使用Subversion进行版本控制

针对 Subversion 1.1

（本书编译对应1876修订版本）

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

使用Subversion进行版本控制: 针对 Subversion 1.1: （本书编译对应1876修订版本）

由 Ben Collins-Sussman、Brian W. Fitzpatrick和C. Michael Pilato

出版方 (TBA)

译者序

最早接触这本书是在2004上半年，当时Subversion 1.0刚刚发布，而我很快把它引入到我们的项目当中，相对于CVS的简陋，Subversion显得非常的完备，是一个经过了深思熟虑的产品，是新一代开源项目的代表。

当我看到这本免费共享的图书，注意到了它已经在O'Reilly出版，而网站上有最新的版本可以下载，对于这种开源文化赞叹不已，萌生了自己翻译这本书的想法，但是苦于当时对DocBook非常不熟悉，于是使用文本格式，利用闲暇时间翻译了前四章，但后来杂事渐多，竟然慢慢忘了此事。

一转眼到了2005年，Subversion 1.2发布了，我的注意力又转到了这个领域，正好我有了做一个网站的念头，所以就有了Subversion中文站（http://www.subversion.org.cn），而同时我也开始申请成为这本书的中文官方翻译。

这本书的官方翻译要求我必须使用DocBook，要求我必须有一个团队，于是我在这两方面进行了努力，于是有人开始与我并肩工作了。在这段翻译的时间里陆续有人加入进来，按照时间顺序是rocksun、jerry、nashwang、gxio、MichaelDuan、viv、lifengliu2000、genedna、luyongshou、leasun和nannan。但是必须要说明这不是对翻译贡献大小的排序，大家都在自己的能力范围内为这个翻译做出了自己的贡献，感谢我们成员的努力，也感谢许多对我们提出建议的朋友。

开始的时候并没有觉得做好这件事有多难，但当看到翻译的东西自己都读不懂的时候，我感到了一种压力。如果这翻译还不如英文，我们还有没有必要继续。好在在大家的支持下，我越来越喜欢这本书了，渐渐的发现自己可以把这本书当作自己的参考材料了。

但是，我也有过许多疑惑，在中国人们似乎只是把版本控制工具当做一个代码分享的工具，而没有把它融入到整个软件开发的生命周期当中，这也难怪，大多数中国软件的寿命似乎并不长，不需要那么多复杂的配置管理。所以我们的这些翻译能够给大家带来多大的帮助要由中国软件的发展决定，希望我们的工作能够伴随着中国软件的腾飞不断成长。

让我们一起努力吧！

— Rock Sun，青岛，2005年11月29日

前言

一个不太好的常见问题列表（FAQ），常常并不是由人们实际上的问题组成，而经常是由作者期待的问题组成。或许你曾经见过这种类型的问题：

> Q：怎样使用Glorbosoft XYZ提高生产率？
>
> A：许多客户希望知道怎样通过革新我们特许的办公室群件来提高生产率，回答非常简单：首先点击“文件” 菜单，鼠标移到“提高生产率”，然后…

这样的FAQ并不是其字面意义上的FAQ，没有人会这样询问支持者，“怎样提高生产率？”相反，人们经常询问一些更具体的问题，像“怎样修改日程系统提前两天而不是提前一天去提醒被提醒人？”等等。但是通过想象比去发现一个这样的问题列表更容易，编辑一个真实的问题列表需要持续的，有组织的工作，覆盖软件的整个生命周期，提出的问题必须被追踪，要监控反馈，所有问题要收集为一个一致的，可查询的整体，并且能够反映所有用户的经验。这需要耐心，像实地博物学家一样严谨的态度，不应该有浮华的假设，虚幻的断言—而需要开放的视野和精确的记录。

之所以会喜欢这本书，是因为这本书非凡的成长过程，这体现在每一页里，这是作者与用户直接交流的结果。这一切的基础是Ben Collins-Sussman's关于Subversion常见问题邮件列表的研究：使用subversion通常的流程是怎样的？分支与标签同其它版本控制系统的工作方式是一样的吗？我怎样知道某一处修改是谁做的？

由于每天看到相同问题的失落，Ben在2002年夏天努力工作了一个月，撰写了一本Subversion手册，一本六十页，涵盖了所有基础使用知识的手册。这本手册没有说明什么时候要结束，伴随着Subversion的版本，帮助用户开始最初的学习。当O'Reilly决定出版一本完备的Subversion图书的时候，最快捷的方式很明显，就是扩充这本书。

三个联合作者因而面临了一个不寻常的机会。从职责上讲，他们的任务是以一些原始内容为基础，从头到尾写一个草稿。但实际上他们正在使用着一些丰富的自下而上的原材料，像一条稳定的河流，也可能是一口不可预料的间歇泉。Subversion被数以千计的用户采用，这些用户提供了大量的反馈，不仅仅针对Subversion，还包括业已存在的文档。

在写这本书的过程里，Ben，Mike 和 Brian想鬼魂一样一直游荡在Subversion邮件列表和聊天室中，仔细的研究用户实际遇到的问题。监视这些反馈是他们在CollabNet工作的一部分，这给他们开始写这本书时提供了巨大的便利。这本书建立在丰富经验的根基之上，并不是在流沙一样的想象之上；它结合了用户手册和FAQ最好的一面，在第一次阅读时，这种二元性并不明显，按照顺序，从前到后，这本书只是简单的从头到尾的关于软件细节的描述。有一个总的看法，有一个教程，有一章关于管理配置，还有一些高级主题，当然也有一个命令参考和故障指南。只有当你过一段时间之后，返回来找一些特定问题的解决方案时，这种二元

性才得以显现：这些生动的细节一定来自不可预料的实际用例的提炼，大多是源于用户的需要和视点。

当然，没人可以承诺这本书可以回答所有问题。尽管有时候一些前人提问的惊人一致性让你感觉是心灵感应；你仍有可能在社区的知识库里摔跤，空手而归。如果有这种情况，最好的办法是写明问题发送email到<users@subversion.tigris.org>，作者还在那里关注着社区，不仅仅封面提到的三位，还包括许多曾经作出修正与提供原始材料的人。从社区的视角，帮你解决问题只是逐步的调整这本书，进一步调整Subversion本身以更合理的适合用户使用这样一个大工程的一个有趣的额外效用。他们渴望你的信息，不仅仅可以帮助你，也因为可以帮助他们。与Subversion这样活跃的自由软件项目一起，你并不孤单。

让这本书将成为你的第一个伙伴。

— Karl Fogel，芝加哥，2004年3月15日

序言

> “如果C给你足够的绳子吊死你自己，试着用Subversion作为一种存放绳子的工具。” —Brian W. Fitzpatrick

在开源软件领域，并行版本系统（CVS）一直是版本控制的选择。恰如其分的是，CVS本身是一个自由软件，它的非限制性的技法和对网络操作的支持—允许大量的不同地域分散的程序员可以共享他们工作的特性—非常符合开源软件领域合作的精神，CVS和它半混乱状态的开发模型成为了开源文化的基石。

但是像许多其他工具一样，CVS开始显露出衰老的迹象。Subversion是一个被设计成为CVS继任者的新版本控制系统。设计者通过两个办法来争取现有的CVS用户：使用它构建一个开源软件系统的版本控制过程，从感觉和体验上与CVS类似，同时Subversion努力弥补CVS许多明显的缺陷，结果就是不需要版本控制系统一个大的革新。Subversion是非常强大、有用及灵活的工具。

这本书是为Subversion 1.1 系列撰写的，我们试图涵盖Subversion的所有内容，但是Subversion有一个兴盛和充满活力的开发社区，已经有许多特性和改进计划在新的版本中实现，可能会与目前这本书中的命令与细节不一致。

1. 读者

这本书是为了那些希望使用Subversion管理他们数据的在计算机领域有丰富知识的人士准备的。Subversion可以在多种不同的操作系统上运行，它的主要用户操作界面是基于命令行的，这就是我们将要在本书中提到的命令行工具（svn）。出于一致性，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。

那就是说，svn程序可以运行在像Microsoft Windows这样的非Unix平台上。除了一些微小的不同，像用反斜线（\）代替正斜线（/）作为路径分隔符，在Windows上运行svn工具进行输入输出和Unix平台上的功能是完全一致的。Windows用户甚至可以发现这个程序能够在Unix仿真环境Cygwin中成功运行。

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最普通的用途，因此这个场景贯穿于整本书的例子。但是Subversion能够被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，所有的数据仅仅是数据而已。

这本书假定读者从来没有使用过任何版本控制系统，我们也努力使CVS用户能够轻而易举的跳跃到Subversion中。偶尔一些特殊的工具条可能讨论CVS，在最后一个特别的附录中将总结Subversion和CVS的不同。

2. 怎样阅读本书

这本书的目标读者非常的广泛—从从未使用过版本控制的新手到经验丰富的系统管理员。根据你的基础，特定的章节可能对你更有用，下面的内容可以看作是为

各类用户提供的“推荐阅读清单”：

资深管理员
: 假设你以前已经使用过CVS，希望得到一个Subversion服务器并且尽快运行起来，第5、6章将会告诉你怎样建立第一个版本库，并且使之在网络上可用，此后，根据你的CVS使用经验，第3章和附录A告诉你怎样使用Subversion客户端。

新用户
: 你的管理员已经为你准备好了Subversion服务，你将学习如何使用客户端。如果你没有使用过版本控制系统（像CVS），那么第2、3章是重要的介绍，如果你是CVS的老手，最好从第3章和附录A开始。

高级用户
: 无论你只是个使用者还是管理员，最终你的项目会长大，你想通过Subversion作许多高级的事情，像如何使用分支和执行合并（第4章），怎样使用Subversion的属性支持，怎样配制运行参数（第7章）等等。第4、7章一开始并不重要，但你适应了基本操作之后一定要读一下。

开发者
: 大概你已经很熟悉Subversion了，你想扩展它并在它的API基础之上开发新软件，第8章将是为你准备的。

这本书以一个参考材料作为结束—第9章包括了所有命令的参考，这个附录包括了许多有用的主题，当你完成了本书的阅读，你会经常去看这个章节。

3. 本书约定

这一部分包括书中各种约定。

3.1. 排版习惯

等宽
: 用在命令，命令输出和转换

等宽斜体
: 用在代码和文本中可替换的部分

斜体
: 用在文件和路径名

3.2. 图标

注意

这部分内容需要注意的情况。

提示

表明有用的小技巧。

警告

一些警告信息

注意这些源代码只是一例子，需要通过正确编译咒语进行编译，它们只是为了描述在手边的问题，没有必要注意好的编码样式。

4. 本书组织结构

以下是章节和其中的内容介绍：

第1章，介绍
记述了Subversion的历史与特性、架构、组件和安装方法，还包括一个快速入门指南。

第2章，基本概念
解释了版本控制的基础知识，介绍了不同的版本模型，随同讲述了Subversion的版本库，工作拷贝和修订版本的概念。

第3章，指导教程
带领你作为一个Subversion用户开始工作，示范了怎样使用Subversion获得、修改和提交数据。

第4章，分支和合并
讨论分支、合并与标签，包括一个最佳实践，通常的用例，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，以及你可以使用的工具。

第6章，配置服务器
解释了怎样配置Subversion服务器，以及三种访问版本库的方式，HTTP、svn协议和本地访问。这里也介绍了认证的细节，以及授权与匿名访问方式。

第7章，高级主题
研究Subversion客户配置文件，文件和目录属性，怎样忽略工作拷贝中的文件

，怎样引入外部版本树到工作拷贝，最后介绍了如何掌握卖主分支。

第8章，开发者信息
介绍了Subversion的核心，Subversion文件系统，以及从程序员的角度如何看待工作拷贝的管理区域，介绍了如何使用公共APIs写程序使用Subversion，最重要的是，怎样投身到Subversion的开发当中去。

第9章，Subversion完全手册
深入研究研究所有的命令，包括 svn、svnadmin、和svnlook以及大量的相关实例

附录A，Subversion对于CVS用户
比较Subversion与CVS的异同点，消除多年使用CVS养出的坏习惯的建议，包括subversion版本号、标记版本的目录、离线操作、update与status、分支、标签、元数据、冲突和认证。

附录B，故障解决
叙述常见的问题，以及使用和编译Subversion的难点。

附录C，WebDAV与自动版本化
描述了WebDAV与DeltaV的细节，和怎样将你的Subversion版本库作为可读/写的DAV共享装载。

附录D，第三方工具
讨论一些支持和使用Subversion的工具，包括可选的客户端工具，版本库浏览工具等等。

5. Subversion 1.1的新特性

本版图书覆盖了Subversion 1.1的新特性，下面是一个1.1主要变化的列表。

非数据库的版本库
现在可以创建不使用Berkeley DB数据库的版本库，作为替代，这个新的版本库使用普通的文件系统，使用自定义的文件格式，这个版本库不是一个脆弱的“楔入”，它和Berkeley DB版本库一样经过很好的测试，见第 5.1.3 节 “版本库数据存储”。

对象链接纳入版本控制
Unix用户可以创建一个对象链接，使用svn add放置到版本控制，见svn add和第 7.2.3.7 节 “svn:special”。

客户可以追踪拷贝和改名
文件和目录的分支（拷贝）维护着他们与历史的联系，但是在Subversion 1.0中svn log追踪历史的方式与svn diff、svn merge、svn list或svn cat都不同，在Subversion 1.1，所有的客户端子命令可以透明的回溯到拷贝和改名之前的历史文件和目录。

客户端自动转化URI和IRI

在1.0的命令行客户端，用户需要手工的回避URL，客户端只能接收“合法正确的”URL，例如http://host/path%20with%20space/project/espa%F1a。1.1命令行客户端现在知道了web浏览器长久以来所做的事情：它会自动回避用户在shell放置的空格和重音字符之类的字符："http://host/path with space/project/española"

本地化的用户信息
Subversion 1.1现在使用gettext()来为用户显示翻译的错误信息和帮助消息。现在有的翻译包括德国、西班牙、波兰、瑞典、繁体中文、日本、巴西、葡萄牙和挪威Bokmal，为了本地化你的Subversion客户端，只需要设置你的shell的LANG环境变量为支持的某个值（例如de_DE）。

可分享的工作拷贝
允许多个用户分享一个工作拷贝有一些历史问题，现在相信已经修正了。

store-passwords运行变量
这是一个新的运行变量用来关闭密码缓存，所以服务器证书可以缓存，见第 7.1.3.2 节 “config”。

优化和bug修正
svn checkout、svn update、svn status和 svn blame会更快，超过50个小bug被修正，都在项目的CHANGES文件（在http://svn.collab.net/repos/svn/trunk/CHANGES ）里描述。

新的命令选项

- svn blame --verbose:见 svn blame.
- svn export --native-eol EOL:见 svn export.
- svn add --force:见 svn add.
- svnadmin dump --deltas:见 第 5.3.5 节 “版本库的移植”.
- svnadmin create --fs-type TYPE:见 svnadmin create.
- svnadmin recover --wait:见 svnadmin recover.
- svnserve --tunnel-user=NAME:见 第 9.4.1 节 “svnserve选项”.

6. 这本书是免费的

这本书是作为Subversion项目的文档，由开发者开始撰写的，后来成为一个独立工作并进行了重写，因此，它一直有一个免费许可证（见附录 E，版权。）实际上，这本书是在公众关注中写出来的，作为Subversion的一部分，它有两种含义：

- 你一直可以在Subversion的版本库里找到本书的最新版本。

- 对于这本书，你可以任意分发或者修改—它是免费许可证，当然，相对于发布你的私有版本，你最好向Subversion开发社区提供反馈。为了能够参与到社区，见第 8.6 节 “为Subversion做贡献”来学习如何加入到社区。

你可以向O'Reilly发布评论和问题：###insert boilerplate.

一个相对新的在线版本可以在http://svnbook.red-bean.com找到。

7. 致谢

没有Subversion就没有可能（或者有用）有这本书，所以作者很乐意去感谢Brian Behlendorf和CollabNet，有眼光开始这样一个冒险和野心勃勃的开源项目；Jim Blandy给了Subversion这个名字和最初的设计—我们爱你。还有Karl Fogel，伟大社区领导和好朋友。[1]

感谢O'Reilly和我们的编辑Linda Mui和Tatiana对我们的耐心的支持。

最后，我们要感谢数不清的曾经为社区作出贡献的人们，他们提供了非正式的审计、建议和修正：这一定不是一个最终的完整列表，离开了这些人的帮助，这本书不会这样完整和正确：Jani Averbach, Ryan Barrett, Francois Beausoleil, Jennifer Bevan, Matt Blais, Zack Brown, Martin Buchholz, Brane Cibej, John R. Daily, Peter Davis, Olivier Davy, Robert P. J. Day, Mo DeJong, Brian Denny, Joe Drew, Nick Duffek, Ben Elliston, Justin Erenkrantz, Shlomi Fish, Julian Foad, Chris Foote, Martin Furter, Dave Gilbert, Eric Gillespie, Matthew Gregan, Art Haas, Greg Hudson, Alexis Huxley, Jens B. Jorgensen, Tez Kamihira, David Kimdon, Mark Benedetto King, Andreas J. Koenig, Nuutti Kotivuori, Matt Kraai, Scott Lamb, Vincent Lefevre, Morten Ludvigsen, Paul Lussier, Bruce A. Mah, Philip Martin, Feliciano Matias, Patrick Mayweg, Gareth McCaughan, Jon Middleton, Tim Moloney, Mats Nilsson, Joe Orton, Amy Lyn Pilato, Kevin Pilch-Bisson, Dmitriy Popkov, Michael Price, Mark Proctor, Steffen Prohaska, Daniel Rall, Tobias Ringstrom, Garrett Rooney, Joel Rosdahl, Christian Sauer, Larry Shatzer, Russell Steicke, Sander Striker, Erik Sjoelund, Johan Sundstroem, John Szakmeister, Mason Thomas, Eric Wadsworth, Colin Watson, Alex Waugh, Chad Whitacre, Josef Wolf, Blair Zajac, 以及整个Subversion社区。

7.1. 来自Ben Collins-Sussman

感谢我的妻子Frances，在几个月里，我一直在对你说，“但是亲爱的，我还在为这本书工作”，非比寻常，“但是亲爱的，我还在处理邮件”。我不知道她为什么会如此耐心！她是我完美的平衡点。

感谢我的家人对我的鼓励，无论是否对我的题目感兴趣。（你知道的，一个人说“哇，你正在写一本书？”，然后当他知道你是写一本计算机书时，那种惊讶就

[1]噢，要感谢Karl自己，为了这本书的超时工作。

变得没有那么多了。）

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。

7.2. 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，感谢我的母亲和外祖母的支持，对我在圣诞夜里埋头工作的理解。

Mike和Ben：与你们一起工作非常快乐，Heck，我们在一起工作很愉快！

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。

最后，感谢我的祖父，他一直跟我说“自由等于责任”，我深信不疑。

7.3. 来自C. Michael Pilato

特别感谢我的妻子Amy，由于她的耐心照顾，由于她对我熬夜的容忍，由于她用难以想象的优雅方式修订我的每一个章节—你总能先行一步。Gavin，你已经大到可以阅读了，我希望你能为我这样一个爸爸感到骄傲，像我为你骄傲一样。爸爸妈妈（家庭的其余部分），感谢你们恒久不变的支持和鼓励。

向你们致敬，Shep Kendall，为我打开了计算机世界的大门；Ben Collins Sussman，我在开源世界的导师；Karl Fogel—你是我的.emacs；Greg Stain，让我在困境中知道怎样编程；Brain Fitzpatrick—同我分享他的写作经验。所有我曾经从你们那里获得知识的人—尽管又不断忘记。

最后，对所有为我展现完美卓越创造力的人们—感谢。

第 1 章 介绍

版本控制是管理信息变化的艺术，它很早就成为了程序员重要的工具，程序员经常会花时间做一点小修改然后第二天又把它改回来。但是版本控制的作用不仅在软件开发领域，任何需要管理频繁信息改变的地方都需要它，这就是Subversion发挥的舞台。

这一章是一个对Subversion高层次的介绍—它是什么；它能做什么；它是怎样做到的。

1.1. Subversion是什么？

Subversion是一个自由/开源版本控制系统，它管理文件和目录可以超越时间。一组文件存放在中心版本库，这个版本库很像一个普通的文件服务器，只是它可以记录每一次文件和目录的修改，这便使你可以取得数据以前的版本，从而可以检查所作的更改。从这个方面看，许多人把版本控制系统当作一种“时间机器”。

Subversion可以通过网络访问它的版本库，从而使用户可以在不同的电脑上使用。一定程度上可以说，允许用户在各自的地方修改同一份数据是促进协作。进展可能非常的迅速，并没有一个所有的改变都会取得效果的通道，由于所有的工作都有历史版本，你不必担心由于失去某个通道而影响质量，如果存在不正确的改变，只要取消改变。

一些版本控制系统也是软件配置管理（SCM）系统，这种系统经过特定的精巧设计来管理源代码，有许多关于软件开发的特性—本身理解编程语言、或者提供构建程序的工具。然而，Subversion不是这样一个系统，它是一个通用系统，可以管理任何类型的文件集，对你这可能是源代码—对别人，可能是一个货物清单或者是数字电影。

1.2. Subversion的历史

早在2000年，CollabNet, Inc. (http://www.collab.net) 开始寻找CVS替代产品的开发人员，CollabNet提供了一个协作软件套件SourceCast，它的一个组件是版本控制系统。尽管SourceCast在初始时使用CVS作为其版本控制系统，但是CVS的局限性在一开始就很明显，CollabNet知道迟早要找到一个更好的替代品。遗憾的是，CVS成为了开源世界事实上的标准，因为没有更好的产品，至少是没有可以自由使用的。所以CollabNet决定写一个新的版本控制系统，建立在CVS思想之上的，但是修正其错误和不合理的特性。

2000年2月，他们联系Open Source Development with CVS(Coriolis, 1999)的作者Karl Fogel，并且询问他是否希望为这个新项目工作，巧合的是，当时Karl正在与朋友Jim Blandy讨论设计一个新的版本控制系统。在1995年，他们两个曾经开办一个提供CVS支持的公司Cyclic Software，尽管他们最终卖掉了公司，但还是天天使用CVS进行日常工作，在使用CVS时的挫折最终促使他们认真地去考虑如何管理标记版本的数据，而且他们当时不仅仅提出了“Subversion”这个名字，并且做出了Subversion版本库的基础设计。所以当CollabNet提出邀请的时候，

Karl马上同意为这个项目工作，同时Jim也得到了他的雇主，RedHat软件赞助他到这个项目并提供了一个宽松的时间。CollabNet雇佣了Karl和Ben Collins Sussman，详细的设计从三月开始，在Behlendorf 、CollabNet、Jason Robbins和 Greg Stein（当时是一个独立开发者，活跃在WebDAV/DeltaV系统规范阶段）的恰当激励的帮助下，Subversion很快吸引了许多活跃的开发者，结果是许多有CVS经验的人们很乐于有机会为这个项目做些事情。

最初的设计小组固定在简单的目标上，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匹配CVS的特性，保留相同的开发模型，但不复制CVS明显的缺陷。尽管它不需要成为CVS的继任者，它也应该与CVS保持足够的相似性，使得CVS用户可以轻松的做出转换。

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

当CollabNet开始这个项目的时候，曾经资助了大量的工作（它为全职的Subversion开发者提供薪水），Subversion像许多开源项目一样，被一些激励知识界精英的宽松透明的规则支配着。CollabNet的版权许可证完全符合Debian的自由软件方针，也就是说，任何人可以自由的下载，修改和重新发布，不需要经过CollabNet或其他人的允许。

1.3. Subversion的特性

当讨论Subversion为版本控制领域带来的特性的时候，通过学习它在CVS基础上所作的改进会是比较有效的方法。如果你不熟悉CVS，你会不太明白所有的特性，如果你根本就不熟悉版本控制，你会瞪着眼无所适从，你最好首先阅读一下第 2 章基本概念，它提供了一个版本控制的简单介绍。

Subversion提供：

版本化的目录
: CVS只记录单个文件的历史，但是Subversion实现了一个可以跟踪目录树更改的“虚拟”版本化文件系统，文件和目录都是有版本的。

真实的版本历史
: 因为CVS只记录单个文件的版本，对于拷贝和改名—这些文件经常发生的操作，会改变一个目录的内容—在CVS中并不支持。在CVS里你也不可以用一个完全不同的文件覆盖原来的同名文件而又不继承原来文件的历史。通过Subversion，你可以对文件或是目录进行增加、拷贝和改名操作，也可以新增一个具有干净历史的文件。

原子提交
: 一系列的改动，要么全部提交到版本库，要么一个也不提交，这样可以让用户构建一个所要提交修改的逻辑块，防止部分修改提交到版本库。

版本化的元数据
: 每一个文件或目录都有一套属性—键和它们的值，你可以建立并存储任何键/值对，属性也是随时间的流逝而纳入版本控制的，很像文件的内容。

可选的网络层
Subversion在版本库访问方面有一个抽象概念，利于人们去实现新的网络机制，Subversion可以作为一个扩展模块与Apache结合，这给了Subversion在稳定性和交互性方面很大的好处，可以直接使用服务器的特性—认证、授权和传输压缩等等。也有一个轻型的，单独运行的Subversion服务，这个服务使用自己的协议可以轻松的用SSH封装。

一致的数据操作
Subversion表示文件是建立在二进制文件区别算法基础上的，对于文本（可读）和二进制（不可读）文件具备一致的操作方式，两种类型的文件都压缩存放在版本库中，区别信息是在网络上双向传递的。

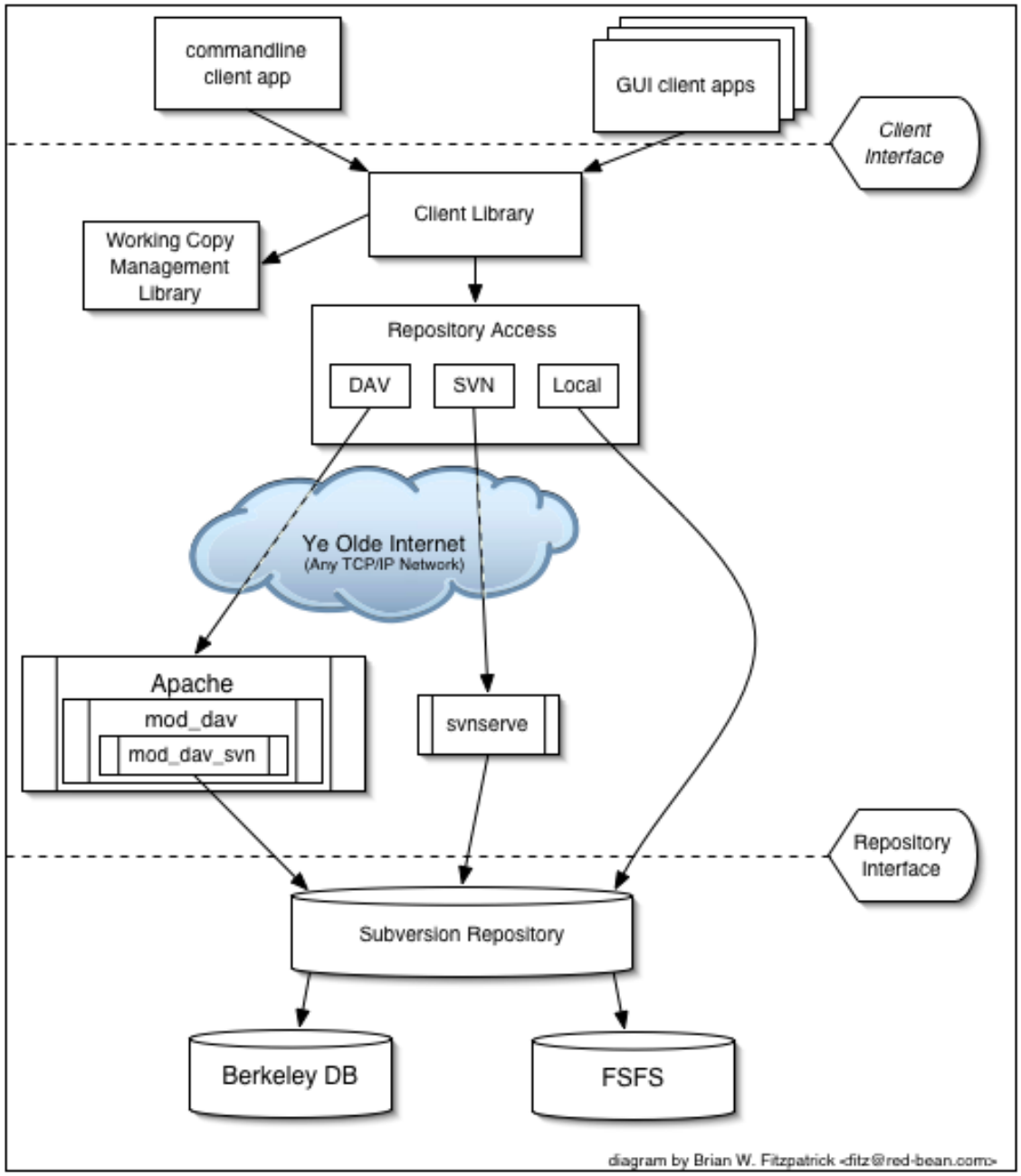
有效率的分支和标签
分支与标签的代价不与工程的大小成比例，Subversion建立分支与标签时只是拷贝整个工程，使用了一种类似于硬链接的机制，因而这类操作通常只会花费很少并且相对固定的时间。

可修改性
Subversion没有历史负担，它由一系列良好的共享C库实现，具有定义良好的API，这使得Subversion非常容易维护，可以轻易的用其他语言操作。

1.4. Subversion的架构

图 1.1 “Subversion的架构”从高处“俯视”Subersion的设计。

图 1.1. Subversion的架构



一端是保存你所有纳入版本控制的数据的Subversion版本库，在另一端是你的Subvesion客户端程序，管理着所有纳入版本控制数据的本地影射（叫做“工作拷贝”），在这两极之间是各种各样的版本库访问（RA）层，一些使用电脑网络通过网络服务器访问版本库，一些则绕过网络服务器直接访问版本库。

1.5. 安装Subversion

Subversion建立在一个可移植层上，叫做APR（Apache Portable Runtime library），这意味着Subversion可以在所有可运行Apache服务器的平台上工作：Windows、Linux、各种BSD、Mac OS X、Netware以及其他。

最简单的安装办法就是下载相应操作系统的二进制包，Subversion的网站（

http://subversion.tigris.org）上通常会有志愿者提供的包可以下载，对于微软操作系统，网站上通常会有图形化的安装包，对于类Unix系统，你可以使用它们本身的打包系统（PRMs、DEBs、ports tree等等）得到Subversion。

你也可以选择从源代码直接编译Subversion，从网站下载最新的源代码，解压缩，根据INSTALL文件的指导进行编译。注意，通过这些源代码可以完全编译访问服务器的命令行客户端工具（通常是apr，apr-util和neno库）。但是可选部分有许多依赖，如Berkeley DB和Apache httpd。如果你希望做一个完全的编译，确定你有所有INSTALL文件中记述的包。如果你计划通过Subversiong本身工作，你可以使用客户端程序取得最新的，带血的源代码，这部分内容见第 8.6.2 节 “取得源代码”。

1.6. Subversion的组件

Subversion安装之后，分为几个部分，这是一个快速浏览。不要害怕这些让你挠头的简略描述，本书有足够的内容来减少这种混乱。

svn
: 命令行客户端。

svnversion
: 报告工作拷贝状态（当前修订版本的项目）的工具。

svnlook
: 检查版本库的工具。

svnadmin
: 建立、调整和修补版本库的工具。

svndumpfilter
: 过滤Subversion版本库转储文件的工具。

mod_dav_svn
: Apache HTTP服务器的一个插件，可以让版本库在网络上可见。

svnserve
: 一种单独运行的服务器，可以作为守护进程由SSH调用，另一种让版本库在网络上可见的方式。

假定你已经将Subversion正确安装，你已经准备好开始，下两章将带领你使用svn，Subversion的客户端程序。

1.7. 快速入门

许多人为“从头到尾”的方式读一本介绍有趣新技术的书感到发愁，这一小节是一个很短的介绍，给许多“实用”的用户一个实战的机会，如果你是一个喜欢通过实验进行学习的用户，以下将告诉你怎么做，相对应，我们给出这本书相关的

链接。

如果版本控制或者Subversion和CVS都用到的“拷贝-修改-合并”模型对于你来说是完全的新概念，在进一步阅读之前，你首先要读第 2 章 基本概念。

注意

以下的例子假定你有了svn这个客户端程序，也有svnadmin这个管理程序，你的svn也应该在Berkeley DB的基础上进行编译。为了验证这些，运行svn --version，确定ra_local模块存在，如果没有，这个程序不能访问file://的URL。

Subversion存储所有版本控制的数据到一个中心版本库，作为开始，新建一个版本库：

```
$ svnadmin create /path/to/repos
$ ls /path/to/repos
conf/  dav/  db/  format  hooks/  locks/  README.txt
```

这个命令建立了一个新的目录 /path/to/repos，包含了一个Subversion版本库。确定这个目录在本地磁盘上，而不是一个网络共享，这个新的目录保存着一些Berkeley DB的数据库文件，你打开后看不到你的已经版本化的文件。更多的版本库创建和维护信息，见第5章第 5 章 版本库管理。

第二步，建立一些将要导入到版本库的文件与目录，为了以后使用更清楚（见第 4 章 分支与合并），你的文件应该包括三个顶级子目录，分别是branches、tags和trunk：

```
/tmp/project/branches/
/tmp/project/tags/
/tmp/project/trunk/
                foo.c
                bar.c
                Makefile
                ...
```

一旦你有了树形结构和数据你就可以继续了，使用svn import导入数据到版本库（见第 3.7.2 节 “svn import”部分）：

```
$ svn import /tmp/project file:///path/to/repos -m "initial import"
Adding         /tmp/project/branches
Adding         /tmp/project/tags
Adding         /tmp/project/trunk
```

```
Adding         /tmp/project/trunk/foo.c
Adding         /tmp/project/trunk/bar.c
Adding         /tmp/project/trunk/Makefile
…
Committed revision 1.
$
```

现在版本库已经包含你的目录和数据了，注意原先的/tmp/project目录没有任何变化；Subversion不管这个，（事实上，你甚至可以任意删除这个目录）。为了处理 版本库的数据，你需要创建一个新的包含数据的“工作拷贝”，一个私人的工作空间。告诉Subversion来“取出”版本库的trunk目录：

```
$ svn checkout file:///path/to/repos/trunk project
A  project/foo.c
A  project/bar.c
A  project/Makefile
…
Checked out revision 1.
```

你现在在project目录里有了一个版本库的个人拷贝，你可以编辑你的工作备份中的文件，并且提交到版本库。

- 进入到你的工作备份，编辑一个文件的内容。
- 运行svn diff来查看你的修改的标准区别输出。
- 运行svn commit来提交你的改变到版本库。
- 运行svn update将你的工作拷贝与版本库“同步”。

对于你对工作拷贝可做操作的完全教程可以察看第 3 章 指导教程。

目前，你可以选择使你的版本库在网络上可见，可以参考第 6 章 配置服务器，学习使用不同的服务器以及配置。

第 2 章 基本概念

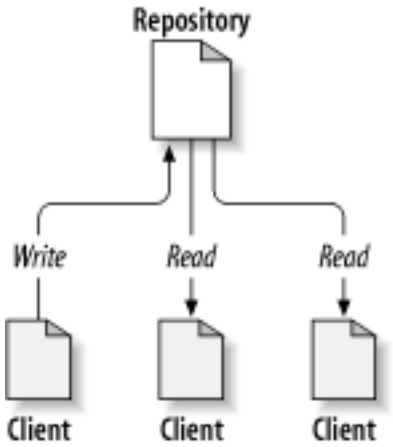
这一章是对Subversion一个简短和随意的介绍，如果你对版本控制很陌生，这一章节完全为你准备的，我们从讨论基本概念开始，深入理解Subversion的思想，然后展示许多简单的实例。

尽管我们的例子展示了人们如何分享程序源代码，仍然要记住Subversion可以控制所有类型的文件—它并没有限制在只为程序员工作。

2.1. 版本库

Subversion是一种集中的分享信息的系统，它的核心是版本库，它储存所有的数据，版本库按照文件树形式储存数据—包括文件和目录。任意数量的客户端可以连接到版本库，读写这些文件。通过写，别人可以看到这些信息，通过读数据，可以看到别人的修改。图 2.1 “一个典型的客户/服务器系统”描述了这种关系：

图 2.1. 一个典型的客户/服务器系统



所以为什么这很有趣呢？讲了这么多，让人感觉这是一种普通的文件服务器，但实际上，版本库是另一种文件服务器，而不是你常见的那一种。最特别的是Subversion会记录每一次的更改，不仅针对文件也包括目录本身，包括增加、删除和重新组织文件和目录。

当一个客户端从版本库读取数据时，通常只会看到最新的版本，但是客户端也可以去看以前的任何一个版本。举个例子，一个客户端可以发出这样的历史问题“上个星期三的目录是怎样的？”或是“谁最后一个更改了这个文件，更改了什么？”，这些是每一种版本控制系统的核心问题：系统是设计来记录和跟踪每一次改动的。

2.2. 版本模型

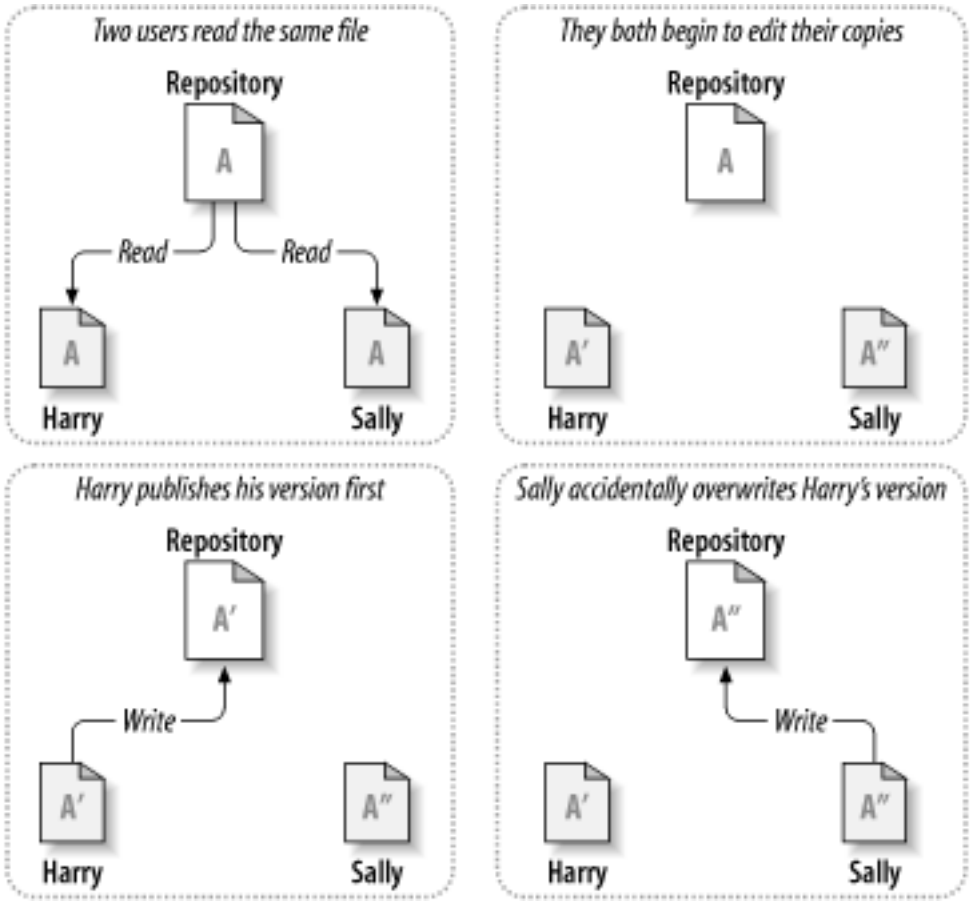
版本控制系统的核心任务是提供协作编辑和数据共享，但是不同的系统使用不同的策略来达到目的。

2.2.1. 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。

考虑图 2.2 “需要避免的问题”的情景，我们有两个共同工作者，Harry和Sally，他们想同时编辑版本库里的同一个文件，如果首先Harry保存它的修改，过了一会，Sally可能凑巧用自己的版本覆盖了这些文件，Harry的更改不会永远消失（因为系统记录了每次修改），Harry所有的修改不会出现在Sally的文件中，所以Harry的工作还是丢失了—至少是从最新的版本中丢失了—而且是意外的，这就是我们要明确避免的情况！

图 2.2. 需要避免的问题

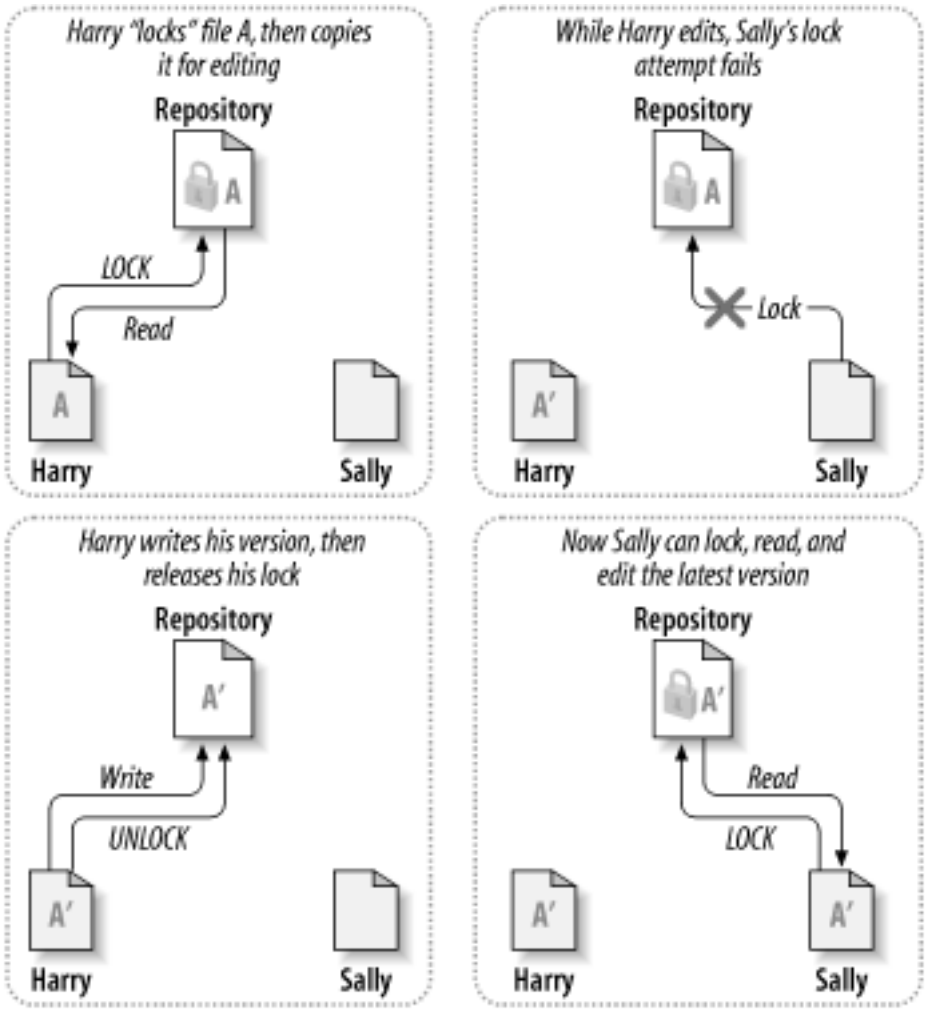


2.2.2. 锁定-修改-解锁 方案

许多版本控制系统使用锁定-修改-解锁这种机制解决这种问题，在这样的系统里，在一个时间段里版本库的一个文件只允许被一个人修改。首先在修改之前，

Harry要“锁定”住这个文件，锁定很像是从图书馆借一本书，如果Harry锁住这个文件，Sally不能做任何修改，如果Sally想请求得到一个锁，版本库会拒绝这个请求。在Harry结束编辑并且放开这个锁之前，她只可以阅读文件。Harry解锁后，就要换班了，Sally得到自己的轮换位置，锁定并且开始编辑这个文件。图 2.3 “锁定-修改-解锁 方案”描述了这样的解决方案。

图 2.3. 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。

2.2.3. 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。

这是一个例子，Harry和Sally为同一个项目各自建立了一个工作拷贝，工作是并行的，修改了同一个文件A，Sally首先保存修改到版本库，当Harry想去提交修改的时候，版本库提示文件A已经过期，换句话说，A在他上次更新之后已经更改了，所以当他通过客户端请求合并版本库和他的工作拷贝之后，碰巧Sally的修改和他的不冲突，所以一旦他把所有的修改集成到一起，他可以将工作拷贝保存到版本库，图 2.4 “拷贝-修改-合并 方案”和图 2.5 “拷贝-修改-合并 方案（续）”展示了这一过程。

