1.4 and later, the libraries Subversion depends on (apr, apr-util, and neon) are distributed in a separate source package suffixed with `-deps`. These libraries are now common enough that they may already be installed on your system. If not, you'll need to unpack the dependency package into the same directory where you unpacked the main Subversion source. Regardless, it's possible that you may want to fetch other optional dependencies such as Berkeley DB and possibly Apache **httpd**. If you want to do a complete build, make sure you have all of the packages documented in the `INSTALL` file.

如果你是一个喜欢使用最新软件的人，你可以从Subversion本身的版本库得到Subversion最新的源代码，显然，你首先需要一个Subversion客户端，有了之后，你就可以从`http://svn.collab.net/repos/svn/trunk/`检出一个Subversion源代码的工作拷贝：[1]

```
$ svn checkout http://svn.collab.net/repos/svn/trunk subversion
```

[1]注意上面例子中检出的URL并不是以`svn`结尾，而是它的一个叫做`trunk`的子目录，可以看我们对Subversion的分支和标签模型的讨论来理解背后的原因。

```
A    subversion/HACKING
A    subversion/INSTALL
A    subversion/README
A    subversion/autogen.sh
A    subversion/build.conf
…
```

上面的命令会检出一个流血的，最新的Subversion源代码版本到你的叫做`subversion`的当前工作目录。很明显，你可以调整最后的参数改为你需要的。不管你怎么称呼你的新的工作拷贝目录，在操作之后，你现在已经有了Subversion的源代码。当然，你还是需要得到一些帮助库(apr，apr-util等等)—见工作拷贝根目录的`INSTALL`来得到更多细节。

A.2. 快速指南

> “请确定你坐在了正确的位置，你的盘桌已经关闭，乘务员们，准备起飞…。”

What follows is a quick tutorial that walks you through some basic Subversion configuration and operation. When you finish it, you should have a general understanding of Subversion's typical usage.

注意

The examples used in this appendix assume that you have **svn**, the Subversion command-line client, and **svnadmin**, the administrative tool, ready to go on a Unix-like operating system. (This tutorial also works at the Windows command-line prompt, assuming you make some obvious tweaks.) We also assume you are using Subversion 1.2 or later (run **`svn --version`** to check.)

Subversion的所有版本化数据都储存在中心版本库中。因此首先，我们需要创建一个版本库：

```
$ svnadmin create /var/svn/repos
$ ls /var/svn/repos
conf/  dav/  db/  format  hooks/  locks/  README.txt
```

这个命令创建了一个新目录`/var/svn/repos`，并在其中创建了一个Subversion版本库。这个目录里主要保存了一些数据库文件(还有其它一些文件)，而不像CVS那样保存着版本化的文件。需要更多版本库创建和维护方面的内容，参见第 5 章 版本库管理。

Subversion has no concept of a “project.” The repository is just a virtual versioned filesystem, a large tree that can hold anything you wish. Some administrators prefer to store only one project in a repository, and others prefer to store multiple projects in a repository by placing them into separate directories. We discuss the merits of each approach in 第 5.2.1 节 “规划你的版本库结构”. Either way, the repository manages only files and directories, so it's up to humans to interpret particular directories as “projects.” So while you might see references to projects throughout this book, keep in mind that we're only ever talking about some directory (or collection of directories) in the repository.

In this example, we assume you already have some sort of project (a collection of files and directories) that you wish to import into your newly created Subversion repository. Begin by organizing your data into a single directory called `myproject` (or whatever you wish). For reasons explained in 第 4 章 分

支与合并, your project's tree structure should contain three top-level directories named `branches`, `tags`, and `trunk`. The `trunk` directory should contain all of your data, and the `branches` and `tags` directories should be empty:

```
/tmp/myproject/branches/
/tmp/myproject/tags/
/tmp/myproject/trunk/
                      foo.c
                      bar.c
                      Makefile
                      …
```

`branches`、`tags`和`trunk`这三个子目录不是Subversion必须的。但这样做是Subversion的习惯用法，我们还是遵守这个约定吧。

准备好了数据之后，就可以使用**svn import**命令(参见第 2.2 节 “导入数据到你的版本库”)将其导入到版本库中：

```
$ svn import /tmp/myproject file:///var/svn/repos/myproject -m "initial im
Adding         /tmp/myproject/branches
Adding         /tmp/myproject/tags
Adding         /tmp/myproject/trunk
Adding         /tmp/myproject/trunk/foo.c
Adding         /tmp/myproject/trunk/bar.c
Adding         /tmp/myproject/trunk/Makefile
…
Committed revision 1.
$
```

现在版本库中已经保存了目录中的数据。如前所述，直接查看版本库是看不到文件和目录的；它们存放在数据库之中。但是版本库的虚拟文件系统中则包含了一个名为`myproject`的顶级目录，其中依此保存了所有的数据。

Note that the original `/tmp/myproject` directory is unchanged; Subversion is unaware of it. (In fact, you can even delete that directory if you wish.) To start manipulating repository data, you need to create a new “working copy” of the data, a sort of private workspace. Ask Subversion to “check out” a working copy of the `myproject/trunk` directory in the repository:

```
$ svn checkout file:///var/svn/repos/myproject/trunk myproject
A  myproject/foo.c
A  myproject/bar.c
A  myproject/Makefile
…
Checked out revision 1.
```

现在，在`myproject`目录下生成了一个版本库数据的独立拷贝。我们可以在这个工作拷贝中编辑文件，并将修改提交到版本库中。

- 进入工作拷贝目录，编辑某个文件的内容。
- Run **`svn diff`** to see unified diff output of your changes.
- Run **`svn commit`** to commit the new version of your file to the repository.
- Run **`svn update`** to bring your working copy “up to date” with the repository.

完整的工作拷贝操作指南，请参见第 2 章 基本使用。

现在，Subversion版本库可以通过网络方式访问。参考第 6 章 服务配置，了解不同服务器软件的使用以及配置方法。

附录 B. CVS 用户的 Subversion 指南

This appendix is a guide for CVS users new to Subversion. It's essentially a list of differences between the two systems as “viewed from 10,000 feet.” For each section, we provide references to relevant chapters when possible.

尽管Subversion的目标是接管当前和未来的CVS用户基础，需要一些新的特性设计来修正一些CVS“不好的”行为习惯，这意味着，作为一个CVS用户，你或许需要打破习惯—忘记一些奇怪的习惯来作为开始。

B.1. 版本号现在不同了

在CVS中，修订版本号是每文件的，这是因为CVS使用RCS文件保存数据，每个文件都在版本库有一个对应的RCS文件，版本库几乎就是根据项目树的结构创建。

In Subversion, the repository looks like a single filesystem. Each commit results in an entirely new filesystem tree; in essence, the repository is an array of trees. Each of these trees is labeled with a single revision number. When someone talks about “revision 54”, he's talking about a particular tree (and indirectly, the way the filesystem looked after the 54th commit).

技术上讲，谈论“文件`foo.c`的修订版本5”是不正确的，相反，一个人会说“`foo.c`在修订版本5出现”。同样，我们在假定文件的进展时也要小心，在CVS，文件`foo.c`的修订版本5和6一定是不同的，在Subversion，`foo.c`可能在修订版本5和6之间没有改变。

类似的，在CVS中标签或分支是文件的一种标注，或者是单个文件的版本信息，而在Subversion中，标签和分支是整个目录树的拷贝(为了方便，进入版本库顶级目录的`/branches`或`/tags`子目录，`/trunk`旁边)。版本库作为一个整体，每个文件的许多版本可见：每个分支的最新版本，每个标签的最新版本以及trunk本身的最新版本。所以，我们再精炼一下术语，我们说“`foo.c`在修订版本5出现在`/branches/REL1`。”

更多细节见第 1.3.3 节 “修订版本”.

B.2. 目录的版本

Subversion会记录目录树的结构，不仅仅是文件的内容。这是编写Subversion替代CVS最重要的一个原因。

以下是对你这意味着什么的说明，作为一个前CVS用户：

- The **svn add** and **svn delete** commands work on directories now, just as they work on files. So do **svn copy** and **svn move**. However, these commands do *not* cause any kind of immediate change in the repository. Instead, the working items are simply “scheduled” for addition or deletion. No repository changes happen until you run **`svn commit`**.
- 目录不再是哑容器了；它们也有文件一样的修订版本号。(更准确一点，谈论“修订版本5的目录`foo/`”是正确的。)

让我们再讨论一下最后一点，目录版本化是一个困难的问题；因为我们希望允许混合修订版本的工作拷贝，有一些防止我们滥用这个模型的限制。

从理论观点，我们定义“目录foo的修订版本5”意味着一组目录条目和属性。现在假定我们从foo开始添加和删除文件，然后提交。如果说我们还有foo的修订版本5就是一个谎言。然而，如果说我们在提交之后增加了一位foo的修订版本号码，这也是一个谎言；foo还有一些修改我们没有得到，因为我们还没有更新。

Subversion deals with this problem by quietly tracking committed adds and deletes in the `.svn` area. When you eventually run **`svn update`**, all accounts are settled with the repository, and the directory's new revision number is set correctly. *Therefore, only after an update is it truly safe to say that you have a "perfect" revision of a directory.* Most of the time, your working copy will contain "imperfect" directory revisions.

同样的，如果你尝试提交目录的属性修改会有一个问题，通常情况下，提交应该会提高工作目录的本地修订版本号，但是再一次，这还是一个谎言，因为这个目录还没有添加和删除发生，因为还没有更新发生。因此，在你的目录不是最新的时候不允许你提交属性修改。

关于目录版本的更多讨论见第 1.3.5 节 “混合修订版本的工作副本”。

B.3. 更多离线操作

近些年来，磁盘空间变得异常便宜和丰富，但是网络带宽还没有，因此Subversion工作拷贝为紧缺资源进行了优化。

The `.svn` administrative directory serves the same purpose as the `CVS` directory, except that it also stores read-only, "pristine" copies of your files. This allows you to do many things offline:

svn status
显示你所做的本地修改(见第 2.4.3.1 节 “查看你的修改概况”)

svn diff
显示修改的详细信息(见see 第 2.4.3.2 节 “检查你的本地修改的详情”)

svn revert
删除你的本地修改(见第 2.4.4 节 “取消本地修改”)