图 2.4. 拷贝-修改-合并 方案

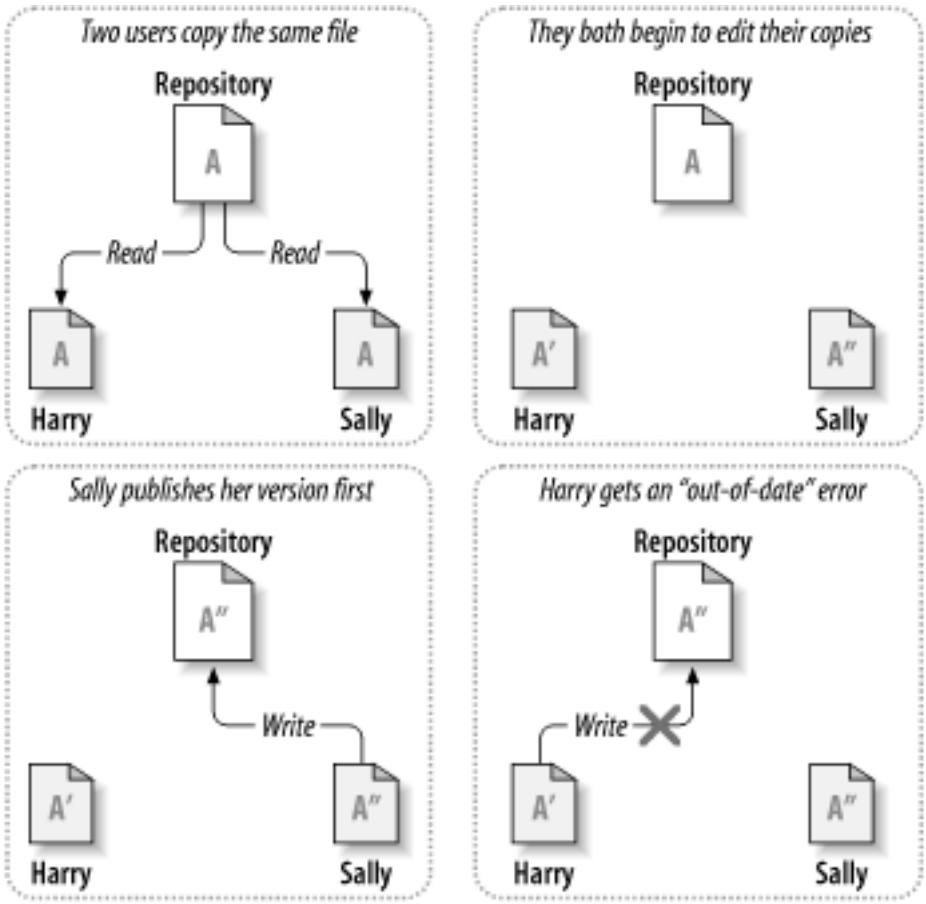
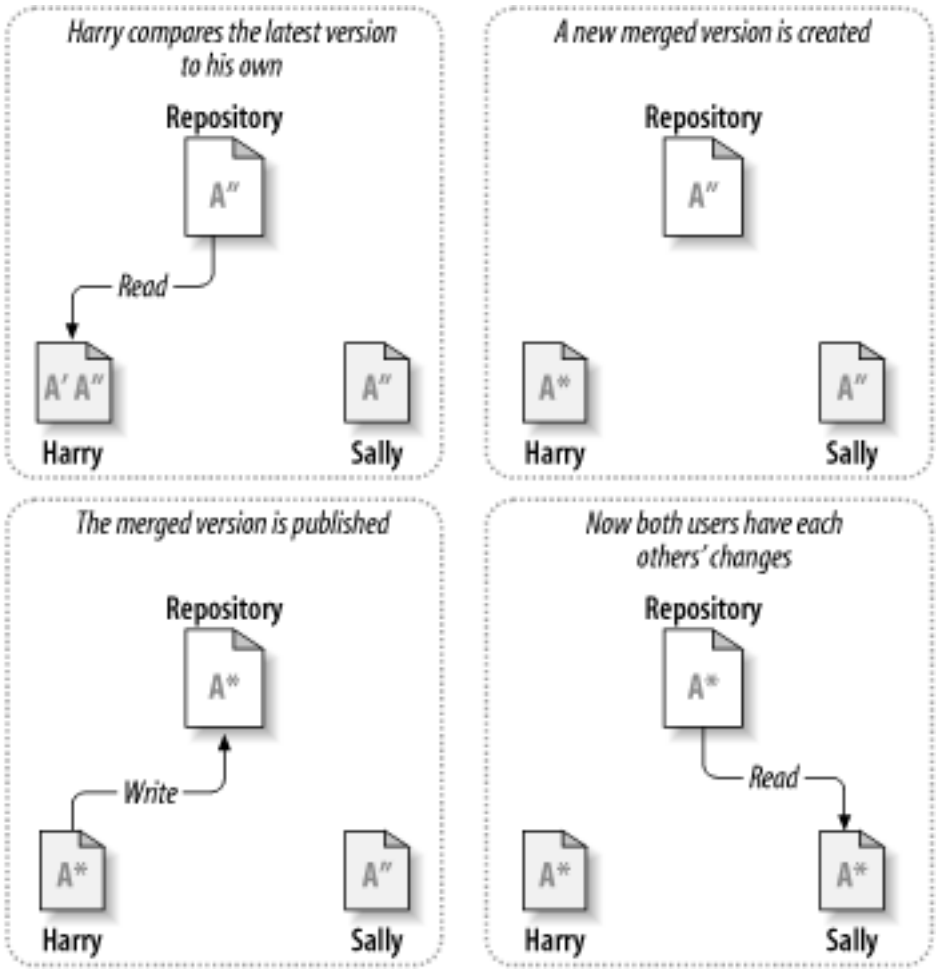


图 2.5. 拷贝-修改-合并 方案（续）



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-合并模型感觉是有一点混乱，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。

2.3. Subversion实战

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。

2.3.1. 工作拷贝

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一个普通的目录，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。

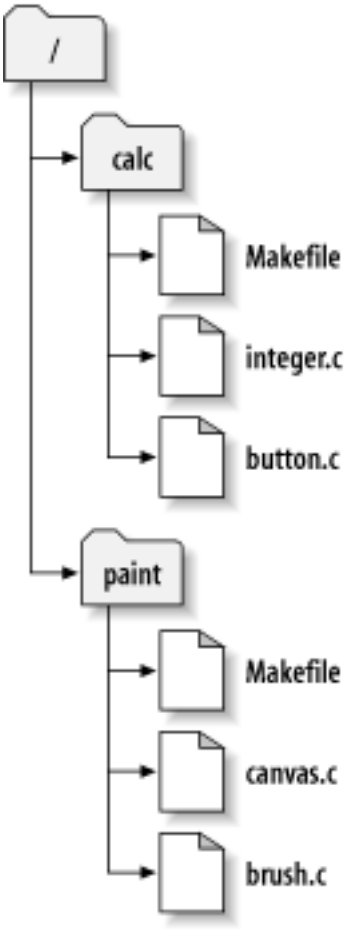
当你在工作拷贝作了一些修改并且确认它们工作正常之后，Subversion提供了一个命令可以“发布”你的修改给项目中的其他人（通过写到版本库），如果别人发布了各自的修改，Subversion提供了手段可以把这些修改与你的工作目录进行合并（通过读取版本库）。

一个工作拷贝也包括一些由Subversion创建并维护的额外文件，用来协助执行这些命令。通常情况下，你的工作拷贝每一个文件夹有一个以.svn为名的文件夹，也被叫做工作拷贝管理目录，这个目录里的文件能够帮助Subversion识别哪一个文件做过修改，哪一个文件相对于别人的工作已经过期了。

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，在这种安排下，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 2.6 “版本库的文件系统”。

图 2.6. 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存资源，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
A  calc/Makefile
A  calc/integer.c
A  calc/button.c

$ ls -A calc
Makefile  integer.c  button.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。

> 版本库的URL
>
> Subversion可以通过多种方式访问—本地磁盘访问，或各种各样不同的网络

协议，但一个版本库地址永远都是一个URL，表格2.1描述了不同的URL模式对应的访问方法。

表 2.1. 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 7.7 节 “Subversion版本库URL”。

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。

发布你的修改给别人，你可以使用Subversion的提交（commit）命令：

```
$ svn commit button.c
Sending        button.c
Transmitting file data .
Committed revision 57.
```

这时你对button.c的修改已经提交到了版本库，如果其他人取出了/calc的一个工作拷贝，他们会看到这个文件最新的版本。

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。

要使项目最新，Sally可以要求Subversion更新她的工作备份，通过使用更新（update）命令，将结合你和所有其他人在她上次更新之后的改变到她的工作拷贝。

```
$ pwd
/home/sally/calc
```

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

$ svn update
U button.c
```

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。

2.3.2. 修订版本

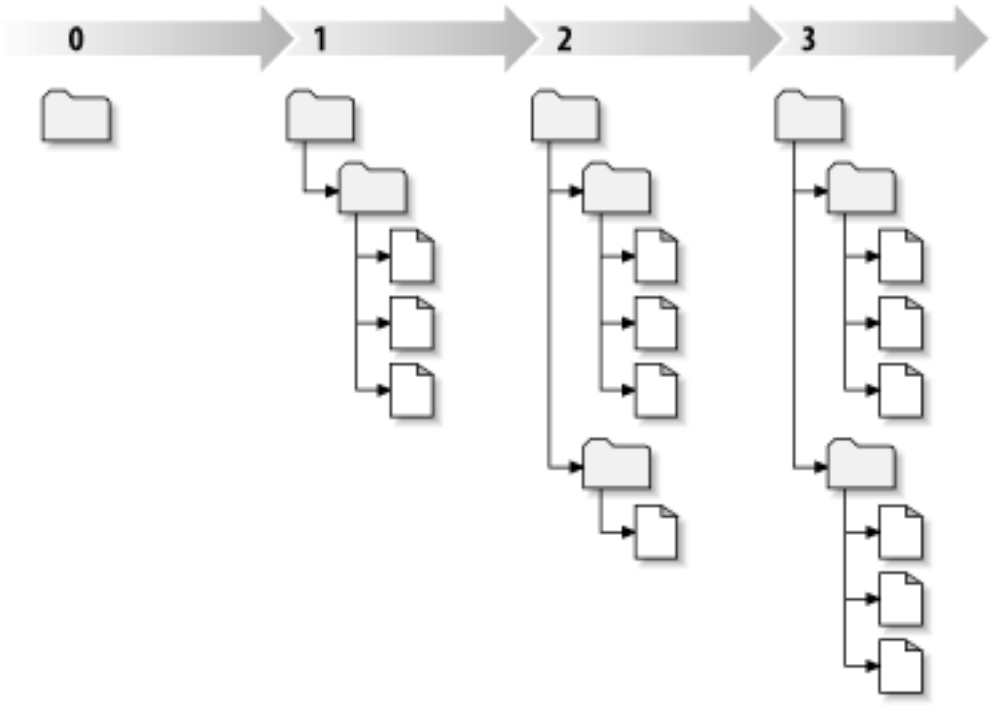
一个svn commit操作可以作为一个原子事务操作发布任意数量文件和目录的修改，在你的工作拷贝里，你可以改变文件内容、删除、改名和拷贝文件和目录，然后作为一个整体提交。

在版本库中，每一次提交被当作一次原子事务操作：要么所有的改变发生，要么都不发生，Subversion努力保持原子性以应对程序错误、系统错误、网络问题和其他用户行为。

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 2.7 “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 2.7. 版本库



全局修订号

不像其他版本控制系统，Subversion的修订号是针对整个目录树的，而不是单个文件。每一个修订号代表了一次提交后版本库整个目录树的特定状态，另一种理解是修订号N代表版本库已经经过了N次提交。当Subversion用户讨论“foo.c的修订号5”时，他们的实际意思是“在修订号5时的foo.c”。需要注意的是，修订号N和M并不表示一个文件需要不同。因为CVS使用每一个文件一个修订号的策略，CVS用户可能希望察看附录 A，Subversion对于CVS用户来得到更多细节。

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
```

假设此刻，Sally提交了对integer.c的修改，建立修订版本6，如果你使用svn update来更新你的工作拷贝，你会看到：

```
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。

2.3.3. 工作拷贝怎样追踪版本库

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：

- 工作文件所作为基准的修订版本（叫做文件的工作修订版本）和
- 一个本地拷贝最后更新的时间戳。

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与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
: 在工作目录已经修改，从基本修订版本之后没有修改提交到版本库。本地修改没有提交，因此svn commit会成功的提交，svn update不做任何事情。

未修改且不是当前的了
: 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。

本地已修改且不是最新的
: 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 3.5.3.1 节 “svn status”。

2.3.4. 修订版本混合的限制

作为通常的原则，Subversion期望尽可能的灵活，一个灵活性的表现就是能够在工作拷贝中混合有不同的修订版本。

起初，为什么把这种灵活性看作一种特性并没有完全看清楚，这也不是一个任务。完成了提交之后，干净的提交的文件比其他文件有更加新的版本，这看起来有些混乱，但是像以前说过的，通过svn update可以使整个版本统一起来， 怎么会有人故意的混合版本呢？

假设你的项目非常复杂，有时候需要强制地使工作拷贝的一部分“回到”某一个日期，你可以在第3章学习如何操作。或许你也希望测试某一目录下子模块早期的版本，或许你想检查某一文件过去的一系列版本在最新目录树环境下的表现。

无论你在工作拷贝中如何利用混合版本，对于这种灵活性是有限制的。

首先，你不可以提交一个不是完全最新的文件或目录，如果有个新的版本存在于版本库，你的删除操作会被拒绝，这防止你不小心破坏你没有见到的东西。

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6章学习附加“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。

2.4. 摘要

我们在这一章里学习了许多Subversion的基本概念：

- 我们介绍了中央版本库的概念、客户工作拷贝和版本修订树。
- 我们介绍了两个协作者如何使用Subversion发布和获得对方的修改，使用“拷贝-修改-合并”模型。
- 我们讨论了一些Subversion跟踪和管理工作拷贝信息的方式。

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 3 章 指导教程

现在，我们将要深入到Subversion到使用细节当中，完成本章，你将学会所有日常使用的Subversion命令，你将从一个初始化检出开始，做出修改并检查，你也将会学到如何将别人的修改取到工作拷贝，检查他们，并解决所有可能发生的冲突。

这一章并不是Subversion命令的完全列表—而是你将会遇到的最常用任务的介绍，这一章假定你已经读过并且理解了第 2 章 基本概念，而且熟悉Subversion的模型，如果想查看所有命令的参考，见第 9 章 Subversion完全参考。

3.1. 帮助！

在继续阅读之前，需要知道Subversion使用中最重要的命令：svn help，Subversion命令行工具是一个自文档的工具—在任何时候你可以运行svn help <subcommand>来查看子命令的语法、参数以及行为方式。

3.2. 导入

使用svn import来导入一个新项目到Subversion的版本库，这恐怕是使用Subversion必定要做的第一步操作，但不是经常发生的事情，详细介绍可以看本章后面的第 3.7.2 节 “svn import”。

3.3. 修订版本：号码、关键字和日期，噢，我的！

在继续之前你一定要知道如何识别版本库的一个修订版本，像你在第 2.3.2 节 “修订版本”看到的，一个修订版本就是版本库的一个“快照”，当你的版本库持续扩大，你必须有手段来识别这些快照。

你可以使用--revision（-r）参数来选择特定修订版本（svn --revision REV），你也可以指定在两个修订版本之间的一个范围 （svn --revision REV1:REV2）。你可以在Subversion中通过修订版本号、关键字或日期指定特定修订版本。

3.3.1. 修订版本号

当你新建了一个Subversion版本库，从修订版本号0开始，每一次成功的提交加1，当你提交成功，Subversion告诉客户端这个新版本号：

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

如果你想在未来使用这个版本（我们将在此章的后面讲述我们这样做的方式和原

因），你可以通过号码“3”指定。

3.3.2. 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的数字参数，这会被Subversion解释到特定版本：

注意

工作拷贝中的每一个目录都有一个叫作.svn的管理目录，工作目录中的每一个文件，Subversion在管理区域为它保留了一个备份，这是上一个版本（叫做“BASE”版本）没有修改的（没有关键字变化，没有行结束符号转化，没有任何改动）拷贝，我们把这个文件当作原始拷贝或基准文件使用，它与版本库中的文件完全一样。

HEAD
　版本库中最新的版本。

BASE
　工作拷贝中的“原始”修订版本。

COMMITTED
　在BASE版本之前（或在Base）一个项目最后修改的版本。

PREV
　一个项目最后修改版本之前的那个版本（技术上为COMMITTED -1）。

注意

PREV、BASE、和COMMITTED指的都是本地路径而不是URL。

下面是一些关键字使用的例子，不要担心现在没有意义，我们将在本章的后面解释这些命令：

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

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

$ svn diff --revision HEAD
# compares your working file (with local mods) to the latest version
# in the repository.
```

```
$ svn diff --revision BASE:HEAD foo.c
# compares your “pristine” foo.c (no local mods) with the
# latest version in the repository

$ svn log --revision BASE:HEAD
# shows all commit logs since you last updated

$ svn update --revision PREV foo.c
# rewinds the last change on foo.c.
# (foo.c's working revision is decreased.)
```

这些关键字允许你执行许多常用（而且有用）的操作，而不必去查询特定的修订版本号，或者记住本地拷贝的修订版本号。

3.3.3. 修订版本日期

在任何你使用特定版本号和版本关键字的地方，你也可以在“{}”中使用日期，你也可通过日期或者版本号配合使用来访问一段时间的修改！

如下是一些Subversion能够接受的日期格式，注意在日期中有空格时需要使用引号。

```
$ svn checkout --revision {2002-02-17}
$ svn checkout --revision {15:30}
$ svn checkout --revision {15:30:00.200000}
$ svn checkout --revision {"2002-02-17 15:30"}
$ svn checkout --revision {"2002-02-17 15:30 +0230"}
$ svn checkout --revision {2002-02-17T15:30}
$ svn checkout --revision {2002-02-17T15:30Z}
$ svn checkout --revision {2002-02-17T15:30-04:00}
$ svn checkout --revision {20020217T1530}
$ svn checkout --revision {20020217T1530Z}
$ svn checkout --revision {20020217T1530-0500}
…
```

当你指定一个日期，Subversion会在版本库找到接近这个日期的最新版本：

```
$ svn log --revision {2002-11-28}
------------------------------------------------------------------------
r12 | ira | 2002-11-27 12:31:51 -0600 (Wed, 27 Nov 2002) | 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2002-11-27），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2002-11-27，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（{"2002-11-27 23:59"}），或是直接使用（{2002-11-28}）。

你可以使用时间段，Subversion会找到这段时间的所有版本：

```
$ svn log --revision {2002-11-20}:{2002-11-29}
…
```

我们也曾经指出，你可以混合日期和修订版本号：

```
$ svn log --revision {2002-11-20}:4040
```

用户一定要认识到这种精巧会成为处理日期的绊脚石，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—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。这将大大破坏Subversion的这种时间—版本转化功能的表现。

3.4. 初始化的Checkout

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，这个拷贝包括版本库中的HEAD（最新的）版本：

```
$ svn checkout http://svn.collab.net/repos/svn/trunk
A  trunk/subversion.dsw
A  trunk/svn_check.dsp
A  trunk/COMMITTERS
A  trunk/configure.in
A  trunk/IDEAS
…
Checked out revision 2499.
```

版本库规划

> 你也许会为在每个URL上包括trunk感到好奇，我们将在第 4 章 分支与合并详细论述这种推荐的规划方式。

尽管上面的例子取出了trunk目录，你也完全可以通过输入特定URL取出任意深度的子目录：

```
$ svn checkout http://svn.collab.net/repos/svn/trunk/doc/book/tools
A  tools/readme-dblite.html
A  tools/fo-stylesheet.xsl
A  tools/svnbook.el
A  tools/dtd
A  tools/dtd/dblite.dtd
…
Checked out revision 2499.
```

因为Subversion使用“拷贝-修改-合并”模型而不是“锁定-修改-解锁”模型（见第 2 章 基本概念），你可以开始修改工作拷贝中的目录和文件，你的工作拷贝和你的系统中的其它文件和目录完全一样，你可以编辑并改变它，移动它，也可以完全的删掉它，把它忘了。

注意

因为你的工作拷贝“同你的系统上的文件和目录没有什么区别”，如果你希望重新规划工作拷贝，你必须要让Subversion知道，当你希望拷贝或者移动工作拷贝的一个项目时，你应该使用svn copy或者 svn move而不要使用操作系统的命令，我们会在以后的章节详细介绍。

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。

> .svn目录包含什么？
>
> 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。

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/subversion.dsw
```

```
A  subv/svn_check.dsp
A  subv/COMMITTERS
A  subv/configure.in
A  subv/IDEAS
…
Checked out revision 2499.
```

这样将把你的工作拷贝放到subv而不是和前面那样放到trunk。

3.5. 基本的工作周期

Subversion有许多特性、选项和华而不实的高级功能，但日常的工作中你只使用其中的一小部分，有一些只在特殊情况才会使用，在这一节里，我们会介绍许多你在日常工作中常见的命令。

典型的工作周期是这样的：

- 更新你的工作拷贝
  - svn update
- 做出修改
  - svn add
  - svn delete
  - svn copy
  - svn move
- 检验修改
  - svn status
  - svn diff
  - svn revert
- 合并别人的修改到工作拷贝
  - svn update
  - svn resolved
- 提交你的修改
  - svn commit

3.5.1. 更新你的工作拷贝

当你在一个团队的项目里工作时，你希望更新你的工作拷贝得到所有其他人这段时间作出的修改，使用svn update让你的工作拷贝与最新的版本同步。

```
$ 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。

让我们认真检查svn update的输出，当服务器发送修改到你的工作拷贝，一个字母显示在每一个项目之前，来让你知道Subversion对你的工作拷贝做了什么操作：

U foo
文件foo更新了（从服务器收到修改）。

A foo
文件或目录foo被添加到工作拷贝。

D foo
文件或目录foo在工作拷贝被删除了。

R foo
文件或目录foo在工作拷贝已经被替换了，这是说，foo被删除，而一个新的同样名字的项目添加进来，它们具有同样的名字，但是版本库会把它们看作具备不同历史的不同对象。

G foo
文件foo接收到版本库的更改，你的本地版本也已经修改，但改变没有互相影响，Subversion成功的将版本库和本地文件合并，没有发生任何问题。

C foo
文件foo的修改与服务器冲突，服务器的修改与你的修改交迭在一起，不要恐慌，这种冲突需要人（你）来解决，我们在后面的章节讨论这种情况。

3.5.2. 修改你的工作拷贝

现在你可以开始工作并且修改你的工作拷贝了，你很容易决定作出一个修改（或者是一组），像写一个新的特性，修正一个错误等等。这时可以使用的Subversion命令包括svn add、 svn delete、svn copy和svn move。如果你只是修改版本库中已经存在的文件，在你提交之前，不必使用上面的任何一个命令。你可以对工作备份作的修改包括：

文件修改
: 这是最简单的一种修改，你不必告诉Subversion你想修改哪一个文件，只需要去修改，然后Subversion会自动地探测到哪些文件已经更改了。

目录树修改
: 你可以“标记”目录或者文件为预定要删除、增加、复制或者移动，也许这些改动在你的工作拷贝马上发生，而版本库只在你提交的时候才发生改变。

修改文件，可以使用文本编辑器、字处理软件、图形程序或任何你常用的工具，Subverion处理二进制文件像同文本文件一样—效率也一样。

这些是常用的可以修改目录树结构的子命令（我们会在后面包括svn import和svn mkdir）。

警告

你可以使用任何你喜欢的工具编辑文件，但你不可以在修改目录结构时不通知Subversion，需要使用svn copy、svn delete和svn move命令修改工作拷贝的结构，使用svn add增加版本控制的新文件或目录。

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，使用--non-recursive（-N）参数。

svn delete foo
: 预定将文件、目录或者符号链foo从版本库中删除掉，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[1]，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。

svn copy foo bar
: 建立一个新的项目bar作为foo的复制品，当在下次提交时会将bar添加到版本库，这种拷贝历史会记录下来（按照来自foo的方式记录），svn copy并不建立中介目录。

svn move foo bar
: 这个命令与与运行svn copy foo bar; svn delete foo完全相同，bar作为foo的拷贝准备添加，foo已经预定要被删除，svn move不建立中介的目录。

不通过工作拷贝修改版本库

[1]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。

本章的前面曾经说过，为了使版本库反映你的改动，你应该提交所有改动。这并不完全正确—有一些方式是可以直接操作版本库的，当然只有子命令直接操作URL而不是本地拷贝路径时才可以实现，通常svn mkdir、svn copy、svn move、和 svn delete可以使用URL工作。

指定URL的操作方式有一些区别，因为在使用工作拷贝的运作方式时，工作拷贝成为一个“集结地”，可以在提交之前整理组织所要做的修改，直接对URL操作就没有这种奢侈，所以当你直接操作URL的时候，所有以上的动作代表一个立即的提交。

3.5.3. 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、svn diff和svn revert精确地察看所做的修改。你可以使用前两个命令察看工作拷贝中的修改，使用第三个来撤销部分（或全部)的修改。

Subversion已经被优化来帮助你完成这个任务，可以在不与版本库通讯的情况下做许多事情，详细来说，对于每一个文件，你的的工作拷贝在.svn包含了一个“原始的”拷贝，所以Subversion可以快速的告诉你那些文件修改了，甚至允许你在不与版本库通讯的情况下恢复修改。

3.5.3.1. svn status

相对于其他命令，你会更多地使用这个svn status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。

在Subversion，update只是做这件事—将工作拷贝更新到版本库的最新版本，你可以消除使用update察看本地修改的习惯。

如果你在工作拷贝的顶级目录运行不带参数的svn status命令，它会检测你做的所有的文件或目录的修改，以下的例子是来展示svn status可能返回的状态码（注意，#之后的不是svn status打印的）。

```
  L     abc.c                 # svn已经在.svn目录锁定了abc.c
M       bar.c                 # bar.c的内容已经在本地修改过了
 M      baz.c                 # baz.c属性有修改，但没有内容修改
```

```
X      3rd_party           # 这个目录是外部定义的一部分
?      foo.o               # svn并没有管理foo.o
!      some_dir            # svn管理这个，但它可能丢失或者不完整
~      qux                 # 作为file/dir/link进行了版本控制，但类型已经改变
I      .screenrc           # svn不管理这个，配置确定要忽略它
A  +   moved_dir           # 包含历史的添加，历史记录了它的来历
M  +   moved_dir/README    # 包含历史的添加，并有了本地修改
D      stuff/fish.c        # 这个文件预定要删除
A      stuff/loot/bloo.h   # 这个文件预定要添加
C      stuff/loot/lump.c   # 这个文件在更新时发生冲突
R      xyz.c               # 这个文件预定要被替换
    S  stuff/squawk        # 这个文件已经跳转到了分支
```

在这种格式下，svn status打印五列字符，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的状态或它的内容，返回代码解释如下：

A item
文件、目录或是符号链item预定加入到版本库。

C item
文件item发生冲突，在从服务器更新时与本地版本发生交迭，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。

R item
文件、目录或是符号链item预定将要替换版本库中的item，这意味着这个对象首先要被删除，另外一个同名的对象将要被添加，所有的操作发生在一个修订版本。

X item
目录没有版本化，但是与Subversion的外部定义关联，关于外部定义，可以看第 7.4 节 “外部定义”。

? item
文件、目录或是符号链item不在版本控制之下，你可以通过使用svn status的--quiet（-q）参数或父目录的svn:ignore属性忽略这个问题，关于忽略文件的使用，见第 7.2.3.3 节 “svn:ignore”。

! item
文件、目录或是符号链item在版本控制之下，但是已经丢失或者不完整，这可能因为使用非Subversion命令删除造成的，如果是一个目录，有可能是检出或是更新时的中断造成的，使用svn update可以重新从版本库获得文件或者目录

，也可以使用svn revert file恢复原来的文件。

~ item
文件、目录或是符号链item在版本库已经存在，但你的工作拷贝中的是另一个。举一个例子，你删除了一个版本库的文件，新建了一个在原来的位置，而且整个过程中没有使用svn delete或是svn add。

I item
文件、目录或是符号链item不在版本控制下，Subversion已经配置好了会在svn add、svn import和svn status命令忽略这个文件，关于忽略文件，见第 7.2.3.3 节 “svn:ignore”。注意，这个符号只会在使用svn status的参数--no-ignore时才会出现—否则这个文件会被忽略且不会显示!

第二列说明文件或目录的属性的状态（更多细节可以看第 7.2 节 “属性”），如果一个M出现在第二列，说明属性被修改了，否则显示空白。

第三列只显示空白或者L，L表示Subversion已经在.svn工作区域锁定了这个项目，当你的svn commit正在运行的时候—也许正在输入log信息，运行svn status你可以看到L标记，如果这时候Subversion并没有运行，可以推测Subversion发生中断并且已经锁定，你必须运行svn cleanup来清除锁定（本节后面将有更多论述）。

第四列只会显示空白或+，+的意思是一个有附加历史信息的文件或目录预定添加或者修改到版本库，通常出现在svn move或是svn copy时，如果是看到A +就是说要包含历史的增加，它可以是一个文件或是拷贝的根目录。+表示它是即将包含历史增加到版本库的目录的一部分，也就是说他的父目录要拷贝，它只是跟着一起的。 M +表示将要包含历史的增加，并且已经更改了。当你提交时，首先会随父目录进行包含历史的增加，然后本地的修改提交到更改后的版本。

第五列只显示空白或是S，表示这个目录或文件已经转到了一个分支下了（使用svn switch）。

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：

```
$ svn status --verbose
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。

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

```
$ svn status --show-updates --verbose
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
```

注意这两个星号：如果你现在执行svn update，你的README和trout.c会被更新，这告诉你许多有用的信息—你可以在提交之前，需要使用更新操作得到文件README的更新，或者说文件已经过时，版本库会拒绝了你的提交。（后面还有更多关于此主题）。

3.5.3.2. svn diff

另一种检查修改的方式是svn diff命令，你可以通过不带参数的svn diff精确的找出你所做的修改，这会输出统一区别格式：[2]

```
$ 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>
```

[2] Subversion使用内置区别引擎，缺省情况下输出为统一区别格式。如果你期望不同的输出格式，你可以使用--diff-cmd指定外置的区别程序，并且通过--extensions传递其他参数，举个例子，察看本地文件foo.c的区别，同时忽略空格修改，你可以运行svn diff --diff-cmd /usr/bin/diff --extensions '-bc' foo.c。

```
 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
```

举个例子，你可以把补丁文件发送邮件到其他开发者，在提交之前审核和测试。

3.5.3.3. svn revert

假设你通过上面的diff输出发现你不小心用编辑器在README中输入了一些字符。

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的效果与删除ITEM然后执行svn update -r BASE ITEM完全一样，但是，如果你使用svn revert它不必通知版本库就可以恢复文件。

或许你不小心删除了一个文件：

```
$ svn status README
       README

$ svn delete README
D         README

$ svn revert README
Reverted 'README'

$ svn status README
       README
```

看！没有网络！

这三个命令（svn status、svn diff和 svn revert）都可以在没有网络的情

况下工作，这让你在没有网络连接时的管理修改过程更加简单，像在飞机上旅行，乘坐火车往返或是在海滩上奋力工作时。

Subversion通过在.svn管理区域使用原始的版本缓存来做到这一点，这使得恢复本地版本而不必访问网络，这个缓存（叫做“text-base”）也允许Subversion可以根据原始版本生成一个压缩的增量（“区别”） 提交—即使你有个非常快的网络，有这样一个缓存有极大的好处，非常的快，只向服务器提交修改的部分，这一点乍一看好像并不重要，但当你要提交一个400M大小的文件的修改时，你就会明白!

3.5.4. 解决冲突（合并别人的修改）

我们可以使用svn status -u来预测冲突，当你运行svn update一些有趣的事情发生了：

```
$ svn update
U  INSTALL
G  README
C  bar.c
Updated to revision 46.
```

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，你需要自己手工去解决。

当冲突发生了，有三件事可以帮助你注意到这种情况和解决问题：

- Subversion打印C标记，并且标记这个文件已冲突。
- 如果Subversion认为这个文件是可合并的，它会置入冲突标记—特殊的横线分开冲突的“两面”—在文件里可视化的描述重叠的部分（Subversion使用svn:mime-type属性来决定一个文件是否可以使用上下文的，以行为基础合并，更多信息可以看第 7.2.3.2 节 “svn:mime-type”）。
- 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
  你更新前的文件，没有冲突标志，只是你最新更改的内容。（如果Subversion认为这个文件不可以合并，.mine文件不会创建，因为它和工作文件相同。）

  filename.rOLDREV

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.rNEWREV
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。

举一个例子，Sally修改了sandwich.txt，Harry刚刚改变了他的本地拷贝中的这个文件并且提交到服务器，Sally在提交之前更新它的工作拷贝得到了冲突：

```
$ svn update
C  sandwich.txt
Updated to revision 2.
$ ls -1
sandwich.txt
sandwich.txt.mine
sandwich.txt.r1
sandwich.txt.r2
```

在这种情况下，Subversion不会允许你提交sandwich.txt，直到你的三个临时文件被删掉。

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

如果你遇到冲突，三件事你可以选择：

- “手动”合并冲突文本（检查和修改文件中的冲突标志）。
- 用某一个临时文件覆盖你的工作文件。
- 运行svn revert <filename>来放弃所有的修改。

一旦你解决了冲突，你需要通过命令svn resolved让Subversion知道，这样就会删除三个临时文件，Subversion就不会认为这个文件是在冲突状态了。[3]

[3]你也可以手工的删除这三个临时文件，但是当Subversion会给你做时你会自己去做吗？我们是这样想的。

```
$ svn resolved sandwich.txt
Resolved conflicted state of 'sandwich.txt'
```

3.5.4.1. 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。

这里一个简单的例子，由于不良的交流，你和同事Sally，同时编辑了sandwich.txt。Sally提交了修改，当你准备更新你的版本，冲突发生了，我们不得不去修改sandwich.txt来解决这个问题。首先，看一下这个文件：

```
$ 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
```

小于号、等于号和大于号串是冲突标记，并不是冲突的数据，你一定要确定这些内容在下次提交之前得到删除，前两组标志中间的内容是你在冲突区所做的修改：

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

后两组之间的是Sally提交的修改冲突：

```
=======
Sauerkraut
```

```
Grilled Chicken
>>>>>>> .r2
```

通常你并不希望只是删除冲突标志和Sally的修改—当她收到三明治时，会非常的吃惊。所以你应该走到她的办公室或是拿起电话告诉Sally，你没办法从从意大利熟食店得到想要的泡菜。[4]一旦你们确认了提交内容后，修改文件并且删除冲突标志。

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

现在运行svn resolved，你已经准备好提交了：

```
$ svn resolved sandwich.txt
$ svn commit -m "Go ahead and use my sandwich, discarding Sally's edits."
```

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。你也可以使用第三方的合并工具检验这三个文件。

3.5.4.2. 拷贝覆盖你的工作文件

如果你只是希望取消你的修改，你可以仅仅拷贝Subversion为你生成的文件替换你的工作拷贝：

```
$ svn update
C  sandwich.txt
Updated to revision 2.
$ ls sandwich.*
sandwich.txt  sandwich.txt.mine  sandwich.txt.r2  sandwich.txt.r1
$ cp sandwich.txt.r2 sandwich.txt
$ svn resolved sandwich.txt
```

[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。

3.5.4.3. 下注：使用svn revert

如果你得到冲突，经过检查你决定取消自己的修改并且重新编辑，你可以恢复你的修改：

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

注意，当你恢复一个冲突的文件时，不需要再运行svn resolved。

现在我们准备好提交修改了，注意svn resolved不像我们本章学过的其他命令一样需要参数，在任何你认为解决了冲突的时候，只需要小心运行svn resolved，——一旦删除了临时文件，Subversion会让你提交这文件，即使文件中还存在冲突标记。

3.5.5. 提交你得修改

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。

svn commit命令发送所有的修改到版本库，当你提交修改时，你需要提供一些描述修改的日志信息，你的信息会附到这个修订版本上，如果信息很简短，你可以在命令行中使用--message（-m）选项：

```
$ svn commit --message "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，你也许会希望通过告诉Subversion一个文件名得到日志信息，使用--file选项：

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

如果你没有指定--message或者--file选项，Subversion会自动地启动你最喜欢的编辑器（见第 7.1.3.2 节 “config”的editor-cmd部分）来编辑日志信息。

提示

如果你使用编辑器撰写日志信息时希望取消提交，你可以直接关掉编辑器，不要保存，如果你已经做过保存，只要简单的删掉所有的文本并再次保存。

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

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

版本库不知道也不关心你的修改作为一个整体是否有意义，它只检查是否有其他人修改了同一个文件，如果别人已经这样做了，你的整个提交会失败，并且提示你一个或多个文件已经过时了：

```
$ svn commit --message "Add another rule"
Sending        rules.txt
svn: Commit failed (details follow):
svn: Out of date: 'rules.txt' in transaction 'g'
```

此刻，你需要运行svn update来处理所有的合并和冲突，然后再尝试提交。

我们已经覆盖了Subversion基本的工作周期，还有许多其它特性可以管理你得版本库和工作拷贝，但是只使用前面介绍的命令你就可以很轻松的工作了。

3.6. 检验历史

我们曾经说过，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。

有许多命令可以为你提供版本库历史：

`svn log`
展示给你主要信息：附加在版本上的日志信息和所有版本的路径修改。

`svn diff`
展示一个文件改变的详细情况。

svn cat
取得在特定版本的某一个文件显示在当前屏幕。

svn list
显示一个目录在某一版本存在的文件。

3.6.1. svn log

找出一个文件或目录的历史信息，使用svn log命令，svn log将会提供你一条记录，包括：谁对文件或目录作了修改、哪个修订版本作了修改、修订版本的日期和时间、还有如果你当时提供了日志信息，也会显示。

```
$ svn log
------------------------------------------------------------------------
r3 | sally | Mon, 15 Jul 2002 18:03:46 -0500 | 1 line

Added include lines and corrected # of cheese slices.
------------------------------------------------------------------------
r2 | harry | Mon, 15 Jul 2002 17:47:57 -0500 | 1 line

Added main() methods.
------------------------------------------------------------------------
r1 | sally | Mon, 15 Jul 2002 17:40:08 -0500 | 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 --revision 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。

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 | 2002-07-14 08:15:29 -0500 | 1 line
Changed paths:
M /trunk/code/foo.c
M /trunk/code/bar.h
A /trunk/code/doc/README

Frozzled the sub-space winch.

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

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
------------------------------------------------------------------------
$
```

乍一看，好像是一个错误，但是想一下修订版本号是作用在版本库整体之上的，如果你没有提供路径，svn log会使用当前目录作为默认的目标，所以，作为结果，如果你对一个本身和子目录在指定版本到现在没有做过修改的目录运行这个命令，你会得到空的日志。如果你希望察看某个版本做的修改的日志，只需要直接告诉svn log使用版本库顶级的目录作为参数，例如svn log -r 2 http://svn.collab.net/repos/svn。

3.6.2. svn diff

我们已经看过svn diff—使用标准区别文件格式显示区别，它在提交前用来显示本地工作拷贝与版本库的区别。

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
- 比较版本库和版本库

3.6.2.1. 比较本地修改

像我们看到的，不使用任何参数调用时，svn diff将会比较你的工作文件与缓存在.svn的“原始”拷贝：

```
$ 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
$
```

3.6.2.2. 比较工作拷贝和版本库

如果传递一个--revision（-r）参数，你的工作拷贝会与指定的版本比较。

```
$ svn diff --revision 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
$
```

3.6.2.3. 比较版本库与版本库

如果通过--revision （-r）传递两个版本号，通过冒号分开，这两个版本会进行

比较。

```
$ svn diff --revision 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
$
```

你不仅可以用svn diff比较你工作拷贝中的文件，你甚至可以通过提供一个URL参数来比较版本库中两个文件的的区别，通常在本地机器没有工作拷贝时非常有用：

```
$ svn diff --revision 4:5 http://svn.red-bean.com/repos/example/trunk/text/rules.txt
…
$
```

3.6.3. 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 --revision 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 --revision 2 rules.txt > rules.txt.v2
$
```

你一定疑惑为什么不只是使用svn update --revision ，将文件更新到旧的文件，我们有使用svn cat的原因。

首先，你或许希望使用外置的比较工具（或许是一个图形化的工具，或者你的格式无法用标准区别格式察看）察看这两个版本的区别，这种情况下，你需要得到一个旧的版本的拷贝，所以重定向到一个文件，并且在你的比较工具中指定这两个版本来察看区别。

有时候察看整个文件比只看区别要容易。

3.6.4. 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 --verbose http://svn.collab.net/repos/svn
   2755 harry          1331 Jul 28 02:07 README
   2773 sally               Jul 29 15:07 branches/
   2769 sally               Jul 29 12:07 clients/
   2698 harry               Jul 24 18:07 tags/
   2785 sally               Jul 29 19:07 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。

3.6.5. 关于历史的最后一个词

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

[5]看到了吧？我们说过Subversion是一个时间机器。

3.7. 其他有用的命令

不象这章前面讨论的那些经常用到的命令，这些命令只是偶尔被用到。

3.7.1. svn cleanup

当Subversion改变你的工作拷贝（或是.svn中的任何信息），它会尽可能的小心，在修改任何事情之前，它把意图写到日志文件中去，然后执行log文件中的命令，然后删掉日志文件，这与分类帐的文件系统架构类似。如果Subversion的操作中断了（举个例子：进程被杀死了，机器死掉了），日志文件会保存在硬盘上，通过重新执行日志文件，Subversion可以完成上一次开始的操作，你的工作拷贝可以回到一致的状态。

这就是svn cleanup所作的：它查找工作拷贝中的所有遗留的日志文件，删除进程中的锁。如果Subversion告诉你工作拷贝中的一部分已经“锁定”了，你就需要运行这个命令了。同样，svn status将会使用L 显示锁定的项目：

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

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

3.7.2. svn import

svn import命令是拷贝用户的一个未被版本化的目录树到版本库最快的方法，如果需要，它也要建立一些中介文件。

```
$ svnadmin create /usr/local/svn/newrepos
$ svn import mytree file:///usr/local/svn/newrepos/some/project
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:///usr/local/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。

3.8. 摘要

我们已经覆盖了大多数Subversion的客户端命令，引人注目的例外是处理分支与合并（见第 4 章 分支与合并）以及属性（见第 7.2 节 “属性”）的命令，然而你也许会希望跳到第 9 章 Subversion完全参考来察看所有不同的命令—怎样利用它们使你的工作更容易。

第 4 章 分支与合并

分支、标签和合并是所有版本控制系统的共同概念，如果你并不熟悉这些概念，我们会在这一章里很好的介绍，如果你很熟悉，非常希望你有兴趣知道Subversion是怎样实现这些概念的。

分支是版本控制的基础组成部分，如果你允许Subversion来管理你的数据，这个特性将是你所必须依赖的 ，这一章假定你已经熟悉了Subversion的基本概念（第 2 章 基本概念）。

4.1. 什么是分支?

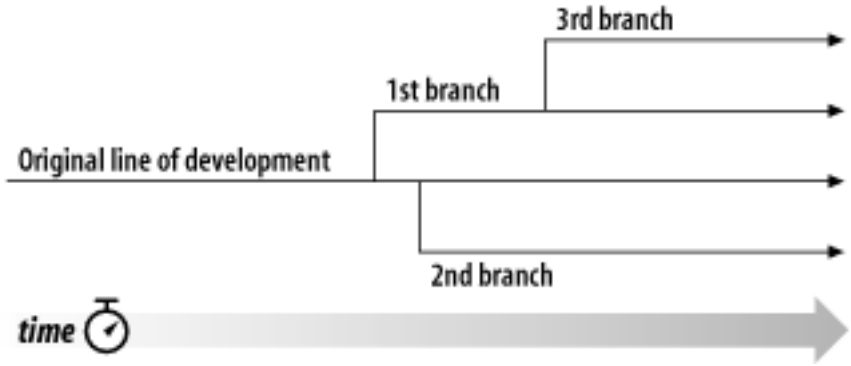
假设你的工作是维护本公司一个部门的手册文档，一天，另一个部门问你要相同的手册，但一些地方会有“区别”，因为他们有不同的需要。

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 “分支开发”）。

图 4.1. 分支开发



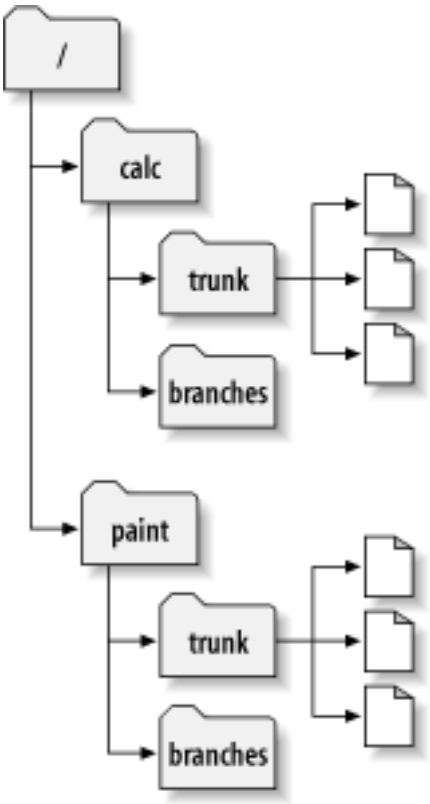
Subversion允许你并行的维护文件和目录的分支，它允许你通过拷贝数据建立分支，记住，分支互相联系，它也帮助你从一个分支复制修改到另一个分支。最终，它可以让你的工作拷贝反映到不同的分支上，所以你在日常工作可以“混合和比较”不同的开发线。

4.2. 使用分支

在这一点上，你必须理解每一次提交是怎样建立整个新的文件系统树（叫做“修订版本”）的，如果没有，可以回头去读第 2.3.2 节 “修订版本”。

对于本章节，我们会回到第2章的同一个例子，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”。

假定你有一个任务，将要对项目做基本的重新组织，这需要花费大量时间来完成，会影响项目的所有文件，问题是你不会希望打扰Sally，她正在处理这样或那样的程序小Bug，一直使用整个项目（/calc/trunk）的最新版本，如果你一点一点的提交你的修改，你一定会干扰Sally的工作。

一种策略是自己闭门造车：你和Sally可以停止一个到两个星期的共享，也就是说，开始作出本质上的修改和重新组织工作拷贝的文件，但是在完成这个任务之前不做提交和更新。这样会有很多问题，首先，这样并不安全，许多人习惯频繁的保存修改到版本库，工作拷贝一定有许多意外的修改。第二，这样并不灵活，如果你的工作在不同的计算机（或许你在不同的机器有两份/calc/trunk的工作拷贝），你需要手工的来回拷贝修改，或者只在一个计算机上工作，这时很难做到共

享你即时的修改，一项软件开发的“最佳实践”就是允许审核你做过的工作，如果没有人看到你的提交，你失去了潜在的反馈。最后，当你完成了公司主干代码的修改工作，你会发现合并你的工作拷贝和公司的主干代码会是一件非常困难的事情，Sally（或者其他人）也许已经对版本库做了许多修改，已经很难和你的工作拷贝结合—当你单独工作几周后运行svn update时就会发现这一点。

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

建立分支非常的简单—使用svn copy命令给你的工程做个拷贝，Subversion不仅可以拷贝单个文件，也可以拷贝整个目录，在目前情况下，你希望作/calc/trunk的拷贝，新的拷贝应该在哪里？在你希望的任何地方—它只是在于项目的政策，我们假设你们项目的政策是在/calc/branches建立分支，并且你希望把你的分支叫做my-calc-branch，你希望建立一个新的目录/calc/branches/my-calc-branch，作为/calc/trunk的拷贝开始它的生命周期。

有两个方法作拷贝，我们首先介绍一个混乱的方法，只是让概念更清楚，作为开始，取出一个工程的根目录，/calc：

```
$ svn checkout http://svn.example.com/repos/calc bigwc
A  bigwc/trunk/
A  bigwc/trunk/Makefile
A  bigwc/trunk/integer.c
A  bigwc/trunk/button.c
A  bigwc/branches/
Checked out revision 340.
```

建立一个备份只是传递两个目录参数到svn copy命令：

```
$ cd bigwc
$ svn copy trunk branches/my-calc-branch
$ svn status
A  +   branches/my-calc-branch
```

在这个情况下，svn copy命令迭代的将trunk工作目录拷贝到一个新的目录branhes/my-calc-branch，像你从svn status看到的，新的目录是准备添加到版本库的，但是也要注意A后面的“+”号，这表明这个准备添加的东西是一份备份，而不是新的东西。当你提交修改，Subversion会通过拷贝/calc/trunk建立/calc/branches/my-calc-branch目录，而不是通过网络传递所有数据：

```
$ svn commit -m "Creating a private branch of /calc/trunk."
```

```
Adding         branches/my-calc-branch
Committed revision 341.
```

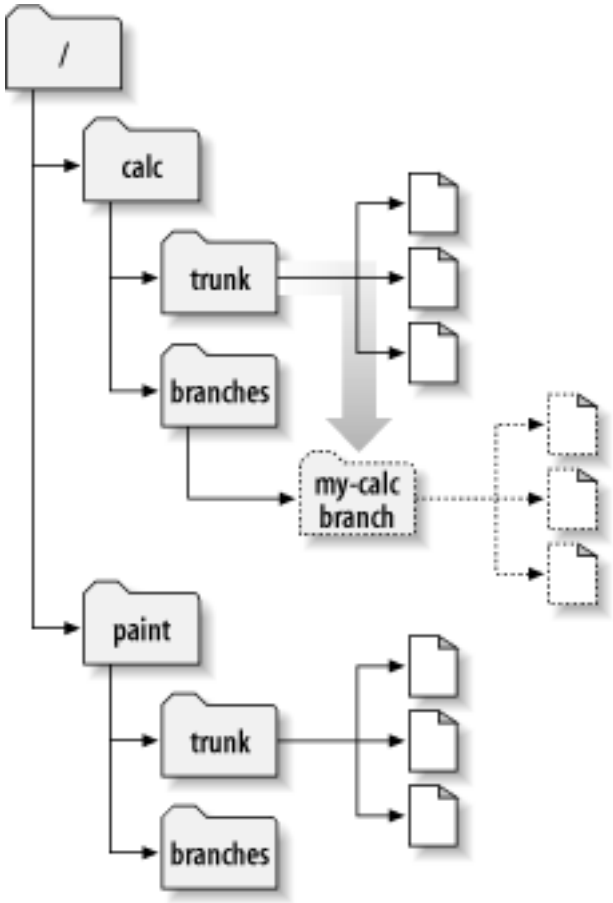
现在，我们必须告诉你建立分支最简单的方法：svn copy可以直接对两个URL操作。

```
$ 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.
```

其实这两种方法没有什么区别，两个过程都在版本341建立了一个新目录作为/calc/trunk的一个备份，这些可以在图 4.3 “拷贝后的版本库”看到，注意第二种方法，只是执行了一个立即提交。[1]这是一个简单的过程，因为你不需要取出版本库一个庞大的镜像，事实上，这个技术不需要你有工作拷贝。

图 4.3. 拷贝后的版本库



[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。

> 代价低廉的拷贝
>
> Subversion的版本库有特殊的设计，当你复制一个目录，你不需要担心版本库会变得十分巨大—Subversion并不是拷贝所有的数据，相反，它建立了一个已存在目录树的入口，如果你是Unix用户，可以把它理解成硬链接，在这里，这个拷贝被可以被认为是“懒的”，如果你提交一个文件的修改，只有这个文件改变了—余下的文件还是作为原来文件的链接存在。
>
> 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）。
>
> 当然，拷贝与分享的内部机制对用户来讲是不可见的，用户只是看到拷贝树，这里的要点是拷贝的时间与空间代价很小，所以你可以随意做想要的分支。

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.
```

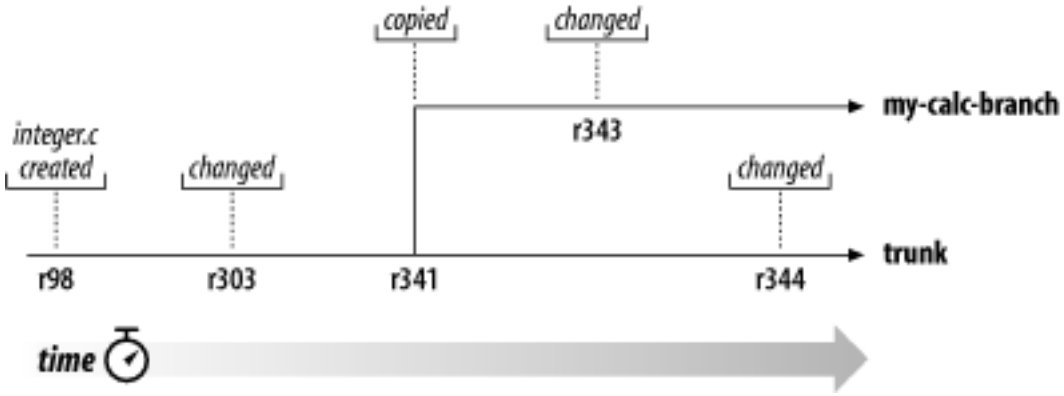
这一份工作拷贝没有什么特别的，它只是版本库另一个目录的一个镜像罢了，当你提交修改时，Sally在更新时不会看到改变，她是/calc/trunk的工作拷贝。（确定要读本章后面的第 4.5 节 “转换工作拷贝”，svn switch命令是建立分支工作拷贝的另一个选择。）

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，生成版本号342。
- 你修改了/calc/branches/my-calc-branch/integer.c，生成版本号343。
- Sally修改了/calc/trunk/integer.c，生成了版本号344。

现在有两个独立开发线，图 4.4 “一个文件的分支历史”显示了integer.c的历史。

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 --verbose 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:
   M /calc/trunk/integer.c
```

```
* integer.c:  adding this file to the project.

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

注意，Subversion追踪分支上的integer.c的历史，包括所有的操作，甚至追踪到拷贝之前。这表示了建立分支也是历史中的一次事件，因为在拷贝整个/calc/trunk/时已经拷贝了一份integer.c。现在看Sally在她的工作拷贝运行同样的命令：

```
$ pwd
/home/sally/calc

$ svn log --verbose 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:
   M /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

在这个章节你需要记住两个重要的经验。

1. 不像其他版本控制系统，Subversion的分支存在于真实的正常文件系统中，并不是存在于另外的维度，这些目录只是恰巧保留了额外的历史信息。

2. Subversion并没有内在的分支概念—只有拷贝，当你拷贝一个目录，这个结果目录就是一个“分支”，只是因为你给了它这样一个含义而已。你可以换一种角度考虑，或者特别处理，但是对于Subversion它只是一个普通的拷贝的结果。

4.3. 在分支间拷贝修改

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。

4.3.1. 拷贝特定的修改

在上一章节，我们提到你和Sally对integer.c在不同的分支上做过修改，如果你看了Sally的344版本的日志信息，你会知道她修正了一些拼写错误，毋庸置疑，你的拷贝的文件也一定存在这些拼写错误，所以你以后的对这个文件修改也会保留这些拼写错误，所以你会在将来合并时得到许多冲突。最好是现在接收Sally的修改，而不是作了许多工作之后才来做。

是时间使用svn merge命令，这个命令的结果非常类似svn diff命令（在第3章的内容），两个命令都可以比较版本库中的任何两个对象并且描述其区别，举个例子，你可以使用svn diff来查看Sally在版本344作的修改：

```
$ svn diff -r 343:344 http://svn.example.com/repos/calc/trunk

Index: integer.c
===================================================================
--- integer.c (revision 343)
+++ integer.c (revision 344)
@@ -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, "CPM"); break;
+    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;
@@ -164,7 +164,7 @@
     low = (unsigned short) read_byte(gzfile);  /* read LSB */
     high = (unsigned short) read_byte(gzfile); /* read MSB */
     high = high << 8;  /* interpret MSB correctly */
-    total = low + high; /* add them togethe for correct total */
+    total = low + high; /* add them together for correct total */

     info->extra_header = (unsigned char *) my_malloc(total);
     fread(info->extra_header, total, 1, gzfile);
@@ -241,7 +241,7 @@
      Store the offset with ftell() ! */

   if ((info->data_offset = ftell(gzfile))== -1) {
-    printf("error: ftell() retturned -1.\n");
+    printf("error: ftell() returned -1.\n");
     exit(1);
   }

@@ -249,7 +249,7 @@
   printf("I believe start of compressed data is %u\n", info->data_offset);
   #endif

-  /* Set postion eight bytes from the end of the file. */
+  /* Set position eight bytes from the end of the file. */

   if (fseek(gzfile, -8, SEEK_END)) {
     printf("error: fseek() returned non-zero\n");
```

svn merge命令几乎完全相同，但不是打印区别到你的终端，它会直接作为本地修改作用到你的本地拷贝：

```
$ svn merge -r 343:344 http://svn.example.com/repos/calc/trunk
U  integer.c

$ svn status
M  integer.c
```

svn merge的输出告诉你的integer.c文件已经作了补丁（patched），现在已经保留了Sally修改—修改从主干“拷贝”到你的私有分支的工作拷贝，现在作为一个

本地修改，在这种情况下，要靠你审查本地的修改来确定它们工作正常。

在另一种情境下，事情并不会运行得这样正常，也许integer.c也许会进入冲突状态，你必须使用标准过程（见第三章）来解决这种状态，或者你认为合并是一个错误的决定，你只需要运行svn revert放弃。

但是当你审查过你的合并结果后，你可以使用svn commit提交修改，在那一刻，修改已经合并到你的分支上了，在版本控制术语中，这种在分支之间拷贝修改的行为叫做搬运修改。

当你提交你的修改时，确定你的日志信息中说明你是从某一版本搬运了修改，举个例子：

```
$ svn commit -m "integer.c: ported r344 (spelling fixes) from trunk."
Sending        integer.c
Transmitting file data .
Committed revision 360.
```

你将会在下一节看到，这是一条非常重要的“最佳实践”。

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？举个例子：

```
$ svn diff -r 343:344 http://svn.example.com/repos/calc/trunk > patchfile
$ 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。[2] 但是svn merge命令会通过直接作用你的工作拷贝来表示目录树的修改。

一个警告：为什么svn diff和svn merge在概念上是很接近，但语法上有许多不同

[2]在将来，Subversion项目将会计划（或者发明）一种扩展补丁格式来描述目录树改变。

，一定阅读第9章来查看其细节或者使用svn help查看帮助。举个例子，svn merge需要一个工作拷贝作为目标，就是一个地方来施展目录树修改，如果一个目标都没有指定，它会假定你要做以下某个普通的操作：

1. 你希望合并目录修改到工作拷贝的当前目录。

2. 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。

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。

如果你希望修改应用到别的目录，你需要说出来。举个例子，你在工作拷贝的父目录，你需要指定目标目录：

```
$ svn merge -r 343:344 http://svn.example.com/repos/calc/trunk my-calc-branch
U   my-calc-branch/integer.c
```

4.3.2. 合并背后的关键概念

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。

迷惑的主要原因是这个命令的名称，术语“合并”不知什么原因被用来表明分支的组合，或者是其他什么神奇的数据混合，这不是事实，一个更好的名称应该是svn diff-and-apply，这是发生的所有事件：首先两个版本库树比较，然后将区别应用到本地拷贝。

这个命令包括三个参数：

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）。

一旦这三个参数指定以后，两个目录树将要做比较，比较结果将会作为本地修改应用到目标工作拷贝，当命令结束后，结果同你手工修改或者是使用svn add或svn delete没有什么区别，如果你喜欢这结果，你可以提交，如果不喜欢，你可以使用svn revert恢复修改。

svn merge的语法允许非常灵活的指定参数，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
```

第一种语法使用URL@REV的形式直接列出了所有参数，第二种语法可以用来作为比较同一个URL的不同版本的简略写法，最后一种语法表示工作拷贝是可选的，如果省略，默认是当前目录。

4.3.3. 合并的最佳实践

4.3.3.1. 手工追踪合并

合并修改听起来很简单，但是实践起来会是很头痛的事，如果你重复合并两个分支，你也许会合并两次同样的修改。当这种事情发生时，有时候事情会依然正常，当对文件打补丁时，Subversion如果注意到这个文件已经有了相应的修改，而不会作任何操作，但是如果已经应用的修改又被修改了，你会得到冲突。

理想情况下，你的版本控制系统应该会阻止对一个分支做两次改变操作，必须自动的记住那一个分支的修改已经接收了，并且可以显示出来，用来尽可能帮助自动化的合并。

不幸的是，Subversion不是这样一个系统，类似于CVS，Subversion并不记录任何合并操作，当你提交本地修改，版本库并不能判断出你是通过svn merge还是手工修改得到这些文件。

这对你这样的用户意味着什么？这意味着除非Subversion以后发展这个特性，你必须手工的记录这些信息。最佳的方式是使用提交日志信息，像前面的例子提到的，推荐你在日志信息中说明合并的特定版本号（或是版本号的范围），之后，你可以运行svn log来查看你的分支包含哪些修改。这可以帮助你小心的依序运行svn merge命令而不会进行多余的合并。

在下一小节，我们要展示一些这种技巧的例子。

4.3.3.2. 预览合并

因为合并只是导致本地修改，它不是一个高风险的操作，如果你在第一次操作错误，你可以运行svn revert来再试一次。

有时候你的工作拷贝很可能已经改变了，合并会针对存在的那一个文件，这时运行svn revert不会恢复你在本地作的修改，两部分的修改无法识别出来。

在这个情况下，人们很乐意能够在合并之前预测一下，一个简单的方法是使用运行svn merge同样的参数运行svn diff，另一种方式是传递--dry-run选项给merge命令：

```
$ svn merge --dry-run -r 343:344 http://svn.example.com/repos/calc/trunk
U  integer.c

$ svn status
#  nothing printed, working copy is still unchanged.
```

--dry-run选项实际上并不修改本地拷贝，它只是显示实际合并时的状态信息，对于得到“整体”的印象，这个命令很有用，因为svn diff包括太多细节。

> Subversion与修改集
>
> 每一个人对于“修改集”的概念都有些不一样，至少对于版本控制系统的“修改集特性”这一概念有着不同的期望，根据我们的用途，可以说修改集只是一个有唯一名字的一系列修改集合，修改也许包括文件内容的修改，目录树结构的修改，或是元数据的调整，更通常的说法，一个修改集就是我们可以引用的有名字的补丁。
>
> 在Subversion里，一个全局的修订版本号N标示一个版本库中的树：它代表版本库在N次提交后的样子，它也是一个修改集的隐含名称：如果你比较树N与树N-1，你可以得到你提交的补丁。出于这个原因，想象“版本N”并不只是一棵树，也是一个修改集。如果你使用一个问题追踪工具来管理bug，你可以使用版本号来表示特定的补丁修正了bug—举个例子，“这个问题是在版本9238修正的”，然后其他人可以运行svn log -r9238来查看修正这个bug的修改集，或者使用svn diff -r9237:9238来看补丁本身。Subversion合并命令也使用版本号作为参数，可以将特定修改集从一个分支合到另一个分支：svn merge -r9237:9238将会合并修改集#9238到本地拷贝。

4.3.3.3. 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。

作为开始，我们假定本地没有修改，当你svn update到一个特定修订版本时，修改会“干净的”应用到工作拷贝，服务器产生比较两树的增量数据：一个工作拷贝和你关注的版本树的虚拟快照，因为比较的左边同你拥有的完全相同，增量数据确保你把工作拷贝转化到右边的树。

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
C  glorb.h

$
```

在前一个例子中，baz.c也许会存在于比较的两个分支快照里，但工作拷贝里不存在，比较的增量数据要应用到这个文件，这种情况下会发生什么？“skipped”信息意味着用户可能是在比较错误的两棵树，这是经典的驱动器错误，当发生这种情况，可以使用迭代恢复（svn revert --recursive）合并所作的修改，删除恢复后留下的所有未版本化的文件和目录，并且使用另外的参数运行svn merge。

也应当注意前一个例子显示glorb.h发生了冲突，我们已经规定本地拷贝没有修改：冲突怎么会发生呢？因为用户可以使用svn merge将过去的任何变化应用到当前工作拷贝，变化包含的文本修改也许并不能干净的应用到工作拷贝文件，即使这些文件没有本地修改。

另一个svn update和svn merge的小区别是冲突产生的文件的名字不同，在第 3.5.4 节 “解决冲突（合并别人的修改）”，我们看到过更新产生的文件名字为filename.mine、filename.rOLDREV和filename.rNEWREV，当svn merge产生冲突时，它产生的三个文件分别为 filename.working、filename.left 和filename.right。在这种情况下，术语“left”和“right”表示了两棵树比较时的两边，在两种情况下，不同的名字会帮助你区分冲突是因为更新造成的还是合并造成的。

4.3.3.4. 关注还是忽视祖先

当与Subversion开发者交谈时你一定会听到提及术语祖先，这个词是用来描述两个对象的关系：如果他们互相关联，一个对象就是另一个的祖先，或者相反。

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，区别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，你会看到A foo.c后面紧跟D 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，你也许会希望合并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卖主软件项目的不同版本（见第 7.5 节 “卖主分支”），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，你只是希望svn merge能够做一个以路径为基础的比较，忽略所有文件和目录的关系，增加--ignore-ancestry选项会导致命令象svn diff一样。（相应的，--notice-ancestry选项会使svn diff象合并命令一样行事。）

4.4. 常见用例

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。

4.4.1. 合并一条分支到另一支

为了完成这个例子，我们将时间往前推进，假定已经过了几天，在主干和你的分支上都有许多更改，假定你完成了分支上的工作，已经完成了特性或bug修正，你想合并所有分支的修改到主干上，让别人也可以使用。

这种情况下如何使用svn merge？记住这个命令比较两个目录树，然后应用比较结果到工作拷贝，所以要接受这种变化，你需要主干的工作拷贝，我们假设你有一个最初的主干工作拷贝（完全更新），或者是你最近取出了/calc/trunk的一个干净的工作拷贝。

但是要哪两个树进行比较呢？乍一看，回答很明确，只要比较最新的主干与分支。但是你要意识到—这个想法是错误的，伤害了许多新用户！因为svn merge的操作很像svn diff，比较最新的主干和分支树不仅仅会描述你在分支上所作的修改，这样的比较会展示太多的不同，不仅包括分支上的增加，也包括了主干上的删除操作，而这些删除根本就没有在分支上发生过。

为了表示你的分支上的修改，你只需要比较分支的初始状态与最终状态，在你的分支上使用svn log命令，你可以看到你的分支在341版本建立，你的分支最终的状态用HEAD版本表示，这意味着你希望能够比较版本341和HEAD的分支目录，然后应用这些分支的修改到主干目录的工作拷贝。

提示

查找分支产生的版本（分支的“基准”）的最好方法是在svn log中使用--stop-on-copy选项，log子命令通常会显示所有关于分支的变化，包括 创建分支的过程，就好像你在主干上一样，--stop-on-copy会在svn log检测到目标拷贝或者改名时中止日志输出。

所以，在我们的例子里，

```
$ svn log --verbose --stop-on-copy \
          http://svn.example.com/repos/calc/branches/my-calc-branch
...
```

```
------------------------------------------------------------------------
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)

$
```

正如所料，最后的打印出的版本正是my-calc-branch生成的版本。

如下是最终的合并过程，然后：

```
$ cd calc/trunk
$ svn update
At revision 405.

$ svn merge -r 341:405 http://svn.example.com/repos/calc/branches/my-calc-branch
U   integer.c
U   button.c
U   Makefile

$ svn status
M   integer.c
M   button.c
M   Makefile

# ...examine the diffs, compile, test, etc...

$ svn commit -m "Merged my-calc-branch changes r341:405 into the trunk."
Sending        integer.c
Sending        button.c
Sending        Makefile
Transmitting file data ...
Committed revision 406.
```

再次说明，日志信息中详细描述了合并到主干的的修改范围，记住一定要这么做，这是你以后需要的重要信息。

举个例子，你希望在分支上继续工作一周，来进一步加强你的修正，这时版本库的HEAD版本是480，你准备好了另一次合并，但是我们在第 4.3.3 节 “合并的最佳实践”提到过，你不想合并已经合并的内容，你只想合并新的东西，技巧就是指出什么是“新”的。

第一步是在主干上运行svn log察看最后一次与分支合并的日志信息：

```
$ cd calc/trunk
$ svn log
…
------------------------------------------------------------------------
r406 | user | 2004-02-08 11:17:26 -0600 (Sun, 08 Feb 2004) | 1 line

Merged my-calc-branch changes r341:405 into the trunk.
------------------------------------------------------------------------
…
```

阿哈！因为分支上341到405之间的所有修改已经在版本406合并了，现在你只需要合并分支在此之后的修改—通过比较406和HEAD。

```
$ cd calc/trunk
$ svn update
At revision 480.

# We notice that HEAD is currently 480, so we use it to do the merge:

$ svn merge -r 406:480 http://svn.example.com/repos/calc/branches/my-calc-branch
U   integer.c
U   button.c
U   Makefile

$ svn commit -m "Merged my-calc-branch changes r406:480 into the trunk."
Sending        integer.c
Sending        button.c
Sending        Makefile
Transmitting file data ...
Committed revision 481.
```

现在主干有了分支上第二波修改的完全结果，此刻，你可以删除你的分支（我们会在以后讨论），或是继续在你分支上工作，重复这个步骤。

4.4.2. 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：

```
$ svn merge -r 303:302 http://svn.example.com/repos/calc/trunk
U  integer.c
```

```
$ svn status
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.
```

我们可以把版本库修订版本想象成一组修改（一些版本控制系统叫做修改集），通过-r选项，你可以告诉svn merge来应用修改集或是一个修改集范围到你的工作拷贝，在我们的情况例子里，我们使用svn merge合并修改集#303到工作拷贝。

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。

继续，你也许会想：好吧，这不是真的取消提交吧！是吧？版本303还依然存在着修改，如果任何人取出calc的303-349版本，他还会得到错误的修改，对吧？

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，每个修订版本都是不可变的目录树 ，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。[3]

4.4.3. 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？”

第一步首先要知道需要拯救的项目是什么，这里有个很有用的比喻：你可以认为任何存在于版本库的对象生活在一个二维的坐标系统里，第一维是一个特定的版本树，第二维是在树中的路径，所以你的文件或目录的任何版本可以有这样一对坐标定义。

Subversion没有向CVS一样的古典目录，[4] 所以你需要svn log来察看你需要找回

[3] Subversion项目有计划，不管用什么方式，总有一天要实现svnadmin obliterate命令来进行永久删除操作，而此时可以看第 5.3.1.3 节 “svndumpfilter”。

[4] 因为CVS没有版本树，它会在每个版本库目录创建一个古典区域用来保存增量数据。

的坐标对，一个好的策略是使用svn log --verbose来察看你删除的项目，--verbose选项显示所有改变的项目的每一个版本 ，你只需要找出你删除文件或目录的那一个版本。你可以通过目测找出这个版本，也可以使用另一种工具来检查日志的输出 （通过grep或是在编辑器里增量查找）。

```
$ cd parent-dir
$ svn log --verbose
…
------------------------------------------------------------------------
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。

一种是对版本反向使用svn merge到808（我们已经学会了如何取消修改，见第 4.4.2 节 “取消修改”），这样会重新添加real.c，这个文件会列入增加的计划，经过一次提交，这个文件重新回到HEAD。

在这个例子里，这不是一个好的策略，这样做不仅把real.c加入添加到计划，也取消了对integer.c的修改，而这不是你期望的。确实，你可以恢复到版本808，然后对integer.c执行取消svn revert操作，但这样的操作无法扩大使用，因为如果从版本808修改了90个文件怎么办？

所以第二个方法不是使用svn merge，而是使用svn copy命令，精确的拷贝版本和路径“坐标对”到你的工作拷贝：

```
$ svn copy --revision 807 \
           http://svn.example.com/repos/calc/trunk/real.c ./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。

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。

4.4.4. 常用分支模式

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。

4.4.4.1. 发布分支

大多数软件存在这样一个生命周期：编码、测试、发布，然后重复。这样有两个问题，第一，开发者需要在质量保证小组测试假定稳定版本时继续开发新特性，新工作在软件测试时不可以中断，第二，小组必须一直支持老的发布版本和软件；如果一个bug在最新的代码中发现，它一定也存在已发布的版本中，客户希望立刻得到错误修正而不必等到新版本发布。

这是版本控制可以做的帮助，典型的过程如下：

- 开发者提交所有的新特性到主干。 每日的修改提交到/trunk：新特性，bug修正和其他。
- 这个主干被拷贝到“发布”分支。 当小组认为软件已经做好发布的准备（如，版本1.0）然后/trunk会被拷贝到/branches/1.0。
- 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。
- 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。
- 分支多次维护。当继续在/trunk上为版本2.0工作，bug修正继续从/trunk运送到/branches/1.0，如果积累了足够的bug修正，管理部门决定发布1.0.1版本：拷贝/branches/1.0到/tags/1.0.1，标签被打包发布。

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。

4.4.4.2. 特性分支

一个特性分支是本章中那个重要例子中的分支，你正在那个分支上工作，而Sally还在/trunk继续工作，这是一个临时分支，用来作复杂的修改而不会干扰/trunk的稳定性，不象发布分支（也许要永远支持），特性分支出生，使用了一段时间，合并到主干，然后最终被删除掉，它们在有限的时间里有用。

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。

许多项目采用折中的方式，坚持每次编译/trunk并进行回归测试，只有需要多次不稳定提交时才需要一个特性分支，这个规则可以用这样一个问题检验：如果开发者在好几天里独立工作，一次提交大量修改（这样/trunk就不会不稳定。），是否会有太多的修改要来回顾？如果答案是“是”，这些修改应该在特性分支上进行，因为开发者增量的提交修改，你可以容易的回头检查。

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。

这种情况最好通过有规律的将主干合并到分支来避免，制定这样一个政策：每周将上周的修改合并到分支，注意这样做时需要小心，需要手工记录合并的过程，以避免重复的合并（在第 4.3.3.1 节 “手工追踪合并”描述过），你需要小心的撰写合并的日志信息，精确的描述合并包括的范围（在第 4.4.1 节 “合并一条分支到另一支”中描述过），这看起来像是胁迫，可是实际上是容易做到的。

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，所以在这个特别的例子里，你会通过直接比较分支和主干来进行合并：

```
$ cd trunk-working-copy

$ svn update
At revision 1910.

$ svn merge http://svn.example.com/repos/calc/trunk@1910 \
            http://svn.example.com/repos/calc/branches/mybranch@1910
U  real.c
```

```
U  integer.c
A  newdirectory
A  newdirectory/newfile
…
```

通过比较HEAD修订版本的主干和HEAD修订版本的分支，你确定了只在分支上的增量信息，两条开发线都有了分枝的修改。

可以用另一种考虑这种模式，你每周按时同步分支到主干，类似于在工作拷贝执行svn update的命令，最终的合并操作类似于在工作拷贝运行svn commit，毕竟，工作拷贝不就是一个非常浅的分支吗？只是它一次只可以保存一个修改。

4.5. 转换工作拷贝

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
```

完成了到分支的“跳转”，你的目录与直接取出一个干净的版本没有什么不同。这样会更有效率，因为分支只有很小的区别，服务器只是发送修改的部分来使你的工作拷贝反映分支。

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最新版本。

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。

换句话说，如果一个用户知道分支工作只发生在部分子目录，我们使用svn switch来跳转部分目录（有时候只是单个文件），这样的话，他们依然可以继续得到普通的“trunk”主干的更新，但是已经跳转的部分则被免去了更新（除非分支上有更新）。这个特性给“混合工作拷贝”概念添加了新的维度—不仅工作拷贝的版本可以混合，在版本库中的位置也可以混合。

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。

注意，因为你的工作拷贝可以在混合位置的情况下工作正常，但是所有的位置必须在同一个版本库，Subversion的版本库不能互相通信，这个特性还不在Subversion 1.0的计划里。[5]

> 跳转和更新
>
> 你注意到svn switch和svn update的输出很像？switch命令只是update命令的一个超集。
>
> 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svn update会一直比较同一路径。
>
> 也就是了，如果你的工作拷贝是/calc/trunk的一个镜像，当运行svn update时会自动地比较你的工作拷贝的/calc/trunk与HEAD版本的/calc/trunk。如果你使用svn switch跳转工作拷贝到分支，则会比较你的工作拷贝的/calc/trunk与相应分支目录的HEAD版本。
>
> 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，这里允许你作各种聪明的把戏。

举个例子，你的工作拷贝目录是/calc/trunk，你已经做了很多修改，然后你突然发现应该在分支上修改更好，没问题！你可以使用svn switch，而你本地修改还会保留，你可以测试并提交它们到分支。

[5]当你的服务器位置改变，而你不想放弃存在的本地拷贝，你可以使用带选项--relocate的svn switch命令转换URL，见第 9 章 Subversion完全参考的svn switch查看更多信息和例子。

4.6. 标签

另一个常见的版本控制系统概念是标签（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. 建立最简单的标签

svn copy再次登场，你希望建立一个/calc/trunk的一个快照，就像HEAD修订版本，建立这样一个拷贝：

```
$ 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 351.
```

这个例子假定/calc/tags目录已经存在（如果不是，见svn mkdir），拷贝完成之后，一个表示当时HEAD版本的/calc/trunk目录的镜像已经永久的拷贝到release-1.0目录。当然，你会希望更精确一点，以防其他人在你不注意的时候提交修改，所以，如果你知道/calc/trunk的版本350是你想要的快照，你可以使用svn copy加参数 -r 350。

但是等一下：标签的产生过程与建立分支是一样的？是的，实际上在Subversion中标签与分支没有区别，都是普通的目录，通过copy命令得到，与分支一样，一个目录之所以是标签只是人们决定这样使用它，只要没有人提交这个目录，它永远是一个快照，但如果人们开始提交，它就变成了分支。

如果你管理一个版本库，你有两种方式管理标签，第一种方法是禁止命令：作为项目的政策，我们要决定标签所在的位置，确定所有用户知道如何处理拷贝的目录（也就是确保他们不会提交他们），第二种方法看来很过分：使用访问控制脚本来阻止任何想对标签目录做的非拷贝的操作（见第 6 章 配置服务器）这种方法通常是不必要的，如果一个人不小心提交了到标签目录一个修改，你可以简单的取消，毕竟这是版本控制啊。

4.6.2. 建立复杂的标签

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。

举个例子，假定你的项目比我们的的例子calc大的多：假设它保存了一组子目录和许多文件，在你工作时，你或许决定创建一个包括特定特性和Bug修正的工作拷

贝，你可以通过选择性的回溯文件和目录到特定修订版本（使用svn update -r）来实现，或者转换文件和目录到特定分支（使用svn switch），这样做之后，你的工作拷贝成为版本库不同版本和分支的司令部，但是经过测试，你会知道这是你需要的一种精确数据组合。

是时候进行快照了，拷贝URL在这里不能工作，在这个例子里，你希望把本地拷贝的布局做镜像并且保存到版本库中，幸运的是，svn copy包括四种不同的使用方式（在第9章可以详细阅读），包括拷贝工作拷贝到版本库：

```
$ ls
my-working-copy/

$ svn copy my-working-copy http://svn.example.com/repos/calc/tags/mytag

Committed revision 352.
```

现在在版本库有一个新的目录/calc/tags/mytag，这是你的本地拷贝的一个快照—混合了修订版本，URL等等。

一些人也发现这一特性一些有趣的使用方式，有些时候本地拷贝有一组本地修改，你希望你的协作者看到这些，不使用svn diff并发送一个不定文件（不会捕捉到目录修改），而是使用svn copy来“上传”你的工作拷贝到一个版本库的私有区域，你的协作者可以选择完整的取出你的工作拷贝，或使用svn merge来接受你的精确修改。

4.7. 分支维护

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

有一些标准的，推荐的组织版本库的方式，许多人创建一个trunk目录来保存开发的“主线”，一个branches目录存放分支拷贝，一个目录保存标签拷贝，如果一个版本库只是存放一个项目，人们会在顶级目录创建这些目录：

```
/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 5.4.1 节 “选择一种版本库布局”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.
```

你的分支已经消失了，当然不是真的消失了：这个目录只是在HEAD修订版本里消失了，如果你使用svn checkout、svn switch或者svn list来检查一个旧的版本，你仍会见到这个旧的分支。

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 -r来从旧的版本拷贝出来：

```
$ svn copy -r 374 http://svn.example.com/repos/calc/branches/my-calc-branch \
                  http://svn.example.com/repos/calc/branches/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“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。

4.8. 摘要

我们已经在本章覆盖了许多基础知识，我们讨论了标签和分支的概念，然后描述了Subversion怎样用svn copy命令拷贝目录实现了这些概念，我们也已经展示了怎样使用svn merge命令来在分支之间拷贝修改，或是撤销错误的修改。我们仔细研究了使用svn switch来创建混合位置的工作拷贝，然后我们也讨论了怎样管理和组织版本库中分支的生命周期。

记住Subversion的曼特罗（mantra）：分支和标签是廉价的，自由的使用它们吧！

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（它们都包含在Subversion中）。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。

如果您只是以普通用户的身份访问版本库对数据进行版本控制（就是说通过Subversion客户端），您完全可以跳过本章。但是如果您已经是或打算成为Subversion版本库的管理员，[1]您一定要关注一下本章的内容。

5.1. 版本库基本知识

在进入版本库管理这块宽广的主题之前，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，你应该既从逻辑视角－数据在版本库中如何展示，又能从物理具体细节的视角－版本库如何响应一个非Subversion的工具，来理解版本库的组成。下面的小节从一个比较高的层面覆盖了这些基本概念。

5.1.1. 理解事务和修订版本

从概念上来说，Subversion的版本库就是一串目录树。每一个目录树，就是版本库的文件和目录在某一时刻的快照。这些快照是客户端使用者操作的结果，叫做修订版本。

每一个修订版本都是以事务树开始其生命周期。做提交操作时，客户端建立了一个映射本地修改的Subversion事务（加上客户端提交操作后任何对版本库的更改），然后指导版本库将该树存储为下一个快照。要是提交成功，这个事务就会成为新的修订版本树，并被赋予新的修订版本号。如果因为某些原因提交失败，事务会被销毁，客户端将被通知这个事务失败。

更新的动作也类似这样。客户端建立一个临时的事务树，映射工作文件的状态。然后版本库比较事务树和被请求的修订版本树（通常是最新的，也就是最“年轻”的修订版本树），然后发回消息通知客户端哪些变更需要将拷贝发送到修订版本树。更新完成后，临时事务将被删除。

事务树的使用是对版本库中版本控制文件系统产生永久变更的唯一方法。一个事务的生命周期非常灵活，了解这一点很重要。在更新的情况下，事务只是马上会被销毁的临时树。在提交的情况下，事务会变成固定的修订版本（如果失败的情况下，则会被删除）。在出现错误或bug的情况下，事务可能会被留在版本库中（

[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。

不会影响任何东西，但是会占据空间）。

理论上，有一天整个流程能够发展到对事务进行更加细密的流程控制。可以想象一个系统，在客户端完成操作，将要保存到版本库中时，每个加到它的事务都变成一个修订版本。这将会使每一个新的提交都可以被别人查看到，也许是主管，也许是质量保证小组，他们可以决定是要接收这个事务成为修订版本，还是放弃它。

5.1.2. 未受版本控制的属性

事务和修订版本在Subversion版本库中可以附加属性。这些属性就是普通的属性名和属性值的映射，被用来存储与对应目录树有关的信息。这些属性名和属性值跟你的其他数据一样，被存储在版本库文件系统中。

修订版本和事务的属性对于存储一个跟目录树相关，但与树中的某个具体目录或文件不相关的性质很有用－即并不被客户端工作拷贝所管理的属性。举例来说，当一个新的提交事务在版本库中被创建时，Subversion给这个事务添加一个叫做svn:date的属性—一个表示事务何时被创建的时间戳。当提交进程结束，该事务成为一个固定的修订版本，这个目录树被赋予一个用来存储这个版本作者名称的属性（svn:author）和一个用来存储与这个修订版本关联日志信息的属性（svn:log）。

修订版本和事务的属性都是未受版本控制的－因为当它们被修改时，先前的值就被完全舍弃了。修订版本树自身是不能变更的，与之关联的属性可以修改。你可在日后添加、删除、修改修订版本的属性。如果你提交一个新的修订版本之后意识到遗漏了一些信息或在日志中的拼写错误，你可以直接以正确的信息覆盖svn:log它的值。

5.1.3. 版本库数据存储

在Subversion1.1中，版本库中存储数据有两种方式。一种是在Berkeley DB数据库中存储数据；另一种是使用普通的文件，使用自定义格式。因为Subversion的开发者称版本库为[版本化的]文件系统，他们接受了称后一种存储方式为FSFS的习惯，也就是说，使用本地操作系统文件系统来存储数据的版本化文件系统。

建立一个版本库时，管理员必须决定使用Berkeley DB还是FSFS。它们各有优缺点，我们将详细描述。这两个中并没有一个是更正式的，访问版本库的程序与采用哪一种实现方式无关。访问程序并不知道版本库如何存储数据，它们只是从版本库的API读取到修订版本和事务树。

下面的表从总体上比较了Berkeley DB和FSFS版本库，下一部分将会详细讲述细节。

表 5.1. 版本库数据存储对照表

特性	Berkeley DB	FSFS
对操作中断的敏感	很敏感；系统崩溃或者权	不敏感。

特性	Berkeley DB	FSFS
	限问题会导致数据库“塞住”，需要定期进行恢复。	
可只读加载	不能	可以
存储平台无关	不能	可以
可从网络文件系统访问	不能	可以
版本库大小	稍大	稍小
可扩展性：修订版本树的数量	数据库，没有限制	许多古老的本地文件系统在处理单一目录包含上千个条目时出现问题。
可扩展性：文件较多的目录	较慢	较快
速度：检出最新的代码	较快	较慢
速度： 大的提交	较慢，但是时间被分配在整个提交操作中	较快，但是最后较长的延时可能会导致客户端操作超时
组访问权处理	对于用户的umask设置十分敏感，最好只由一个用户访问。	对umask设置不敏感
功能成熟时间	2001年开始使用	2004年开始使用

5.1.3.1. Berkeley DB

在Subversion的初始设计阶段，开发者因为多种原因而决定采用Berkeley DB，比如它的开源协议、事务支持、可靠性、性能、简单的API、线程安全、支持游标等。

Berkeley DB提供了真正的事务支持－这或许是它最强大的特性，访问你的Subversion版本库的多个进程不必担心偶尔会破坏其他进程的数据。事务系统提供的隔离对于任何给定的操作，Subversion版本库代码看到的只是数据库的静态视图－而不是一个在其他进程影响不断变化的数据库－并能够根据该视图作出决定。如果该决定正好同其他进程所做操作冲突，整个操作会回滚，就像什么都没有发生一样，并且Subversion会优雅的再次对更新的静态视图进行操作。

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 5.3.6 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

5.3.3 节 “管理磁盘空间”。

但是每朵玫瑰都有刺，我们也必须记录一些Berkeley DB已知的缺陷。首先，Berkeley DB环境不是跨平台的。你不能简单的拷贝一个在Unix上创建的Subversion版本库到一个Windows系统并期望它能够正常工作。尽管Berkeley DB数据库的大部分格式是不受架构约束的，但环境还是有一些方面没有独立出来。其次，使用Berkeley DB的Subversion不能在95/98系统上运行—如果你需要将版本库建在一个Windows机器上，请装到Windows2000或WindowsXP上。另外，Berkeley DB版本库不能放在网络共享文件夹中，尽管Berkeley DB承诺如果按照一套特定规范的话，可以在网络共享上正常运行，但实际上已知的共享类型几乎都不满足这套规范。

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。因为Berkeley DB版本库非常快，并且可以扩展，非常适合使用一个单独的服务进程，通过一个用户来访问—比如Apache的httpd或svnserve（参见第 6 章 配置服务器）—而不是多用户通过file:///或svn+ssh://URL的方式多用户访问。如果将Berkeley DB版本库直接用作多用户访问，请先阅读第 6.5 节 “支持多种版本库访问方法”。

5.1.3.2. 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。

修订版本文件格式代表了一个修订版本的目录结构，文件内容，和其它修订版本树中相关信息。不像Berkeley DB数据库，这种存储格式可跨平台并且与CPU架构无关。因为没有日志或用到共享内存的文件，数据库能被网络文件系统安全的访问和在只读环境下检查。缺少数据库花消同时也意味着版本库的总体体积可以稍小一点。

FSFS也有一种不同的性能特性。当提交大量文件时，FSFS使用O(N)算法来追加条目，而Berkeley DB则用(N^2)算法来重写整个目录。另一方面，FSFS通过写入与上一个版本比较的变化来记录新版本，这也意味着获取最新修订版本时会比Berkeley DB慢一点，提交时FSFS也会有一个更长的延迟，在某些极端情况下会导致客护端在等待回应时超时。

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。

唯一真正对FSFS不利的是相对于Berkeley DB的不成熟，缺乏足够的使用和压力测试，许多关于速度和可扩展性的判断都是建立在良好的猜测之上。在理论上，它承诺会降低管理员新手的门槛并且更加不容易发生问题。在实践中，只有时间可以证明。

5.2. 版本库的创建和配置

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。要建立一个新的版本库，只需要运行：

```
$ svnadmin create /path/to/repos
```

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。

在 Subversion 1.1中，版本库默认使用Berkeley DB后端存储方式来创建。在以后的发行版中这个行为会被改变。不管怎样，存储类型可以使用--fs-type参数明确说明。:

```
$ svnadmin create --fs-type fsfs /path/to/repos
$ svnadmin create --fs-type bdb /path/to/other/repos
```

警告

不要在网络共享上创建Berkeley DB版本库—它不能存在于诸如NFS，AFS或Windows SMB的远程文件系统中，Berkeley 数据要求底层文件系统实现严格的POSIX锁定语义，几乎没有任何网络文件系统提供这些特性，假如你在网络共享上使用Berkeley DB，结果是不可预知的——许多错误可能会立刻发现，也有可能在几个月之后才能发现

假如你需要多台计算机来访问，你需要在网络共享上创建FSFS版本库，而不是Berkeley DB的版本库。或者更好的办法，你建立一个真正的服务进程（例如Apache或svnserve），把版本库放在服务器能访问到的本地文件系统中，以便能通过网络访问。详情请参看第 6 章 配置服务器

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。

所以，当你运行svnadmin create命令后，就会在运行目录创建一个崭新的Subversion版本库，让我们看一下在这个目录创建中创建了什么。

```
$ ls repos
conf/  dav/  db/  format  hooks/  locks/  README.txt
```

除了README.txt和format文件，版本库目录就是一些子目录了。就像Subversion其它部分的设计一样，模块化是一个很重要的原则，而且层次化的组织要比杂乱无章好。下面是对新的版本库目录中各个项目的简要介绍：

conf
: 一个存储版本库配置文件的目录。

dav
: 提供给Apache和mod_dav_svn的目录，让它们存储自己的数据。

db
: 你所有的受版本控制数据的所在之处。这个目录或者是个Berkeley DB环境（满是数据表和其他东西），或者是一个包含修订版本文件的FSFS环境。

format
: 包含了用来表示版本库布局版本号的整数。

hooks
: 一个存储钩子脚本模版的目录（还有钩子脚本本身， 如果你安装了的话）。

locks
: 一个存储Subversion版本库锁定数据的目录，被用来追踪对版本库的访问。

README.txt
: 这个文件只是用来告诉它的阅读者，他现在看的是 Subversion 的版本库。

一般来说，你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。不过还是会有些例外情况，我们会在这里提到。