另外，原始文件的缓存允许Subversion客户端在提交时只提交区别，这是CVS做不到的。

The last subcommand in the list—**svn revert**—is new. It will not only remove local changes, but also unschedule operations such as adds and deletes. Although deleting the file and then running **`svn update`** will still work, doing so distorts the true purpose of updating. And, while we're on this subject…

B.4. 区分状态和更新

在Subversion，我们已经设法抹去**cvs status**和**cvs update**之间的混乱。

The **cvs status** command has two purposes: first, to show the user any local modifications in the working copy, and second, to show the user which files are out of date. Unfortunately, because of CVS's hard-to-read status output, many CVS users don't take advantage of this command at all. Instead, they've developed a habit of running **`cvs update`** or **`cvs -n update`** to quickly see their changes.

If users forget to use the -n option, this has the side effect of merging repository changes they may not be ready to deal with.

对于Subversion，我们通过修改**svn status**的输出使之同时满足阅读和解析的需要来努力消除这种混乱，同样，**svn update**只会打印将要更新的文件信息，而不是本地修改。

B.4.1. 状态

svn status打印所有本地修改的文件，缺省情况下，不会联系版本库，然而这个命令接受一些选项，如下是一些最常用的：

-u
访问版本库检测并显示过期的信息。

-v
显示所有版本控制下的文件。

-N
非递归方式运行(不会访问子目录)。

The **svn status** command has two output formats. In the default “short” format, local modifications look like this:

```
$ svn status
M       foo.c
M       bar/baz.c
```

如果你指定--show-updates(-u)选项，就会使用较长的格式输出：

```
$ svn status -u
M             1047   foo.c
       *      1045   faces.html
       *             bloo.png
M             1050   bar/baz.c
Status against revision:   1066
```

在这个例子里，出现了两列，第二列的星号表示了文件或目录是否过期，第三列显示了工作拷贝修订版本号，在上面的例子里，星号表示如果进行更新，faces.html会被合并，而bloo.png则是版本库新加的文件。(bloo.png前面的修订版本号为空表示了这个文件在工作拷贝已经不存在了。)

此刻，你必须赶快看一下svn status中所说的可能属性代码，下面是一些你会看到的常用状态代码：

```
A     Resource is scheduled for Addition
D     Resource is scheduled for Deletion
M     Resource has local Modifications
C     Resource has Conflicts (changes have not been completely merged
```

```
        between the repository and working copy version)
X     Resource is eXternal to this working copy (may come from another
        repository).  See 第 3.8 节 “外部定义”
?     Resource is not under version control
!     Resource is missing or incomplete (removed by a tool other than
        Subversion)
```

关于**svn status**的详细讨论，见第 2.4.3.1 节 “查看你的修改概况”。

B.4.2. 更新

svn update会更新你的工作拷贝，只打印这次更新的文件。

Subversion将CVS的P和U合并为U，当合并或冲突发生时，Subversion会简单的打印G或C，而不是大段相关内容。

关于**svn update**的详细讨论，见第 2.4.1 节 “更新你的工作副本”。

B.5. 分支和标签

Subversion不区分文件系统空间和“分支”空间；分支和标签都是普通的文件系统目录，这恐怕是CVS用户需要逾越的最大心理障碍，所有信息在第 4 章 分支与合并。

警告

因为Subversion把分支和标签看作普通目录看待，你项目不同的开发线存在于主项目目录的不同子目录里。所以要记住检出你所要的保存特定开发线的子目录，而不是项目的根目录。如果你错误的检出了项目本身，你会紧张的发现你的项目拷贝包含了所有的分支和标签。[1]

B.6. 元数据属性

Subversion的一个新特性就是你可以对文件和目录任意附加元数据(或者是“属性”)，属性是关联在工作拷贝文件或目录的任意名称/值对。

为了设置或得到一个属性名称，使用**svn propset**和**svn propget**子命令，列出对象所有的属性，使用**svn proplist**。

更多信息见第 3.2 节 “属性”。

B.7. 解决冲突

CVS marks conflicts with inline “conflict markers,” and then prints a C during an update or merge operation. Historically, this has caused problems, because CVS isn't doing enough. Many users forget about (or don't see) the C after it whizzes by on their terminal. They often forget that the conflict markers are even present, and then accidentally commit files containing those conflict markers.

[1]如果在检出完成之前没有消耗完磁盘空间的话。

Subversion solves this problem in a pair of ways. First, when a conflict occurs in a file, Subversion records the fact that the file is in a state of conflict, and won't allow you to commit changes to that file until you explicitly resolve the conflict. Second, Subversion 1.5 provides interactive conflict resolution, which allows you to resolve conflicts as they happen instead of having to go back and do so after the update or merge operation completes. See 第 2.4.5 节 “解决冲突(合并别人的修改)” for more about conflict resolution in Subversion.