5.2.1. 钩子脚本

所谓钩子就是与一些版本库事件触发的程序，例如新修订版本的创建，或是未版本化属性的修改。每个钩子都会被告知足够多的信息，包括那是什么事件，所操作的对象，和触发事件的用户名。通过钩子的输出或返回状态，钩子程序能让工作继续、停止或是以某种方式挂起。

默认情况下，钩子的子目录中包含各种版本库钩子模板。

```
$ ls repos/hooks/
post-commit.tmpl           pre-revprop-change.tmpl
post-revprop-change.tmpl   start-commit.tmpl
pre-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。

目前Subversion有已实现了五种钩子：

start-commit
: 它在提交事务产生前已运行，通常用来判定一个用户是否有权提交。版本库传给该程序两个参数：到版本库的路径，和要进行提交的用户名。如果程序返回一个非零值，会在事务产生前停止该提交操作。如果钩子程序要在stderr中写入数据，它将排队送至客户端。

pre-commit
: 在事务完成提交之前运行，通常这个钩子是用来保护因为内容或位置（例如，你要求所有到一个特定分支的提交必须包括一个bug追踪的ticket号，或者是要求日志信息不为空）而不允许的提交。版本库传递两个参数到程序：版本库的路径和正在提交的事务名称，如果程序返回非零值，提交会失败，事务也会删除。如果钩子程序在stderr中写入了数据，也会传递到客户端。

  Subversion的分发版本包括了一些访问控制脚本（在Subversion源文件目录树的tools/hook-scripts目录），可以用来被pre-commit调用来实现精密的写访问控制。另一个选择是使用Apache的httpd模块mod_authz_svn，可以对单个目

录进行读写访问控制（见第 6.4.4.2 节 “每目录访问控制”）。在未来的Subversion版本中,我们计划直接在文件系统中实现访问控制列表（ACLs）。

post-commit
它在事务完成后运行，创建一个新的修订版本。大多数人用这个钩子来发送关于提交的描述性电子邮件，或者作为版本库的备份。版本库传给程序两个参数：到版本库的路径和被创建的新的修订版本号。退出程序会被忽略。

Subversion分发版本中包括mailer.py和commit-email.pl脚本（存于Subversion源代码树中的tools/hook-scripts/目录中）可以用来发送描述给定提交的email（并且或只是追加到一个日志文件），这个mail包含变化的路径清单，提交的日志信息、日期和作者以及修改文件的GNU区别样式输出。

Subversion提供的另一个有用的工具是hot-backup.py脚本（在Subversion源代码树中的tools/backup/目录中）。这个脚本可以为Subversion版本库进行热备份（Berkeley DB数据库后端支持的一种特性），可以制作版本库每次提交的快照作为归档和紧急情况的备份。

pre-revprop-change
因为Subversion的修订版本属性不是版本化的，对这类属性的修改（例如提交日志属性svn:log）将会永久覆盖以前的属性值。因为数据在此可能丢失，所以Subversion提供了这种钩子（及与之对应的post-revprop-change），因此版本库管理员可用一些外部方法记录变化。作为对丢失未版本化属性数据的防范，Subversion客户端不能远程修改修订版本属性，除非为你的版本库实现这个钩子。

这个钩子在对版本库进行这种修改时才会运行，版本库给钩子传递四个参数：到版本库的路径，要修改属性的修订版本，经过认证的用户名和属性自身的名字。

post-revprop-change
我们在前面提到过，这个钩子与pre-revprop-change对应。事实上，因为多疑的原因，只有存在pre-revprop-change时这个脚本才会执行。当这两个钩子都存在时，post-revprop-change在修订版本属性被改变之后运行，通常用来发送包含新属性的email。版本库传递四个参数给该钩子：到版本库的路径，属性存在的修订版本，经过校验的产生变化的用户名，和属性自身的名字。

Subversion分发版本中包含propchange-email.pl脚本（在Subversion源代码树中的tools/hook-scripts/目录中），可以用来发送修订版本属性修改细节的email（并且或只是追加到一个日志文件）。这个email包含修订版本和发生变化的属性名，作出修改的用户和新属性值。

警告

不要尝试用钩子脚本修改事务。一个常见的例子就是在提交时自动设置svn:eol-style或svn:mime-type这类属性。这看起来是个好主意，但它会引起问题。主要的问题是客户并不知道由钩子脚本进行的修改，同时没有办法通告客户它的数据是过时的，这种矛盾会导致出人意料和不能

预测的行为。

作为尝试修改事务的替代，我们通过检查pre-commit钩子的事务，在不满足要求时拒绝提交。

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。

5.2.2. Berkeley DB配置

Berkeley DB环境是对一个或多个数据库、日志文件、区域文件和配置文件的封装。Berkeley DB环境对许多参数有自己的缺省值，例如任何时间里可用的锁定数目、日志文件的最大值等。Subversion文件系统会使用Berkeley DB的默认值。 不过，有时候你的特定版本库与它独特的数据集合和访问类型，可能需要不同的配置选项。

Sleepycat（Berkeley DB的制造厂商）的人们清楚不同的数据库有不同的需求，所以他们提供了在运行中覆盖Berkeley DB环境配置参数的机制。Berkeley在每一个环境目录中检查是否存在一个名叫DB_CONFIG的文件，然后解析其中的参数成为Berkeley环境所用的选项。

你的版本库的Berkeley配置文件位于db目录的repos/db/DB_CONFIG， Subversion在创建版本库时自己创建了这个文件。这个文件初始时包含了一些默认选项，也包含了Berkeley DB在线文档，使你能够了解这些选项是做什么的。当然，你也可以为你的DB_CONFIG 文件添加任何Berkeley DB支持的选项。需要注意到，虽然Subversion不会尝试读取并解析这个文件，或使用其中的设置，你一定要避免会导致Berkeley DB按照Subversion代码不习惯的方式工作的修改。另外，DB_CONFIG的修改在复原数据库环境（用svnadmin recover）之前不会产生任何效果。

5.3. 版本库维护

维护一个Subversion版本库是一项令人沮丧的工作，主要因为有数据库后端与生俱来的复杂性。做好这项工作需要知道一些工具——它们是什么，什么时候用以及如何使用。这一节将会向你介绍Subversion自带的版本库管理工具，以及如何使用它们来完成诸如版本库移植、升级、备份和整理之类的任务。

5.3.1. 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。

5.3.1.1. 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。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.
...
```

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 和 --transaction选项指定要查看的修订版本或事务。注意，虽然修订版本号看起来像自然数，但是事务名称是包含英文字母与数字的字符串。请记住文件系统只允许浏览未提交的事务（还没有形成一个新的修订版本的事务）。多数版本库没有这种事务，因为事务通常或者被提交了（这样便不能被查看），或者被中止并删除了。

如果没有--revision和--transaction选项，svnlook会查看版本库中最年轻的修订版本（或“HEAD”）。当版本库中的/path/to/repos的最年轻的修订版本是19时，下边的两个命令执行结果完全相同：

```
$ svnlook info /path/to/repos
$ svnlook info /path/to/repos --revision 19
```

这些子命令的唯一例外，是svnlook youngest命令，它不需要选项，只会显示出HEAD的修订版本号。

```
$ svnlook youngest /path/to/repos
19
```

svnlook的输出被设计为人和机器都易理解，拿info子命令举例来说：

```
$ svnlook info /path/to/repos
sally
2002-11-04 09:29:13 -0600 (Mon,  04 Nov 2002)
```

```
27
Added the usual
Greek tree.
```

info子命令的输出定义如下：

1. 作者，后接换行。
2. 日期，后接换行。
3. 日志消息的字数，后接换行。
4. 日志信息本身， 后接换行。

这种输出是人可阅读的，像是时间戳这种有意义的条目，使用文本表示，而不是其他比较晦涩的方式（例如许多无聊的人推荐的十亿分之一秒的数量）。这种输出也是机器可读的—因为日志信息可以有多行，没有长度的限制，svnlook在日志消息之前提供了消息的长度，这使得脚本或者其他对这个命令进行的封装提供了更强的功能，比如日志消息使用了多少内存，或在这个输出成为最后一个字节之前应该略过多少字节。

另一个svnlook常见的用法是查看修订版本树或事务树的内容。svnlook tree 命令显示在请求的树中的目录和文件。如果你提供了--show-ids选项，它还会显示每个路径的文件系统节点修订版本ID（这一点对开发者往往更有用）。

```
$ svnlook tree /path/to/repos --show-ids
/ <0.0.1>
 A/ <2.0.1>
  B/ <4.0.1>
   lambda <5.0.1>
   E/ <6.0.1>
    alpha <7.0.1>
    beta <8.0.1>
   F/ <9.0.1>
  mu <3.0.1>
  C/ <a.0.1>
  D/ <b.0.1>
   gamma <c.0.1>
   G/ <d.0.1>
    pi <e.0.1>
    rho <f.0.1>
    tau <g.0.1>
   H/ <h.0.1>
    chi <i.0.1>
    omega <k.0.1>
```

```
    psi <j.0.1>
 iota <1.0.1>
```

如果你看过树中目录和文件的布局，你可以使用svnlook cat，svnlook propget ， 和svnlook proplist命令来查看这些目录和文件的细节。

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，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。下面是svnlook命令能接受的子命令的介绍，以及这些子命令的输出：

author
   显示该树的作者。

cat
   显示树中某文件的内容。

changed
   显示树中修改过的所有文件和目录。

date
   显示该树的时间戳。

diff
   使用统一区别格式显示被修改的文件。

dirs-changed
   显示树中本身被修改或者其中文件被修改的目录。

history
   显示受到版本控制的路径（更改和复制发生过的地方）中重要的历史点。

info
   显示树的作者、时间戳、日志大小和日志信息。

log
   显示树的日志信息。

propget
   显示树中路径的属性值。

proplist
   显示树中属性集合的名字与值。

tree
   显示树列表，可选的显示与路径有关的文件系统节点的修订版本号。

uuid

显示版本库的UUID—全局唯一标示。

youngest
显示最年轻的修订版本号。

5.3.1.2. 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的语法跟 svnlook类似：

```
$ svnadmin help
general usage: svnadmin SUBCOMMAND REPOS_PATH  [ARGS & OPTIONS ...]
Type "svnadmin help <subcommand>" for help on a specific subcommand.

Available subcommands:
   create
   deltify
   dump
   help (?,  h)
...
```

我们已经提过svnadmin的create子命令（参照第 5.2 节 “版本库的创建和配置”）。本章中我们会详细讲解大多数其他的命令。现在，我们来简单的看一下每个可用的子命令提供了什么功能。

create
创建一个新的Subversion版本库。

deltify
在指定的修订版本范围内，对其中修改过的路径做增量化操作。如果没有指定修订版本，这条命令会修改HEAD修订版本。

dump
导出版本库修订一定版本范围内的内容，使用可移植转储格式。

hotcopy
对版本库做热拷贝，用这个方法你能任何时候安全的备份版本库而无需考虑是否正在使用。

list-dblogs
（Berkeley DB版本库专有）列出Berkeley DB中与版本库有关的日志文件清单。这个清单包括所有的日志文件—仍然被版本库使用的和不再使用的。

list-unused-dblogs
（Berkeley DB版本库专有)列出Berkeley DB版本库有关的不在使用日志文件路径清单。你能安全的从版本库中删除那些日志文件，也可以将它们存档以用

来在灾难事件后版本库的恢复。

load
导入由dump子命令导出的可移植转储格式的一组修订版本。

lstxns
列出刚刚在版本库的没有提交的Subversion事务清单。

recover
恢复版本库，通常在版本库发生了致命错误的时候，例如阻碍进程干净的关闭同版本库的连接的错误。

rmtxns
从版本库中清除Subversion事务（通过加工lstxns子命令的输出即可）。

setlog
替换给定修订版本的svn:log（提交日志信息）属性值。

verify
验证版本库的内容，包括校验比较本地版本化数据和版本库。

5.3.1.3. svndumpfilter

因为Subversion使用底层的数据库储存各类数据，手工调整是不明智的，即使这样做并不困难。何况，一旦你的数据存进了版本库，通常很难再将它们从版本库中删除。[2]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。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。

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 5.3.5 节 “版本库的移植”）。转储文件易于阅读意味着你可以小心翼翼的查看和修改它。当然，问题是如果你有一个运行了两年的版本库，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。

虽然在管理员的日常工作中并不会经常使用，不过svndumpfilter可以对特定的路径进行过滤。这是一个独特而很有意义的用法，可以帮助你快速方便的修改转储的数据。使用时，只需提供一个你想要保留的（或者不想保留的）路径列表，然后把你的版本库转储文件送进这个过滤器。最后你就可以得到一个仅包含你想保留的路径的转储数据流。

svndumpfilter的语法如下：

[2]顺便说一句，这是Subversion的特性，而不是bug。

```
$ svndumpfilter help
general usage: svndumpfilter SUBCOMMAND [ARGS & OPTIONS ...]
Type "svndumpfilter help <subcommand>" for help on a specific subcommand.

Available subcommands:
   exclude
   include
   help (?,  h)
```

有意义的子命令只有两个。你可以使用这两个子命令说明你希望保留和不希望保留的路径：

exclude
   将指定路径的数据从转储数据流中排除。

include
   将指定路径的数据添加到转储数据流中。

现在我来演示如何使用这个命令。我们会在其它章节（参见 第 5.4.1 节 “选择一种版本库布局”）讨论关于如何选择设定版本库布局的问题，比如应该使用一个版本库管理多个项目还是使用一个版本库管理一个项目，或者如何在版本库中安排数据等等。不过，有些时候，即使在项目已经展开以后，你还是希望对版本库的布局做一些调整。最常见的情况是，把原来存放在同一个版本库中的几个项目分开，各自成家。

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/
      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 /path/to/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：

```
$ cat repos-dumpfile | svndumpfilter include calc > calc-dumpfile
…
$ cat repos-dumpfile | svndumpfilter include calendar > cal-dumpfile
…
$ cat repos-dumpfile | svndumpfilter include spreadsheet > ss-dumpfile
…
$
```

现在你必须要作出一个决定了。这三个转储文件中，每个都可以用来创建一个可用的版本库，不过它们保留了原版本库的精确路径结构。也就是说，虽然项目calc现在独占了一个版本库，但版本库中还保留着名为calc的顶级目录。如果希望trunk、tags和branches这三个目录直接位于版本库的根路径下，你可能需要编辑转储文件，调整Node-path和Copyfrom-path头参数，将路径calc/删除。同时，你还要删除转储数据中创建calc目录的部分。一般来说，就是如下的一些内容：

```
Node-path: calc
Node-action: add
Node-kind: dir
Content-length: 0
```

警告

如果你打算通过手工编辑转储文件来移除一个顶级目录，注意不要让你的编辑器将换行符转换为本地格式（比如将\r\n转换为\n）。否则文件的内容就与所需的格式不相符，这个转储文件也就失效了。

剩下的工作就是创建三个新的版本库，然后将三个转储文件分别导入：

```
$ svnadmin create calc; svnadmin load 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 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 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
   如果空修订版本被保留，保持这些空修订版本的属性（日志信息，作者，日期，自定义属性，等等）。如果不设定这个选项，空修订版本将仅保留初始时间戳，以及一个自动生成的日志信息，表明此修订版本由svndumpfilter处理过。

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Copyfrom-path这两个头参数对你有些帮助。

```
…
Node-path: spreadsheet/Makefile
…
```

如果这些路径以斜线开头，那么你传递给svndumpfilter include和svndumpfilter exclude的路径也必须以斜线开头（反之亦然）。如果因为某些原

因转储文件中的路径没有统一使用或不使用斜线开头，[3]也许需要修正这些路径，统一使用斜线开头或不使用斜线开头。

此外，复制操作生成的路径也会带来麻烦。Subversion支持在版本库中进行复制操作，也就是复制一个存在的路径，生成一个新的路径。问题是，svndumpfilter保留的某个文件或目录可能是由某个svndumpfilter排除的文件或目录复制而来的。也就是说，为了确保转储数据的完整性，svndumpfilter需要切断这些复制自被排除路径的文件与源文件的关系，还要将这些文件的内容以新建的方式添加到转储数据中。但是由于Subversion版本库转储文件格式中仅包含了修订版本的更改信息，因此源文件的内容基本上无法获得。如果你不能确定版本库中是否存在类似的情况，最好重新考虑一下到底保留/排除哪些路径。

5.3.1.4. svnshell.py

Subversion源代码树中有一个类似于shell的版本库访问界面。Python脚本svnshell.py（位于源代码树的tools/examples/下）通过Subversion语言绑定接口（所以运行这个脚本须要正确的编译和安装这些程序包）连接到版本库和文件系统库。

运行这个脚本，你可以浏览版本库中的目录，就像在shell下浏览文件系统一样。一开始，你“位于”修订版本HEAD的根目录中， 在命令提示符中可以看到相应的提示。 任何时候都可以使用help命令显示当前可用的命令帮助。

```
$ svnshell.py /path/to/repos
<rev: 2 />$  help
Available commands:
  cat FILE     : dump the contents of FILE
  cd DIR       : change the current working directory to DIR
  exit         : exit the shell
  ls [PATH]    : list the contents of the current directory
  lstxns       : list the transactions available for browsing
  setrev REV   : set the current revision to browse
  settxn TXN   : set the current transaction to browse
  youngest     : list the youngest browsable revision number
<rev: 2 />$
```

浏览版本库的目录结构就像在Unix或Windows shell中一样——使用cd命令。任何时候，命令提示符中都会显示当前所在的修订版本（前缀为rev:）或事务（前缀为txn:，以及你所在的路径。你可以用setrev和settxn切换到其它修订版本或事务中去。你可以像在Unix shell中那样，使用ls命令列出目录的内容，使用cat命令列出文件的内容。

例 5.1. 使用svnshell浏览版本库

[3]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。

```
<rev: 2 />$ ls
   REV   AUTHOR  NODE-REV-ID     SIZE         DATE NAME
--------------------------------------------------------------------------------
     1    sally <     2.0.1>          Nov 15 11:50 A/
     2    harry <     1.0.2>       56 Nov 19 08:19 iota
<rev: 2 />$ cd A
<rev: 2 /A>$ ls
   REV   AUTHOR  NODE-REV-ID     SIZE         DATE NAME
--------------------------------------------------------------------------------
     1    sally <     4.0.1>          Nov 15 11:50 B/
     1    sally <     a.0.1>          Nov 15 11:50 C/
     1    sally <     b.0.1>          Nov 15 11:50 D/
     1    sally <     3.0.1>       23 Nov 15 11:50 mu
<rev: 2 /A>$ cd D/G
<rev: 2 /A/D/G>$ ls
   REV   AUTHOR  NODE-REV-ID     SIZE         DATE NAME
--------------------------------------------------------------------------------
     1    sally <     e.0.1>       23 Nov 15 11:50 pi
     1    sally <     f.0.1>       24 Nov 15 11:50 rho
     1    sally <     g.0.1>       24 Nov 15 11:50 tau
<rev: 2 /A>$ cd ../..
<rev: 2 />$ cat iota
This is the file 'iota'.
Added this text in revision 2.

<rev: 2 />$ setrev 1; cat iota
This is the file 'iota'.

<rev: 1 />$ exit
$
```

在上例中可以看到，可以将几条命令现在同一行中，并以分号隔开。此外，这个shell也能正确处理相对路径和绝对路径，以及特殊的路径.和..。

youngest命令将列出最年轻的修订版本。这可以用来确定setrev命令参数的范围—你可以浏览所有0到最年轻修订版本中的任何一个（它们都以整数为标识）。确定可以浏览的事务就不这么简单了。你需要使用lstxns命令列出哪些事务可以浏览。lstxns命令的输出与svnadmin lstxns的输出相同，设置了--transaction选项的svnlook命令也可以得到相同的结果。

使用exit命令可以退出这个shell。也可以使用文件结束符—Control-D（在某些Win32的Python版本中用Control-Z代替）。

5.3.1.5. Berkeley DB工具

如果你使用Berkeley DB版本库，那么所有纳入版本控制的文件系统结构和数据都储存在一系列数据库的表中，而这个位于版本库的db子目录下。这个子目录是一个标准的Berkeley DB环境目录，可以应用任何Berkeley数据库工具进行操作（参考SleepyCat网站http://www.sleepycat.com/上关于这些工具的介绍）。

对于Subversion的日常使用来说，这些工具并没有什么用处。大多数Subversion版本库必须的数据库操作都集成到svnadmin工具中。比如，svnadmin list-unused-dblogs和svnadmin list-dblogs实现了Berkeley db_archive命令功能的一个子集，而svnadmin recover则起到了 db_recover工具的作用。

当然，还有一些Berkeley DB工具有时是有用的。db_dump将Berkeley DB数据库中的键值对以特定的格式写入文件中，而db_load则可以将这些键值对注入到数据库中。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。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。

5.3.2. 版本库清理

Subversion版本库一旦按照需要配置完成，一般情况下不需要特别的关照。不过有些时候还是需要管理员手工干预一下。svnadmin工具就能够帮你完成以下这类工作：

- 修改提交日志信息，
- 移除中止的事务，
- 恢复“塞住”的版本库，以及
- 将一个版本库中的内容搬移到另一个版本库中。

svnadmin的子命令中最经常用到的恐怕就是setlog。用户在提交时输入的日志信息随着相关事务提交到版本库并升级成为修订版本后，便作为新修订版本的非版本化（即没有进行版本管理）属性保存下来。换句话说，版本库只记得最新的属性值，而忽略以前的。

有时用户输入的日志信息有错误（比如拼写错误或者内容错误）。如果配置版本库时设置了（使用pre-revprop-change和 post-revprop-change钩子；参见第 5.2.1 节 “钩子脚本”）允许用户在提交后修改日志信息的选项，那么用户可以使用svn程序的propset命令（参见第 9 章 Subversion完全参考）“修正”日志信息中的错误。不过为了避免永远丢失信息，Subversion版本库通常设置为仅能由管理员修改非版本化属性（这也是默认的选项）。

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。

svnadmin的另一个常见用途是查询异常的—可能是已经死亡的—Subversion事务。通常提交操作失败时，与之相关的事务就会被清除。也就是说，事务本身及所有与该事务相关（且仅与该事务相关）的数据会从版本库中删除。不过偶尔也会出现操作失败而事务没有被清除的情况。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的lstxns 命令列出当前的异常事务名。

```
$ svnadmin lstxns myrepos
19
3a1
a45
$
```

将输出的结果条目作为svnlook（设置--transaction选项）的参数，就可以获得事务的详细信息，如事务的创建者、创建时间，事务已作出的更改类型，由这些信息可以判断出是否可以将这个事务安全的删除。如果可以安全删除，那么只需将事务名作为参数输入到svnadmin rmtxns，就可以将事务清除掉了。其实rmtxns子命令可以直接以lstxns的输出作为输入进行清理。

```
$ 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。下面例子中的shell脚本可以用来迅速获得版本库中异常事务的信息：

例 5.2. txn-info.sh（异常事务报告）

```
#!/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}" --transaction "${TXN}"
done
```

可以用下面的命令使用上例中脚本： /path/to/txn-info.sh /path/to/repos。该命令的输出主要由多个svnlook info参见第 5.3.1.1 节 “svnlook”）的输出组成，类似于下面的例子：

```
$ 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。

简言之，作出事务清理的决定前应该仔细考虑一下。许多信息源—比如Apache的错误和访问日志，已成功完成的Subversion提交日志等等—都可以作为决策的参考。管理员还可以直接和那些似乎已经死亡事务的提交者直接交流（比如通过邮件），来确认该事务确实已经死亡了。

5.3.3. 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留。哪些数据需要备份、哪些数据可以安全的删除等等。本节的内容专注于Berkeley DB类型的版本库。FSFS类型的版本库不需要进行数据清理和回收。

目前为止，Subversion版本库中耗费磁盘空间的最大凶手是日志文件，每次Berkeley DB在修改真正的数据文件之前都会进行预写入（pre-writes）操作。这些文件记录了数据库从一个状态变化到另一个状态的所有动作——数据库文件反应了特定时刻数据库的状态，而日志文件则记录了所有状态变化的信息。因此，日志文件会以很快的速度膨胀起来。

幸运的是，从版本4.2开始，Berkeley DB的数据库环境无需额外的操作即可删除无用的日志文件。如果编译svnadmin时使用了高于4.2版本的Berkeley DB，那么由此svnadmin程序创建的版本库就具备了自动清除日志文件的功能。如果想屏蔽这个功能，只需设置svnadmin create命令的--bdb-log-keep选项即可。如果创建版本库以后想要修改关于此功能的设置，只需编辑版本库中db目录下的DB_CONFIG文件，注释掉包含set_flags DB_LOG_AUTOREMOVE内容的这一行，然后运行svnadmin recover强制设置生效就行了。查阅第 5.2.2 节 “Berkeley DB配置”获得更多关于数据库配置的帮助信息。

如果不自动删除日志文件，那么日志文件会随着版本库的使用逐渐增加。这多少应该算是数据库系统的特性，通过这些日志文件可以在数据库严重损坏时恢复整个数据库的内容。但是一般情况下，最好是能够将无用的日志文件收集起来并删除，这样就可以节省磁盘空间。使用svnadmin list-unused-dblogs命令可以列出无用的日志文件：

```
$ svnadmin list-unused-dblogs /path/to/repos
/path/to/repos/log.0000000031
/path/to/repos/log.0000000032
/path/to/repos/log.0000000033

$ svnadmin list-unused-dblogs /path/to/repos | xargs rm
## disk space reclaimed!
```

为了尽可能减小版本库的体积，Subversion在版本库中采用了增量化技术（或称

为“增量存储技术”）。增量化技术可以将一组数据表示为相对于另一组数据的不同。如果这两组数据十分相似，增量化技术就可以仅保存其中一组数据以及两组数据的差别，而不需要同时保存两组数据，从而节省了磁盘空间。每次一个文件的新版本提交到版本库，版本库就会将之前的版本（之前的多个版本）相对于新版本做增量化处理。采用了这项技术，版本库的数据量大小基本上是可以估算出来的—主要是版本化的文件的大小—并且远小于“全文”保存所需的数据量。

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。

5.3.4. 版本库的恢复

第 5.1.3.1 节 “Berkeley DB”中曾提到，Berkeley DB版本库如果没有正常关闭可能会进入冻结状态。这时，就需要管理员将数据库恢复到正常状态。

Berkeley DB使用一种锁机制保护版本库中的数据。锁机制确保数据库不会同时被多个访问进程修改，也就保证了从数据库中读取到的数据始终是稳定而且正确的。当一个进程需要修改数据库中的数据时，首先必须检查目标数据是否已经上锁。如果目标数据没有上锁，进程就将它锁上，然后作出修改，最后再将锁解除。而其它进程则必须等待锁解除后才能继续访问数据库中的相关内容。

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“塞住”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）。

首先，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取保只有灾难性的事件[4]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. 确保没有其它进程访问（或者试图访问）版本库。对于网络版本库，关闭Apache HTTP服务器是个好办法。

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. 运行命令svnadmin recover /path/to/repos。 输出如下：

[4]比如：硬盘 + 大号电磁铁 = 毁灭。

```
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Subversion服务器。

这个方法能修复几乎所有版本库锁住的问题。记住，要以数据库的拥有者和管理员的身份运行这个命令，而不一定是root用户。恢复过程中可能会使用其它数据存储区（例如共享内存区）重建一些数据库文件。如果以root用户身份恢复版本库，这些重建的文件拥有者将变成root用户，也就是说，即使恢复了到版本库的连接，一般的用户也无权访问这些文件。

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。

5.3.5. 版本库的移植

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的数据（或者一部分数据）保存在一个独立的、可移植的、普通格式的文件中。Subversion通过svnadmin的两个子命令dump和load提供了类似的功能。

对版本库的转储和装载的需求主要还是由于Subversion自身处于变化之中。在Subversion的成长期，后端数据库的设计多次发生变化，这些变化导致之前的版本库出现兼容性问题。当然，将Berkeley DB版本库移植到不同的操作系统或者CPU架构上，或者在Berkeley DB和FSFS后端之间进行转化也需要转储和装载功能。按照下面的介绍，只需简单几步就可以完成数据库的移植：

1. 使用当前版本的svnadmin将版本库转储到文件中。

2. 升级Subversion。

3. 移除以前的版本库，并使用新版本的svnadmin在原来版本库的位置建立空的版本库。

4. 还是使用新版本的svnadmin从转储文件中将数据装载到新建的空版本库中。

5. 记住从以前的版本库中复制所有的定制文件到新版本库中，包括DB_CONFIG文件和钩子脚本。最好阅读一下新版本的release notes，看看此次升级是否会影响钩子和配置选项。

6. 如果移植的同时改变的版本库的访问地址（比如移植到另一台计算机或者改变了访问策略），那么可以通知用户运行svn switch --relocate来切换他们的工作副本。参见svn switch。

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流，而提示信息会输出到标准错误流。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.
```

最后，版本库中的指定的修订版本数据被转储到一个独立的文件中（在上面的例子中是dumpfile）。注意，svnadmin dump从版本库中读取修订版本树与其它“读者”（比如svn checkout）的过程相同，所以可以在任何时候安全的运行这个命令。

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) >>>
```

既然svnadmin使用标准输入流和标准输出流作为转储和装载的输入和输出，那么更漂亮的用法是（管道两端可以是不同版本的svnadmin：

```
$ svnadmin create newrepos
$ svnadmin dump myrepos | svnadmin load newrepos
```

默认情况下，转储文件的体积可能会相当庞大——比版本库自身大很多。这是因为在转储文件中，每个文件的每个版本都以完整的文本形式保存下来。这种方法速度很快，而且很简单，尤其是直接将转储数据通过管道输入到其它进程中时（比如一个压缩程序，过滤程序，或者一个装载进程）。不过如果要长期保存转储文件，那么可以使用--deltas选项来节省磁盘空间。设置这个选项，同一个文件的数个连续修订版本会以增量式的方式保存—就像储存在版本库中一样。这个方法较慢，但是转储文件的体积则基本上与版本库的体积相当。

之前我们提到svnadmin dump输出指定的修订版本。使用--revision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 --revision 23 > rev-23.dumpfile
$ svnadmin dump myrepos --revision 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。

默认情况下，Subversion不会把转储的第一个修订版本看作对前一个修订版本的更改。 首先，转储文件中没有比第一个修订版本更靠前的修订版本了！其次，Subversion不知道装载转储数据时（如果真的需要装载的话）的版本库是什么样的情况。为了保证每次运行svnadmin dump都能得到一个独立的结果，第一个转储的修订版本默认情况下会完整的保存目录、文件以及属性等数据。

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 --revision 0:1000 > dumpfile1
$ svnadmin dump myrepos --revision 1001:2000 --incremental > dumpfile2
$ svnadmin dump myrepos --revision 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的dump和load命令就变成了很好的版本库备份工具，万一出现系统崩溃或其它灾难性事件，它的价值就体现出来了。

转储还可以用来将几个独立的版本库合并为一个版本库。使用svnadmin load的--parent-dir选项，可以在装载的时候指定根目录。也就是说，如果有三个不同版本库的转储文件，比如calc-dumpfile，cal-dumpfile，和ss-dumpfile，可以在一个新的版本库中保存所有三个转储文件中的数据：

```
$ svnadmin create /path/to/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:///path/to/projects/calc \
      file:///path/to/projects/calendar \
      file:///path/to/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 /path/to/projects --parent-dir calc < calc-dumpfile
…
$ svnadmin load /path/to/projects --parent-dir calendar < cal-dumpfile
…
$ svnadmin load /path/to/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，[5]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.py工具（参见 第 A.11 节 “转化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。

5.3.6. 版本库备份

尽管现代计算机的诞生带来了许多便利，但有一件事听起来是完全正确的—有时候，事情变的糟糕，很糟糕，动力损耗、网络中断、坏掉的内存和损坏的硬盘都是对魔鬼的一种体验，即使对于最尽职的管理员，命运也早已注定。所以我们来到了这个最重要的主题—怎样备份你的版本库数据。

Subversion版本库管理员通常有两种备份方式—增量的和完全的。我们在早先的章节曾经讨论过如何使用svnadmin dump --incremental命令执行增量备份（见第 5.3.5 节 “版本库的移植”），从本质上讲，这个方法只是备份了从你上次备份版本库到现在的变化。

一个完全的版本库备份照字面上讲就是对整个版本库目录的复制（包括伯克利数据库或者文件FSFS环境），现在，除非你临时关闭了其他对版本库的访问，否则仅仅做一次迭代的拷贝会有产生错误备份的风险，因为有人可能会在并行的写数据库。

如果是伯克利数据库，恼人的文档描述了保证安全拷贝的步骤，对于FSFS的数据，也有类似的顺序。我们有更好的选择，我们不需要自己去实现这个算法，因为Subversion开发小组已经为你实现了这些算法。Subversion源文件分发版本的tools/backup/目录有一个hot-backup.py文件。只要给定了版本库路径和备份路径，hot-backup.py—一个包裹了svnadmin hotcopy但更加智能的命令—将会执行必要的步骤来备份你的活动的版本库—不需要你首先禁止公共的版本库访问—而且之后会从你的版本库清理死掉的伯克利日志文件。

甚至当你用了一个增量备份时，你也会希望有计划的运行这个程序。举个例子，你考虑在你的调度程序（如Unix下的cron）里加入hot-backup.py，或者你喜欢更加细致的备份解决方案，你可以让你的post-commit的钩子脚本执行hot-backup.py（见see 第 5.2.1 节 “钩子脚本”），这样会导致你的版本库的

[5]Subversion版本库的转储文件格式类似于RFC-822格式，后者广泛的应用于电子邮件系统中。

每次提交执行一次备份，只要在你的hooks/post-commit脚本里添加如下代码：

```
(cd /path/to/hook/scripts; ./hot-backup.py ${REPOS} /path/to/backups &)
```

作为结果的备份是一个完全功能的版本库，当发生严重错误时可以作为你的活动版本库的替换。

两种备份方式都有各自的优点，最简单的方式是完全备份，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。

增量备份会使用的版本库转储格式，在Subversion的数据库模式改变时非常完美，因此当我们升级Subversion数据库模式的时候，一个完整的版本库导出和导入是必须的，做一半工作非常的容易（导出部分），不幸的是，增量备份的创建和恢复会占用很长时间，因为每一次提交都会被重放。

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。[6] 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。

通常情况下，最好的版本库备份方式是混合的，你可以平衡完全和增量备份，另外配合提交邮件的归档，Subversion开发者，举个例子，在每个新的修订版本建立时备份Subversion的源代码版本库，并且保留所有的提交和属性修改通知文件。你的解决方案类似，必须迎合你的需要，平衡便利和你的偏执。然而这些不会改变你的硬件来自钢铁的命运。[7] 这一定会帮助你减少尝试的时间。

5.4. 添加项目

一旦你的版本库已经建立并且配置好了，剩下的就是使用了。如果你已经准备好了需要版本控制的数据，那么可以使用客户端软件svn的import子命令来实现你的期望。不过在这样做之前，你最好对版本库仔细的作一个长远的规划。本节，我们会给你一些好的建议，这些建议可以帮助你设计版本库的文件布局，以及如何

[6] svnadmin setlog可以被绕过钩子程序被调用。
[7] 你知道的—只是对各种变化莫测的问题的统称。

在特定的布局中安排你的数据。

5.4.1. 选择一种版本库布局

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，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。

在建立Subversion版本库之前，有很多事情需要考虑。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，还是为其中的某些项目提供独立的版本库支持，而将另外一些项目分布在几个版本库中。

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集，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提交邮件列表或者不同的权限认证和权限要求。还有，别忘了Subversion的修订版本号是针对整个版本库的。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。

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。

在决定了如何用版本库组织项目以后，就该决定如何设置版本库的目录层次了。由于Subversion按普通的目录复制方式完成分支和标签操作（参见第 4 章 分支与合并），Subversion社区建议为每一个项目建立一个项目根目录—项目的“顶级”目录—然后在根目录下建立三个子目录：trunk，保存项目的开发主线；branches，保存项目的各种开发分支；tags，保存项目的标签，也就是创建后永远不会修改的分支（可能会删除）。

举个例子，一个版本库可能会有如下的布局：

```
/
   calc/
      trunk/
      tags/
      branches/
   calendar/
      trunk/
      tags/
      branches/
   spreadsheet/
```

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

项目在版本库中的根目录地址并不重要。如果每个版本库中只有一个项目，那么就可以认为项目的根目录就是版本库的根目录。如果版本库中包含多个项目，那么可以将这些项目划分成不同的组合（按照项目的目标或者是否需要共享代码甚至是字母顺序）保存在不同子目录中，下面的例子给出了一个类似的布局：

```
/
   utils/
      calc/
         trunk/
         tags/
         branches/
      calendar/
         trunk/
         tags/
         branches/
      ...
   office/
      spreadsheet/
         trunk/
         tags/
         branches/
      ...
```

按照你因为合适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。

5.4.2. 创建布局，导入初始数据

设计好版本库的布局后，就该在版本库中实现布局和导入初始数据了。在Subversion中，有很多种方法完成这项工作。可以使用svn mkdir命令（参见第 9 章 Subversion完全参考）在版本库中逐个创建需要的目录。更快捷的方法是使用svn import命令（参见第 3.7.2 节 “svn import”）。首先，在硬盘上创建一个临时目录，并按照设计好的布局在其中创建子目录，然后通过导入命令一次性的提交整个布局到版本库中：

```
$ mkdir tmpdir
$ cd tmpdir
$ mkdir projectA
```

```
$ mkdir projectA/trunk
$ mkdir projectA/branches
$ mkdir projectA/tags
$ mkdir projectB
$ mkdir projectB/trunk
$ mkdir projectB/branches
$ mkdir projectB/tags
…
$ svn import . file:///path/to/repos --message 'Initial repository layout'
Adding         projectA
Adding         projectA/trunk
Adding         projectA/branches
Adding         projectA/tags
Adding         projectB
Adding         projectB/trunk
Adding         projectB/branches
Adding         projectB/tags
…
Committed revision 1.
$ cd ..
$ rm -rf tmpdir
$
```

然后可以使用svn list命令确认导入的结果是否正确：：

```
$ svn list --verbose file:///path/to/repos
      1 harry               May 08 21:48 projectA/
      1 harry               May 08 21:48 projectB/
…
$
```

创建了版本库布局以后，如果有项目的初始数据，那么可以将这些数据导入到版本库中。同样有很多种方法完成这项工作。首先，可以使用svn import命令。也可以先从版本库中取出工作副本，将已有的项目数据复制到工作副本中，再使用svn add和svn commit命令提交修改。不过这些工作就不属于版本库管理方面的内容了。如果对svn 客户端程序还不熟悉，请阅读第 3 章 指导教程。

5.5. 摘要

现在，你应该已经对如何创建、配置以及维护Subversion版本库有了个基本的认识。我们向您介绍了几个可以帮助您工作的工具。通过这一章，我们说明了一些常见的管理误区，并提出了避免陷入误区的建议。

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 配置服务器

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。

本小节描述了怎样将一个Subversion的版本库暴露给远程客户端，我们会覆盖Subversion已存在的服务器机制，讨论各种方式的配置和使用。经过阅读本小节，你可以决定你需要哪种网络设置，并且明白怎样在你的主机上进行配置。

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，目前实践中存在着两种服务器。

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，另一个是svnserve：一个小的，独立服务器，使用自己定义的协议和客户端，表格6-1比较了这两种服务器。

需要注意到Subversion作为一个开源的项目，并没有官方的指定何种服务器是“主要的”或者是“官方的”，并没有那种网络实现被视作二等公民，每种服务器都有自己的优点和缺点，事实上，不同的服务器可以并行工作，分别通过自己的方式访问版本库，它们之间不会互相阻碍（见第 6.5 节 “支持多种版本库访问方法”）。以下是对两种存在的Subversion服务器的比较—作为一个管理员，你更加胜任给你和你的用户挑选服务器的任务。

表 6.1. 网络服务器比较

特性	Apache + mod_dav_svn	svnserve
认证选项	HTTP(S) basic auth 、X.509 certificates 、LDAP、NTLM或任何Apache httpd已经具备的方式	CRAM-MD5或SSH
用户帐号选项	私有的'users'文件	私有的'users'文件，或存在的系统(SSH)帐户
授权选项	整体的读/写访问，或者是每目录的读/写访问	整体的读/写访问，或者是使用pre-commit钩子的每目录写访问（但不是读）
加密	通过选择SSL	通过选择SSH通道
交互性	可以部分的被其他WebDAV客户端使用	不能被其他客户端使用
Web浏览能力	有限的内置支持，或者通	通过第三方工具，如

特性	Apache + mod_dav_svn	svnserve
	过第三方工具，如ViewCVS	ViewCVS
速度	有些慢	快一点
初始化配置	有些复杂	相当简单

6.2. 网络模型

这部分是讨论了Subversion客户端和服务器怎样互相交流，不考虑具体使用的网络实现，通过阅读，你会很好的理解服务器的行为方式和多种客户端与之响应的配置方式。

6.2.1. 请求和响应

Subversion客户端花费大量的时间来管理工作拷贝，当它需要版本库信息，它会做一个网络请求，然后服务器给一个恰当的回答，具体的网络协议细节对用户不可见，客户端尝试去访问一个URL，根据URL模式的不同，会使用特定的协议与服务器联系（见版本库的URL），用户可以运行svn --version来查看客户端可以使用的URL模式和协议。

当服务器处理一个客户端请求，它通常会要求客户端确定它自己的身份，它会发出一个认证请求给客户端，而客户端通过提供凭证给服务器作为响应，一旦认证结束，服务器会响应客户端最初请求的信息。注意这个系统与CVS之类的系统不一样，它们会在请求之前，预先提供凭证（“logs in”）给服务器，在Subversion里，服务器通过请求客户端适时地“拖入”凭证，而不是客户端“推”出。这使得这种操作更加的优雅，例如，如果一个服务器配置为世界上的任何人都可以读取版本库，在客户使用svn checkout时，服务器永远不会发起一个认证请求。

如果客户端请求往版本库写入新的数据（例如svn commit），这会建立新的修订版本树，如果客户端的请求是经过认证的，认证过的用户的用户名就会作为svn:author属性的值保存到新的修订本里（见第 5.1.2 节 “未受版本控制的属性”）。如果客户端没有经过认证（换句话说，服务器没有发起过认证请求），这时修订本的svn:author的值是空的。[1]

6.2.2. 客户端凭证缓存

许多服务器配置为在每次请求时要求认证，这对一次次输入用户名和密码的用户来说是非常恼人的事情。

令人高兴的是，Subversion客户端对此有一个修补：存在一个在磁盘上保存认证凭证缓存的系统，缺省情况下，当一个命令行客户端成功的在服务器上得到认证，它会保存一个认证文件到用户的私有运行配置区—类Unix系统下会在~/.subversion/auth/，Windows下在%APPDATA%/Subversion/auth/（运行区在第 7.1 节 “运行配置区”会有更多细节描述）。成功的凭证会缓存在磁盘，以主机名、端口和认证域的组合作为唯一性区别。

[1]这个问题实际上是一个FAQ，源自错误的服务器配置。

当客户端接收到一个认证请求，它会首先查找磁盘中的认证凭证缓存，如果没有发现，或者是缓存的凭证认证失败，客户端会提示用户需要这些信息。

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！”但是请保持冷静，首先，auth/是访问保护的，只有用户（拥有者）可以读取这些数据，不是整个世界，如果这对你还不够安全，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配置文件（坐落在auth/目录），只需要把store-auth-creds设置为no，这样就不会有凭证缓存在磁盘。

```
[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。

客户端认证的行为的最后一点：对使用--username和--password选项的一点说明，许多客户端和子命令接受这个选项，但是要明白使用这个选项不会主动地发送凭证信息到服务器，就像前面讨论过的，服务器会在需要的时候才会从客户端“拖”入凭证，客户端不会随意“推”出。如果一个用户名和/或者密码作为选项传入，它们只会在服务器需要时展现给服务器。[2]通常，只有在如下情况下才会使用这些选项：

- 用户希望使用与登陆系统不同的名字认证，或者
- 一段不希望使用缓存凭证但需要认证的脚本

这里是Subversion客户端在收到认证请求的时候的行为方式：

1. 检查用户是否通过--username和/或--password命令选项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. 查找运行中的auth/区域保存的服务器域信息，来确定用户是否已经有了恰当的认证缓存，如果没有，或者缓存凭证认证失败，然后
3. 提示用户输入。

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。）

[2]再次重申，一个常见的错误是把服务器配置为从不会请求认证，当用户传递--username和--password给客户端时，他们惊奇的发现它们没有被使用，如新的修订版本看起来始终是由匿名用户提交的！

6.3. svnserve，一个自定义的服务器

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

有许多调用svnserve的方式，如果调用时没有参数，你只会看到一些帮助信息，然而，如果你计划使用inetd启动进程，你可以传递-i（--inetd）选项：

```
$ svnserve -i
( success ( 1 2 ( ANONYMOUS ) ( edit-pipeline ) ) )
```

当用参数--inetd调用时，svnserve会尝试使用自定义协议通过stdin和stdout来与Subversion客户端通话，这是使用inetd工作的标准方式，IANA为Subversion协议保留3690端口，所以在类Unix系统你可以在/etc/services添加如下的几行（如果他们还不存在）：

```
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。

在一个Windows系统，有第三方工具可以将svnserve作为服务运行，请看Subversion的网站的工具列表。

svnserve的第二个选项是作为独立“守护”进程，为此要使用-d选项：