B.8. 二进制文件和行结束标记转换

In the most general sense, Subversion handles binary files more gracefully than CVS does. Because CVS uses RCS, it can only store successive full copies of a changing binary file. Subversion, however, expresses differences between files using a binary differencing algorithm, regardless of whether they contain textual or binary data. That means all files are stored differentially (compressed) in the repository.

CVS users have to mark binary files with `-kb` flags to prevent data from being garbled (due to keyword expansion and line-ending translations). They sometimes forget to do this.

Subversion takes the more paranoid route. First, it never performs any kind of keyword or line-ending translation unless you explicitly ask it to do so (see 第 3.5 节 “关键字替换” and 第 3.3.3 节 “行结束字符序列” for more details). By default, Subversion treats all file data as literal byte strings, and files are always stored in the repository in an untranslated state.

第二，Subversion维护了一个内部的概念来区别一个文件是“文本”还是“二进制”文件，但这个概念只在工作拷贝非常重要，在**svn update**，Subversion会对本地修改的文本文件执行上下文的合并，但是对二进制文件不会。

To determine whether a contextual merge is possible, Subversion examines the `svn:mime-type` property. If the file has no `svn:mime-type` property, or has a MIME type that is textual (e.g., `text/*`), Subversion assumes it is text. Otherwise, Subversion assumes the file is binary. Subversion also helps users by running a binary-detection algorithm in the **svn import** and **svn add** commands. These commands will make a good guess and then (possibly) set a binary `svn:mime-type` property on the file being added. (If Subversion guesses wrong, the user can always remove or hand-edit the property.)

B.9. 版本化的模块

Unlike CVS, a Subversion working copy is aware that it has checked out a module. That means if somebody changes the definition of a module (e.g., adds or removes components), a call to **svn update** will update the working copy appropriately, adding and removing components.

Subversion定义了模块作为一个目录属性的目录列表：见第 3.8 节 “外部定义”。

B.10. 认证

通过CVS的pserver，你需要在读写操作之前“登陆”到服务器(使用**cvs login**命令)—即使是匿名操作。Subversion版本库使用Apache的**httpd**或**svnserve**作为服务器，你不需要开始时提供认证凭证—如果一个操作需要认证，服务器会要求你的凭证(不管这凭证是用户名与密码，

客户证书还是两个都有)。所以如果你的工作拷贝是全局可读的，在所有的读操作中不需要任何认证。

相对于CVS，Subversion会一直在磁盘(在你的~/.subversion/auth/目录)缓存凭证，除非你通过--no-auth-cache选项告诉它不这样做。

这个行为也有例外，当使用SSH管道的**svnserve**服务器时，使用svn+ssh://的URL模式这种情况下，**ssh**会在通道刚开始时无条件的要求认证。

B.11. 迁移 CVS 版本库到 Subversion

Perhaps the most important way to familiarize CVS users with Subversion is to let them continue to work on their projects using the new system. And while that can be somewhat accomplished using a flat import into a Subversion repository of an exported CVS repository, the more thorough solution involves transferring not just the latest snapshot of their data, but all the history behind it as well, from one system to another. This is an extremely difficult problem to solve; it involves deducing changesets in the absence of atomicity and translating between the systems' completely orthogonal branching policies, among other complications. Still, a handful of tools claim to at least partially support the ability to convert existing CVS repositories into Subversion ones.

The most popular (and mature) conversion tool is cvs2svn (http://cvs2svn.tigris.org/), a Python program originally created by members of Subversion's own development community. This tool is meant to run exactly once: it scans your CVS repository multiple times and attempts to deduce commits, branches, and tags as best it can. When it finishes, the result is either a Subversion repository or a portable Subversion dump file representing your code's history. See the web site for detailed instructions and caveats.

附录 C. WebDAV 和自动版本

WebDAV is an extension to HTTP, and it is growing more and more popular as a standard for file sharing. Today's operating systems are becoming extremely web-aware, and many now have built-in support for mounting “shares” exported by WebDAV servers.

If you use Apache as your Subversion network server, to some extent you are also running a WebDAV server. This appendix gives some background on the nature of this protocol, how Subversion uses it, and how well Subversion interoperates with other software that is WebDAV-aware.

C.1. 什么是 WebDAV?

DAV stands for “Distributed Authoring and Versioning.” RFC 2518 defines a set of concepts and accompanying extension methods to HTTP 1.1 that make the Web a more universal read/write medium. The basic idea is that a WebDAV-compliant web server can act like a generic file server; clients can “mount” shared folders over HTTP that behave much like other network filesystems (such as NFS or SMB).

悲惨的是，RFC规范并没有提供任何版本控制模型。基本的DAV客户端和服务器只是假定每个文件或目录只有一个版本存在，可以重复的覆盖。

因为RFC 2518漏下了版本概念，几年之后，另一个委员会留下来负责撰写RFC 3253来添加WebDAV的版本化，也就是“DeltaV”。WebDAV/DeltaV客户端和服务器经常叫做“DeltaV”客户端和服务器，因为DeltaV暗含了基本的WebDAV。

The original WebDAV standard has been widely successful. Every modern computer operating system has a general WebDAV client built in (details to follow), and a number of popular standalone applications are also able to speak WebDAV—Microsoft Office, Dreamweaver, and Photoshop, to name a few. On the server end, Apache HTTP Server has been able to provide WebDAV services since 1998 and is considered the de facto open source standard. Several other commercial WebDAV servers are available, including Microsoft's own IIS.