```
$ svnserve -d
$               # svnserve is now running, listening on port 3690
```

当以守护模式运行svnserve时，你可以使用--listen-port=和--listen-host=选项来自定义“绑定”的端口和主机名。

也一直有第三种方式，使用-t选项的“管道模式”，这个模式假定一个分布式服务程序如RSH或SSH已经验证了一个用户，并且以这个用户调用了一个私有svnserve进程，svnserve运作如常（通过stdin和stdout通讯），并且可以设想通讯是自动转向到一种通道传递回客户端，当svnserve被这样的通道代理调用，确定认证用户对版本数据库有完全的读写权限，（见服务器和访问许可：一个警告。）这与本地用户通过file:///URl访问版本库同样重要。

> 服务器和访问许可：一个警告
>
> 首先需要记住，一个Subversion版本库是一组数据库文件，任何进程直接访问版本库需要对整个版本库有正确的读写许可，如果你不仔细处理，这会变得很头痛，特别是当你使用Berkeley DB数据库而不是FSFS时，详细信息可以阅读第 6.5 节 “支持多种版本库访问方法”。
>
> 第二点，当配置svnserve、Apache httpd或者其它任何服务器时，不要使用root用户（或者其它具备无限制权限的用户）启动服务器进程，根据所有权和版本库允许的权限，通常应该创建一个新的自定义用户，例如很多管理员会创建一个叫做svn的用户，赋予这个用户排他的拥有权和对Subversion版本库的导出权利，只让服务器以这个用户运行。

一旦svnserve已经运行，它会将你系统中所有版本库发布到网络，一个客户端需要指定版本库在URL中的绝对路径，举个例子，如果一个版本库是位于 /usr/local/repositories/project1 ， 则一个客户端可以使用 svn://host.example.com/usr/local/repositories/project1 来进行访问，为了提高安全性，你可以使用svnserve的-r选项，这样会限制只输出指定路径下的版本库：

```
$ svnserve -d -r /usr/local/repositories
…
```

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
…
```

6.3.2. 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。
- 依赖于位置和授权政策，
  - 如果没有收到认证请求，客户端可能被允许匿名访问，或者
  - 客户端收到认证请求，或者
  - 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。

在撰写本文时，服务器还只知道怎样发送CRAM-MD5[3]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。

当然也有可能，如果客户端在外部通过通道代理认证，如SSH，在那种情况下，服务器简单的检验作为那个用户的运行，然后使用它作为认证用户名，更多信息请看第 6.3.3 节 “SSH认证和授权”。

像你已经猜测到的，版本库的svnserve.conf文件是控制认证和授权政策的中央机构，这文件与其它配置文件格式相同（见第 7.1 节 “运行配置区”）：小节名称使用方括号标记（[和]），注释以井号（#）开始，每一小节都有一些参数可以设置（variable = value），让我们浏览这个文件并且学习怎样使用它们。

6.3.2.1. 创建一个用户文件和域

此时，svnserve.conf文件的[general]部分包括所有你需要的变量，开始先定义一个保存用户名和密码的文件和一个认证域：

```
[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 6.2.2 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
sally = barpassword
```

[3] 见RFC 2195。

password-db的值可以是用户文件的绝对或相对路径，对许多管理员来说，把文件保存在版本库conf/下的svnserve.conf旁边是一个简单的方法。另一方面，可能你的多个版本库使用同一个用户文件，此时，这个文件应该在更公开的地方，版本库分享用户文件时必须配置为相同的域，因为用户列表本质上定义了一个认证域，无论这个文件在哪里，必须设置好文件的读写权限，如果你知道运行svnserve的用户，限定这个用户对这个文件有读权限是必须的。

6.3.2.2. 设置访问控制

svnserve.conf有两个或多个参数需要设置：它们确定未认证（匿名）和认证用户可以做的事情，参数anon-access和auth-access可以设置为none、read或者write，设置为none会限制所有方式的访问，read允许只读访问，而write允许对版本库完全的读/写权限：

```
[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
```

注意svnserve只能识别“整体”的访问控制，一个用户可以有全体的读/写权限，或者只读权限，或没有访问权限，没有对版本库具体路径访问的细节控制，很多项目和站点，这种 访问控制已经完全足够了，然而，如果你希望单个目录访问控制，你会需要使用包括mod_authz_svn（见第 5.2.1 节 “钩子脚本”）的Apache，或者是使用pre-commit钩子脚本来控制写访问（见第 5.2.1 节 “钩子脚本”），Subversion的分发版本包含一个commit-access-control.pl和一个更加复杂的svnperms.py脚本可以作为pre-commit脚本使用。

6.3.3. SSH认证和授权

svnserve的内置认证会非常容易得到，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
harry@host.example.com's password:  *****

foo
bar
baz
…
```

在这个例子里，Subversion客户端会调用一个ssh进程，连接到host.example.com，使用用户harry认证，然后会有一个svnserve私有进程以用户harry运行。svnserve是以管道模式调用的（-t），它的网络协议是通过ssh“封装的”，被管道代理的svnserve会知道程序是以用户harry运行的，如果客户执行一个提交，认证的用户名会作为版本的参数保存到新的修订本。

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 6.2.2 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。

当在一个管道上运行时，认证通常是基于操作系统对版本库数据库文件的访问控制，这同Harry直接通过file:///的URL直接访问版本库非常类似，如果有多个系统用户要直接访问版本库，你会希望将他们放到一个常见的组里，你应该小心的使用umasks。（确定要阅读第 6.5 节 “支持多种版本库访问方法”）但是即使是在管道模式时，文件svnserve.conf还是可以阻止用户访问，如auth-access = read或者auth-access = none。

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
```

现在你可以通过指定与定义匹配的URL模式来使用新的管道定义：svn+rsh://host/path。当使用新的URL模式时，Subversion客户端实际上会在后台运行rsh host svnserve -t这个命令，如果你在URL中包括一个用户名（例如，svn+rsh://username@host/path），客户端也会在自己的命令中包含这部分（rsh username@host svnserve -t），但是你可以定义比这个更加智能的新的管道模式：

```
[tunnels]
joessh = $JOESSH /opt/alternate/ssh -p 29934
```

这个例子里论证了一些事情，首先，它展现了如何让Subversion客户端启动一个特定的管道程序（这个在/opt/alternate/ssh），在这个例子里，使用svn+joessh://的URL会以-p 29934参数调用特定的SSH程序—对连接到非标准端口的程序非常有用。

第二点，它展示了怎样定义一个自定义的环境变量来覆盖管道程序中的名字，设置SVN_SSH环境变量是覆盖缺省的SSH管道的一种简便方法，但是如果你需要为多个服务器做出多个不同的覆盖，或许每一个都联系不同的端口或传递不同的SSH选项，你可以使用本例论述的机制。现在如果我们设置JOESSH环境变量，它的值会覆盖管道中的变量值—会执行$JOESSH而不是/opt/alternate/ssh -p 29934。

6.3.4. SSH配置技巧

不仅仅是可以控制客户端调用ssh方式，也可以控制服务器中的sshd的行为方式，在本小节，我们会展示怎样控制sshd执行svnserve，包括如何让多个用户分享同一个系统帐户。

6.3.4.1. 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SSHauthorized_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
```

当command字段设置后，SSH守护进程运行命名的程序而不是通常Subversion客户端询问的svnserve -t。这为实施许多服务器端技巧开启了大门，在下面的例子里，我们简写了文件的这些行：

```
command="program" TYPE KEY COMMENT
```

6.3.4.2. 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 6.5 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。

多个用户也可以共享同一个帐号，作为为每个用户创建系统帐户的替代，我们创建一个公开/私有密钥对，然后在authorized_users文件里放置各自的公钥，一个用户一行，使用--tunnel-user选项：

```
command="svnserve -t --tunnel-user=harry" TYPE1 KEY1 harry@example.com
command="svnserve -t --tunnel-user=sally" TYPE2 KEY2 sally@example.com
```

这个例子允许Harry和Sally通过公钥认证连接同一个的账户，每个人自定义的命令将会执行。--tunnel-user选项告诉svnserve -t命令采用命名的参数作为经过认证的用户，如果没有--tunnel-user，所有的提交会作为共享的系统帐户提交。

最后要小心：设定通过公钥共享账户进行用户访问时还会允许其它形式的SSH访问，即使你设置了authorized_keys的command值，举个例子，用户仍然可以通过SSH得到shell访问，或者是通过服务器执行X11或者是端口转发。为了给用户尽可能少的访问权限，你或许希望在command命令之后指定一些限制选项：

```
command="svnserve -t --tunnel-user=harry",no-port-forwarding,\
        no-agent-forwarding,no-X11-forwarding,no-pty \
```

```
TYPE1 KEY1 harry@example.com
```

6.4. httpd，Apache的HTTP服务器

Apache的HTTP服务器是一个Subversion可以利用的“重型”网络服务器，通过一个自定义模块，httpd可以让Subversion版本库通过WebDAV/DeltaV协议在客户端前可见，WebDAV/DeltaV协议是HTTP 1.1的扩展（见http://www.webdav.org/来查看详细信息）。这个协议利用了无处不在的HTTP协议是广域网的核心这一点，添加了写能力—更明确一点，版本化的写—能力。结果就是这样一个标准化的健壮的系统，作为Apache 2.0软件的一部分打包，被许多操作系统和第三方产品支持，网络管理员也不需要打开另一个自定义端口。[4]这样一个Apache-Subversion服务器具备了许多svnserve没有的特性，但是也有一点难于配置，灵活通常会带来复杂性。

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。

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?
>
> 如果你系统管理员，很有可能是你已经运行了Apache服务器，并且有一些高级经验。写本文的时候，Apache 1.3是Apache最流行的版本，这个世界因为许多原因而放缓升级到2.X系列：如人们害怕改变，特别是像web服务器这种重要的变化，有些人需要一些在Apache 1.3 API下工作的插件模块，在等待2.X的版本。无论什么原因，许多人会在首次发现Subversion的Apache模块只是为Apache 2 API写的后开始担心。
>
> 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
> ...
> ```

[4]他们讨厌这样做。

6.4.1. 必备条件

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，
- 为mod_dav安装mod_dav_svn插件，它会使用Subversion的库访问版本库，并且
- 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 /absolute/path/to/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，如果你知道你会在/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 /usr/local/svn/foo
  SVNParentPath /usr/local/svn
</Location>
```

使用上面的语法，Apache会代理所有URL路径部分为/svn/的请求到Subversion的DAV提供者，Subversion会认为SVNParentPath指定的目录下的所有项目是真实的Subversion版本库，这通常是一个便利的语法，不像是用SVNPath指示，我们在此不必为创建新的版本库而重启Apache。

请确定当你定义新的位置，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。

> 服务器名称和拷贝请求
>
> 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
> ```
>
> 如果你通过NameVirtualHost指示使用Apache的虚拟主机，你或许需要ServerAlias指示来指定额外的名称，再说一次，可以查看Apache文档的来得到更多细节。

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。（见服务器和访问许可：一个警告。）

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

此时，如果你配置的httpd.conf保存如下的内容

```
<Location /svn>
  DAV svn
  SVNParentPath /usr/local/svn
</Location>
```

这样你的版本库对全世界是可以“匿名”访问的，直到你配置了一些认证授权政策，你通过Location指示来使Subversion版本库可以被任何人访问，换句话说，

- 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，并且
- 任何人可以提交到版本库。

6.4.3.1. 基本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
$
```

下一步，你需要在httpd.conf的Location区里添加一些指示来告诉Apache如何来使用这些密码文件，AuthType指示指定系统使用的认证类型，这种情况下，我们需要指定Basic认证系统，AuthName是你提供给认证域一个任意名称，大多数浏览器会在向用户询问名称和密码的弹出窗口里显示这个名称，最终，使用AuthUserFile指示来指定使用htpasswd创建的密码文件的位置。

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 /usr/local/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 /usr/local/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。

需要警惕：HTTP基本认证的密码是用明文传输，因此非常不可靠的，如果你担心密码偷窥，最好是使用某种SSL加密，所以客户端认证使用https://而不是http://，为了方便，你可以配置Apache为自签名认证。[5] 参考Apache的文档（和

OpenSSL文档）来查看怎样做。

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。

如果Subversion使用OpenSSL编译，它就会具备与Subversion服务器使用https://的URL通讯的能力，Subversion客户端使用的Neon库不仅仅可以用来验证服务器证书，也可以必要时提供客户端证书，如果客户端和服务器交换了SSL证书并且成功地互相认证，所有剩下的交流都会通过一个会话关键字加密。

怎样产生客户端和服务器端证书以及怎样使用它们已经超出了本书的范围，许多书籍，包括Apache自己的文档，描述这个任务，现在我们可以覆盖的是普通的客户端怎样来管理服务器与客户端证书。

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

如果客户端接收了一个服务器证书，它需要去验证它是可以相信的：这个服务器是它自称的那一个吗？OpenSSL库会去检验服务器证书的签名人或者是核证机构（CA）。如果OpenSSL不可以自动信任这个CA，或者是一些其他的问题（如证书过期或者是主机名不匹配），Subversion命令行客户端会询问你是否愿意仍然信任这个证书：

```
$ 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:7b

(R)eject, accept (t)emporarily or accept (p)ermanently?
```

这个对话看起来很熟悉，这是你会在web浏览器（另一种HTTP客户端，就像

[5]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。

Subversion）经常看到的问题，如果你选择(p)ermanent选项，服务器证书会存放在你存放那个用户名和密码缓存（见第 6.2.2 节 “客户端凭证缓存”。）的私有运行区auth/中，缓存后，Subversion会自动记住在以后的交流中信任这个证书。

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。

当与Apache通话时，Subversion客户端也会收到一个证书的要求，Apache是询问客户端来证明自己的身份：这个客户端是否是他所说的那一个？如果一切正常，Subversion客户端会发送回一个通过Apache信任的CA签名的私有证书，一个客户端证书通常会以加密方式存放在磁盘，使用本地密码保护，当Subversion收到这个要求，它会询问你证书的路径和保护用的密码：

```
$ 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。[6]

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 /usr/local/svn

  # how to authenticate a user
  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。

[6]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。

```
<Location /svn>
  DAV svn
  SVNParentPath /usr/local/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。

例 6.1. 匿名访问的配置实例。

```
<Location /repos>
  DAV svn
  SVNParentPath /usr/local/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。

例 6.2. 一个认证访问的配置实例。

```
<Location /repos>
  DAV svn
  SVNParentPath /usr/local/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。

例 6.3. 一个混合认证/匿名访问的配置实例。

```
<Location /repos>
  DAV svn
  SVNParentPath /usr/local/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>
```

一旦你的基本Location区块已经配置了，你可以创建一个定义一些授权规则的访问文件。

访问文件的语法与svnserve.conf和运行中配置文件非常相似，以（#）开头的行会被忽略，在它的简单形式里，每一小节命名一个版本库和一个里面的路径，认证用户名是在每个小节中的选项名，每个选项的值描述了用户访问版本库的级别：r（只读）或者rw（读写），如果用户没有提到，访问是不允许的。

具体一点：这个小节的名称是[repos-name:path]或者[path]的形式，如果你使用SVNParentPath指示，指定版本库的名字是很重要的，如果你漏掉了他们，[/some/dir]部分就会与/some/dir的所有版本库匹配，如果你使用SVNPath指示，因此在你的小节中只是定义路径也很好—毕竟只有一个版本库。

```
[calc:/branches/calc/bug-142]
harry = rw
sally = r
```

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。

当然，访问控制是父目录传递给子目录的，这意味着我们可以为Sally指定一个子目录的不同访问策略：

```
[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
```

现在Sally可以读取分支的testing子目录，但对其他部分还是只可以读，同时，Harry对整个分支还继续有完全的读写权限。

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 =
```

在这个例子里，Harry对bug-142目录树有完全的读写权限，但是对secret子目录没有任何访问权利。

有一件事需要记住的是需要找到最匹配的目录，mod_authz_svn模块首先找到匹配自己的目录，然后父目录，然后父目录的父目录，就这样继续下去，更具体的路径控制会覆盖所有继承下来的访问控制。

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*实现，用来代表“所有用户”：

```
[/]
* = r
```

这是一个普通的设置；注意在小节名中没有提到版本库名称，这让所有版本库对所有的用户可读，不管你是使用SVNPath或是SVNParentPath。当所有用户对版本库有了读权利，你可以赋予特定用户对特定子目录的rw权限。

星号（*）参数需要在这里详细强调：这是匹配匿名用户的唯一模式，如果你已经配置了你的Location区块允许匿名和认证用户的混合访问，所有用户作为Apache匿名用户开始访问，mod_authz_svn会在要访问路径的定义中查找*值；如果找不到，Apache就会要求真实的客户端认证。

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]
@paint-developers = rw
jane = r
```

...并且非常接近。

6.4.4.3. 关闭路径为基础的检查

mod_dav_svn模块做了许多工作来确定你标记为“不可读”的数据不会因意外而泄露，这意味着需要紧密监控通过svn checkout或是svn update返回的路径和文件内容，如果这些命令遇到一些根据认证策略不是可读的路径，这个路径通常会被一起忽略，在历史或者重命名操作时—例如运行一个类似svn cat -r OLD foo.c的命令来操作一个很久以前改过名字的文件 — 如果一个对象的以前的名字检测到是只读的，重命令追踪就会终止。

所有的路径检查在有时会非常昂贵，特别是svn log的情况。当检索一列修订版本时，服务器会查看所有修订版本修改的路径，并且检查可读性，如果发现了一个不可读路径，它会从修订版本的修改路径中忽略（可以查看--verbose选项），并且整个的日志信息会被禁止，不必多说，这种影响大量文件修订版本的操作会非常耗时。这是安全的代价：即使你并没有配置mod_authz_svn模块，mod_dav_svn还是会询问httpd来对所有路径运行认证检查，mod_dav_svn模块没有办法知道那个认证模块被安装，所以只有询问Apache来调用所提供的模块。

在另一方面，也有一个安全舱门允许你用安全特性来交换速度，如果你不是坚持要求有每目录授权（如不使用 mod_authz_svn和类似的模块），你就可以关闭所有的路径检查，在你的httpd.conf文件，使用SVNPathAuthz指示：

例 6.4. 关闭所有的路经检查

```
<Location /repos>
  DAV svn
  SVNParentPath /usr/local/svn
```

```
  SVNPathAuthz off
</Location>
```

SVNPathAuthz指示缺省是打开的，当设置为“off”时，所有的路径为基础的授权都会关闭；mod_dav_svn停止对每个目录调用授权检查。

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

一个非常有用的好处是使用Apache/WebDAV配置Subversion版本库时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。

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：

```
<Location /svn>
  DAV svn
  SVNParentPath /usr/local/svn
  SVNIndexXSLT "/svnindex.xsl"
  ...
</Location>
```

使用SVNIndexXSLT指示和创建一个XSLT样式表，你可以让你的目录列表的颜色模式与你的网站的其它部分匹配，否则，如果你愿意，你可以使用Subversion源分发版本中的tools/xslt/目录下的样例样式表。记住提供给SVNIndexXSLT 指示的路径是一个URL路径—浏览器需要阅读你的样式表来利用它们！

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。

为了回答这些问题，一个明显的看老版本文件和目录的方式是带--revision参数的svn list和svn cat命令，为了在浏览器里察看老版本，你可以使用第三方的软件，一个好的例子是ViewCVS（http://viewcvs.sourceforge.net/），ViewCVS最初写出来是为了在web显示CVS版本库，最新的带血的（此时正在编写）版本也已经可以理解Subversion版本库了。

6.4.5.2. 其它特性

Apache作为一个健壮的Web服务器的许多特性也可以用来增加Subversion的功能性和安全性，Subversion使用Neon与Apache通讯，这是一种一般的HTTP/WebDAV库，可以支持SSL和Deflate压缩（是gzip和PKZIP程序用来“压缩”文件为数据块的一样的算法）之类的机制。你只需要编译你希望Subversion和Apache需要的特性，并且正确的配置程序来使用这些特性。

Deflate压缩给服务器和客户端带来了更多地负担，压缩和解压缩减少了网络传输的实际文件的大小，如果网络带宽比较紧缺，这种方法会大大提高服务器和客户端之间发送数据的速度，在极端情况下，这种最小化的传输会造成超时和成功的区别。

不怎么有趣，但同样重要，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和自动版本化来得到更多细节。

6.5. 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（以它们自己运行）；
- 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。

最通常的一个问题是管理进入到版本库的所有权和访问许可，是前面例子的所有进程 （或者说是用户）都有读写Berkeley DB的权限？假定你有一个类Unix的操作系统，一个直接的办法是在新的svn组添加所有潜在的用户，然后让这个组完全拥有版本库，但这样还不足够，因为一个进程会使用不友好的umask来写数据库文件—用来防止别的用户的访问。

所以下一步我们不选择为每个版本库用户设置一个共同的组的方法，而是强制每个版本库访问进程使用一个健全的umask。对直接访问版本库的用户，你可以使用svn的包裹脚本来首先设置umask 002，然后运行真实的svn客户端程序，你可以为svnserve写相同的脚本，并且增加umask 002命令到Apache自己的启动脚本apachectl中。例如：

```
$ 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。

一旦你跳过了这些障碍，你的版本库一定是可以通过各种可能的手段访问了，这看起来有点凌乱和复杂，但是这个让多个用户分享对一个文件的写权限的问题是一个经典问题，并且经常是没有优雅的解决。

幸运的是，大多数版本库管理员不需要这样复杂的配置，用户如果希望访问本机的版本库，并不是一定要通过file://的URL—他们可以用localhost机器名联系

Apache的HTTP服务器或者是svnserve，协议分别是http://或svn://。为你的Subversion版本库维护多个服务器进程，版本库会变得超出需要的头痛，我们建议你选择最符合你的需要的版本库，并且坚持使用！

svn+ssh://服务器检查列表

让一些用户通过存在的SSH帐户来共享版本库而没有访问许可问题是一件很有技巧的事情，如果你为自己需要在（作为一个管理员）类Unix系统上做的事情感到迷惑，这里是一些快速的检查列表，总结了本小节讨论的事情：

- 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，让版本库完全属于这个组，设置组的权限是读/写。
- 你的用户在访问版本库时需要使用一个健全的umask，确定svnserve（/usr/bin/svnserve或者是任何一个$PATH说明的位置）是一个设置了umask 002和执行真正的svnserve程序的包裹脚本，对svnlook和svnadmin使用相同的措施，或者是使用一个健全的umask运行或者是使用上面说明的包裹。

第 7 章 高级主题

如果你是从头到尾按章节阅读本书，你一定已经具备了使用Subversion客户端执行大多数不同的版本控制操作足够的知识，你理解了怎样从Subversion版本库取出一个工作拷贝，你已经熟悉了通过svn commit和svn update来提交和接收修改，你甚至也经常下意识的使用svn status，无论目的是什么，你已经可以正常使用Subversion了。

但是Subversion的特性集可不只是“一般的版本控制操作”。

本章重点介绍一些Subversion不常用的特性，在这里，我们会讨论Subversion的属性（或者说“元数据”）支持，和如何通过更改运行配置区来改变Subversion的缺省行为方式，我们会描述怎样使用外部定义来指导Subversion从多个版本库得到数据，我们会覆盖一些Subversion分发版本附加的客户端和服务器端的工具的细节。

在阅读本章之前，你一定要熟悉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 2 章 基本概念和第 3 章 指导教程，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！

7.1. 运行配置区

Subversion提供了许多用户可以控制的可选行为方式，许多是用户希望添加到所有的Subversion操作中的选项，为了避免强制用户记住命令行参数并且在每个命令中使用，Subversion使用配置文件，并且将配置文件保存在独立的Subversion配置区。

Subversion配置区是一个双层结构，保存了可选项的名称和值。通常，Subversion配置区是一个保存配置文件的特殊目录（第一层结构），目录中保存了一些标准INI格式的文本文件（文件中的“section”形成第二层结构）。这些文件可以简单用你喜欢的文本编辑器编辑（如Emacs或vi），而且保存了客户端可以读取的指示，用来指导用户的一些行为选项。

7.1.1. 配置区布局

svn命令行客户端第一次执行时，会创建一个用户配置区，在类Unix系统中，配置区位于用户主目录中，名为.subversion。在Win32系统，Subversion创建一个名为Subversion的目录，这个目录通常位于用户配置目录（顺便说一句，通常是一个隐藏目录）的Application Data子目录下。然而，在Win32平台上，此目录的具体位置在不同的系统上是不一样的，由Windows注册表决定。[1] 我们以Unix下的名字.subversion来表示用户配置区。

除了用户配置区，Subversion也提供了系统配置区，通过系统配置区，系统管理员可以为某个机器的所有用户建立缺省配置值。注意系统配置区不会规定强制性

[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。

的策略—每个用户配置区都可以覆盖系统配置区中的配置项，而svn的命令行参数决定了最后的行为。在类Unix的平台上，系统配置区位于/etc/subversion目录下，在Windows平台上，系统配置区位于Application Data（再说一次，是由Windows注册表决定的）的Subversion目录中。与用户配置区不同，svn不会试图创建系统配置区。

目前，Subversion的配置区包含三个文件—两个配置文件（config和servers），和一个INI文件格式的README.txt描述文件。配置文件创建的时候，Subversion的选项都设置为默认值。配置文件中的选项都按功能划分成组，大多数选项还有详细的文字描述注释，说明这些选项的值对Subversion的主要影响。要修改选项，只需用文本编辑器打开并编辑配置文件。如果想要恢复缺省的配置，可以直接删除（或者重命名）配置目录，并且运行一些如svn --version之类的无关紧要的svn命令，一个包含缺省值的新配置目录就会创建起来。

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

除了基于INI文件的配置区，运行在Windows平台的Subversion客户端也可以使用Windows注册表来保存配置数据。注册表中保存的选项名称和值的含义与INI文件中相同，“file/section”在注册表中表现为注册表键树的层级，使得双层结构得以保留下来。

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。

基于注册表的配置项在基于文件的配置项之前解析，所以其配置项的值会被配置文件中相同配置项的值覆盖，换句话说，在Windows系统下配置项的优先级是：

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。

svn命令行客户端不会尝试写Windows注册表，也不会在注册表中创建默认配置区。不过可以使用REGEDIT创建所需的键。此外，还可以创建一个.reg文件，并在文件浏览器中双击这个文件，文件中的数据就会合并到注册表中。

例 7.1. 注册表条目（.reg）样本文件。

```
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-auth-creds"="no"

[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\miscellany]
"#global-ignores"="*.o *.lo *.la #*# .*.rej *.rej .*~ *~ .#* .DS_Store"
"#log-encoding"=""
"#use-commit-times"=""
"#template-root"=""
"#enable-auto-props"=""

[HKEY_CURRENT_USER\Software\Tigris.org\Subversion\Config\tunnels]

[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

本节我们会详细讨论Subversion目前支持的运行配置选项。

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，值则是一个可以包含通配符的字符序列，对应于接收Subversion请求的主机名，称为glob。

```
[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-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
这说明是否在与设置好DAV的服务器通讯时使用网络压缩请求，缺省值是yes（尽管只有在这个功能编译到网络层时压缩才会有效），设置no来关闭压缩，如调试网络传输时。

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 开发者信息。

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
如果你的SSL客户端证书文件是用密码加密的，Subversion会在每次使用证书时请你输入密码，如果你发现这很讨厌（并且不介意把密码存放在servers文件中），你可以设置这个参数为证书的密码，这样就不会再收到密码输入提示了。

7.1.3.2. config

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

6.2.2 节 “客户端凭证缓存”。

store-auth-creds
这个设置与store-passwords相似，不过设置了这个选项将会保存所有认证信息，如用户名、密码、服务器证书，以及其他任何类型的可以缓存的凭证。

helpers小节控制完成Subversion任务的外部程序，正确的选项包括：

editor-cmd
Subversion在提交操作时用来询问用户日志信息的程序，例如使用svn commit而没有指定--message（-m）或者--file（-F）选项。这个程序也会与svn propedit一起使用——一个临时文件跳出来包含已经存在的用户希望编辑的属性，然后用户可以对这个属性进行编辑（见第 7.2 节 “属性”），这个选项的缺省值为空，如果这个选项没有设置，Subversion会依次检查环境变量SVN_EDITOR、VISUAL和EDITOR（这个顺序）来找到一个编辑器命令。

diff-cmd
这个命令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。

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。

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。

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 6.3.3 节 “SSH认证和授权”。

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
当运行svn status命令时，Subversion会和版本化的文件一样列出未版本化的文件和目录，并使用?字符（见see 第 3.5.3.1 节 “svn status”）标记，有时候察看无关的未版本化文件会很讨厌—比如程序编译产生的对象文件—的显示出来。global-ignores选项是一个空格分隔的列表，用来描述Subversion在它们版本化之前不想显示的文件和目录，缺省值是*.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，关于更加细致的控

[2]就是一个大杂烩？

制忽略的项目，见第 7.2.3.3 节 “svn:ignore”。

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UTF8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
通常你的工作拷贝文件会有最后一次被进程访问的时间戳，不管是你自己的编辑器还是用svn子命令。这通常对人们开发软件提供了便利，因为编译系统通常会通过查看时间戳来决定那些文件需要重新编译。

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。

auto-props小节控制Subversion客户端自动设置提交和导入的文件的属性的能力，它可以包含任意数量的键-值对，格式是PATTERN = PROPNAME=PROPVALUE，其中PATTERN是一个文件模式，匹配一系列文件名，此行其它两项为属性和值。如果一个文件匹配多次，会导致有多个属性集；然而，没有手段保障自动属性不会按照配置文件中的顺序应用，所以你可以一个规则“覆盖”另一个。你可以在config文件找到许多自动属性的用法实例。最后，如果你希望开启自动属性，不要忘了设置miscellany小节的enable-auto-props为yes。

7.2. 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。到此为止，Subversion还仅仅表现出一个普通的版本控制理念。但是Subversion并没有就此止步。

作为目录和文件版本化的补充，Subversion提供了对每一个版本化的目录和文件添加、修改和删除版本化的元数据的接口，我们用属性来表示这些元数据。我们可以认为它们是一个两列的表，附加到你的工作拷贝的每个条目上，映射属性名到任意的值。一般来说，属性的名称和值可以是你希望的任何值，限制就是名称必须是可读的文本，并且最好的一点是这些属性也是版本化的，就像你的文本内容文件，你可以像提交文本修改一样修改、提交和恢复属性修改，当你更新时也会接收到别人的属性修改。

Subversion的其他属性

> 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—除了修订版本不是版本化的，参见第 5.1.2 节 “未受版本控制的属性”获得版本化的属性信息。

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，希望你会感到Subversion的属性可以提高你的版本控制体验。

7.2.1. 为什么需要属性？

属性可能会是工作拷贝的有益补充，实际上，Subversion本身使用属性来存放特殊的信息，作为支持特别操作的一种方法，同样，你也可以使用属性来实现自己的目的，当然，你对属性作的任何事情也可以针对普通的版本化文件，但是先考虑下面Subversion使用属性的例子。

假定你希望设计一个网站存放许多数码图片，并且显示他们的标题和时间戳，现在你的图片集经常修改，所以你希望你的网站能够尽量的自动化，这些图片可能非常大，所以根据这个网站的特性，你希望在网站给用户提供图标图像。你可以用传统的文件做这件事，你可以有一个image123.jpg和一个image123-thumbnail.jpg对应在同一个目录，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。很快你的目录树会是一团糟，每个新图片的添加都会成倍的增加混乱。

现在考虑使用Subversion文件的属性来做相同的设置，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 8.2.3 节 “使用C和C++以外的语言”）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。

你怎样（而且如果）使用Subversion完全在你，像我们提到的，Subversion拥有它自己的属性集，我们会在后面的章节讨论，但首先，让我们讨论怎样使用svn的属性处理选项。

7.2.2. 处理属性

svn命令提供一些方法来添加和修改文件或目录的属性，对于短的，可读的属性，最简单的添加方法是在propset子命令里指定正确的名称和值。

```
$ svn propset copyright '(c) 2003 Red-Bean Software' calc/button.c
```

```
property 'copyright' set on 'calc/button.c'
$
```

但是我们已经“吹嘘”过Subversion为属性值提供的灵活性，如果你计划有一个多行的可读文本，甚至是二进制文件的属性值，你通常不希望在命令行里指定，所以propset子命令使用--file（-F）选项来指定一个保存新属性值的文件的名字。

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

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 “config”）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 changes
No changes to property 'copyright' on 'calc/button.c'
$
```

我们也应该注意导，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。举个例子，我们可以：

```
$ svn propset copyright '(c) 2002 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) 2003 Red-Bean Software
```

还有一个proplist变种命令会列出所有属性的名称和值，只需要设置--verbose（-v）选项。

```
$ svn proplist --verbose calc/button.c
Properties on 'calc/button.c':
  copyright : (c) 2003 Red-Bean Software
  license : ================================================================
Copyright (c) 2003 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.
…
```

最后一个与属性相关的子命令是propdel，因为Subversion允许属性值为空，所有不能用propedit或者propset命令删除一个属性。举个例子，这个命令不会产生预期的效果：

```
$ svn propset license '' calc/button.c
property 'license' set on 'calc/button.c'
$ svn proplist --verbose calc/button.c
Properties on 'calc/button.c':
  copyright : (c) 2003 Red-Bean Software
  license :
$
```

你需要用propdel来删除属性，语法与其它与属性相关命令相似：

```
$ svn propdel license calc/button.c
property 'license' deleted from ''.
$ svn proplist --verbose calc/button.c
Properties on 'calc/button.c':
  copyright : (c) 2003 Red-Bean Software
$
```

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。

修改修订版本的属性

还记的这些未版本化的属性？你也可以使用svn命令修改这些属性。只需要添加--revprop命令参数，并且说明希望修改属性的修订版本。因为修订版本是全局的，你不需要指定一个路径，只要你已经位于你希望修改属性的工作拷贝路径，举个例子，你希望修改一个存在版本的提交日志信息。[3]

```
$ svn propset svn:log '* button.c: Fix a compiler warning.' -r11 --revprop
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 5.2.1 节 “钩子脚本”）。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。

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，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) 2003 Red-Bean Software

$
```

注意status子命令显示的M在第二列而不是在第一列，这是因为我们修改了calc/button.c的属性，而不是它的文本内容，如果我们都修改了，我们也会看到M出现在第一列（见第 3.5.3.1 节 “svn status”）。

[3]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。

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
 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
prop 'linecount': user set to '1256', but update set to '1301'.
$
```

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。

你也许已经注意到了Subversion在显示属性时的非标准方式。你还可以运行svn diff并且重定向输出来产生一个有用的补丁文件，patch程序会忽略属性补丁—作为规则，它会忽略任何不理解的噪音。很遗憾，这意味着完全应用svn diff产生的补丁时，任何属性修改必须手工实施。

就象你看到的，属性修改的出现并没有对典型的Subversion工作流有显著的影响，更新工作拷贝、检查文件和目录的状态、报告所作的修改和提交修改到版本库等等的工作方式完全与属性的存在与否无关。svn程序有一些额外的子命令用来进行属性修改，但那是唯一显而易见不对称的命令。

7.2.3. 特别属性

Subversion没有关于属性的特殊政策—你可以通过它们实现自己的目的。Subversion只是要求你不要使用svn:开头的命名空间作为属性名，这是Subversion自己使用的命名空间。实际上，Subversion定义了某些特殊的属性，这些属性对它们所附加的文件和目录有特殊的影响。在本小节，我们会解开这个

谜团，并且描述这些属性怎样让你的生活更加容易。

7.2.3.1. svn:executable

svn:executable属性用来控制一个版本化的文件自动执行文件权限设定，这个属性没有特定的值—它只是说明一个Subversion可以保留的文件权限的期望值，删除这个属性会恢复操作系统对这些权限的完全控制。

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，这些位通常是关闭的，必须由用户显式的指定，这意味着你必须改变文件的执行位，然后更新你的工作拷贝，燃火如果你的文件成为更新的一部分，它的执行位会被关闭，所以Subversion提供了svn:executable这个属性来保持打开执行位。

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。

7.2.3.2. svn:mime-type

svn:mime-type属性为Subversion的许多目的服务，除了保存一个文件的多用途网际邮件扩展（MIME）分类以外，这个属性值也描述了一些Subversion自己使用的行为特性。

举个例子，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通常在更新到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，但是对于保存二进制数据的文件，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，任何时候你在本地修改的一个二进制文件有了更新，你的文件会被重命名为.orig为扩展名，然后Subversion保存一个新的工作拷贝文件，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。

另外，如果svn:mime-type属性被设置，Subversion的Apache模块会使用这个值来在HTTP头里输入Content-type:，这给了web浏览器如何显示一个文件提供了至关重要的线索。

7.2.3.3. svn:ignore

这个svn:ignore属性保存了一个Subversion特定操作忽略的文件模式列表，或许这个是最常用的属性，它可以与global-ignores运行配置选项配合使用（见第 7.1.3.2 节 “config”）来过滤svn status、svn add和svn import命令中操作的未版本化文件。

svn:ignore背后的基本原理很容易解释，Subversion不会假定工作拷贝中的所有文件或子目录是版本控制的一部分，资源必须被显式的使用svn add或者svn import放到Subversion的管理控制之下，作为结果，经常有许多工作拷贝的资源并没有版本化。

[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。

现在，svn status命令会的显示会包括所有未纳入版本控制且没有用global-ignores（或是内置的缺省值）过滤掉的文件和子目录，这样可以帮助用户查看是否忘记了把某些自愿加入到版本控制。

但是Subversion不可能猜测到每个需要忽略的资源的名字，但是也有一些资源是所有特定版本库的工作拷贝都有忽略的，强制版本库的每个用户来添加这些模式到他们的运行配置区域不仅仅是一个负担，也会与用户取出的其他工作拷贝配置需要存在潜在的冲突。

解决方案是保存的忽略模式必须对出现在给定目录和这个目录本身的资源是独立的，一个常见的例子就是一个未版本化资源对一个目录来说是唯一的，会出现在那个位置，包括程序编译的输出，或者是—用一个本书的例子—DocBook的文件生成的HTML、PDF或者是PostScript文件。

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，但是Subversion不会识别重置回到没有忽略模式的!模式的使用。

为了这个目的，svn:ignore属性是解决方案，它的值是一个多行的文件模式集，一行一个模式，这个属性已经设置到这个你希望应用模式的目录。[5] 举个例子，你的svn status有如下的输出：

```
$ 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
```

在这个例子里，你对button.c文件作了一些属性修改，但是你的工作拷贝也有一

[5]这个模式对那个目录是严格的—不会迭代的应用到子目录。

些未版本化的文件：你从源代码编译的最新的计算器程序是data.c，一系列调试输出日志文件，现在你知道你的编译系统会编译生成计算器程序。[6] 就像你知道的，你的测试组件总是会留下这些调试日志，这对所有的工作拷贝都是一样的，不仅仅使你的。你也知道你不会有兴趣在svn status命令中显示这些信息，所以使用svn propedit svn:ignore calc来为calc目录增加一些忽略模式，举个例子，你或许会添加如下的值作为svn:ignore属性：

```
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，这些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删除了，你留下了更加感兴趣的项目—如你忘记添加到版本控制的源代码文件。

如果想查看被忽略的文件，可以设置Subversion的--no-ignore选项：

```
$ svn status --no-ignore
 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
```

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。

7.2.3.4. svn:keywords

Subversion具备有添加关键字的能力—一些有用的，关于版本化的文件动态信息的片断—不必直接添加到文件本身。关键字通常会用来描述文件最后一次修改的

[6]这不是编译系统的基本功能吗？

一些信息，因为这些信息每次都有改变，更重要的一点，这是在文件修改之后，除了版本控制系统，对于任何处理完全保持最新的数据都是一场争论，作为人类作者，信息变得陈旧是不可避免的。

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，同时会改变描述这部分细细的部分，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，在你的文档需要放置这个关键字的地方放置一个keyword anchor，这个anchor只是一个格式为$KeywordName$字符串。

所有作为anchor出现在文件里的关键字是大小写敏感的：为了关键字的扩展，你必须使用正确的按顺序大写。你必须考虑svn:keywords的属性值也是大小写敏感—特定的关键字名会忽略大小写，但是这个特性已经被废弃了。

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: 2002-07-22 21:42:37 -0700 (Mon, 22 Jul 2002) $，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的版本的完全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 2002-07-28 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2002年7月28日，作者是sally。

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，你可以撰写一个文档 [7] 关于如何使用关键字，你希望Subversion不会替代你漂亮的关于不需要替换的关键字anchor实例！

为了告诉Subversion是否替代某个文件的关键字，我们要再次求助于属性相关的

[7]… 或者可能是一本书的一个小节 …

子命令，当svn:keywords属性设置到一个版本化的文件，这些属性控制了那些关键字将会替换到那个文件。这个值是空格分隔的前面列表的名称或是别名列表。

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。

> 关键字和虚假的差异
>
> 用户可见的关键字替换会让你以为每一个具有此特性的文件的每个版本都会与前一个版本至少在关键字替换的地方不同，但是实际上并不是如此，当用svn diff检查本地修改时，或者是在使用svn commit传输修改之前，Subversion不会“取消替换”任何上次替换的关键字，结果就是版本库保存的文件只保存用户实际做的修改。

在你提交了属性修改后，Subversion会立刻更新你的工作文件为新的替代文本，你将无法找到$LastChangedDate$的关键字anchor，你会看到替换的结果，这个结果也保存了关键字的名字，与美元符号（$）绑定在一起，而且我们预测的，Rev关键字不会被替换，因为我们没有要求这样做。

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: 2002-07-22 21:42:37 -0700 (Mon, 22 Jul 2002) $
$Rev$
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。

7.2.3.5. svn:eol-style

不像我们说过的版本化文件的svn:mime-type属性，Subversion假定这个文件保存了可读的数据，一般来讲，Subversion因为这个属性来判断一个文件是否可以用上下文区别报告，否则，对Subversion来说只是字节。

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SCII控制字符—回车（CR）和一个移行（LF）。Unix软件，只使用一个LF来表示一个行的结束。

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标志的敏感会让在各个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。

但实践中，许多常用的工具不会正确的读取外来的EOL标志，或者是将文件的行结束转化为本地的样式，如果是前者，他需要一个外部的转化工具（如dos2unix或是他的伴侣，unix2dos）来准备需要编辑的文件。后一种情况不需要额外的准备工作，两种方法都会造成文件会与原来的文件在每一行上都不一样！在提交之前，用户有两个选择，或者选择用一个转化工具恢复文件的行结束样式，或者是简单的提交文件—包含新的EOL标志。

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，判断那个文件是通过正常的方式修改的会是一件复杂的工作，bug在那一行修正的？那一行引入了语法错误？

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，Subversion使用它来判断针对行结束样式执行何种特殊的操作，而不会因为多种操作系统的每次提交发生震荡。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一个文件有svn:eol-style的属性设置为native，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
这会导致文件使用CR字符作为EOL标志，不管使用何种操作系统。这种行结束样式不是很常见，它用在一些老的苹果机（Subversion不会运行的机器上）。

7.2.3.6. svn:externals

svn:externals属性保存了指导Subversion从一个或多个取出的工作拷贝移出目录的指示，关于这个关键字的更多信息，见第 7.4 节 “外部定义”。

7.2.3.7. svn:special

svn:special是唯一一个不是用户直接设置和修改的svn:属性，当“特别的”对象如一个对象链接计划加入到版本库，Subversion会自动设置这个属性。版本库像普通文件一样保存svn:special对象，然而，当一个客户端在检出和更新操作时看到这个属性时，就会翻译这个文件的内容，并且将文件转化为特殊类型的对象，在Subversion1.1中，只有版本化的符号链接有这个属性附加，但在以后的版本中其它特殊的节点也有可能使用这个属性。

注意：Windows客户端不会有符号链接，因此会忽略含有svn:special声明为符号链的文件，在Windows，用户会以一个工作拷贝中的版本化的文件作为结束。

7.2.4. 自动属性设置

属性是Subversion一个强大的特性，成为本章和其它章讨论的许多Subversion特性的关键组成部分—文本区别和合并支持、关键字替换、新行的自动转换等等。但是为了从属性得到完全的利益，他们必须设置到正确的文件和目录。不幸的是，在日常工作中很容易忘记这一步工作，特别是当没有设置属性不会引起明显的错误时（至少相对与未能添加一个文件到版本控制这种操作），为了帮助你在需要添加属性的文件上添加属性，Subversion提供了一些简单但是有用的特性。

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会运行一个基本探测来检查文件是包括了可读还是不可读的内容，如果Subversion猜测错误，或者是你希望使用svn:mime-type属性更精确的设置—或许是image/png或者application/x-shockwave-flash—你可以一直删除或编辑那个属性。

Subversion也提供了自动属性特性，允许你创建文件名到属性名称与值影射，这个影射在你的运行配置区域设置，它们会影响添加和导入操作，而且不仅仅会覆盖Subversion所有缺省的MIME类型判断操作，也会设置额外的Subversion或者自定义的属性。举个例子，你会创建一个影射文件说在任何时候你添加了一个JPEG

文件—一些符合*.jpg的文件—Subversion一定会自动设置它们的svn:mime-type属性为image/jpeg。或者是任何匹配*.cpp的文件，必须把svn:eol-style设置为native，并且svn:keywords设置为Id。自动属性支持是Subversion工具箱中属性相关最垂手可得的工具，见第 7.1.3.2 节 “config”来查看更多的配置支持。

7.3. Peg和实施修订版本

文件和目录的拷贝、移动和改名能力可以让我们可以删除一个对象，然后在同样的路径添加一个新的—这是我们在电脑上对文件和目录经常作的操作，我们认为这些操作都是理所当然的。Subversion很乐意你认为这些操作已经赋予给你，Subversion的文件管理操作是这样的解放，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。

Subversion可以非常聪明的注意到一个对象的版本历史变化包括一个“地址改变”，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，清除了你删除的东西，然后你添加的不是同样的东西，它们仅仅是有同样的路径，我们会把它叫做/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。

由于移动，版本化资源历史会变得非常扭曲。举个例子，你会有一个目录叫做concept，保存了一些你用来试验的初生的软件项目，最终，这个项目变得足够成熟，说明这个注意确实需要一些翅膀了，所以你决定给这个项目一个名字。[8] 假定你叫你的软件为Frabnaggilywort，此刻，有必要把你的目录命名为反映项目名称的名字，所以concept改名为frabnaggilywort。生活还在继续，Frabnaggilywort发布了1.0版本，并且被许多希望改进他们生活的分散用户天天使用。

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。

当这样的情景发生时，指导Subversion工作在重新使用的路径上的尝试就像指导一个芝加哥西郊的乘客驾车到东面的罗斯福路并且左转到主大道。仅仅20分钟，你可以穿过惠顿、格伦埃林何朗伯德的“主大道”，但是它们不是一样的街道，我们的乘客—和我们的Subversion—需要更多的细节来做正确的事情。

[8] “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

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，这是一个提供给Subversion的一个区别一个独立历史线路的单独目的修订版本，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”（@）。

但是我们在本书多次提到的--revision (-r)到底是什么？修订版本（或者是修订版本集）叫做实施的修订版本（或者叫做实施的修订版本范围），一旦一个特定历史线路通过一个路径和peg修订版本指定，Subversion会使用实施的修订版本执行要求的操作。类似的，为了指出这个到我们芝加哥的道路，如果我们被告知到惠顿主大道606号，[9] 我们可以把“主大道”看作路径，把“惠顿”当作我们的peg修订版本。这两段信息确认了我们可以旅行（主大道的北方或南方）的唯一路径，也会保持我们不会在前前后后寻找目标时走到错误的主大道。现在我们把“606 N.”作为我们实施的修订版本，我们精确的知道到哪里。

当使用peg和实施修订版本来查找我们需要工作文件时，Subversion会执行一个很直接的算法。首先，找到与peg修订版本相关的路径坐落于版本库的那个修订版本，Subversion开始从那里开始向后查询这个对象的历史前辈。每个前辈表示这个对象的以前的一个版本，每一个版本的对象都保存了自己被创建的修订版本和路径，所以通过前辈集，Subversion可以注意到哪个版本是这个实施修订版本最年轻的版本，如果是，可以将实施修订版本影射到前辈创建的路径/创建修订版本对。算法在所有的实施修订版本影射到真实的对象位置后，或者是没有更多的前辈时完成，就是任何未影射的实施修订版本已经标记为不符合操作的对象。

也就是说很久以前我们创建了我们的版本库，在修订版本1添加我们第一个concept目录，并且在这个目录增加一个IDEA文件与concept相关，在几个修订版本之后，真实的代码被添加和修改，我们在修订版本20，修改这个目录为frabnaggilywort。通过修订版本27，我们有了一个新的概念，所以一个新的concept目录用来保存这些东西，一个新的IDEA文件来描述这个概念，然后经过5年20000个修订版本，就像他们都有一个非常浪漫的历史。

现在，一年之后，我们想知道IDEA在修订版本1时是什么样子，但是Subversion需要知道我们是想询问当前文件在修订版本1时的样子，还是希望知道concepts/IDEA在修订版本1时的那个文件？确定这些问题有不同的答案，并且因为peg修订版本，你可以用两种方式询问。为了知道当前的IDEA文件在旧版本1的样子，我们可以运行：

```
$ svn cat -r 1 concept/IDEA
subversion/libsvn_client/ra.c:775: (apr_err=20014)
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：

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

```
$ svn cat -r 1 concept/IDEA@BASE
subversion/libsvn_client/ra.c:775: (apr_err=20014)
svn: Unable to find repository location for 'concept/IDEA' in revision 1
```

当执行时会有预料中的结果，当应用到工作拷贝路径时，Peg修订版本通常缺省值是BASE（在当前工作拷贝现在的修订版本），当应用到URL时，缺省值是HEAD。

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.
```

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@20
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，几乎所有的人们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。

7.4. 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 7.2.1 节 “为什么需要属性？”）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子目录和完全有效的Subversion版本库URL的列表（相对于设置属性的版本化目录）。

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

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。

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
…
```

如果你希望修改外部定义，你可以使用普通的属性修改子命令，当你提交一个svn:externals属性修改后，当你运行svn update时，Subversion会根据修改的外部定义同步检出的项目，同样的事情也会发生在别人更新他们的工作拷贝接受你的外部定义修改时。

svn status命令也认识外部定义，会为外部定义的子目录显示X状态码，然后迭代这些子目录来显示外部项目的子目录状态信息。

Subversion目前对外部定义的支持可能会引起误导，首先，一个外部定义只可以指向目录，而不是文件。第二，外部定义不可以指向相对路径（如../../skins/myskin）。第三，同过外部定义创建的工作拷贝与主工作拷贝没有连接，所以举个例子，如果你希望提交一个或多个外部定义的拷贝，你必须在这些工作拷贝显示的运行svn commit—对主工作拷贝的提交不会迭代到外部定义的部分。

另外，因为定义本身使用绝对路径，移动和拷贝路径他们附着的路径不会影响他们作为外部的检出（尽管相对的本地目标子目录会这样，当然，根据重命名的目录）。这看起来有些迷惑—甚至让人沮丧—在特定情形。举个例子，如果你在/trunk开发线对一个目录使用外部定义，指向同一条线上的其他区域，然后使用svn copy把分支开发线拷贝到/branches/my-branch这个新位置，这个项目新分支的外部定义仍然指向/trunk版本化资源。另外，需要意识到如果你需要一个重新规划你的工作拷贝的父目录（使用svn switch --relocate），外部定义不会重新选择父目录。

7.5. 卖主分支

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。

举个例子，软件开发者会工作在一个使用第三方库的应用，Subversion恰好是和Apache的Portable Runtime library（见第 8.2.1 节 “Apache可移植运行库”）有这样一个关系。Subversion源代码依赖于APR库来实现可移植需求。在Subversion的早期开发阶段，项目紧密地追踪APR的API修改，经常在库代码的“流血的边缘”粘住，现在APR和Subversion都已经成熟了，Subversion只尝试同步APR的经过良好测试的，稳定的API库。

现在，如果你的项目依赖于其他人的信息，有许多方法可以用来尝试同步你的信息，最痛苦的，你可以为项目所有的贡献者发布口头或书写的指导，告诉他们确信他们拥有你们的项目需要的特定版本的第三方信息。如果第三方信息是用Subversion版本库维护，你可以使用Subversion的外部定义来有效的“强制”特定的版本的信息在你的工作拷贝的的位置（见第 7.4 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。

现在你会面对一个有趣的情形，你的项目可以用某种脱节的样式保持它关于第三方数据自己的修改，如使用补丁文件或者是完全的可选版本的文件和目录。但是这很快会成为维护的头痛的事情，需要一种机制来应用你对第三方数据的自定义修改，并且迫使在第三方数据的后续版本重建这些修改。

这个问题的解决方案是使用卖主分支，一个卖主分支是一个目录树保存了第三方实体或卖主的信息，每一个卖主数据的版本吸收到你的项目叫做卖主drop。

卖主分支提供了两个关键的益处，第一，通过在我们的版本控制系统保存现在支持的卖主drop，你项目的成员不需要指导他们是否有了正确版本的卖主数据，他们只需要作为不同工作拷贝更新的一部份，简单的接受正确的版本就可以了。第二，因为数据存在于你自己的Subversion版本库，你可以在恰当的位置保存你的自定义修改—你不需要一个自动的（或者是更坏，手工的）方法来交换你的自定义行为。

7.5.1. 常规的卖主分支管理过程

管理卖主分支通常会像这个样子，你创建一个顶级的目录（如/vendor）来保存卖主分支，然后你导入第三方的代码到你的子目录。然后你将拷贝这个子目录到主要的开发分支（例如/trunk）的适当位置。你一直在你的主要开发分支上做本地修改，当你的追踪的代码有了新版本，你会把带到卖主分支并且把它合并到你的/trunk，解决任何你的本地修改和他们的修改的冲突。

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。我们从卖主分支的初始创建开始，并且导入卖主drop，我们会把每株分支目录叫做libcomplex，我们的代码drop会进入到卖主分支的子目录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卖主数据的拷贝版本。

```
$ 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—现在包括了第一个卖主drop的拷贝—我们开始自定义libcomplex的代码，我们知道，我们的libcomplex修改版本是已经与我们的计算器程序完全集成。[10]

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，我们取出一个我们卖主分支的拷贝，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current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。

[10]而且完全没有bug，当然！

我们的current分支现在保存了新的卖主drop，我们为这个新的版本创建一个新的标签（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      \
            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。

但是事情总不是这样简单，实际上源文件在不同的版本间的移动是很常见的，这种过程复杂性可以确保我们的修改会一直对新的版本代码有效，可以很快使形势退化到我们需要在新版本手工的重新创建我们的自定义修改。一旦Subversion知道了给定文件的历史—包括了所有以前的位置—合并到新版本的进程就会很简单，但是我们需要负责告诉Subversion卖主drop之间源文件布局的改变。

7.5.2. svn_load_dirs.pl

不仅仅包含一些删除、添加和移动的卖主drops使得升级第三方数据后续版本的过程变得复杂，所以Subversion提供了一个svn_load_dirs.pl脚本来辅助这个过程，这个脚本自动进行我们前面提到的常规卖主分支管理过程的导入步骤，从而使得错误最小化。你仍要负责使用合并命令合并第三方的新 版本数据合并到主要开发分支，但是svn_load_dirs.pl帮助你快速到达这一步骤。

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利用三个强制的参数，第一个参数是Subversion工作的基本目录URL，第二个参数在URL之后—相对于第一个参数—指向当前的卖主分支将会导

入的目录，最后，第三个参数是一个需要导入的本地目录，使用前面的例子，一个典型的svn_load_dirs.pl调用看起来如下：

```
$ 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，需要制定一个标签名。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
…
```

当你运行svn_load_dirs.pl，它会检验你的存在的“current”卖主drop，并且与提议的新卖主drop比较，在这个琐碎的例子里，没有文件只出现在一个版本里，脚本执行新的导入而不会发生意外。然而如果版本之间有了文件布局的区别，svn_load_dirs.pl会询问你如何解决这个区别，例如你会有机会告诉脚本libcomplex版本1.0的math.c文件在1.1已经重命名为arithmetic.c，任何没有解释为移动的差异都会被看作是常规的添加和删除。

这个脚本也接受单独配置文件用来为添加到版本库的文件和目录设置匹配正则表达式的属性。配置文件通过svn_load_dirs.pl的-p命令行选项指定，这个配置文件的每一行都是一个空白分割的两列或者四列值：一个Perl样式的正则表达式来匹配添加的路径、一个控制关键字（break或者是cont）和可选的属性名和值。

```
\.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
```

对每一个添加的路径，会按照顺序为匹配正则表达式的文件配置属性，除非控制标志是break（意味着不需要更多的路径匹配应用到这个路径）。如果控制说明是cont—continue的缩写—然后匹配工作会继续到配置文件的下一行。

任何正则表达式，属性名或者属性值的空格必须使用单引号或者双银行环绕，你可以使用反斜杠（\）换码符来回避引号，反斜杠只会在解析配置文件时回避引号，所以不要保护对正则表达式不需要的其它字符。

7.6. 本地化

本地化是让程序按照地区特定方式运行的行为，如果一个程序的格式、数字或者是日期是你的本地方式，或者是打印的信息（或者是接受的输入）是你本地的语言，这个程序被叫做已经本地化了，这部分描述了针对本地化的Subversion的步骤。

7.6.1. 理解地区

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"
```

输出是一个本地相关的环境变量和它们的值，在这个例子里，所有的变量设置为缺省的C地区，但是用户可以设置这些变量为特定的国家/语言代码组合。举个例子，如果有人设置LC_TIME变量为fr_CA，然后程序会知道使用讲法语的加拿大期望的格式来显示时间和日期信息。如果一个人会设置LC_MESSAGES变量为zh_TW，程序会知道使用繁体中文显示可读信息。如果设置LC_ALL的效果同分别设置所有的位置变量为同一个值有相同的效果。LANG用来作为没有设置地区变量的缺省值，为了查看Unix系统所有的地区列表，运行locale -a命令。

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。

7.6.2. Subversion对地区的支持

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: Ausgabe des Inhaltes der angegebenen Dateien oder URLs
```

```
Aufruf: cat ZIEL...
…
```

这个行为在Unix和Windows上同样工作，注意，尽管有时你的操作系统支持某个地区，Subversion客户端可能不能讲特定的语言。为了制作本地化信息，志愿者可以提供各种语言的翻译。翻译使用GNU gettext包编写，相关的翻译模块使用.mo作为后缀名。举个例子，德国翻译文件为de.mo。翻译文件安装到你的系统的某个位置，在Unix它们会在/usr/share/locale/，而在Windows它们通常会在Subversion安装的\share\locale\目录。一旦安装，一个命名在程序后面的模块会为此提供翻译。举个例子，de.mo会最终安装到/usr/share/locale/de/LC_MESSAGES/subversion.mo，通过查看安装的.mo文件，我们可以看到Subversion支持的语言。

第二种支持地区设置的方式包括svn怎样解释你的输入，版本库使用UTF-8保存了所有的路径，文件名和日志信息。在这种情况下，版本库是国际化的—也就是版本库准备接受任何人类的语言。这意味着，无论如何Subversion客户端要负责发送UTF-8的文件名和日志信息到版本库，为此，必须将数据从本地位置转化为UTF-8。

举个例子，你创建了一个文件叫做caffè.txt，然后提交了这个文件，你写的日志信息是“Adesso il caffè è più forte”，文件名和日志信息都包含非ASCII字符，但是因为你的位置设置为it_IT，Subversion知道把它们作为意大利语解释，在发送到版本库之前，它用一个意大利字符集转化数据为UTF-8。

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 recode string.
> ```
>
> 这个信息是神秘的，但是通常会发生在Subversion客户端从版本库接收到一个UTF-8串，但是字符不能转化为当前的地区文字，举个例子，如果你的地区设置是en_US，但是一个写作者使用日本文件名提交，你会在svn update接受文件时会看到这个错误。
>
> 解决方案或者是设置你的地区为可以表示即将到来的UTF-8数据，或者是修改版本库的文件名或信息。（不要忘记和你的合作者拍拍手—项目必须首先决定通用的语言，这样所有的参与者会使用相同的地区设置。）

7.7. Subversion版本库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:///path/to/repos
…
$ svn checkout file://localhost/path/to/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:/path/to/repos
…
C:\> svn checkout "file:///X|/path/to/repos"
…
```

在第二个语法，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，注意URL使用普通的斜线而不是Windows本地（不是URL）的反斜线。

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"
```

…Subversion会回避这些不安全字符，并且会像你输入了这些字符一样工作：

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

如果URL包含空格，一定要使用引号，这样你的脚本才会把它做一个单独的svn参数。

第 8 章 开发者信息

Subversion是一个开源的软件项目，使用Apache样式的软件许可证。这个项目由位于加利福尼亚的CollabNet, Inc.软件开发公司资助。这个关于Subversion开发的社区一直欢迎新成员贡献自己的时间和注意力。鼓励志愿者做他们能做的任何帮助，不管是发现和诊断bug，精炼已存的代码还是补充新的特性。

本章是为那些希望实际参与源代码编写来帮助Subversion不断进步的人们准备的。我们要知道，在这里我们会涉及到许多软件内在的细节，在开发Subversion本身—或利用Subversion库开发全新工具时—所用到的许多核心技术。如果你无法预测你是否会以这种层级参与到这个软件中来，那么也可以随意跳过这一章，而你作为一个Subversion用户的体验不会受到任何影响。

8.1. 分层的库设计

Subversion有一个模块化的设计，通过一套C库[1]来实现。每一个库都有一套定义良好的目标与接口，据称，大部分模块都属于三层中的某一层—版本库层、版本库访问（RA）层或是客户端层。我们很快就会考察这些层，但首先让我们看一下表 8.1 “Subversion库的摘要目录”中的有关于Subversion库的摘要目录，为了一致性，我们将通过它们的无扩展Unix库名（例如libsvn_fs、libsvn_wc和mod_dav_svn）来引用它们。

表 8.1. Subversion库的摘要目录

库	描述
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_fs	Subversion文件系统库
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_dav	WebDAV版本库访问模块
libsvn_ra_local	本地版本库访问模块
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

[1]译者：这里的“库（library）”指函数库，与文中大量出现的“版本库（Repository）”不同，一般情况下，作为独词出现的“库”应属于前者。

库	描述
	Apache模块

单词“各色各样的”只在列表表 8.1 “Subversion库的摘要目录”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。

模块化的另一个好处是我们有能力去构造一个全新的，能够完全实现相同API功能的库，以替换整个给定的模块，而又不会影响基础代码。在某种意义上，Subversion已经这样做了。libsvn_ra_dav、libsvn_ra_local和libsvn_ra_svn all都实现了相同的接口，三者均与版本库层进行通讯—libsvn_ra_dav和libsvn_ra_svn通过网络，而libsvn_ra_local则是直接连接。

客户端设计本身就给模块化设计理念增色不少，Subversion目前只是附带了一个命令行方式的客户端，但已经出现了一些由第三方开发的GUI客户端程序，这些GUI客户端程序全都使用了与原装命令行客户端程序相同的API。为了开发一个实用的Subversion客户端程序，对于绝大部分功能，仅使用Subversion的libsvn_client库就够了（见第 8.1.3 节 “客户端层”）。

8.1.1. 版本库层

当提到Subversion版本库层时，我们通常会讨论两个库—版本库（函数）库和文件系统（函数）库。这两个库为你的版本控制数据的各个修订版本提供了存储和报告机制，该层通过版本库访问层连接到客户层，而且，从Subversion用户的角度看，这是资料存储过程中的“链接的另一端”。

Subversion文件系统通过libsvn_fs API来访问，它并不是一个安装在操作系统之上的内核级的文件系统（例如Linux ext2或NTFS），而是一个虚拟文件系统。它并未将“文件”和“目录”保存为真实的文件和目录（也就是用你熟知的shell程序可以浏览的那种），而是采用了一种抽象的后端存储方式，这个后端存储方式有两种—一个是Berkeley DB数据库环境，另一个是普通文件表示。（要了解更多关于版本库后端的信息，请看第 5.1.3 节 “版本库数据存储”）。除此之外，开发社区也非常有兴趣考虑在Subversion的未来版本 中提供某种使用其它后端数据库系统的能力，也许是开放式数据库连接（ODBC）的机制。

libsvn_fs支持的文件系统API包含了所有其他文件系统的功能：你可以创建和删除文件和目录、拷贝和移动、修改文件内容等等。它也包含了一些不太常用的特性，如对任意文件和目录添加、修改和删除元数据（“properties”）的能力。此外，Subversion文件系统是一个版本化的文件系统，意味着你修改你的目录树时，Subversion会记住修改以前的样子。等等，可以回到所有初始化版本库之后（且仅仅之后）的版本。

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。

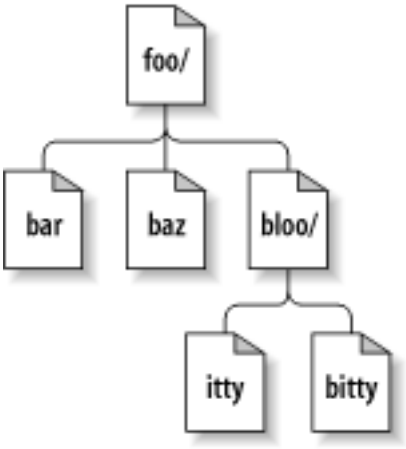
一旦你提交了你的事务，你的文件系统修改就会永久的作为历史保存起来，每个这样的周期会产生一个新的树，所有的修订版本都是永远可以访问的一个不变的快照。

> 事务其它
>
> Subversion的事务概念，特别是在libsvn_fs中的数据库附近的代码，很容易与低层提供支持的数据库事务混淆。两种类型事务都提供了原子和隔离操作，换句话说，事务给你能力可以用“全部或者没有”样式执行一系列的动作—所有的动作都完全成功，或者是所有的没有发生—而且不会干扰别人操作数据。
>
> 数据库事务通常围绕着一些对数据库本身的数据修改相关的小操作（如修改表行的内容），Subversion是更大范围的事务，围绕着一些高一级的操作，如下一个修订版本文件系统的一组文件和目录的修改。如果这还不是很混乱，考虑这个：Subversion在创建Subversion事务（所以如果Subversion创建事务失败，数据库会看起来我们从来没有尝试创建）时会使用一个数据库事务！
>
> 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）。

大多数文件系统接口提供的功能作为一个动作发生在一个文件系统路径上，也就是，从文件系统的外部，描述和访问文件和目录独立版本的主要机制是经过如/foo/bar的路径，就像你在喜欢的shell程序中定位文件和目录。你通过传递它们的路径到相应的API功能来添加新的文件和目录，查询这些信息也是同样的机制。

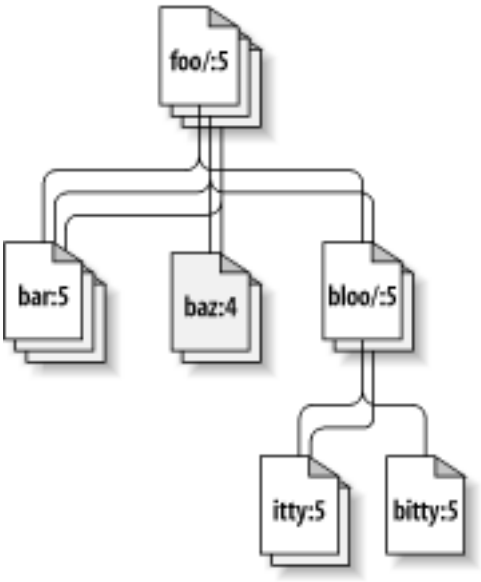
不像大多数文件系统，尽管，一个单独的路径不足以在Subversion定位一个文件或目录，可以把目录树看作一个二维的系统，一个节点的兄弟代表了一种从左到右的动作，并且递减到子目录是一个向下的动作，图 8.1 “二维的文件目录”展示了一个典型的树的形式。

图 8.1. 二维的文件目录



当然，Subversion文件系统有一个其它文件系统所没有的第三维—时间！ [2] 在一个文件系统接口，几乎所有的功能都有个路径参数，也期望一个root参数。svn_fs_root_t参数不仅描述了一个修订版本或一个Subversion事务（通常正好是一个修订版本），而且提供了用来区分修订版本32的/foo/bar和修订版本98在同样路径的三维上下文环境。图 8.2 “版本时间—第三维！”展示了修订版本历史作为添加的纬度进入到Subversion文件系统领域。

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。

[2]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。

Subversion版本库库（libsvn_repos）是文件系统功能的一个基本包裹库，这个库负责创建版本库布局，确定底层的文件系统已经初始化等等。Libsvn_repos也实现了一组钩子—当特定动作发生时版本库执行的脚本。这些脚本用来通知，授权或者任何版本库管理员期望的目的。版本库库提供的这些功能和小工具与版本化的文件系统关系不大，所以放到了自己的库里。

希望使用libsvn_repos的API的开发者会发现它不是文件系统的一个完全包裹，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，在将来，别的事件也将会使用版本库API包裹。所有其它的文件系统交互会直接通过libsvn_fs的API发生。

举个例子，这里是使用版本库和文件系统接口创建文件系统新修订版本的代码块，新版本包括添加一个新目录。注意这个例子（和其它本书中的代码），这个SVN_ERR宏只是简单的检查是否有一个非成功的错误从包裹的函数中返回，如果存在就会返回错误。

例 8.1. 使用版本库层

```
/* Create a new directory at the path NEW_DIRECTORY in the Subversion
   repository located at REPOS_PATH.  Perform all memory allocation in
   POOL.  This function will create a new revision for the addition of
   NEW_DIRECTORY.  */
static svn_error_t *
make_new_directory (const char *repos_path,
                    const char *new_directory,
                    apr_pool_t *pool)
{
  svn_error_t *err;
  svn_repos_t *repos;
  svn_fs_t *fs;
  svn_revnum_t youngest_rev;
  svn_fs_txn_t *txn;
  svn_fs_root_t *txn_root;
  const char *conflict_str;

  /* Open the repository located at REPOS_PATH.  */
  SVN_ERR (svn_repos_open (&repos, repos_path, pool));

  /* Get a pointer to the filesystem object that is stored in
     REPOS.  */
  fs = svn_repos_fs (repos);

  /* Ask the filesystem to tell us the youngest revision that
     currently exists.  */
```

```
  SVN_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 later commit rejected as conflicting if we
     always try to make our changes against a copy of the latest snapshot
     of the filesystem tree.  */
  SVN_ERR (svn_fs_begin_txn (&txn, fs, youngest_rev, pool));

  /* Now that we have started a new Subversion transaction, get a root
     object that represents that transaction.  */
  SVN_ERR (svn_fs_txn_root (&txn_root, txn, pool));

  /* Create our new directory under the transaction root, at the path
     NEW_DIRECTORY.  */
  SVN_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);
    }

  /* Return the result of the attempted commit to our caller.  */
  return err;
}
```

在前面的代码片断中，同时调用了版本库和文件系统接口，我们可以正像这样简单的用svn_fs_commit_txn提交事务。但是文件系统的API对版本库库的钩子一无所知，如果你希望你的Subversion版本库在每次提交一个事务时自动执行一些非Subversion的任务（例如，给开发者邮件组发送一个描述事务修改的邮件），你需要使用libsvn_repos包裹的功能版本—svn_repos_fs_commit_txn。这个功能会实际上首先运行一个如果存在的pre-commit钩子脚本，然后提交事务，最后会运行一个post-commit钩子脚本。钩子提供了一种特别的报告机制，不是真的属于核心文件系统库本身。（关于Subversion版本库钩子的更多信息，见第 5.2.1 节“钩子脚本”。）

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：

1. 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。

2. 描述两个文件系统树的区别。

3. 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

4. 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。

5. 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

如果说Subversion版本库层是在“这条线的另一端”，那版本库访问层就是这条线。负责在客户端库和版本库之间编码数据，这一层包括libsvn_ra模块加载模块，RA模块本身（现在包括了libsvn_ra_dav、libsvn_ra_local和libsvn_ra_svn），和所有一个或多个RA模块需要的附加库，例如与Apache模块mod_dav_svn通讯的libsvn_ra_dav或者是libsvn_ra_svn的服务器，svnserve。

因为Subversion使用URL来识别版本库资源，URL模式的协议部分（通常是file:、http:、https:或svn:）用来监测那个RA模块用来处理通讯。每个模块注册一组它们知道如何“说话”的协议，所以RA加载器可以在运行中监测在手边的任务中使用哪个模块。通过运行svn --version，你可以监测Subversion命令行客户端所支持的RA模块和它们声明支持的协议：

```
$ svn --version
```

```
svn, version 1.0.1 (r9023)
   compiled Mar 17 2004, 09:31:13

Copyright (C) 2000-2004 CollabNet.
Subversion is open source software, see http://subversion.tigris.org/
This product includes software developed by CollabNet (http://www.Collab.Net/).

The following repository access (RA) modules are available:

* ra_dav : Module for accessing a repository via WebDAV (DeltaV) protocol.
  - handles 'http' schema
  - handles 'https' schema
* ra_local : Module for accessing a repository on local disk.
  - handles 'file' schema
* ra_svn : Module for accessing a repository using the svn network protocol.
  - handles 'svn' schema
```

8.1.2.1. RA-DAV（使用HTTP/DAV版本库访问）

libsvn_ra_dav库是给在不同机器使用http:或https:协议访问服务器的用户设计的，为了理解这个模块的工作，我们首先要知道这种版本库访问层中的特定配置的关键组成部分—强大的Apache HTTP服务器，和Neon HTTP/WebDAV客户端库。

Subversion的主要网络服务器是Apache HTTP服务器，Apache是久经考验的用来认真使用的开源服务器，它可以支撑很大的网络压力并且可以运行在多种平台。Apache服务器支持多种认证协议，而且可以通过模块扩展使用其它协议。它也支持流水线和缓存之类的网络优化。通过将Apache作为服务器，Subversion轻易得到这些特性。而且因为许多防火墙已经允许HTTP通过，系统管理员通常不会改变防火墙设置来允许Subversion工作。

Subversion使用HTTP和WebDAV（和DeltaV）来与Apache服务器通讯，你可以在本章的WebDAV读到更多信息，但简而言之，WebDAV和DeltaV是标准HTTP 1.1协议的扩展，允许在web上对文件进行分享和版本操作。Apache 2.0版随着一个mod_dav，一个Apache理解HTTP DAV扩展的模块，Subversion本身提供了mod_dav_svn，尽管，这是另一个Apache模块，它与mod_dav结合（实际上mod_dav_svn是作为后端支持）使用来提供Subversion对WebDAV和DeltaV的实现。

当与版本库通过HTTP通讯时，RA加载器库选择libsvn_ra_dav作为正确的访问模块，Subversion客户端调用原始的RA接口，libsvn_ra_dav把这些调用（包含了大量Subversion操作）影射为一系列HTTP/WebDAV请求。使用Neon库，libsvn_ra_dav把这些请求传递到Apache服务器，Apache接受到这些请求（就像平时web服务器常做的那样处理原始的HTTP请求），注意到这些请求的URL已经配置为DAV的位置（使用httpd.conf的Location指示），并且会使用自己的mod_dav模块来处理。当正确的配置了mod_dav使之知道了使用mod_dav_svn来处理所有文件系统相关的要求，而不是使用默认的Apache自带的原始mod_dav_fs来处理。所以最终客户端是与mod_dav_svn通讯，直接与Subversion版本库层绑定。

有一个实际交换发生的简单描述，举个例子，Subversion版本库可以使用Apache的授权指示进行保护。这会导致初始的与版本库的通讯会被Apache的授权基础拒绝，在此刻，libsvn_ra_dav将提供不足鉴定的通知返回，并且回调客户端层来得到一些更新的认证数据。如果数据是正确提供，而且用户有访问的权限，会赋予libsvn_ra_dav的下一个对原操作的自动尝试权限，并且一切会很好。如果足够的认证信息不能提供，请求会最后失败，客户端也会报告给用户失败信息。

通过使用Neon和Apache，Subversion在许多其它领域的轻易得到复杂的功能。举个例子，如果Neon找到OpenSSL库，它允许Subversion客户端尝试与Apache服务器（它自己的mod_ssl“可以说这个语言”）使用SSL加密的通讯。Neon本身和Apache的mod_deflate都可以理解“deflate”算法（PKZIP和gzip共同使用的程序），所以请求可以压缩块方式传输。其它Subversion今后希望支持的复杂特性包括，自动处理服务器重定向（举个例子，当版本库转移到一个新的规范URL）和利用HTTP流水线的能力。

8.1.2.2. RA-SVN（自定义协议版本库访问）

作为标准HTTP/WebDAV协议的补充，Subversion也提供了一个使用自定义协议的RA实现，libsvn_ra_svn模块实现了自己的网络套接字连接，与一个独立服务器通讯—svnserve程序—在存放版本库的机器上。客户端可以使用svn://访问版本库。

这个RA实现缺乏前面小节提到的Apache的大多数优点；然而虽然如此，系统管理员会非常有兴趣，因为这配置和运行异常的简单；设置一个svnserve几乎是立刻的，它与Apache相比也是非常的小（从代码长度这方面说），让它非常容易进行安全或其它方面原因的审核。此外，一些系统管理员或许已经有了一个SSH安全基础，希望Subversion使用它，客户端使用ra_svn可以容易的使用SSH封装这个协议。

8.1.2.3. RA-Local（直接版本库访问）

并不是所有与Subversion版本库的通讯需要服务器进程和一个网络层。用户如果只是希望简单的访问本地磁盘的版本库，他们会使用file:的URL和libsvn_ra_local提供的功能。RA模块直接与版本库和文件系统库绑定，所以不需要网络通讯。

Subversion需要服务器名称成为file:的URL的一部分，是localhost或者是为空。换句话说，你的URL必须看起来如file://localhost/path/to/repos或者file:///path/to/repos。

也必须意识到Subversion的file: URL不能和在普通的web服务器中的file: URL一样工作。当你尝试在web服务器查看一个file:的URL，它会通过直接检测文件系统读取和显示那个位置的文件内容，但是Subversion的资源存在于虚拟文件系统（见第 8.1.1 节 “版本库层”）中，你的浏览器不会理解怎样读取这个文件系统。

8.1.2.4. 你的RA库在这里

对那些一直希望使用另一个协议来访问Subversion版本库的人，正好是为什么版

本库访问层是模块化的！开发者可以简单的编写一个新的库来在一侧实现RA接口并且与另一侧的版本库通讯。你的新库可以使用存在的网络协议，或者发明你自己的。你可以使用进程间的通讯调用，或者—让我们发狂，我们会吗？—你甚至可以实现一个电子邮件为基础的协议，Subversion提供了API，你提供创造性。

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 8.3 节 “进入工作拷贝的管理区”。

Subversion客户端库libsvn_client具备最广泛的职责；它的工作是结合工作拷贝库和版本库访问库的功能，然后为希望普通版本控制的应用提供最高级的API。举个例子，svn_client_checkout方法是用一个URL作为参数，传递这个URL到RA层然后在特定版本库打开一个会话。然后向版本库要求一个特定的目录树，然后把目录树发送给工作拷贝库，然后把完全的工作拷贝写到磁盘（.svn目录和一切）。

客户端库是为任何程序使用设计的，尽管Subversion的源代码包括了一个标准的命令行客户端，用客户端库编写GUI客户端也是很简单，Subversion新的GUI（或者任何新的客户端，真的）不需要紧密围绕包含的命令行客户端—他们对具有相同功能、数据和回调机制的libsvn_client的API有完全的访问权利。

> 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…）。

8.2. 使用API

使用Subversion库API开发应用看起来相当的直接，所有的公共头文件放在源文件的subversion/include目录，从源代码编译和安装Subversion本身，需要这些头

文件拷贝到系统位置。这些头文件包括了所有用户可以访问的功能和类型。

你首先应该注意Subversion的数据类型和方法是命名空间保护的，每一个公共Subversion对象名以svn_开头，然后紧跟一个这个对象定义（如wc、client和fs其他）所在的库的简短编码，然后是一个下划线（_）和后面的对象名称。半公开的方法（库使用，但是但库之外代码不可以使用并且只可以在库自己的目录看到）与这个命名模式不同，并不是库代码之后紧跟一个下划线，他们是用两个下划线（__）。给定源文件的私有方法没有特殊前缀，使用static声明。当然，一个编译器不会关心命名习惯，只是用来区分给定方法或数据类型。

8.2.1. Apache可移植运行库

伴随Subversion自己的数据类型，你会看到许多apr_开头的数据类型引用—来自Apache可移植运行库（APR）的对象。APR是Apache可移植运行库，源自为了服务器代码的多平台性，尝试将不同的操作系统特定字节与操作系统无关代码隔离。结果就提供了一个基础API的库，只有一些适度区别—或者是广泛的—来自各个操作系统。Apache HTTP服务器很明显是APR库的第一个用户，Subversion开发者立刻发现了使用APR库的价值。意味着Subversion没有操作系统特定的代码，也意味着Subversion客户端可以在Server存在的平台编译和运行。当前这个列表包括，各种类型的Unix、Win32、OS/2和Mac OS X。

除了提供了跨平台一致的系统调用，[3] APR给Subversion对多种数据类型有快速的访问，如动态数组和哈希表。Subversion在代码中广泛使用这些类型，但是或许大多数普遍深入的APR数据类型可以在所有的Subversion的API原型中发现，是apr_pool_t—APR内存池，Subversion使用内部缓冲池用来进行内存分配（除非外部库在API传递参数时需要一个不同的内存管理模式），[4] 而且一个人如果针对Subversion的API编码不需要做同样的事情，他们可以在需要时给API提供缓冲池，这意味着Subversion的API使用者也必须链接到APR，必须调用apr_initialize()来初始化APR字系统，然后必须得到一个缓冲池用来进行Subversion的API调用。详情见第 8.5 节 “使用内存池编程”。

8.2.2. URL和路径需求

因为分布式版本控制操作是Subversion存在的重点，有意义来关注一下国际化（i18n）支持。毕竟，当“分布式”或许意味着“横跨办公室”，它也意味着“横跨全球”。为了更容易一点，Subversion的所有公共接口只接受路径参数,这些参数是传统的，使用UTF-8编码。这意味着，举个例子，任何新的使用libsvn_client接口客户端库，在把这些参数传递给Subversion库前，需要首先将路径从本地代码转化为UTF-8代码，然后将Subversion传递回来的路径转换为本地代码，很幸运，Subversion提供了一组任何程序可以使用的转化方法（见subversion/include/svn_utf.h）。

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，而会传递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

[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。

码和解码。

8.2.3. 使用C和C++以外的语言

除C语言以外，如果你对使用其他语言结合Subversion库感兴趣—如Python脚本或是Java应用—Subversion通过简单包裹生成器（SWIG）提供了最初的支持。Subversion的SWIG绑定位于subversion/bindings/swig，并且慢慢的走向成熟进入可用状态。这个绑定允许你直接调用Subversion的API方法，使用包裹器会把脚本数据类型转化为Subversion需要的C语言库类型。

通过语言绑定访问Subversion的API有一个明显的好处—简单性。通常来讲，Python和Perl之类的语言比C和C++更加的灵活和简单，这些语言的高级数据类型和上下文驱动类型更加易于处理用户提供的信息，就像你知道的，人们精于把程序搞坏，脚本语言可以更优雅的处理这些错误信息，当然，灵活性经常带来性能的损失，这就是为什么使用紧密优化的，C基础的接口和库组件，然后与一种高效的、灵活的绑定语言，是这样的吸引人。

让我们看看Subversion的Python SWIG绑定的实例，这个例子和前面的例子作同样的事，注意比较方法的长度和复杂性！

例 8.2. 使用Python处理版本库层

```
from svn import fs
import os.path

def crawl_filesystem_dir (root, directory, pool):
  """Recursively crawl DIRECTORY under ROOT in the filesystem, and return
  a list of all the paths at or below DIRECTORY.  Use POOL for all
  allocations."""

  # Get the directory entries for DIRECTORY.
  entries = fs.dir_entries(root, directory, pool)

  # Initialize our returned list with the directory path itself.
  paths = [directory]

  # Loop over the entries
  names = entries.keys()
  for name in names:
    # Calculate the entry's full path.
    full_path = os.path.join(basepath, 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 fs.is_dir(fsroot, full_path, pool):
```

```
      subpaths = crawl_filesystem_dir(root, full_path, pool)
      paths.extend(subpaths)

    # Else, it is a file, so add the entry's full path to the FILES list.
    else:
      paths.append(full_path)

  return paths
```

前面C语言的实现确实有一点长，另外C语言的例行公事就是必须关注内存使用，并且需要使用自定义的数据类型来表示条目的哈希值和路径列表。Python有哈希（叫做“dictionaries”）并且列表示内置数据类型，并提供了许多操作这些类型的好方法。而且因为Python使用引用计数来进行垃圾收集，这种语言的用户不需要麻烦自己去分配和回收内存。

在本章的前面小节，我们提到libsvn_client接口，并且解释了它存在的唯一目的是为了简化编写Subversion客户端的过程，下面是一个如何同SWIG绑定访问库的简短例子，简单的几句Python代码，我们就可以检出一个完全功能的Subversion工作拷贝！

例 8.3. 一段检出工作拷贝的简单脚本

```
#!/usr/bin/env python
import sys
from svn import util, _util, _client

def usage():
  print "Usage: " + sys.argv[0] + " URL PATH\n"
  sys.exit(0)

def run(url, path):
  # Initialize APR and get a POOL.
  _util.apr_initialize()
  pool = util.svn_pool_create(None)

  # Checkout the HEAD of URL into PATH (silently)
  _client.svn_client_checkout(None, None, url, path, -1, 1, None, pool)

  # Cleanup our POOL, and shut down APR.
  util.svn_pool_destroy(pool)
  _util.apr_terminate()

if __name__ == '__main__':
  if len(sys.argv) != 3:
```

```
    usage()
run(sys.argv[1], sys.argv[2])
```

非常不幸，Subversion的语言绑定缺乏对核心Subversion模块的关注，但是，使用Python、Perl和Java创建有功能的邦定取得了显著的成就。一旦你的SWIG接口文件正确的配置，对于SWIG支持的语言（我们当前包括的版本有C#、Guile、Java、MzScheme、OCaml、Perl、PHP、Python、Ruby和Tcl）的特定语言绑定的包裹器的生成理论上是非常琐碎的。但是，对复杂API还是需要一些额外的的补充，SWIG需要帮助归纳。对于SWIG的更多信息，见这个项目的网站http://www.swig.org/。

8.3. 进入工作拷贝的管理区

像我们前面提到的，每个Subversion工作拷贝包含了一个特别的子目录叫做.svn，这个目录包含了关于工作拷贝目录的管理数据，Subversion使用.svn中的信息来追踪如下的数据：

- 工作拷贝中展示的目录和文件在版本库中的位置。
- 工作拷贝中当前展示的文件和目录的修订版本。
- 所有附加在文件和目录上的用户定义属性。
- 初始（未编辑）的工作拷贝文件的拷贝。

然而.svn目录中还有一些其他的数据，我们会考察一些最重要的项目。

8.3.1. 条目文件

或许.svn目录中最重要的单个文件就是entries了，这个条目文件是一个XML文档，包含了关于工作拷贝中的版本化的资源的大多数管理性信息，这个文件保留了版本库URL、原始修订版本、可知的最后提交信息（作者、修订版本和时间戳）和本地拷贝历史—实际上是Subversion客户端关于一个版本化（或者是将要版本化的）资源的所有感兴趣的信息！

> 比较Subversion和CVS的管理区域
>
> 扫视一下典型的.svn目录会发现比CVS在CVS目录中的内容多一些，entries文件包含的XML描述了工作拷贝目录的当前状态，而且基本上合并了CVS的Entries、Root和Repository的功能。

如下是一个实际条目文件的例子：

例 8.4. 典型的.svn/entries文件内容

```
<?xml version="1.0" encoding="utf-8"?>
<wc-entries
   xmlns="svn:">
<entry
   committed-rev="1"
   name=""
   committed-date="2002-09-24T17:12:44.064475Z"
   url="http://svn.red-bean.com/tests/.greek-repo/A/D"
   kind="dir"
   revision="1"/>
<entry
   committed-rev="1"
   name="gamma"
   text-time="2002-09-26T21:09:02.000000Z"
   committed-date="2002-09-24T17:12:44.064475Z"
   checksum="QSE4vWd9ZM0cMvr7/+YkXQ=="
   kind="file"
   prop-time="2002-09-26T21:09:02.000000Z"/>
<entry
   name="zeta"
   kind="file"
   schedule="add"
   revision="0"/>
<entry
   url="http://svn.red-bean.com/tests/.greek-repo/A/B/delta"
   name="delta"
   kind="file"
   schedule="add"
   revision="0"/>
<entry
   name="G"
   kind="dir"/>
<entry
   name="H"
   kind="dir"
   schedule="delete"/>
</wc-entries>
```

就像你能看到的，条目文件本质上是一列条目，每个entry标签代表了下面三者之一的事情：工作拷贝目录本身（叫做“本目录”条目，并且name属性的值为空），工作拷贝目录中的一个文件（通过kind属性设置为"file"来标示），或者是工

作拷贝中的一个子目录（kind这时设置为"dir"）。所有在这个文件标记的文件和子目录都是已经纳入版本控制或者是（上面例子中的zeta）预定在下次提交加入到版本控制。每个条目都有一个唯一的名字，每个条目有一个kind节点。

开发者必须意识到一些Subversion读写entries文件的特殊规则，每个条目都有一个修订版本和URL与之关联，注意在上面实例文件中并不是每个entry标签都有明确的revision或url属性，Subversion允许一些情况不明确的说明这个两个属性，如属性值与“本目录”的值相同（revision的情况）或者是可以从“本目录”简单计算[5]出的来（url）。注意对于子目录条目，Subversion只保管最重要的信息—名称、类型、URL、修订版本和日程。为了减少重复信息，Subversion指示当要检测目录信息时会跑到这个子目录自己的.svn/entries的“本目录”条目。当然了，对这个子目录的引用还是会保存在父目录的entries文件，这些信息足以在子目录丢失后执行基本的版本操作。

8.3.2. 原始拷贝和属性文件

如我们前面提到的，.svn也包含了一些原始的“text-base”文件版本，可以在.svn/text-base看到。这些原始文件的好处是多方面的—察看本地修改和区别不需要经过网络访问，减少传递修改时的数据—但是随之而来的代价是每个版本化的文件都在磁盘至少保存两次，现在看来这是对大多数文件可以忽略不计的一个惩罚。但是，当你版本控制的文件增多之后形势会变得很严峻，我们已经注意到了应该可以选择使用“text-base”，但是具有讽刺意味的是，当版本化文件增大时，“text-base”文件的存在会更加重要—谁会希望在提交一个小修改时在网络上传递一个大文件？

同“text-base”文件的用途一样的还有属性文件和它们的“prop-base”拷贝，分别位于.svn/props和.svn/prop-base。因为目录也有属性，所以也有.svn/dir-props和.svn/dir-prop-base文件。所有的属性文件（“working”和“base”版本）都使用同样的“hash-on-disk”文件格式来排序属性名称和值。

8.4. WebDAV

WebDAV（“Web-based Distributed Authoring and Versioning”的缩写）是一个标准HTTP协议的扩展，把web变成一个可读写的媒体，作为当今基本的只读媒体的替代。原理就是目录和文件时可以共享的—都是可读写的对象—通过web。RFCs2518和3253描述了WebDAV/DeltaV 对于HTTP的扩展，存放于（随之有许多其它有用的信息）http://www.webdav.org/。

已经有一些操作系统文件浏览器可以使用WebDAV装配网络目录，在Win32中，Windows浏览器可以像普通共享文件夹一样浏览叫做网络文件夹（只是一个设置好WebDAV的网络位置）的目录，在Mac OS X也有这个能力，就像Nautilus和Konqueror作的（分别对应GNOME和KDE）。

这些是如何应用到Subversion中的呢？mod_dav_svn的Apache模块使用HTTP，通过WebDAV和DeltaV扩展，作为它的网络协议之一，Subversion使用mod_dav_svn在

[5]也就是，这个条目的URL就是父目录与名称合并。

Subversion的版本概念和RFCs 2518和3253对应部分建立影射。

关于WebDAV的完全讨论，工作原理和Subversion如何使用，可以看附录 C，WebDAV和自动版本化。在其他事情中，附录讨论了Subversion与一般的WebDAV规范结合的程度，和这些是如何影响普通WebDAV客户端的交互性。

8.5. 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。幸运的是，Subversion所依赖的APR库为了移植性提供了apr_pool_t类型，代表了应用可以分配内存的池。

一个内存池是程序所需要分配内存的一个抽象表示，不选择使用标准的malloc()从操作系统直接申请内存，而使用向APR申请的池申请创建的（使用apr_pool_create()方法）内存。APR会从操作系统分配合适的内存块这些内存可以立刻在程序里使用，当程序需要更多的池内存时，它会使用APR的池API方法，如apr_palloc()，返回池中的基本内存位置，这个程序可以继续从池中请求内存，在超过最初的池的容量后，APR会自动满足程序的要求扩大池的大小，直到系统没有足够的内存。

现在，如果这是池故事的结尾，我们就不应该再作过多的关注，很幸运，不是这个情况。池不可以仅仅被创建；它也可以被清空和销毁，分别使用apr_pool_clear()和apr_pool_destroy()。这给了用户灵活性来分配许多—或者是数千—东西自这个池，然后使用一个命令来清空！更进一步，池可以分级，你可以为前一步创建的池创建“子池”。当你清空一个池，所有的子池会被销毁；如果你销毁一个池，它和所有的子池也会被销毁。

在我们进一步研究之前，开发者会发现在Subversion源代码中并没有对前面提到的APR池方法有很多的调用，APR提供了许多扩展机制，像使用自定义的附加到池的“用户数据”的能力，注册当池销毁时的所要调用的清理方法的机制，Subversion使用一些不太琐碎的方法来利用这些扩展，所以Subversion提供了（大多数代码使用的）包裹方法svn_pool_create()、svn_pool_clear()和svn_pool_destroy()。

尽管池帮助我们基本的内存管理，池的创建确实投射出了循环和迭代场景，因为反复在循环中经常没有界限，在深度迭代中，一定区域的内存消耗变得不可预料，很幸运，使用嵌套的内存池可以简单的管理这种潜在的混乱情形，下面的例子描述了在这个情形下嵌套池的基本使用非常平常—迭代的对目录树的遍历，对树上的每一个部分做一些任务。

例 8.5. 有效地池使用