DeltaV, unfortunately, has not been so successful. It's very difficult to find any DeltaV clients or servers. The few that do exist are relatively unknown commercial products, and thus it's very difficult to test interoperability. It's not entirely clear as to why DeltaV has remained stagnant. Some opine that the specification is just too complex. Others argue that while WebDAV's features have mass appeal (even the least technical users appreciate network file sharing), its version control features just aren't interesting or necessary for most users. Finally, some believe that DeltaV remains unpopular because there's still no open source server product that implements it well.

当Subversion还在设计阶段时，使用Apache的httpd作为主要网络服务器就是一个很好的想法，已经有了支持WebDAV服务的模块(**mod_dav_svn**)。DeltaV有一个很新的规范，希望就是Subversion服务器模块最终能够成为一个开源的DeltaV参考实现，但非常不幸，DeltaV得版本模型过于详细，与Subversion的模型并不匹配，虽然有些概念可以对应起来，但有些则不能。

这是什么意思呢?

首先，Subversion客户端不是一个完全实现的DeltaV客户端，它需要从服务器得到DeltaV不能提供的东西，因此非常依赖于只有**mod_dav_svn**理解的Subversion特定的REPORT请求。

Second, **mod_dav_svn** is not a fully realized DeltaV server. Many portions of the DeltaV specification were irrelevant to Subversion, and thus were left unimplemented.

在开发者社区一直有这样的讨论，是否值得弥补这种形势。改变Subversion的设计来匹配DeltaV看起来并不现实，所以可能没有办法让客户端从普通的DeltaV服务器上得到所有的东西。另一方面，mod_dav_svn可以继续开发来实现所有的DeltaV，但缺乏这样做的动力—几乎没有能与之交户的DeltaV客户端。

C.2. 自动版本化

While the Subversion client is not a full DeltaV client, and the Subversion server is not a full DeltaV server, there's still a glimmer of WebDAV interoperability to be happy about: *autoversioning*.

Autoversioning is an optional feature defined in the DeltaV standard. A typical DeltaV server will reject an ignorant WebDAV client attempting to do a PUT to a file that's under version control. To change a version-controlled file, the server expects a series of proper versioning requests: something like MKACTIVITY, CHECKOUT, PUT, CHECKIN. But if the DeltaV server supports autoversioning, write requests from basic WebDAV clients are accepted. The server behaves as though the client *had* issued the proper series of versioning requests, performing a commit under the hood. In other words, it allows a DeltaV server to interoperate with ordinary WebDAV clients that don't understand versioning.

Because so many operating systems already have integrated WebDAV clients, the use case for this feature can be incredibly appealing to administrators working with non-technical users. Imagine an office of ordinary users running Microsoft Windows or Mac OS. Each user “mounts” the Subversion repository, which appears to be an ordinary network folder. They use the shared folder as they always do: open files, edit them, and save them. Meanwhile, the server is automatically versioning everything. Any administrator (or knowledgeable user) can still use a Subversion client to search history and retrieve older versions of data.

这个场景不是小说：对于Subversion 1.2来说，是真实的和有效的。为了激活**mod_dav_svn**的自动版本化，需要使用httpd.conf中Location区块的SVNAutoversioning指示，例如：

```
<Location /repos>
  DAV svn
  SVNPath /var/svn/repository
  SVNAutoversioning on
</Location>
```

当激活了SVNAutoversioning，来自WebDAV的客户端请求会导致自动提交，每个修订版本会自动附加一个原始的日志信息。

Before activating this feature, however, understand what you're getting into. WebDAV clients tend to do *many* write requests, resulting in a huge number of automatically committed revisions. For example, when saving data, many clients will do a PUT of a 0-byte file (as a way of reserving a name) followed by another PUT with the real file data. The single file-write results in two separate commits. Also consider that many applications auto-save every few minutes, resulting in even more commits.

If you have a post-commit hook program that sends email, you may want to disable email generation either altogether or on certain sections of the repository; it depends on whether you think the influx of emails will still prove to be valuable notifications or not. Also, a smart post-commit hook program can distinguish between a transaction created via autoversioning and one created through a normal Subversion commit operation. The trick is to look for a revision property named `svn:autoversioned`. If present, the commit was made by a generic WebDAV client.

Another feature that may be a useful complement for Subversion's autoversioning comes from Apache's `mod_mime` module. If a WebDAV client adds a new file to the repository, there's no opportunity for the user to set the the `svn:mime-type` property. This might cause the file to appear as a generic icon when viewed within a WebDAV shared folder, not having an association with any application. One remedy is to have a sysadmin (or other Subversion-knowledgeable person) check out a working copy and manually set the `svn:mime-type` property on necessary files. But there's potentially no end to such cleanup tasks. Instead, you can use the `ModMimeUsePathInfo` directive in your Subversion `<Location>` block:

```
<Location /repos>
  DAV svn
  SVNPath /var/svn/repository
  SVNAutoversioning on

  ModMimeUsePathInfo on

</Location>
```

This directive allows `mod_mime` to attempt automatic deduction of the MIME type on new files that enter the repository via autoversioning. The module looks at the file's named extension and possibly the contents as well; if the file matches some common patterns, the file's `svn:mime-type` property will be set automatically.