```
/* Recursively crawl over DIRECTORY, adding the paths of all its file
   children to the FILES array, and doing some task to each path
```

```
   encountered.  Use POOL for the all temporary allocations, and store
   the hash paths in the same pool as the hash itself is allocated in.  */
static apr_status_t
crawl_dir (apr_array_header_t *files,
           const char *directory,
           apr_pool_t *pool)
{
  apr_pool_t *hash_pool = files->pool;  /* array pool */
  apr_pool_t *subpool = svn_pool_create (pool);  /* iteration pool */
  apr_dir_t *dir;
  apr_finfo_t finfo;
  apr_status_t apr_err;
  apr_int32_t flags = APR_FINFO_TYPE | APR_FINFO_NAME;

  apr_err = apr_dir_open (&dir, directory, pool);
  if (apr_err)
    return apr_err;

  /* Loop over the directory entries, clearing the subpool at the top of
     each iteration.  */
  for (apr_err = apr_dir_read (&finfo, flags, dir);
       apr_err == APR_SUCCESS;
       apr_err = apr_dir_read (&finfo, flags, dir))
    {
      const char *child_path;

      /* Clear the per-iteration SUBPOOL.  */
      svn_pool_clear (subpool);

      /* Skip entries for "this dir" ('.') and its parent ('..').  */
      if (finfo.filetype == APR_DIR)
        {
          if (finfo.name[0] == '.'
              && (finfo.name[1] == '\0'
                  || (finfo.name[1] == '.' && finfo.name[2] == '\0')))
            continue;
        }

      /* Build CHILD_PATH from DIRECTORY and FINFO.name.  */
      child_path = svn_path_join (directory, finfo.name, subpool);

      /* Do some task to this encountered path. */
      do_some_task (child_path, subpool);

      /* Handle subdirectories by recursing into them, passing SUBPOOL
         as the pool for temporary allocations.  */
      if (finfo.filetype == APR_DIR)
```

```
        {
          apr_err = crawl_dir (files, child_path, subpool);
          if (apr_err)
            return apr_err;
        }

      /* Handle files by adding their paths to the FILES array.  */
      else if (finfo.filetype == APR_REG)
        {
          /* Copy the file's path into the FILES array's pool.  */
          child_path = apr_pstrdup (hash_pool, child_path);

          /* Add the path to the array.  */
          (*((const char **) apr_array_push (files))) = child_path;
        }
    }

  /* Destroy SUBPOOL.  */
  svn_pool_destroy (subpool);

  /* Check that the loop exited cleanly. */
  if (apr_err)
    return apr_err;

  /* Yes, it exited cleanly, so close the dir. */
  apr_err = apr_dir_close (dir);
  if (apr_err)
    return apr_err;

  return APR_SUCCESS;
}
```

在前一个例子里描述了在循环和迭代情况下有效地池使用，每次迭代会从为方法传递一个新建的子池开始，池在循环区域中使用，在每次迭代清理。结果是内存使用比例和深度成比例，而不是顶级目录包含所有的子目录的总数量。当迭代的第一个调用最终结束时，实际上只有很小的传递过来的数据存放在池中，现在想想一下如果在每片数据使用时使用alloc()和free()时会面临的复杂性！

池并不是对所有的应用是理想的，但是在Subversion中非常有用，作为一个Subversion开发者，你会需要学会适应池并且正确地使用它，内存使用的bug和膨胀可能会非常难于诊断和修正，但是APR提供的pool结构被证明了是非常的方便的，节约时间的功能。

8.6. 为Subversion做贡献

Subversion项目的官方信息源当然是项目的网站http://subversion.tigris.org/。这里你可以发现如何得到源代码和参与到讨论列表。Subversion社区一致欢迎新成员，如果你有兴趣通过贡献源代码来参与到社区，以下是一下作为开始的提示。

8.6.1. 加入社区

加入社区的第一步是关注最新发生的事情，最有效的办法是订阅主要的开发邮件列表（<dev@subversion.tigris.org>）和提交邮件列表（<svn@subversion.tigris.org>）。通过轻松的跟踪这些列表，你可以参与最重要的设计讨论，并且可以看到实际发生效果的Subversion代码，并且可以见证这些修改并提出修改建议。这些基于邮件列表的讨论是我们Subversion开发最主要的交流媒体。如果你对其他Subversion相关的列表有兴趣，可以查看本网站的邮件列表部分。

但是你如何知道需要做什么？这是一个希望参与帮助我们开发的程序员最关心的问题，很难找到一个好的开始。毕竟，来到社区的很多人并没有已经决定好了要做什么，但是通过阅读开发者的讨论，你会看到很感兴趣的已知bug或特性需求。另外也可以在问题追踪数据库找出那些突出的没有人做的任务，这里你会发现当前列表的已知bug和特性需求，如果你希望从一些小事开始，可以查看那些标记为“bite-sized”的问题。

8.6.2. 取得源代码

为了编辑源代码，你需要得到源代码，这意味着你需要从Subversion源代码版本库检出一个工作拷贝，听起来如此直接，这个任务可能有一点微妙。因为Subversion的源代码使用Subversion本身版本管理，你实际上需要使用别的方法得到工作的Subversion客户端来启动这个过程。最通常的方法是下载最新的二进制分发版本（如果有你的平台的版本存在），或者是下载最新的源程序包并且自己编译Subversion客户端，如果你从源代码编译，确定要阅读源代码顶级目录的INSTALL文件作为指导。

在你有了工作的Subversion客户端后，你可以泰然自若的从Subversion源代码版本库http://svn.collab.net/repos/svn/trunk/检出一个工作拷贝：[6]

```
$ svn checkout http://svn.collab.net/repos/svn/trunk subversion
A  subversion/HACKING
A  subversion/INSTALL
A  subversion/README
A  subversion/autogen.sh
A  subversion/build.conf
...
```

[6]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。

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。

8.6.3. 开始熟悉社区政策

现在你有了包含最新Subversion源代码的工作拷贝，你一定希望来通过工作拷贝顶级目录下的HACKING文件来做一次浏览。这个HACKING文件包含了如何对Subversion做贡献的说明，包括如何正确地格式化代码与余下的代码基保持一致性，如何使用有效的提交日志描述你的被提议修改，如何测试修改，等等。对Subversion源代码的提交特权是需要争取得到的—被精英所管理。[7] HACKING文件是一个无价的资源，它可以确保你被提议作的修改能够取得承认，而不会因为技术原因被拒绝。

8.6.4. 作出修改并测试

当理解了代码和社区政策，你已经准备好了作出修改，最好是努力作出小的但是相关的修改，即使在处理大的任务阶段，不要选择作出巨大的扫除试的修改。如果你搞乱最少的代码来完成修改，你被提议的修改就会很容易理解（而且因此应该很容易去审核）。当完成了每个提议的修改集，你的Subversion树一定要处于编译无警告的状态。

Subversion有一个相当彻底[8]的回归测试套件，你提议的修改期望不会带来任何这种测试失败，通过在源代码根目录运行make check（在Unix）你可以完全测试你的修改。提交会导致测试套间失败的代码是拒绝（或者是提供一个好的日志信息）你贡献的代码的最快方法。

在最好的情况下，你实际上应该添加适当的测试到测试套件来验证你提议的修改工作正常，实际上，有时候一个人可以做到的最好贡献就是让添加的测试能够独立起来。你可以添加回归测试来保护当前工作的代码在将来修改时这个区域里不会触发失败。另外，你也可以写测试来描述已知的失败，为了这个目的，Subversion测试套件允许你指定一个给定的测试是期望会失败的（叫做XFAIL），而且只要Subversion按照预期失败，一个XFAIL测试会认为是一个成功。最后，测试组件越好，就会花费更少的时间来诊断潜在的晦涩的回归bug。

8.6.5. 贡献你的修改

当完成了对源代码的修改，写一个干净的和细致的日志信息来描述那些修改和原因。然后，发送一个包含日志信息和svn diff（在Subversion工作拷贝顶级目录运行）输出的邮件到开发者列表。如果社区成员认为你的修改可以接受，一些有提交权限（允许在Subversion源代码版本库提交新的修订版本）的用户会添加你的新的修改到公共源代码树。回想对版本库直接的提交权限是赋予那些展现能力

[7] 浅薄的看起来这像是某种高人一等的优越感，“赢得你的提交特权”这个概念关于效率—检查和应用别人的修改是否安全和有用会花费大量的时间和精力，与之相比的是取消危险的代码的潜在代价。
[8] 在这个情况下，你或许希望抓一些爆米花，在附近花三十分钟转一下，渡过非交互的机器时间。

的人—如果你展示了对Subversion的理解，编程能力，和“团队精神”，你会很可能授予那个权限。

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

为了使用命令行客户端，只需要输入svn和它的子命令[1]以及相关的选项或操作的对象—输入的子命令和选项没有特定的顺序，下面使用svn status的方式都是合法的：

```
$ svn -v status
$ svn status -v
$ svn status -v myfile
```

你可以在第 3 章 指导教程发现更多使用客户端命令的例子，以及第 7.2 节 “属性”中的管理属性的命令。

9.1.1. svn选项

虽然Subversion的子命令有一些不同的选项，但有的选项是全局的—也就是说，每个选项保证是表示同样的事情，而不管是哪个子命令使用的。举个例子，--verbose（-v）一直意味着“冗长输出”，而不管使用它的命令是什么。

--auto-props
开启auto-props，覆盖config文件中的enable-auto-props指示。

--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。

--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。

[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。

--editor-cmd CMD
指定一个外部程序来编辑日志信息或是属性值。

--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
指定一个或多个Subversion传递给提供文件区别的外部区别程序的参数，如果你要传递多个参数，你一定能够要用引号（例如，svn diff --diff-cmd /usr/bin/diff -x "-b -E"）括起所有的参数。这个选项只有在使用--diff-cmd选项时使用。

--file (-F) FILENAME
使用传递的文件内容作为特定子命令的选项。

--force
强制一个特定的命令或操作运行。Subversion有一些操作防止你做普通的使用，但是你可以传递force选项告诉Subversion“我知道我做的事情，也知道这样的结果，所以让我做吧”。这个选项在程序上等同于在打开电源的情况下做你自己的电子工作—如果你不知道你在做什么，你很有可能会得到一个威胁的警告。

--force-log
将传递给--message（-m）或者--file（-F）的可疑参数指定为有效可接受。缺省情况下，如果选项的参数看起来会成为子命令的目标，Subversion会提出一个错误，例如，你传递一个版本化的文件路径给--file（-F）选项，Subversion会认为出了点错误，认为你将目标对象当成了参数，而你并没有提供其它的—未版本化的文件作为日志信息的文件。为了确认你的意图并且不考虑这类错误，传递--force-log选项给命令来接受它作为日志信息。

--help (-h or -?)
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。

--incremental
打印适合串联的输出格式。

--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。

--no-auth-cache
阻止在Subversion管理区缓存认证信息（如用户名密码）。

--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 “config”和第 7.2.3.3 节 “svn:ignore”查看详情。

--non-interactive
如果认证失败，或者是不充分的凭证时，防止出现要求凭证的提示（例如用户名和密码）。这在运行自动脚本时非常有用，只是让Subversion失败而不是提示更多的信息。

--non-recursive (-N)
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—不是。

--notice-ancestry
在计算区别时关注祖先。

--old ARG
使用ARG作为旧的目标。

--password PASS
指出在命令行中提供你的密码—另外，如果它是需要的，Subversion会提示你输入。

--quiet (-q)
请求客户端在执行操作时只显示重要信息。

--recursive (-R)
让子命令迭代到子目录，大多数子命令缺省是迭代的。

--relocate FROM TO [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。

--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 3.3.2 节 “修订版本关键字”查看更多信息。

--revprop
操作是针对一个修订版本的属性而不是一个文件或目录的属性。这个开关需要你也要通过--revision（-r）传递一个修订版本号，见第 5.1.2 节 “未受版本控制的属性”关于未版本化的属性的细节。

--show-updates (-u)
导致客户端显示本地拷贝哪些文件已经过期，这不会实际更新你的任何文件—只是显示了如果你运行svn update时更新的文件。

--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时。

--strict
导致Subversion使用严格的语法，也就是选择含糊，除非谈论特定的子命令。

--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。

--username NAME
表示你要在命令行提供认证的用户名—否则如果需要，Subversion会提示你这一点。

--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
打印客户端版本信息，这个信息不仅仅包括客户端的版本号，也有所有客户端可以用来访问Subversion版本库的版本库访问模块列表。

--xml
使用XML格式打印输出。

9.1.2. svn子命令

名称

svn add -- 添加文件、目录或符号链。

概要

svn add PATH...

描述

添加文件、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

变化

工作拷贝

是否访问版本库

否

选项

```
--targets FILENAME
--non-recursive (-N)
--quiet (-q)
--config-dir DIR
--auto-props
--no-auto-props
--force
```

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 --non-recursive otherdir
A         otherdir
```

通常情况下，命令svn add *会忽略所有已经在版本控制之下的目录，有时候，你会希望添加所有工作拷贝的未版本化文件，包括那些隐藏在深处的文件，可以使用svn add的--force递归到版本化的目录下：

```
$ svn add * --force
A         foo.c
A         somedir/bar.c
A         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...
```

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

变化

无

是否访问版本库

是

选项

```
--revision (-r) REV
--username USER
--password PASS
--no-auth-cache
--non-interactive
--config-dir DIR
--verbose
```

例子

如果你希望在测试版本库看到blame标记的readme.txt源代码：

```
$ svn blame http://svn.red-bean.com/repos/test/readme.txt
     3      sally This is a README file.
     5      harry You should read this.
```

名称

svn cat -- 输出特定文件或URL的内容。

概要

```
svn cat TARGET[@REV]...
```

描述

输出特定文件或URL的内容。列出目录的内容可以使用svn list。

别名

无

变化

无

是否访问版本库

是

选项

```
--revision (-r) REV
--username USER
--password PASS
--no-auth-cache
--non-interactive
--config-dir DIR
```

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

如果你的工作拷贝已经过期（或者你有本地修改），并且希望察看工作拷贝的HEAD修订版本的一个文件，如果你给定一个路径，svn cat会自

动取得HEAD的修订版本：

```
$ cat foo.c
This file is in my local working copy
and has changes that I've made.

$ svn cat foo.c
Latest revision fresh from the repository!
```

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

变化

创建一个工作拷贝。

是否访问版本库

是

选项

```
--revision (-r) REV
--quiet (-q)
--non-recursive (-N)
--username USER
--password PASS
--no-auth-cache
--non-interactive
--config-dir DIR
```

例子

取出一个工作拷贝到mine：

```
$ svn checkout file:///tmp/repos/test mine
A  mine/a
A  mine/b
Checked out revision 2.
$ ls
mine
```

检出两个目录到两个单独的工作拷贝：

```
$ svn checkout file:///tmp/repos/test  file:///tmp/repos/quiz
A  test/a
A  test/b
Checked out revision 2.
A  quiz/l
A  quiz/m
Checked out revision 2.
$ 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:///tmp/repos/test  file:///tmp/repos/quiz working-copies
A  working-copies/test/a
A  working-copies/test/b
Checked out revision 2.
A  working-copies/quiz/l
A  working-copies/quiz/m
Checked out revision 2.
$ 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:///tmp/repos/test test
A  test/a
A  test/b
^C
svn: The operation was interrupted
svn: caught SIGINT

$ svn checkout file:///tmp/repos/test test
A  test/c
A  test/d
^C
svn: The operation was interrupted
svn: caught SIGINT

$ cd test
```

```
$ svn update
A  test/e
A  test/f
Updated to revision 3.
```

名称

svn cleanup -- 递归清理工作拷贝。

概要

svn cleanup [PATH...]

描述

递归清理工作拷贝，删除未完成的操作锁定。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。见附录 B，故障解决。

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

变化

工作拷贝

是否访问版本库

否

选项

--diff3-cmd CMD
--config-dir DIR

例子

svn cleanup没有输出，没有太多的例子，如果你没有传递路径，会使用“.”。

```
$ svn cleanup

$ svn cleanup /path/to/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

将修改从工作拷贝发送到版本库。如果你没有使用--file或--message提供一个提交日志信息，svn会启动你的编辑器来编写一个提交信息，见第 7.1.3.2 节 “config”的editor-cmd小节。

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（“check in”的缩写；不是“checkout”的缩写“co”。）

变化

工作拷贝，版本库

是否访问版本库

是

选项

```
--message (-m) TEXT
--file (-F) FILE
--quiet (-q)
--non-recursive (-N)
--targets FILENAME
--force-log
--username USER
--password PASS
--no-auth-cache
--non-interactive
--encoding ENC
```

```
--config-dir DIR
```

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.
```

提交一个修改到foo.c（在命令行明确指明），并且msg文件中保存了提交信息：

```
$ 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 DST
```

描述

拷贝工作拷贝的一个文件或目录到版本库。SRC和DST既可以是工作拷贝（WC）路径也可以是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。

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

变化

如果目标是URL则包括版本库。

如果目标是WC路径，则是工作拷贝。

是否访问版本库

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

```
--message (-m) TEXT
--file (-F) FILE
--revision (-r) REV
--quiet (-q)
--username USER
--password PASS
--no-auth-cache
--non-interactive
--force-log
--editor-cmd EDITOR
--encoding ENC
--config-dir DIR
```

例子

拷贝工作拷贝的一个项目（只是预定要拷贝—在提交之前不会影响版本库）：

```
$ svn copy foo.txt bar.txt
A         bar.txt
$ svn status
A  +   bar.txt
```

拷贝你的工作拷贝的一个项目到版本库的URL（直接的提交，所以需要提供一个提交信息）：

```
$ svn copy near.txt file:///tmp/repos/test/far-away.txt -m "Remote copy."

Committed revision 8.
```

拷贝版本库的一个项目到你的工作拷贝（只是预定要拷贝—在提交之前不会影响版本库）：

提示

这是恢复死掉文件的推荐方式！

```
$ svn copy file:///tmp/repos/test/far-away near-here
A         near-here
```

最后，是在URL之间拷贝：

```
$ svn copy file:///tmp/repos/test/far-away file:///tmp/repos/test/over-there -m "rem

Committed revision 9.
```

提示

这是在版本库里作“标签”最简单的方法—svn copy那个修订版本（通常是HEAD）到你的tags目录。

```
$ svn copy file:///tmp/repos/test/trunk file:///tmp/repos/test/tags/0.6.32-prereleas

Committed revision 12.
```

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:///tmp/repos/test/trunk file:///tmp/repos/test/tags/0.6.32-pre

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，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

变化

如果操作对象是文件则是工作拷贝变化，对象是URL则会影响版本库。

是否访问版本库

对URL操作时访问

选项

```
--force
--force-log
--message (-m) TEXT
--file (-F) FILE
--quiet (-q)
--targets FILENAME
--username USER
--password PASS
--no-auth-cache
--non-interactive
--editor-cmd EDITOR
--encoding ENC
--config-dir DIR
```

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:///tmp/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 -- 比较两条路径的区别。

概要

```
diff [-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，svn diff有三种使用方式：

svn diff [-r N[:M]] [--old OLD-TGT] [--new NEW-TGT] [PATH...]会显示OLD-TGT和NEW-TGT的区别。如果给定路径PATH，它会被看作OLD-TGT和NEW-TGT的相对路径，输出也会限制在这些路径的区别上。OLD-TGT和NEW-TGT可以是工作拷贝路径或者是URL[@REV]。OLD-TGT缺省是当前工作目录，而NEW-TGT缺省是OLD-TGT。N缺省是BASE，M缺省时当前目录的版本，但如果NEW-TGT是一个URL，则默认是HEAD。svn diff -r N设置OLD-TGT的修订版本为N，svn diff -r N:M设置NEW-TGT的修订版本是M。

svn diff [-r N[:M]] URL1[@N] URL2[@M]是svn diff [-r N[:M]] --old=URL1 --new=URL2的缩写。

TARGET是一个URL，然后可以使用前面提到的--revision或“@”符号来指定N和M。

如果TARGET是工作拷贝路径，则--revision选项的含义是：

--revision N:M
   服务器比较 TARGET@N和TARGET@M。

--revision N
   客户端比较TARGET@N和工作拷贝。

（无--revision）
   客户端比较base和 TARGET的TARGET。

如果使用其他语法，服务器会比较URL1和URL2各自的N和M。如果省掉N或M，会假定为HEAD。

缺省情况下，svn diff忽略文件的祖先，只会比较两个文件的内容。如果你使用

--notice-ancestry，比较修订版本（也就是，当你运行svn diff比较两个内容相同，但祖先历史不同的对象会看到所有的内容被删除又再次添加）时就会考虑路径的祖先。

别名

di

变化

无

是否访问版本库

获得工作拷贝非BASE修订版本的区别时会

选项

```
--revision (-r) REV
--old OLD-TARGET
--new NEW-TARGET
--extensions (-x) "ARGS"
--non-recursive (-N)
--diff-cmd CMD
--notice-ancestry
--username USER
--password PASS
--no-auth-cache
--non-interactive
--no-diff-deleted
--config-dir DIR
```

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)
```

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 READM
```

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
>
```

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 PATH1[@PEGREV] PATH2
```

描述

第一种从版本库导出干净工作目录树的形式是指定URL，如果指定了修订版本REV，会导出相应的版本，如果没有指定修订版本，则会导出HEAD，导出到PATH。如果省略PATH，URL的最后一部分会作为本地目录的名字。

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

变化

本地磁盘

是否访问版本库

只有当从URL导出时会访问

选项

```
--revision (-r) REV
--quiet (-q)
--force
--username USER
--password PASS
--no-auth-cache
--non-interactive
--config-dir DIR
--native-eol EOL
```

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:///tmp/repos my-export
A  my-export/test
A  my-export/quiz
…
Exported revision 15.
```

当使用操作系统特定的分发版本，使用特定的EOL字符作为行结束符号导出一棵树会非常有用。--native-eol选项会这样做，但是如果影响的文件拥有svn:eol-style = native属性，举个例子，导出一棵使用CRLF作为行结束的树（可能是为了做一个Windows的.zip文件分发版本）：

```
$ svn export file://tmp/repos my-export --native-eol CRLF
A  my-export/test
A  my-export/quiz
…
Exported revision 15.
```

名称

svn help -- 帮助!

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

变化

无

是否访问版本库

不访问

选项

--version
--quiet (-q)

名称

svn import -- 递归提交一个路径的拷贝到URL。

概要

svn import [PATH] URL

描述

递归提交一个路径的拷贝到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。

别名

无

变化

版本库

是否访问版本库

是

选项

--message (-m) TEXT
--file (-F) FILE
--quiet (-q)
--non-recursive (-N)
--username USER
--password PASS
--no-auth-cache
--non-interactive
--force-log
--editor-cmd EDITOR
--encoding ENC
--config-dir DIR
--auto-props
--no-auto-props

例子

这会导入本地目录myproj到版本库的 根目录：

```
$ svn import -m "New import" myproj http://svn.red-bean.com/repos/test
Adding         myproj/sample.txt
…
Transmitting file data .........
Committed revision 16.
```

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/test/trunk/misc/myproj
Adding         myproj/sample.txt
…
Transmitting file data .........
Committed revision 19.
```

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 -- 打印PATH的信息。

概要

svn info [PATH...]

描述

打印你的工作拷贝的路径信息，包括：

- 路经
- 名称
- URL
- 修订版本
- 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
- 最后更新的文本
- 最后更新的属性
- 核对

别名

无

变化

无

是否访问版本库

无

选项

```
--targets FILENAME
--recursive (-R)
--config-dir DIR
```

例子

svn info会展示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
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: trunk
URL: http://svn.red-bean.com/repos/test/vendors
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)
```

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。

伴随--verbose，如下的字段展示了项目的状态：

- 最后一次提交的修订版本号
- 最后一次提交的作者
- 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

变化

无

是否访问版本库

是

选项

```
--revision (-r) REV
--verbose (-v)
--recursive (-R)
--incremental
--xml
```

```
--username USER
--password PASS
--no-auth-cache
--non-interactive
--config-dir DIR
```

例子

如果你希望在没有下载工作拷贝时察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:///tmp/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/
```

更多细节见第 3.6.4 节 “svn list”。

名称

svn log -- 显示提交日志信息。

概要

svn log [PATH]

svn log URL [PATH...]

描述

缺省目标是你的当前目录的路径，如果没有提供参数，svn log会显示当前目录下的所有文件和目录的日志信息，你可以通过指定路径来精炼结果，一个或多个修订版本，或者是任何两个的组合。对于本地路径的缺省修订版本范围BASE:1。

如果你只是指定一个URL，就会打印这个URL上所有的日志信息，如果添加部分路径，只有这条路径下的URL信息会被打印，URL缺省的修订版本范围是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

变化

无

是否访问版本库

是

选项

--revision (-r) REV
--quiet (-q)
--verbose (-v)
--targets FILENAME
--stop-on-copy
--incremental
--xml
--username USER

```
--password PASS
--no-auth-cache
--non-interactive
--config-dir DIR
```

例子

你可以在顶级目录运行svn log看到工作拷贝中所有修改的路径的日志信息：

```
$ 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
…
```

当你想连接多个队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。

提示

如果你在特定路径和修订版本运行svn log，输出结果为空

```
$ svn log -r 20 http://svn.red-bean.com/untouched.txt
------------------------------------------------------------------------
```

这只意味着这条路径在那个修订版本没有修改，如果从版本库的顶级目录运行这个命令，或者是你知道那个修订版本修改了那个文件，你可以明确的指定它：

```
$ svn log -r 20 touched.txt
------------------------------------------------------------------------
r20 | sally | 2003-01-17 22:56:19 -0600 (Fri, 17 Jan 2003) | 1 line

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 -r N:M SOURCE[@REV] [WCPATH]
```

描述

第一种形式，源URL用修订版本号N和M指定，这是要比较的两组源文件，如果省略修订版本号，缺省是HEAD。

第二种形式，SOURCE可以是URL或者工作拷贝项目，与之对应的URL会被使用。在修订版本号N和M的URL定义了要比较的两组源。

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。

不像svn diff，合并操作在执行时会考虑文件的祖先，当你从一个分支合并到另一个分支，而这两个分支有各自重命名的文件时，这一点会非常重要。

别名

无

变化

工作拷贝

是否访问版本库

只有在对URL操作时会

选项

```
--revision (-r) REV
--non-recursive (-N)
--quiet (-q)
--force
--dry-run
```

```
--diff3-cmd CMD
--ignore-ancestry
--username USER
--password PASS
--no-auth-cache
--non-interactive
--config-dir DIR
```

例子

将一个分支合并回主干（假定你有一份主干的工作拷贝，分支在修订版本250创建）：

```
$ svn merge -r 250:HEAD http://svn.red-bean.com/repos/branches/my-branch
U  myproj/tiny.txt
U  myproj/thhgttg.txt
U  myproj/win.txt
U  myproj/flo.txt
```

如果你的分支在修订版本23，你希望将主干的修改合并到分支，你可以在你的工作拷贝的分支上这样做：

```
$ svn merge -r 23:30 file:///tmp/repos/trunk/vendors
U  myproj/thhgttg.txt
...
```

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 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

创建一个目录，名字是提供的PATH或者URL的最后一部分，工作拷贝PATH指定的目录会预定要添加，而通过URL指定的目录会作为一次立即提交在版本库建立。多个目录URL的提交是原子操作，在两种情况下，中介目录必须已经存在。

别名

无

变化

如果是对URL操作则会影响版本库，否则是工作拷贝

是否访问版本库

只有在对URl操作时会

选项

```
--message (-m) TEXT
--file (-F) FILE
--quiet (-q)
--username USER
--password PASS
--no-auth-cache
--non-interactive
--editor-cmd EDITOR
--encoding ENC
--force-log
--config-dir DIR
```

例子

在工作拷贝创建一个目录：

```
$ svn mkdir newdir
A         newdir
```

在版本库创建一个目录（立即提交，所以需要日志信息）：

```
$ 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。

注意

Subversion不支持在工作拷贝和URL之间拷贝，此外，你只可以一个版本库内移动文件—Subversion不支持跨版本库的移动。

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

变化

如果对URL操作会影响版本库，否则只影响工作拷贝

是否访问版本库

只有在对URL操作时会

选项

--message (-m) TEXT
--file (-F) FILE
--revision (-r) REV

```
--quiet (-q)
--force
--username USER
--password PASS
--no-auth-cache
--non-interactive
--editor-cmd EDITOR
--encoding ENC
--force-log
--config-dir DIR
```

例子

移动工作拷bede一个文件：

```
$ svn move foo.c bar.c
A         bar.c
D         foo.c
```

移动版本库中的一个文件（一个立即提交，所以需要提交信息）：

```
$ 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 [URL]
```

描述

这会删除文件、目录或修订版本的属性。第一种形式是在工作拷贝删除版本化属性，第二种是在一个版本库修订版本中删除未版本化的属性。

别名

pdel, pd

变化

如果对URL操作会影响版本库，否则只影响工作拷贝

是否访问版本库

只有在对URL操作时会

选项

```
--quiet (-q)
--recursive (-R)
--revision (-r) REV
--revprop
--username USER
--password PASS
--no-auth-cache
--non-interactive
--config-dir DIR
```

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 -- 修改一个或多个版本控制之下文件的属性。

概要

```
svn propedit PROPNAME PATH...

svn propedit PROPNAME --revprop -r REV [URL]
```

描述

使用喜欢的编辑器编辑一个或多个属性，第一种形式是在工作拷贝编辑版本化的属性，第二种形式是远程编辑未版本化的版本库修订版本属性。

别名

pedit, pe

变化

如果对URL操作会影响版本库，否则只影响工作拷贝

是否访问版本库

只有在对URL操作时会

选项

```
--revision (-r) REV
--revprop
--username USER
--password PASS
--no-auth-cache
--non-interactive
--encoding ENC
--editor-cmd EDITOR
--config-dir DIR
```

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

打印一个文件、目录或修订版本的一个属性的值，第一种形式是打印工作拷贝中一个或多个项目的版本化的属性，第二种形式是远程打印版本库修订版本的未版本化的属性。属性的详情见第 7.2 节 “属性”。

别名

pget, pg

变化

如果对URL操作会影响版本库，否则只影响工作拷贝

是否访问版本库

只有在对URL操作时会

选项

```
--recursive (-R)
--revision (-r) REV
--revprop
--strict
--username USER
--password PASS
--no-auth-cache
--non-interactive
--config-dir DIR
```

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.
```

名称

svn proplist -- 列出所有的属性。

概要

```
svn proplist [TARGET[@REV]...]

svn proplist --revprop -r REV [URL]
```

描述

列出文件、目录或修订版本的属性，第一种形式是列出工作拷贝的所有版本化的属性，第二种形式是列出版本库修订版本的未版本化的属性。

别名

plist, pl

变化

如果对URL操作会影响版本库，否则只影响工作拷贝

是否访问版本库

只有在对URL操作时会

选项

```
--verbose (-v)
--recursive (-R)
--revision (-r) REV
--quiet (-q)
--revprop
--username USER
--password PASS
--no-auth-cache
--non-interactive
--config-dir DIR
```

例子

你可以使用proplist察看你的工作拷贝的一个项目的属性：

```
$ svn proplist foo.c
```

```
Properties on 'foo.c':
  svn:mime-type
  svn:keywords
  owner
```

通过--verbose选项，svn proplist也可以非常便利的显示属性的值：

```
$ svn proplist --verbose foo.c
Properties on 'foo.c':
  svn:mime-type : text/plain
  svn:keywords : Author Date Rev
  owner : sally
```

名称

svn propset -- 设置文件、目录或者修订版本的属性PROPNAME为PROPVAL。

概要

```
svn propset PROPNAME [PROPVAL | -F VALFILE] PATH...

svn propset PROPNAME --revprop -r REV [PROPVAL | -F VALFILE] [URL]
```

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。

提示

Subversion有一系列“特殊的”影响行为方式的属性，关于这些属性的详情请见第 7.2.3 节 “特别属性”。

别名

pset, ps

变化

如果对URL操作会影响版本库，否则只影响工作拷贝

是否访问版本库

只有在对URL操作时会

选项

```
--file (-F) FILE
--quiet (-q)
--revision (-r) REV
--targets FILENAME
--recursive (-R)
--revprop
--username USER
--password PASS
--no-auth-cache
--non-interactive
--encoding ENC
```

```
--force
--config-dir DIR
```

例子

设置文件的mimetype：

```
$ 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'
```

最后，你可以告诉propset从一个文件得到输入，你甚至可以使用这个方式来设置一个属性为二进制内容：

```
$ 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 5.2.1 节 “钩子脚本”。

名称

svn resolved -- 删除工作拷贝文件或目录的“冲突”状态。

概要

```
svn resolved PATH...
```

描述

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，从而重新允许路径提交；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 3.5.4 节 “解决冲突（合并别人的修改）”。

别名

无

变化

工作拷贝

是否访问版本库

否

选项

```
--targets FILENAME
--recursive (-R)
--quiet (-q)
--config-dir DIR
```

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

恢复所有对文件和目录的修改，并且解决所有的冲突状态。svn revert不会只是恢复工作拷贝中一个项目的内容，也包括了对属性修改的恢复。最终，你可以使用它来取消所有已经做过的预定操作（例如，文件预定要添加或删除可以“恢复”）。

别名

无

变化

工作拷贝

是否访问版本库

否

选项

```
--targets FILENAME
--recursive (-R)
--quiet (-q)
--config-dir DIR
```

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--recursive选项：

```
$ svn revert --recursive .
```

```
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
```

注意

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。

输出的前五列都是一个字符宽，每一列给出了工作拷贝项目的每一方面的信息。

第一列指出一个项目的是添加、删除还是其它的修改。

' '
  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。

'C'
  项目与从版本库的更新冲突。

'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。

'!'
  项目已经丢失（例如，你使用svn移动或者删除了它）。这也说明了一个目录不是完整的（一个检出或更新中断）。

'~'
项目作为一种对象（文件、目录或链接）纳入版本控制，但是已经被另一种对象替代。

第二列告诉一个文件或目录的属性的状态。

' '
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。

第三列只在工作拷贝锁定时才会出现。

' '
项目没有锁定。

'L'
项目已经锁定。

第四列只在预定包含历史添加的项目出现。

' '
没有历史预定要提交。

'+'
历史预定要伴随提交。

第五列只在项目跳转到相对于它的父目录时出现（见第 4.5 节 “转换工作拷贝”）。

' '
项目是它的父目录的孩子。

'S'
项目已经转换。

过期信息出现在第八列（只在使用--show-updates选项时出现）。

' '
    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。

余下的字段是可变得宽度且使用空格分隔，如果使用--show-updates或--verbose选项，工作修订版本是下一个字段。

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

变化

无

是否访问版本库

只有使用--show-updates时会访问

选项

```
--show-updates (-u)
--verbose (-v)
--non-recursive (-N)
--quiet (-q)
--no-ignore
--username USER
--password PASS
--no-auth-cache
--non-interactive
--config-dir
```

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
```

如果你希望找出工作拷贝哪些文件是最新的，使用--show-updates选项（这不会对工作拷贝有任何修改）。这里你会看到wc/foo.c在上次更新后有了修改：

```
$ 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 只会在过期的项目（如果你运行svn update，就会更新的项目）旁边安置一个星号。--show-updates不会导致状态列表反映项目的版本库版本。

最后，是你能从status子命令得到的所有信息：

```
$ 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
```

关于svn status的更多例子可以见第 3.5.3.1 节 “svn status”。

名称

svn switch -- 把工作拷贝更新到别的URL。

概要

```
svn switch URL [PATH]

switch --relocate FROM TO [PATH...]
```

描述

这个子命令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 “转换工作拷贝”。

别名

sw

变化

工作拷贝

是否访问版本库

是

选项

```
--revision (-r) REV
--non-recursive (-N)
--quiet (-q)
--diff3-cmd CMD
--relocate
--username USER
--password PASS
--no-auth-cache
--non-interactive
--config-dir DIR
```

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.
```

为了跳转回来，只需要提供最初取出工作拷贝的版本库URL：

```
$ 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

如果你不希望跳转所有的工作拷贝，你可以只跳转一部分。

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。使用--relocate来做这种替换，没有文件内容会改变，访问的版本库也不会改变。只是像在工作拷贝.svn/运行了一段Perl脚本s/OldRoot/NewRoot/。

```
$ svn checkout file:///tmp/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:///tmp/repos'

$ svn switch --relocate file:///tmp/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：

- 如果工作拷贝需要反映一个版本库的新目录，只需要使用svn switch。
- 如果你的工作拷贝还是反映相同的版本库目录，但是版本库本身的位置改变了，使用svn switch --relocate。

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。

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

D
: 删除

U
: 更新

C
: 冲突

G
: 合并

第一列的字符反映文件本身的更新，而第二列会反映文件属性的更新。

别名

up

变化

工作拷贝

是否访问版本库

是

选项

```
--revision (-r) REV
--non-recursive (-N)
--quiet (-q)
--diff3-cmd CMD
--username USER
--password PASS
--no-auth-cache
--non-interactive
--config-dir DIR
```

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 A，Subversion对于CVS用户）：

```
$ 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是一个用来监控和修改Subversion版本库的管理工具，详情请见第 5.3.1.2 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 Switches

--bdb-log-keep
（Berkeley DB特定）关闭数据库日志自动日志删除功能。

--bdb-txn-nosync
（Berkeley DB特定）当提交数据库事务时关闭fsync。

--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。

--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/。

--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。

9.2.2. svnadmin Subcommands

名称

svnadmin create -- 创建一个新的空的版本库。

概要

```
svnadmin create REPOS_PATH
```

描述

在提供的路径上创建一个新的空的版本库，如果提供的目录不存在，它会为你创建。[1]

选项

```
--bdb-txn-nosync
--bdb-log-keep
--config-dir DIR
--fs-type TYPE
```

例子

创建一个版本库就是这样简单：

```
$ svnadmin create /usr/local/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 /usr/local/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因为历史原因之存在于1.0.x，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

--revision (-r)
--quiet

名称

svnadmin dump -- 将文件系统的内容转储到标准输出。

概要

```
svnadmin dump REPOS_PATH [-r LOWER[:UPPER]] [--incremental]
```

描述

使用“dumpfile”可移植格式将文件系统的内容转储到标准输出，将反馈发送到标准错误，导出的修订版本从LOWER到UPPER。如果没有提供修订版本，会导出所有的修订版本树，如果只提供LOWER，导出一个修订版本树，通常的用法见第 5.3.5 节 “版本库的移植”。

如果Subversion的转储文件很大，你可以使用--deltas来减小svnadmin创建的导出文件的大小。通过这个选项，就不会写出每次修订版本的内容，svnadmin dump只会写出版本之间的区别。然而，创建增量导出文件的坏处—对CPU的要求更高，svndumpfilter不可以对这个文件操作，而且非增量的转储文件可以更好的压缩。

选项

```
--revision (-r)
--incremental
--quiet
--deltas
```

例子

转储整个版本库：

```
$ svnadmin dump /usr/local/svn/repos
SVN-fs-dump-format-version: 1
Revision-number: 0
* Dumped revision 0.
Prop-content-length: 56
Content-length: 56
…
```

从版本库增量转储一个单独的事务：

```
$ svnadmin dump /usr/local/svn/repos -r 21 --incremental
* Dumped revision 21.
```

```
SVN-fs-dump-format-version: 1
Revision-number: 21
Prop-content-length: 101
Content-length: 101
…
```

名称

svnadmin help

概要

svnadmin help [SUBCOMMAND...]

描述

当你困于一个没有网络连接和本书的沙漠岛屿时，这个子命令非常有用。

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

这个子命令会制作一个版本库的完全“热”拷贝，包括所有的钩子，配置文件，当然还有数据库文件。如果你传递--clean-logs选项，svnadmin会执行热拷贝操作，然后删除不用的Berkeley DB日志文件。你可以在任何时候运行这个命令得到一个版本库的安全拷贝，不管其它进程是否使用这个版本库。

选项

--clean-logs

名称

svnadmin list-dblogs -- 询问Berkeley DB在给定的Subversion版本库有哪些日志文件存在（只有在版本库使用bdb作为后端时使用）。

概要

```
svnadmin list-dblogs REPOS_PATH
```

描述

Berkeley DB创建了记录所有版本库修改的日志，允许我们在面对大灾难时恢复。除非你开启了DB_LOGS_AUTOREMOVE，否则日志文件会累积，尽管大多数是不再使用可以从磁盘删除得到空间。详情见第 5.3.3 节 “管理磁盘空间”。

名称

svnadmin list-unused-dblogs -- 询问Berkeley DB哪些日志文件可以安全的删除（只有在版本库使用bdb作为后端时使用）。

概要

```
svnadmin list-unused-dblogs REPOS_PATH
```

描述

Berkeley DB创建了记录所有版本库修改的日志，允许我们在面对大灾难时恢复。除非你开启了DB_LOGS_AUTOREMOVE，否则日志文件会累积，尽管大多数是不再使用，可以从磁盘删除得到空间。详情见第 5.3.3 节 “管理磁盘空间”。

例子

删除所有不用的日志文件：

```
$ svnadmin list-unused-dblogs /path/to/repos
/path/to/repos/log.0000000031
/path/to/repos/log.0000000032
/path/to/repos/log.0000000033

$ svnadmin list-unused-dblogs /path/to/repos | xargs rm
## disk space reclaimed!
```

名称

svnadmin load -- 从标准输出读取“转储文件”格式流。

概要

```
svnadmin load REPOS_PATH
```

描述

从标准输出读取“转储文件”格式流，提交新的修订版本到版本库文件系统，发送进展反馈到标准输出。

选项

```
--quiet (-q)
--ignore-uuid
--force-uuid
--parent-dir
```

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 /usr/local/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 /usr/local/svn/restored < repos-
<<< Started new txn, based on original revision 1
     * adding path : test ... done.
     * adding path : test/a ... done.
...
```

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 5.3.2 节 “版本库清理”。

例子

列出版本库所有突出的事物。

```
$ svnadmin lstxns /usr/local/svn/repos/
1w
1x
```

名称

svnadmin recover -- 将版本库数据库恢复到稳定状态（只有在版本库使用bdb作为后端时使用）。

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 /usr/local/svn/repos/
Repository lock acquired.
Please wait; recovering the repository may take some time...

Recovery completed.
The latest repos revision is 34.
```

恢复数据库需要一个版本库的独占锁，如果另一个进程访问版本库，svnadmin recover会出错：

```
$ svnadmin recover /usr/local/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 /usr/local/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 rmtxns -- 从版本库删除事物。

概要

svnadmin rmtxns REPOS_PATH TXN_NAME...