C.3. 客户端交互性

All WebDAV clients fall into one of three categories—standalone applications, file-explorer extensions, or filesystem implementations. These categories broadly define the types of WebDAV functionality available to users. 表 C.1 “常用的 WebDAV 客户端” gives our categorization as well as a quick description of some common pieces of WebDAV-enabled software. You can find more details about these software offerings, as well as their general category, in the sections that follow.

表 **C.1.** 常用的 **WebDAV** 客户端

软件	类型	**Windows**	**Mac**	**Linux**	描述
Adobe Photoshop	独立的 WebDAV应用程序	X			Image editing software, allowing direct opening from, and writing to, WebDAV URLs

软件	类型	**Windows**	**Mac**	**Linux**	描述
cadaver	独立的WebDAV应用程序		X	X	Command-line WebDAV client supporting file transfer, tree, and locking operations
DAV Explorer	独立的WebDAV应用程序	X	X	X	Java GUI tool for exploring WebDAV shares
Adobe Dreamweaver	独立的WebDAV应用程序	X			Web production software able to directly read from and write to WebDAV URLs
Microsoft Office	独立的WebDAV应用程序	X			Office productivity suite with several components able to directly read from and write to WebDAV URLs
Microsoft Web文件夹	文件浏览器WebDAV扩展	X			GUI file explorer program able to perform tree operations on a WebDAV share
GNOME Nautilus	文件浏览器WebDAV扩展			X	GUI file explorer able to perform tree operations on a WebDAV share
KDE Konqueror	文件浏览器WebDAV扩展			X	GUI file explorer able to perform tree operations on a WebDAV share
Mac OS X	WebDAV文件系统实现		X		Operating system that has built-in support for mounting

软件	类型	**Windows**	**Mac**	**Linux**	描述
					WebDAV shares.
驱动器映射程序，可以将Windows驱动器加载为远程的WebDAV共享	WebDAV文件系统实现	X			Drive-mapping program for assigning Windows drive letters to a mounted remote WebDAV share
文件传输软件，可以将Windows驱动器加载为远程的WebDAV共享	WebDAV文件系统实现	X			File transfer software, which, among other things, allows the assignment of Windows drive letters to a mounted remote WebDAV share
davfs2	WebDAV文件系统实现			X	Linux filesystem driver that allows you to mount a WebDAV share

C.3.1. 独立的 WebDAV 应用程序

WebDAV应用使用WebDAV协议与WebDAV服务器通讯，我们将会介绍一些支持WebDAV的流行程序。

C.3.1.1. Microsoft Office，Dreamweaver，Photoshop

On Windows, several well-known applications contain integrated WebDAV client functionality, such as Microsoft's Office, [1] Adobe's Photoshop and Dreamweaver programs. They're able to directly open and save to URLs, and tend to make heavy use of WebDAV locks when editing a file.

Note that while many of these programs also exist for Mac OS X, they do not appear to support WebDAV directly on that platform. In fact, on Mac OS X, the File→Open dialog box doesn't allow one to type a path or URL at all. It's likely that the WebDAV features were deliberately left out of Macintosh versions of these programs, since OS X already provides such excellent low-level filesystem support for WebDAV.

[1] WebDAV support was removed from Microsoft Access for some reason, but it exists in the rest of the Office suite.

C.3.1.2. Cadaver，DAV 浏览器

Cadaver是一个简单的Unix命令行的WebDAV共享浏览程序，就像Subversion客户端，它使用neon的HTTP库，毫不奇怪，因为其作者就是neon的作者，Cadaver是一个自由软件(是用GPL许可证)，可以通过`http://www.webdav.org/cadaver/`访问。

Using cadaver is similar to using a command-line FTP program, and thus it's extremely useful for basic WebDAV debugging. It can be used to upload or download files in a pinch, to examine properties, and to copy, move, lock, or unlock files:

```
$ cadaver http://host/repos
dav:/repos/> ls
Listing collection `/repos/': succeeded.
Coll: > foobar                                  0  May 10 16:19
      > playwright.el                        2864  May  4 16:18
      > proofbypoem.txt                      1461  May  5 15:09
      > westcoast.jpg                       66737  May  5 15:09

dav:/repos/> put README
Uploading README to `/repos/README':
Progress: [============================>] 100.0% of 357 bytes succeeded.

dav:/repos/> get proofbypoem.txt
Downloading `/repos/proofbypoem.txt' to proofbypoem.txt:
Progress: [============================>] 100.0% of 1461 bytes succeeded.
```

DAV Explorer是另一个独立运行的WebDAV客户端，使用Java编写，有一个类Apache的许可证，网站是`http://www.ics.uci.edu/~webdav/`。DAV Explorer与cadaver功能差不多，优点可移植，并有一个用户友好的GUI程序。它也是最早的支持WebDAV访问控制协议(RFC 3744)的客户端之一。

当然，在这个情况下DAV Explorer的ACL支持没有任何用处，因为**mod_dav_svn**不支持它，事实上，Cadaver和DAV Explorer支持的一些有限的DeltaV命令也并不有效，因为他们不允许`MKACTIVITY`请求，但是这都不相干；我们假定这些客户端都是针对自动版本化版本库工作。

C.3.2. 文件浏览器的 WebDAV 扩展

Some popular file explorer GUI programs support WebDAV extensions that allow a user to browse a DAV share as though it was just another directory on the local computer, and to perform basic tree editing operations on the items in that share. For example, Windows Explorer is able to browse a WebDAV server as a “network place.” Users can drag files to and from the desktop, or can rename, copy, or delete files in the usual way. But because it's only a feature of the file explorer, the DAV share isn't visible to ordinary applications. All DAV interaction must happen through the explorer interface.

C.3.2.1. Microsoft Web 文件夹

Microsoft was one of the original backers of the WebDAV specification, and first started shipping a client in Windows 98, which was known as Web Folders. This client was also shipped in Windows NT 4.0 and Windows 2000.

The original Web Folders client was an extension to Explorer, the main GUI program used to browse filesystems. It works well enough. In Windows 98, the feature might need to be explicitly installed if Web Folders aren't already visible inside My Computer. In Windows 2000, simply add a new “network place,” enter the URL, and the WebDAV share will pop up for browsing.

With the release of Windows XP, Microsoft started shipping a new implementation of Web Folders, known as the WebDAV Mini-Redirector. The new implementation is a filesystem-level client, allowing WebDAV shares to be mounted as drive letters. Unfortunately, this implementation is incredibly buggy. The client usually tries to convert HTTP URLs (`http://host/repos`) into UNC share notation (`\\host\repos`); it also often tries to use Windows Domain authentication to respond to basic-auth HTTP challenges, sending usernames as `HOST\username`. These interoperability problems are severe and are documented in numerous places around the Web, to the frustration of many users. Even Greg Stein, the original author of Apache's WebDAV module, bluntly states that XP Web Folders simply can't operate against an Apache server.

Windows Vista's initial implementation of Web Folders seems to be almost the same as XP's, so it has the same sort of problems. With luck, Microsoft will remedy these issues in a Vista Service Pack.

However, there seem to be workarounds for both XP and Vista that allow Web Folders to work against Apache. Users have mostly reported success with these techniques, so we'll relay them here.

On Windows XP, you have two options. First, search Microsoft's web site for update KB90730, “Software Update for Web Folders.” This may fix all your problems. If it doesn't, it seems that the original pre-XP Web Folders implementation is still buried within the system. You can unearth it by going to Network Places and adding a new network place. When prompted, enter the URL of the repository, but *include a port number* in the URL. For example, you should enter **`http://host/repos`** as **`http://host:80/repos`** instead. Respond to any authentication prompts with your Subversion credentials.

On Windows Vista, the same KB90730 update may clear everything up. But there may still be other issues. Some users have reported that Vista considers all `http://` connections insecure, and thus will always fail any authentication challenges from Apache unless the connection happens over `https://`. If you're unable to connect to the Subversion repository via SSL, you can tweak the system registry to turn off this behavior. Just change the value of the `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\WebClient\Parameters\BasicAuthLevel` key from **`1`** to **`2`**. A final warning: be sure to set up the Web Folder to point to the repository's root directory (`/`), rather than some subdirectory such as `/trunk`. Vista Web Folders seems to work only against repository roots.

In general, while these workarounds may function for you, you might get a better overall experience using a third-party WebDAV client such as WebDrive or NetDrive.

C.3.2.2. Nautilus，Konqueror

Nautilus是GNOME桌面(`http://www.gnome.org)http://www.kde.org)`

GNOME的Nautilus里，打开File→Open location，并且输入URL。版本库就会显示出来，就像其他文件系统。

In KDE's Konqueror, you need to use the `webdav://` scheme when entering the URL in the location bar. If you enter an `http://` URL, Konqueror will behave like an ordinary web browser. You'll likely see the generic HTML directory listing produced by **mod_dav_svn**. When you enter **`webdav://host/repos`** instead of **`http://host/repos`**, Konqueror becomes a WebDAV client and displays the repository as a filesystem.

C.3.3. WebDAV 的文件系统实现

The WebDAV filesystem implementation is arguably the best sort of WebDAV client. It's implemented as a low-level filesystem module, typically within the operating system's kernel. This means that the DAV share is mounted like any other network filesystem, similar to mounting an NFS share on Unix or attaching an SMB share as a drive letter in Windows. As a result, this sort of client provides completely transparent read/write WebDAV access to all programs. Applications aren't even aware that WebDAV requests are happening.

C.3.3.1. WebDrive，NetDrive

Both WebDrive and NetDrive are excellent commercial products that allow a WebDAV share to be attached as drive letters in Windows. As a result, you can operate on the contents of these WebDAV-backed pseudodrives as easily as you can against real local hard drives, and in the same ways. You can purchase WebDrive from South River Technologies (`http://www.southrivertech.com`). Novell's NetDrive is freely available online, but requires users to have a NetWare license.

C.3.3.2. Mac OS X

Apple的OS X操作系统是集成的文件系统级的WebDAV客户端，通过Finder，选择Go menu的Connect to Server条目，输入WebDAV的URL，会在桌面显示一个磁盘，就像其他装载的卷。你也可以从Darwin终端通过**mount**类型为`webdav`的文件系统实现。

```
$ mount -t webdav http://svn.example.com/repos/project /some/mountpoint
$
```

Note that if your **mod_dav_svn** is older than version 1.2, OS X will refuse to mount the share as read/write; it will appear as read-only. This is because OS X insists on locking support for read/write shares, and the ability to lock files first appeared in Subversion 1.2.

Also, OS X's WebDAV client can sometimes be overly sensitive to HTTP redirects. If OS X is unable to mount the repository at all, you may need to enable the `BrowserMatch` directive in the Apache server's `httpd.conf`:

```
BrowserMatch "^WebDAVFS/1.[012]" redirect-carefully
```

C.3.3.3. Linux davfs2

Linux davfs2 is a filesystem module for the Linux kernel, whose development is organized at `http://dav.sourceforge.net/`. Once you install davfs2, you can mount a WebDAV network share using the usual Linux mount command:

```
$ mount.davfs http://host/repos /mnt/dav
```

附录 D. 版权

REGARDING THE INFORMATION PROVIDED, AND DISCLAIMS LIABILITY FOR DAMAGES RESULTING FROM ITS USE.

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions

   a. "Collective Work" means a work, such as a periodical issue, anthology or encyclopedia, in which the Work in its entirety in unmodified form, along with a number of other contributions, constituting separate and independent works in themselves, are assembled into a collective whole. A work that constitutes a Collective Work will not be considered a Derivative Work (as defined below) for the purposes of this License.

   b. "Derivative Work" means a work based upon the Work or upon the Work and other pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which the Work may be recast, transformed, or adapted, except that a work that constitutes a Collective Work will not be considered a Derivative Work for the purpose of this License. For the avoidance of doubt, where the Work is a musical composition or sound recording, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered a Derivative Work for the purpose of this License.

   c. "Licensor" means the individual or entity that offers the Work under the terms of this License.

   d. "Original Author" means the individual or entity who created the Wor

   e. "Work" means the copyrightable work of authorship offered under the terms of this License.

   f. "You" means an individual or entity exercising rights under this

License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

2. Fair Use Rights. Nothing in this license is intended to reduce, limit, or restrict any rights arising from fair use, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

   a. to reproduce the Work, to incorporate the Work into one or more Collective Works, and to reproduce the Work as incorporated in the Collective Works;

   b. to create and reproduce Derivative Works;

   c. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission the Work including as incorporated in Collective Works;

   d. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission Derivative Works.

   e.

   For the avoidance of doubt, where the work is a musical composition:

      i. Performance Royalties Under Blanket Licenses. Licensor waives the exclusive right to collect, whether individually or via a performance rights society (e.g. ASCAP, BMI, SESAC), royalties for the public performance or public digital performance (e.g. webcast) of the Work.

      ii. Mechanical Rights and Statutory Royalties. Licensor waives the exclusive right to collect, whether individually or via a music rights agency or designated agent (e.g. Harry Fox Agency), royalties for any phonorecord You create from the Work ("cover version") and distribute, subject to the compulsory license created by 17 USC Section 115 of the US

Copyright Act (or the equivalent in other jurisdictions).

f. Webcasting Rights and Statutory Royalties. For the avoidance of doubt, where the Work is a sound recording, Licensor waives the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance (e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions).

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. All rights not expressly granted by Licensor are hereby reserved.

4. Restrictions.The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

a. You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work itself to be made subject to the terms of this License. If You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any reference to such Licensor or the Original Author, as requested. If You create a Derivative Work, upon notice from any Licensor You must, to the extent practicable, remove from the Derivative Work any reference to such Licensor or the Original Author, as requested.

b. If you distribute, publicly display, publicly perform, or publicly digitally perform the Work or any Derivative Works or Collective Works, You must keep intact all copyright notices for

the Work and give the Original Author credit reasonable to the medium or means You are utilizing by conveying the name (or pseudonym if applicable) of the Original Author if supplied; the title of the Work if supplied; to the extent reasonably practicable, the Uniform Resource Identifier, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and in the case of a Derivative Work, a credit identifying the use of the Work in the Derivative Work (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). Such credit may be implemented in any reasonable manner; provided, however, that in the case of a Derivative Work or Collective Work, at a minimum such credit will appear where any other comparable authorship credit appears and in a manner at least as prominent as such other comparable authorship credit.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

   a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Derivative Works or Collective Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

   b. Subject to the above terms and conditions, the license granted

here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

   a. Each time You distribute or publicly digitally perform the Work or a Collective Work, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

   b. Each time You distribute or publicly digitally perform a Derivative Work, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

   c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

   d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

   e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it

索引

符号

B

C

H

P

S