描述

删除版本库突出的事物，更多细节在第 5.3.2 节 “版本库清理”。

选项

--quiet (-q)

例子

删除命名的事物：

$ svnadmin rmtxns /usr/local/svn/repos/ 1w 1x

很幸运，lstxns的输出作为rmtxns输入工作良好：

$ svnadmin rmtxns /usr/local/svn/repos/ `svnadmin lstxns /usr/local/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。

这与使用svn propset --revprop设置某一修订版本的svn:log属性效果一样，除了你也可以使用--bypass-hooks选项绕过的所有pre-或post-commit的钩子脚本，这在pre-revprop-change钩子脚本中禁止修改修订版本属性时非常有用。

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

```
--revision (-r) ARG
--bypass-hooks
```

例子

设置修订版本19的日志信息为文件msg的内容：

```
$ svnadmin setlog /usr/local/svn/repos/ -r 19 msg
```

名称

svnadmin verify -- 验证版本库保存的数据。

概要

svnadmin verify REPOS_PATH

描述

如果希望验证版本库的完整性可以运行这个命令，原理是通过在内部转储遍历所有的修订版本并且丢掉输出。

例子

检验挂起的版本库：

```
$ svnadmin verify /usr/local/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。

因为svnlook通过直接版本库访问（因此只可以在保存版本库的机器上工作）工作，所以他通过版本库的路径访问，而不是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

svnlook的选项是全局的，就像svn和svnadmin；然而，大多数选项只会应用到一个子命令，因为svnlook的功能是（有意的）限制在一定范围的。

--no-diff-deleted
防止svnlook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。

--revision (-r)
指定要进行检查的特定修订版本。

--transaction (-t)

指定一个希望检查的特定事物ID。

--show-ids
显示文件系统树中每条路径的文件系统节点修订版本ID。

9.3.2. svnlook

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)
--transaction (-t)
```

例子

svnlook author垂手可得，但是并不令人激动：

```
$ svnlook author -r 40 /usr/local/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)
--transaction (-t)
```

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 /usr/local/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，也是在第一列使用“svn update样式的”状态字符：A表示添加，D表示删除，U表示更新（修改）。

选项

```
--revision (-r)
--transaction (-t)
```

例子

显示在测试版本库修订版本39修改的文件列表：

```
$ svnlook changed -r 39 /usr/local/svn/repos
A   trunk/vendors/deli/
A   trunk/vendors/deli/chips.txt
A   trunk/vendors/deli/sandwich.txt
A   trunk/vendors/deli/pickle.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)
--transaction (-t)
```

例子

显示测试版本库修订版本40的日期：

```
$ svnlook date -r 40 /tmp/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

```
--revision (-r)
--transaction (-t)
--no-diff-deleted
```

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 /usr/local/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
===================================================================
```

如果一个文件有非文本的svn:mime-type属性，区别不会明确显示。

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)
--transaction (-t)
```

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 /usr/local/svn/repos
trunk/vendors/deli/
```

名称

svnlook help

概要

Also svnlook -h and svnlook -?.

描述

显示svnlook的帮助信息，这个命令如同svn help的兄弟，也是你的朋友，即使你从不调用它，并且忘掉了邀请它加入你的上一次聚会。

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

```
--revision (-r)
--show-ids
```

例子

这显示了实例版本库中作为修订版本20的路径/tags/1.0的历史输出。

```
$ svnlook history -r 20 /usr/local/svn/repos /tags/1.0 --show-ids
REVISION   PATH <ID>
--------   ---------
      19   /tags/1.0 <1.2.12>
      17   /branches/1.0-rc2 <1.1.10>
      16   /branches/1.0-rc2 <1.1.x>
      14   /trunk <1.0.q>
      13   /trunk <1.0.o>
      11   /trunk <1.0.k>
       9   /trunk <1.0.g>
       8   /trunk <1.0.e>
       7   /trunk <1.0.b>
       6   /trunk <1.0.9>
       5   /trunk <1.0.7>
       4   /trunk <1.0.6>
       2   /trunk <1.0.3>
       1   /trunk <1.0.2>
```

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、日志信息大小和日志信息。

选项

```
--revision (-r)
--transaction (-t)
```

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 /usr/local/svn/repos
sally
2003-02-22 17:44:49 -0600 (Sat, 22 Feb 2003)
15
Rearrange lunch.
```

名称

svnlook log -- 打印日志信息。

概要

```
svnlook log REPOS_PATH
```

描述

打印日志信息。

选项

```
--revision (-r)
--transaction (-t)
```

例子

这显示了实例版本库在修订版本40的日志输出：

```
$ svnlook log /tmp/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)
--transaction (-t)
```

例子

这显示了HEAD修订版本中文件/trunk/sandwich的“seasonings”属性的值：

```
$ svnlook pg /usr/local/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)
--transaction (-t)
--verbose (-v)
```

例子

这显示了HEAD修订版本中/trunk/README的属性名称：

```
$ svnlook proplist /usr/local/svn/repos /trunk/README
  original-author
  svn:mime-type
```

这与前一个例子是同一个命令，但是同时显示了属性值：

```
$ svnlook --verbose proplist /usr/local/svn/repos /trunk/README
  original-author : fitz
  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

```
--revision (-r)
--transaction (-t)
--show-ids
```

例子

这会显示实例版本库中修订版本40的树输出（包括节点ID）：

```
$ svnlook tree -r 40 /usr/local/svn/repos --show-ids
/ <0.0.2j>
 trunk/ <p.0.2j>
  vendors/ <q.0.2j>
   deli/ <1g.0.2j>
    egg.txt <1i.e.2j>
    soda.txt <1k.0.2j>
    sandwich.txt <1j.0.2j>
```

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Universal Unique IDentifier（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。

例子

```
$ svnlook uuid /usr/local/svn/repos
e7fe1b91-8cd5-0310-98dd-2f12e793c5e8
```

名称

svnlook youngest -- 打印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。

例子

这显示了在实例版本库显示最年轻的修订版本：

```
$ svnlook youngest /tmp/repos/
42
```

9.4. svnserve

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或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，一个自定义的服务器”。

9.4.1. svnserve选项

不象前面描述的例子，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。

--listen-port=PORT
   在守护进程模式时导致svnserve监听PORT端口。

--listen-host=HOST
   svnserve监听的HOST，可能是一个主机名或是一个IP地址。

--foreground
   当与-d一起使用，会导致svnserve停留在前台，主要用来调试。

--inetd (-i)
导致svnserve使用标准输出/标准输入文件描述符，更准确的是使用inetd作为守护进程。

--help (-h)
显示有用的摘要和选项。

--version
显示版本信息，版本库后端存在和可用的模块列表。

--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。

--tunnel (-t)
导致svnserve以管道模式运行，很像inetd操作的模式（服务于一个标准输入/标准输出的连接），除了连接是用当前uid的用户名预先认证过的这一点。这个选项在客户端使用如ssh之类的管道时使用。

--tunnel-user NAME
与--tunnel选项结合使用；告诉svnserve假定NAME就是认证用户，而不是svnserve进程的UID用户，当希望多个用户通过SSH共享同一个系统帐户，但是维护各自的提交标示符时非常有用。

--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，svnserve进程本身在启动后会一直在后台。

--listen-once (-X)
导致svnserve在svn端口接受一个连接，维护完成它退出。这个选项主要用来调试。

9.5. 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。

如果提供TRAIL_URL，URL的尾端部分用来监测是否WC_PATH本身已经跳转（监测WC_PATH的跳转不需要依赖TRAIL_URL）。

选项

像svnserve，svnversion没有子命令，只有选项。

--no-newline (-n)
   忽略输出的尾端新行。.

--committed (-c)
   使用最后修改修订版本而不是当前的（例如，本地存在的最高修订版本）修订版本。

--help (-h)
   打印帮助摘要。

--version
   打印svnversion，如果没有错误退出。

例子

如果工作拷贝都是一样的修订版本（例如，在更新后那一刻），会打印修订版本：

```
$ svnversion .
4168
```

添加TRAIL_URL来展示工作拷贝不是从你希望的地方跳转过来的：

```
$ svnversion . /repos/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
```

9.6. 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

这个小节主要描述了Subversion Apache配置的每个指示，关于Apache配置Subversion的更多信息见第 6.4 节 “httpd，Apache的HTTP服务器”。

Directives

DAV svn
: 这个指示必须包含在所有Subversion版本库的Directory或Location块中，它告诉httpd使用Subversion的后端，用mod_dav来处理所有的请求。

SVNPath
: 这个指示指定Subversion版本库文件文件系统的位置，在一个Subversion版本库的配置块里，必须提供这个指示或SVNParentPath，但不能同时存在。

SVNSpecialURI
: 指定特定Subversion资源的URI部分（命名空间），缺省是“!svn”，大多数管理员不会用到这个指示。只有那些必须要在版本库中放一个名字为!svn的文件时需要设置。如果你在一个已经使用中的服务器上这样修改，它会破坏所有的工作拷贝，你的用户会拿着叉子和火炬追杀你。

SVNReposName
: 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
: 目录列表所使用的XSL转化的URI，这个指示可选。

SVNParentPath
: 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
: 控制开启和关闭路径为基础的授权，更多细节见第 6.4.4.3 节 “关闭路径为基础的检查”。

附录 A. Subversion对于CVS用户

目录

A.1. 修订版本号现在不同了 294
A.2. 目录的版本 294
A.3. 更多离线操作 295
A.4. 区分状态和更新 296
A.5. 分支和标签 297
A.6. 元数据属性 297
A.7. 冲突解决 298
A.8. 二进制文件和转化 298
A.9. 版本化的模块 298
A.10. 认证 299
A.11. 转化CVS版本库到Subversion 299

这个附录可以作为CVS用户开始使用Subversion的指南，实质上就是鸟瞰这两个系统之间的区别列表，在每一小节，我们会尽可能提供相关章节的引用。

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。

A.1. 修订版本号现在不同了

在CVS中，修订版本号是每文件的，这是因为CVS使用RCS作为后端；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。

在Subversion，版本库看起来像是一个单独的文件系统，每次提交导致一个新的文件系统；本质上，版本库是一堆树，每棵树都有一个单独的修订版本号。当有人谈论“修订版本54”时，他们是在讨论一个特定的树（并且间接来说，文件系统在提交54次之后的样子）。

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。

更多细节见第 2.3.2 节 “修订版本”.

A.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：

- svn add和svn delete现在也工作在目录上了，就像在文件上一样，还有svn copy和svn move也一样。然而，这些命令不会导致版本库即时的变化，相反，工作的项目只是“预定要”添加和删除，在运行svn commit之前没有版本库的修改。
- 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通过在.svn区域偷偷的纪录添加和删除来处理这些问题，当你最后运行svn update，所有的账目会到版本库结算，并且目录的新修订版本号会正确设置。因此，只有在更新之后才可以真正安全地说我们有了一个“完美的”修订版本目录。在大多数时候，你的工作拷贝会保存“不完美的”目录修订版本。

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 2.3.4 节 “修订版本混合的限制”。

A.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。

.svn管理目录维护者与CVS同样的功能，除了它还保存了只读的文件“原始”拷贝，这允许你做许多离线操作：

svn status
   显示你所做的本地修改（见第 3.5.3.1 节 “svn status”）

svn diff
   显示修改的详细信息（见see 第 3.5.3.2 节 “svn diff”）

svn revert
   删除你的本地修改（见第 3.5.3.3 节 “svn revert”）

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。

列表中最后一个子命令是新的；它不仅仅删除本地修改，也会取消如增加和删除的预定操作，这是恢复文件推荐的方式；运行rm file; svn update还可以工作，但是这样侮辱了更新操作的作用，而且，我们在这个主题…

A.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。

cvs status命令有两个目的：第一，显示用户在工作拷贝的所有本地修改，第二，显示给用户哪些文件是最新的。很不幸，因为CVS难以阅读的状态输出，许多CVS用户并没有充分利用这个命令的好处。相反，他们慢慢习惯运行cvs update或cvs update -n来快速查看区别，如果用户忘记使用-n选项，副作用就是将还没有准备好处理的版本库修改合并到工作拷贝。

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。

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
: 联系版本库来确定，然后显示，过时的信息。

-v
: 显示所有版本控制之下的条目。

-N
: 非递归运行（不传递到子目录）。

status命令有两种输出格式，缺省是“简短”格式，本地修改看起来是这样：

```
% svn status
M     ./foo.c
M     ./bar/baz.c
```

如果你指定--show-updates（-u），输出会使用比较长的格式：

```
% 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，有两个新列，如果文件或目录已经过期了，第二列会显示星号。第三列显示工作拷贝项目的修订版本号，在上面的例子里，星号表示faces.html会在更新时更新，而bloo.png是在版本库新加的文件。（bloo.png前面的-表示它不曾存在与工作拷贝。）

最后，你会想看一个常见状态码的快速总结：

A　资源预定要添加
D　资源预定要删除
M　资源有本地修改
C　资源发生冲突（修改不能完全在版本库和工作拷贝之间合并）
X　资源在工作拷贝之外（来自其他版本库，见第 7.2.3.6 节 “svn:externals”）
?　资源不在版本控制之下
!　资源丢失或者不完整（被Subversion以外的工具删除）

Subversion合并了CVS的P和U代码为U，当一个合并或冲突发生，Subversion只会简单得打印G或C，而不是一段完整的描述语句。

关于svn status的详细讨论见第 3.5.3.1 节 “svn status”。

A.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，一直要记住检出项目的trunk（http://svn.example.com/repos/calc/trunk/），而不是项目本身的（http://svn.example.com/repos/calc/）。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]。

A.6. 元数据属性

[1]如果在检出完成之前没有消耗完磁盘空间的话。

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 7.2 节 “属性”。

A.7. 冲突解决

CVS使用内联“冲突标志”来标记冲突，并且在更新时打印C。历史上讲，这导致了许多问题，因为CVS做得还不够。许多用户在它们快速闪过终端时忘记（或没有看到）C，即使出现了冲突标记，他们也经常忘记，然后提交了带有冲突标记的文件。

Subversion通过让冲突更明显来解决这个问题，它记住一个文件是处于冲突状态，在你运行svn resolved之前不会允许你提交修改，详情见第 3.5.4 节 “解决冲突（合并别人的修改）”。

A.8. 二进制文件和转化

在大多数情况下，Subversion比CVS更好的处理二进制文件，因为CVS使用RCS，它只可以存储二进制文件的完整拷贝，但是，从内部原理上讲，Subversion使用二进制区别算法来表示文件的区别，而不管文件是文本文件还是二进制文件。这意味着所有的文件是以微分的（压缩的）形式存放在版本库，小的区别会通过网络传输。

CVS用户需要使用-kb选项来标记二进制文件，防止数据的混淆（因为关键字解释和行结束转化），他们有时候会忘记这样做。

Subversion使用更加异想天开的方法：第一，如果你不明确的告诉它（详情见第 7.2.3.4 节 “svn:keywords”和第 7.2.3.5 节 “svn:eol-style”）这样做，它不会做任何关键字或行结束转化的操作，缺省情况下Subversion会把所有的数据看作字节串，所有的储存在版本库的文件都处于未转化的状态。

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。

为了检测一个上下文的合并是可能的，Subversion检测svn:mime-type属性，如果没有svn:mime-type属性，或者这个属性是文本的（例如text/*），Subversion会假定它是文本的，否则Subversion认为它是二进制文件。Subversion也会在svn import和svn add命令时通过运行一个二进制检测算法来帮助用户。这些命令会做出很好的猜测，然后（如果可能）设置添加文件的svn:mime-type属性。（如果Subversion猜测错误，用户可以删除或手工编辑这个属性。）

A.9. 版本化的模块

不像CVS，Subversion工作拷贝会意识到它检出了一个模块，这意味着如果有人修改了模块的定义（例如添加和删除组件），然后一个对svn update的调用会适当的更新工作拷贝，添加或删除组件。

Subversion定义了模块作为一个目录属性的目录列表：见第 7.4 节 “外部定义”。

A.10. 认证

通过CVS的pserver，你需要在读写操作之前“登陆”到服务器—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。

A.11. 转化CVS版本库到Subversion

或许让CVS用户熟悉Subversion最好的办法就是让他们的项目继续在新系统下工作，这可以简单得通过平淡的把CVS版本库的导出数据导入到Subversion完成，或者是更加完全的方案，不仅仅包括最新数据快照，还包括所有的历史，从一个系统到另一个系统。这是一个非常困难的问题，包括推导保持原子性的修改集，转化两个系统完全不同的分支政策。但是我们还是有许多工具声称至少部分具备了的转化已存在的CVS版本库为Subversion版本库的能力。

其中一个工具是cvs2svn（http://cvs2svn.tigris.org/），是一个Python脚本，最初是Subversion自己的开发社区的成员编写的。其他的如Chia-liang Kao的Subversion的VCP工具（http://svn.clkao.org/revml/branches/svn-perl/）转化器插件，还有Lev Serebryakov的RefineCVS（http://lev.serebryakov.spb.ru/refinecvs/）。这些工具具备不同程度的完成性，也许会具备完全不同的处理CVS历史的方法。无论你决定使用哪个工具，确定要执行尽可能多的验证来确定你可以接受转化结果—毕竟，你曾经努力创建这些历史！

关于最新的转化工具链接列表，可以访问Subversion的网站（http://subversion.tigris.org/project_links.html）。

附录 B. 故障解决

目录

B.1. 共同问题 300
B.1.1. 使用Subversion的问题 300

B.1. 共同问题

在安装和使用Subversion的过程中会有许多问题，有一些在你更加理解Subversion之后会立刻解决，而有一些会引起麻烦，因为你已经习惯于其它版本控制系统的工作方式。也有一些其它的问题不能解决是因为Subversion在一些操作系统上有bug（考虑到Subversion运行系统的广泛性，没有遇到更多已经是一件让人吃惊的事情了）。

下面的列表是从Subversion多年使用过程中编辑出来的，如果你没有在这里发现问题，可以在Subversion网站上察看最新版本的FAQ。如果你还是有问题，可以发送包含你遇到问题详细描述的邮件到<users@subversion.tigris.org>。[1]

B.1.1. 使用Subversion的问题

这里是Subversion的FAQ最流行的问题。

B.1.1.1. 每当我尝试访问版本库，我的Subversion客户端挂起。

你的版本库和数据都没有损坏，如果你的进程直接访问版本库（mod_dav_svn、svnlook、svnadmin或者你通过file://的URL访问），然后它会使用Berkeley DB来访问你的数据。Berkeley DB是一个日志系统，意味着在做一件事之前会记录下来所有的事情，如果你的进程被中断（例如一个终止信号或段错误），这样就会留下一个锁文件，还有一个描述未完成业务的日志文件。任何其它尝试访问数据库的进程会挂起，等待锁文件消失。为了唤醒你的版本库，你需要询问Berkeley DB是否结束工作，或者是恢复数据库到前一个已知的稳定状态。

确定你使用数据库的拥有者和管理者用户来运行这个命令，而不是root或是其它会在db目录产生root拥有文件的用户，这些文件不可以由管理数据库的非root用户打开，通常是你或你的Apache进程。也要确定在恢复时有正确的umask设置，因为如果失败会把允许访问版本库的用户组锁在外面。

简单的运行：

```
$ svnadmin recover /path/to/repos
```

[1] 记住你所能提供的设置细节和问题的数量与从邮件列表得到答案的可能性成正比，鼓励你包括所有的事情，除了早饭吃了什么和你妈妈的娘家姓。

一旦这个命令完成，检查版本库db/目录的访问限制。

B.1.1.2. 每当我尝试运行svn，它告诉我工作拷贝已经锁定。

Subversion的工作拷贝，就像Berkeley DB使用日志机制来执行所有的操作，也就是它会在事情发生前记录所有的操作。如果svn在一个动作中被中断，就会留下一个或多个锁文件以及相关的描述未完成动作的日志文件。（svn status会在锁定的目录前面显示一个L。）

任何其它尝试访问工作拷贝的进程会在看到锁定后会提示失败，为了唤醒工作拷贝，你需要告诉客户端完成工作，做为修正，在你的工作拷贝顶级目录运行这个命令：

```
$ svn cleanup
```

B.1.1.3. 我在查找和打开版本库时得到错误，而我知道我的版本库URL是正确的。

见第 B.1.1.1 节 “每当我尝试访问版本库，我的Subversion客户端挂起。”。

你也许也遇到了一个版本库访问权限问题，见第 6.5 节 “支持多种版本库访问方法”。

B.1.1.4. 我怎样在file://的URL中指定一个Windows驱动器盘符？

见版本库的URL。

B.1.1.5. 通过网络对Subversion版本库进行写操作发生问题。

如果本地访问的导入工作正常：

```
$ mkdir test
$ touch test/testfile
$ svn import test file:///var/svn/test -m "Initial import"
Adding         test/testfile
Transmitting file data .
Committed revision 1.
```

但不是从一个远程主机：

```
$ svn import test http://svn.red-bean.com/test -m "Initial import"
harry's password: xxxxxxx
```

```
svn_error: … The specified activity does not exist.
```

如果REPOS/dav/目录对httpd进程不是可写的我们会看这些，检查权限来确定Apache的httpd进程可以写访问dav/目录（当然也同样对于db/目录）。

B.1.1.6. 在Windows XP下，Subversion服务器有时候看起来发送损坏的数据。

你需要安装Windows XP Service Pack 1来修正操作系统的TCP/IP堆栈bug，你可以查看http://support.microsoft.com/default.aspx?scid=kb;EN-US;q317949来得到这个Service Pack的所有信息。

B.1.1.7. 跟踪Subversion客户端和Apache服务器通话最好的方法是什么？

使用Ethereal来偷听对话：

注意

如下的指导针对Ethereal的图形化版本，不是应用在命令行版本（二进制文件通常叫做tethereal）。

- 打开Capture菜单，选择Start。
- Filter的port输入80，关闭promiscuous模式。
- 运行Subversion客户端。
- 点击Stop，你现在已经捕捉了，它看起来像是巨大的行列表。
- 点击Protocol列来排序。
- 然后，点击第一个相关的TCP行来选择它。
- 右键，选择Follow TCP Stream，你会看到Subversion客户端的HTTP对话的请求/响应对。

另一种选择，你可以在客户端的servers运行配置文件中设置一个参数，来允许neon调试信息的出现，neon-debug得数字值是头文件ne_utils.h中NE_DBG_*值的组合，设置neon-debug-mask变量为130（例如NE_DBG_HTTP + NE_DBG_HTTPBODY）会导致显示HTTP数据。

你或许也会希望在网络跟踪时关掉压缩，可以通过设置同一个文件的http-compression参数。

B.1.1.8. 我刚刚编译了二进制分发版本，当我尝试检出Subversion，我得

到一个“Unrecognized URL scheme”错误。

Subversion使用一个插件系统来允许访问版本库，当前有三种这样的插件：ra_local允许访问本地版本库，ra_dav允许通过WebDAV访问，而ra_svn允许通过svnserve服务器本地或远程访问。当你尝试执行一个Subversion操作，程序会根据URL模式动态的加载一个插件，file://的URL会加载ra_local，而http://的URL会尝试ra_dav。

你看到的错误意味着动态链接器/加载器不能发现需要加载的插件，这通常是因为你使用共享库编译Subversion，然后尝试在没有首先运行make install时运行它。另一个可能的原因是你运行了make install，但是库安装的位置动态链接器/加载器不能识别。在Linux，你可以通过在库目录添加/etc/ld.so.conf并运行ldconfig来允许链接器/加载器找到这些库。如果你不希望这样做，或者是你没有root权限，你可以直接在LD_LIBRARY_PATH环境变量中指定库目录。

B.1.1.9. 为什么svn revert命令要有一个明确的目标？为什么缺省不是递归的？它的行为方式与大多数其它子命令不同。

一句话：它有自己的好处。

Subversion把保护你的数据作为非常高的优先级，对于已经版本化的文件的修改，和预定要添加到版本控制的文件，必须小心对待。

让svn revert命令需要一个明确的目标—即使目标只是“.”—是实现这个目的的一个方法。这个要求（同样的还有要求使用--recursive来实现递归的行为）是为了让你清楚自己所做的事情，因为一旦你的文件被恢复，你的本地修改就会永远消失。

B.1.1.10. 当我启动Apache，mod_dav_svn抱怨说发现一个“bad database version”，它发现了db-3.X而不是db-4.X。

你的apr-util链接了DB-3，而svn链接了DB-4，很不幸，DB对象并没有区别。当mod_dav_svn加载到Apache的处理空间，它无法解析针对apr-util的DB-3的对象名称。

解决方案是确定apr-util针对DB-4编译，你可以通过指定apr-util或Apache的选项来完成这一点："--with-dbm=db4 --with-berkeley-db=/the/db/prefix"。

B.1.1.11. 我在RedHat 9得到“Function not implemented”错误，无法工作，我如何修正这个问题？

这不是Subversion的问题，但是经常影响Subversion用户。

RedHat 9和Fedora分发版本中包括了Berkeley DB库，依赖于为NPTL（the Native Posix Threads Library）内核支持，RedHat得内核提供了内置的支持，但是如果你编译了你的内核，你或许不再有NPTL的支持，所以这种情况下你会看到这样的错误：

```
svn: Berkeley DB error
svn: Berkeley DB error while creating environment for filesystem tester/db:
Function not implemented
```

可以用以下的任意一种方法修正这个问题：

- 重新为你使用的内核编译db4。
- 使用RedHat 9的内核。
- 为你使用的内核应用NPTL补丁。
- 使用最近的（2.5.x）包括NPTL支持的内核。
- 检查环境变量LD_ASSUME_KERNEL是否设置为2.2.5，如果是，在运行Subversion（Apache）之前取消设置。（在RedHat 9运行Wine或Winex时你通常会设置这个变量）

B.1.1.12. 为什么日志说通过Apache（ra_dav）提交或导入的文件“(no author)”？

如果你允许通过Apache的匿名写访问版本库，Apache从不会要求客户端的用户名，而且在写操作中没有认证，因此Subversion对于谁做的操作一无所知，这导致了这样的日志：

```
$ svn log
------------------------------------------------------------------------
rev 24:  (no author) | 2003-07-29 19:28:35 +0200 (Tue, 29 Jul 2003)
...
```

阅读如何添加认证在第 6 章 配置服务器。

B.1.1.13. 我偶然在Windows得到“Access Denied”错误，它们看起来随即出现。

这看起来是因为不同的监控文件系统变化（杀毒软件、目录服务和COM+事件通知服务）的Windows服务。这不是Subversion的bug，让修正变得很困难。当前状态的研究总结在http://www.contactor.se/~dast/svn/archive-2003-10/0136.shtml，一个用来减少大多数人发生概率的工作在修订版本7598已经实现。

B.1.1.14. 在FreeBSD，某些操作（特别是svnadmin create）有时会挂起。

这通常是因为系统缺乏可用的信息量，Subversion一次次询问APR产生随机数来创建UUID，特定操作系统会阻止高质量的随机性，你可能需要配置系统从硬盘和网络中断等资源中收集信息，咨询你的系统管理员，明确random(4)和rndcontrol(8)怎样影响这些变化。另一个工作区让APR根据/dev/urandom而不是/dev/random编译。

B.1.1.15. 我可以在web浏览器看到我的版本库，但是svn checkout给我一个301 Moved Permanently错误。

这意味着你的httpd.conf错误的配置，通常这个错误发生在你定义Subversion虚拟“location”时使之同时存在于两个不同的范围。

例如，如果你已经以<Location /www/foo>导出了一个版本库，但是你也已经设置了DocumentRoot为/www，然后你会陷入麻烦，当请求要得到/www/foo/bar，Apache不知道是去找一个DocumentRoot下的真实文件/foo/bar还是询问mod_dav_svn从/www/foo版本库得到一个文件/bar，通常是前一种情况发生作用，因此得到"Moved Permanently"错误。

解决方案是确定你的版本库的<Location>没有重叠，或者是没有存在于已经作为普通web共享暴露的目录。

B.1.1.16. 我尝试察看我的文件的一个老版本，但是svn告诉我“path not found”。

Subversion的一个好的特性是版本库理解拷贝和重命名，并且保留历史联系。举个例子，如果你拷贝/trunk到/branches/mybranch，然后版本库理解为在分支的每个文件都在trunk有个“前辈”。运行svn log --verbose会显示历史拷贝，所以你可以看到重命名：

```
r7932 | joe | 2003-12-03 17:54:02 -0600 (Wed, 03 Dec 2003) | 1 line
Changed paths:
   A /branches/mybranch (from /trunk:7931)
```

很不幸，当版本库意识到拷贝和重命名，版本1.0的几乎所有的svn客户端子命令还没有意识到。svn diff、svn merge和svn cat应该理解这些重命名，但是它们没有。它们是1.0之后的特性，举个例子，如果你询问svn diff比较两个早期版本的/branches/mybranch/foo.c，这个命令不会自动理解为实际上我们是要比较两个版本的/trunk/foo.c，因为这个重命名。相反，你会看到一个错误说这个分支路径在早期的修订版本并不存在。

解决所有此类问题的方法是你自己的调查，也就是：你需要知道所有的重命名路径，自己使用svn log -v去发现，然后明确地告诉svn客户端，例如，我们不应该运行

```
$ svn diff -r 1000:2000 http://host/repos/branches/mybranch/foo.c
```

```
svn: Filesystem has no item
svn: '/branches/mybranch/foo.c' not found in the repository at revision 1000
```

...而会运行

```
$ svn diff -r1000:2000 http://host/repos/trunk/foo.c
...
```

附录 C. WebDAV和自动版本化

目录

C.1. WebDAV基本概念 307
C.1.1. 仅是平常的WebDAV 307
C.1.2. DeltaV扩展 308
C.2. Subversion和DeltaV 309
C.2.1. 影射Subversion到DeltaV 309
C.2.2. 自动版本化支持 310
C.2.3. 选择mod_dav_lock 311
C.3. 自动版本化交互性 312
C.3.1. Win32网络文件夹 312
C.3.2. Mac OS X 312
C.3.3. Unix: Nautilus 2 313
C.3.4. Linux davfs2 313

WebDAV是HTTP的一个扩展，作为一个文件共享的标准不断发展。当今的操作系统变得极端的web化，许多内置了对装配WebDAV服务器导出的“共享”的支持。

如果你使用Apache/mod_dav_svn作为你的Subversion网络服务器，某种程度上，你也是在运行一个WebDAV服务器。这个附录提供了这种协议一些背景知识，Subversion如何使用它，Subversion如何和认识WebDAV的软件交互工作。

C.1. WebDAV基本概念

这个小节提供了对WebDAV背后思想的一个非常简短和普通的总体看法，为理解WebDAV在客户端和服务器之间的兼容性问题打下基础。

C.1.1. 仅是平常的WebDAV

RFC 2518为HTTP 1.1定义了一组概念和附加扩展方法来把web变成一个更加普遍的读/写媒体，基本思想是一个WebDAV兼容的web服务器可以像普通的文件服务器一样工作；客户端可以装配类似于NFS或SMB的WebDAV“共享”。

然而，必须注意到RFC 2518并没有提供任何版本控制模型，尽管DAV中有“V”。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。[1]

这是基本的WebDAV引入的新概念和方法：

新的写方法

[1]因为这个原因，一些人开玩笑说WebDAV的客户端是“WebDA”客户端！

超出了标准HTTP的PUT方法（用来创建和覆盖web资源），WebDAV定义了新的COPY和MOVE方法用来复制或重新组织资源。

集合
这是一个简单的WebDAV术语用来表示一组资源（URI），在大多数情况下，你可以说以“/”结尾的东西是一个集合，文件资源可以使用PUT方法写或创建，而集合资源使用MKCOL方法创建。

属性
这与Subversion中是同一个思想—附加在文件和集合上的元数据，一个客户端可以使用新方法PROPFIND列出或检索附加在一个资源上的属性，也可以使用PROPPATCH方法修改这些属性。一些属性是完全由用户控制的（例如，一个“color”属性），还有一些是WebDAV服务器创建和控制的（例如，一个保存文件最后修改时间的属性）。前一种叫做“dead”属性，后一种叫做“live”属性。

锁定
WebDAV服务器可以决定为客户端提供一个锁定特性—这部分的规范是可选的，尽管大多数WebDAV服务器提供了这个特性。如果提供这个特性，客户端可以使用新的LOCK和UNLOCK方法来调节访问资源的过程，在大多数情况下是使用独占写锁（在第 2.2.2 节 “锁定-修改-解锁 方案”讨论的），尽管共享写锁也是可以的。

C.1.2. DeltaV扩展

因为RFC 2518漏下了版本概念，另一个小组留下来负责撰写RFC 3253来添加WebDAV的版本化。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。

DeltaV引入了完全的新的首字母缩写，但并不是被逼迫的。想法相当的直接，如下是DeltaV引入的新概念和方法：

每资源的版本化
像CVS和其他版本控制系统，DeltaV假定每个资源有一个潜在的无穷的状态，一个客户端可以使用VERSION-CONTROL放置一个版本控制之下的资源，这创建了一个新的版本控制资源（VCR），每次你修改VCR（通过PUT、PROPPATCH等），这个资源的新状态就会创建，叫做一个版本资源（Version Resource，VR）。VCR和VR还是普通的web资源，使用URL定义，特定的VR也会拥有易读的名字。

服务器端的工作拷贝模型
一些DeltaV服务器支持在服务器创建虚拟“工作区”，所有的工作在这里执行。客户端使用MKWORKSPACE方法来创建私有区域，然后他们说明修改特定的VCR，“把它们检出到”工作拷贝，编辑，然后再次“检入”。在HTTP术语里，方法的顺序可能是CHECKOUT、PUT、CHECKIN，会创建一个新的VR，每个VCR也有一个“历史”资源用来追踪和排序它的各种VR状态。

客户端工作拷贝模型
: 一些DeltaV服务器也支持客户端可以有完全特定VR的私有工作拷贝的思想，（这就是CVS和Subversion的工作原理。）当客户端希望提交修改到服务器，它通过使用MKACTIVITY方法创建一个临时服务器事务（叫做一个活动）开始。然后客户端在每个希望修改的VR上执行一个CHECKOUT操作，这在活动里创建了一些临时“工作资源”，然后可以使用PUT和PROPPATCH方法修改。最后，客户端对每个工作资源执行一个CHECKIN，在每个VCR创建了一个VR，并且整个活动会被删除。

配置
: DeltaV允许你定义叫做“配置”的灵活的VCR集合，不需要对应特定的目录，每个VCR的内容可以使用UPDATE方法指向特定的VR。一旦配置是理想的，客户端可以创建一个整个配置的“快照”，叫做“基线”。客户端使用CHECKOUT和CHECKIN方法捕捉特定的配置状态，很像它们使用这些方法创建VCR的特定VR状态。

扩展性
: DeltaV定义了新方法REPORT，允许客户端和服务器执行自定义的数据交换。客户端发送一个带有包含自定义数据的完全标记的XML主体的REPORT请求；假定服务器理解特定的报告类型，它使用一个等同的XML主体来响应，这个技术与XML-RPC很类似。

自动版本化
: 对大多数，这是DeltaV的“杀手”特性，如果DeltaV服务器支持这个特性，然后基本的WebDAV客户端（例如，那些不知道版本化的客户端）仍然可以对服务器进行写操作，服务器可以悄无声息的执行版本操作。在最简单的例子里，一个从基本的WebDAV客户端发送的无知的PUT可能会被服务器转化为CHECKOUT、PUT、CHECKIN。

C.2. Subversion和DeltaV

所以Subversion与其他DeltaV的兼容性如何？两个字：不好，至少在Subversion 1.0还不好。

当libsvn_ra_dav发送DeltaV到服务器，Subversion客户端不是一个普通目的的DeltaV客户端。实际上，它希望服务器一些自定义的特性（特别是通过自定义的REPORT请求）。更进一步，mod_dav_svn不是一个普通目的的DeltaV服务器，它只实现了DeltaV的一个严格子集，一个更加普通的WebDAV或DeltaV客户端可能与之很好的交互工作，但是只有在服务器非常窄的已经实现的特性范围之内。Subversion开发团队计划会设法在以后的版本中完成普通的WebDAV交互性。

C.2.1. 影射Subversion到DeltaV

这里是多种Subversion客户端如何使用DeltaV操作的一个非常“高级别”描述。在很多种情况下，这些解释过于粗略，这不能作为阅读Subversion源代码和与开发者交谈的替代。

svn checkout/list
对集合执行一个深度为1的PROPFIND来得到直接孩子的列表，对每个孩子执行一个GET（也可能是一个PROPFIND），递归到集合并且重复。

svn commit
使用MKACTIVITY创建一个活动，然后对每个修改项目执行一个CHECKOUT，紧跟一个对新数据的PUT。最终，一个MERGE请求导致一个隐含的对所有工作资源的CHECKIN。

svn update/switch/status/merge/diff
发送一个自定义的描述工作拷贝混合修订版本（和混合URL）状态的REPORT请求，服务器发送一个描述需要更新的项目和文件内容增量数据的响应。解析响应，对于update和switch，在工作拷贝安装新数据，对于diff和merge，与工作拷贝的数据比较，应用修改作为本地修改。

C.2.2. 自动版本化支持

在写作的时候，有一个事实就是这个世界只有很少的DeltaV客户端；RFC 3253一直是相对比较新。然而用户有一些“普通的”客户端，因为几乎所有的现代操作系统现在拥有集成的基本WebDAV客户端，因为这一点，Subversion开发者认识到如果Subversion 1.0可以支持DeltaV自动版本化这一交互特性会是最好的方法。

为了激活mod_dav_svn的自动版本化，使用httpd.conf Location区块的SVNAutoversioning指示，例如：

```
<Location /repos>
  DAV svn
  SVNPath /absolute/path/to/repository
  SVNAutoversioning on
</Location>
```

通常情况下，如果一个原始的WebDAV客户端尝试PUT到你的版本库位置的一个路径，mod_dav_svn会直接拒绝这个请求。（通常只允许对于DeltaV“活动”里面的对于“工作资源”的操作。）通过打开SVNAutoversioning，无论何时，服务器会把PUT请求转化为内部的MKACTIVITY、CHECKOUT、PUT和CHECKIN。一个普通的日志信息是自动生成的，并且创建一个新的文件系统修订版本。

因为有这样多的操作系统已经集成了WebDAV能力，这个特性的用例近似于空想：想象一个普通用户运行Microsoft Windows或Mac OS的办公室，每个电脑“装配”了一个Subversion版本库，作为一个普通的网络共享。他们向普通目录一样操作服务器：从服务器打开文件，修改并且保存回服务器。但在这个幻想中，服务器自动版本化所有的事情，之后，一个系统管理员可以使用Subversion客户端来查找和检索所有旧的版本。

这个幻想是现实吗？完全不是，主要的障碍是Subversion 1.0不支持WebDAV的

LOCK方法UNLOCK，大多数操作系统的DAV客户端尝试LOCK一个直接从DAV装配的网络共享的资源，到目前为止，用户必须要把文件从DAV共享拷贝到本地磁盘，编辑文件，然后再拷贝回去。没有理想的自动版本化，但还是可行的。

C.2.3. 选择mod_dav_lock

Apache模块mod_dav是一个复杂的野兽：它理解和解析所有的WebDAV和DeltaV方法，然而它依赖于后端“提供者”来访问资源本身。

在最简单的化身里，一个用户可以使用mod_dav_fs可以作为mod_dav的提供者，mod_dav_fs使用普通的文件系统来存放文件和目录，只理解平凡的WebDAV方法，不是DeltaV。

在另一方面，Subversion使用mod_dav_svn作为mod_dav的提供者，mod_dav_svn理解除了LOCK以外的所有WebDAV方法，并且理解相当大的DeltaV方法子集，它访问Subversion版本库的数据，而不是真实的文件系统。Subversion 1.0不支持锁定，因为这会非常难于实现，因为Subversion使用拷贝-修改-合并模型。[2]

在Apache httpd-2.0里，mod_dav可以通过追踪私有数据库的锁来支持LOCK方法，假定提供者会乐于接受这一点。在Apache httpd-2.1或以后的版本，这个锁定支持会拆到一个独立的模块，mod_dav_lock。它允许任何mod_dav提供者利用锁数据库的好处，包括mod_dav_svn，即使mod_dav_svn实际上不理解锁定。

感到困惑？

简言之，你可以使用Apache httpd-2.1（或更晚的）的mod_dav_lock来创建一个错觉，也就是mod_dav_svn负责了LOCK操作。确定mod_dav_lock已经编译到httpd或已经在httpd.conf中加载，然后只需要在Location简单的添加如下的DAVGenericLockDB指示：

```
<Location /repos>
  DAV svn
  SVNPath /absolute/path/to/repository
  SVNAutoversioning on
  DavGenericLockDB /path/to/store/locks
</Location>
```

这个技术是一个有危险的业务；在一些情况，mod_dav_svn现在已经接近WebDAV客户端，它宣称接受LOCK请求，但是实际上锁并不是在所有的级别上强制执行。如果第二个WebDAV客户端尝试LOCK锁住同样的资源，然后mod_dav_lock会注意到并且正确的拒绝这个请求，但是完全没有办法来防止一个普通的Subversion客户端使用svn commit来修改文件！如果你使用这个技术，你给用户权利来践踏其他人的修改，更具体一点，一个WebDAV客户端会不小心覆盖普通Subversion客户端提交的修改。

在另一方面，如果你小心设置你的环境变量，你会减轻这个风险，例如，如果所

[2]Subversion可能有一天会开发一个保留检出的锁定模型，可以与拷贝-修改-合并和平相处，但是可能不会立刻发生。

有用户使用WebDAV客户端（而不是Subversion客户端），然后事情变得美好了。

C.3. 自动版本化交互性

在这个小节，我们会描述最普通的原始WebDAV客户端（写作的时刻），和它们是如何与使用SVNAutoversioning指示的mod_dav_svn服务器的运作。RFC 2518是一个有点大，并且有一点太灵活。每个WebDAV客户端的行为都有些许区别，并且产生许多不同的小问题。

C.3.1. Win32网络文件夹

Windows 98、2000和XP有一个集成的WebDAV客户端叫做“网络文件夹”，在Windows 98，这个特性需要明确安装；如果提供，一个“网络文件夹”就会出现在我的电脑，在Windows 2000和XP，只需要简单得打开我的网络位置，运行添加网络位置图标。当出现提示，输入一个WebDAV的URL，我的网络位置中就会出现一个共享文件夹。

大多数写操作对于自动版本化的mod_dav_svn服务器工作正常，但是有一些问题：

- 如果一个Windows XP电脑是一个NT域的成员，它看起来不能连接到WebDAV共享，重复提示要输入用户名和密码，即使Apache服务器没有要求进行认证！如果这个机器不是NT域的一部分，这个共享可以成功装载。

  这个问题源于Windows XP创建网络文件夹快捷方式（.lnk文件）的方法的bug。它有时候会使用“UNC”（Universal Naming Convention）路径来代替WebDAV共享URL，这导致资源管理器尝试使用SMB而不是HTTP来进行连接。

  这个问题的解决方法是在Windows 2000创建.lnk快捷方式，然后拷贝到Windows XP电脑，如果有人可以逆转.lnk文件的格式，也可以使用十六进制编辑器来“修正”快捷方式。

- 一个文件不可以直接在共享中打开编辑；它可能一直是只读的。mod_dav_lock技术也无能为力，因为网络文件夹根本不使用LOCK方法，前面提到的“拷贝、编辑和再拷贝”根本不工作。在共享中的文件可以成功的被本地修改的拷贝覆盖。

C.3.2. Mac OS X

Apple的OS X操作系统也集成了WebDAV客户端，从Finder选择Go菜单的“Connect to Server”，输入一个WebDAV的URL，它会作为一个磁盘在桌面出现，就像任何文件服务器。[3]

很不幸，客户端拒绝与一个自动版本化的mod_dav_svn工作，因为它缺乏LOCK支持，Mac OS X在初始化HTTP的OPTIONS特性交换时会发现缺失了LOCK能力，因而决定以只读方式装配Subversion版本库，之后，不可以进行写操作。为了将版本库按

[3] Unix用户也可以运行mount -t webdav URL /mountpoint。

照读写方式装配，你必须使用前面讨论的mod_dav_lock技巧。一旦锁定看起来工作了，共享会运作良好：文件可以直接以读/写模式打开，尽管每次存储操作会导致客户端对临时位置执行一个PUT，对原文件的DELETE操作和把临时资源MOVE到原文件。每次存盘会产生三个新的Subversion修订版本！

还要警告一点：OS X的WebDAV客户端可以对HTTP重定向完全敏感，如果你不能装配版本库，你可以在httpd.conf开启BrowserMatch指示：

```
BrowserMatch "^WebDAVFS/1.[012]" redirect-carefully
```

C.3.3. Unix: Nautilus 2

Nautilus是GNOME桌面的官方文件管理器/浏览器，它的主页在http://www.gnome.org/projects/nautilus/，只需要在Nautilus窗口中输入一个WebDAV的URL，DAV共享就会像本地磁盘一样出现。

通常情况下，Nautilus 2与自动版本化的mod_dav_svn一起工作相当的好，只是有下面一些警告：

- 任何在共享里直接打开的文件是只读的，即使mod_dav_lock的技巧也看起来无效。Nautilus看起来从没有关注过LOCK方法，“本地拷贝、编辑和拷贝回去”的技巧还可以工作，但是很不幸，Nautilus的覆盖旧文件是通过首先DELETE进行的，这创建了一个额外的修订版本。
- 当覆盖或创建一个文件，Nautilus首先PUT一个空文件，然后使用第二个PUT覆盖它，这创建了两个Subversion文件系统修订版本，而不是一个。
- 当删除了一个集合，它对每个独立的孩子而不是集合本身发出HTTP的DELETE操作，这会创建一系列新的修订版本。

C.3.4. Linux davfs2

Linux davfs2是一个Linux内核的文件系统模块，它的开发位于http://dav.sourceforge.net/。一旦安装，一个WebDAV网络共享可以使用普通的Linux的mount命令装配。

一个公开的秘密就是DAV客户端不会与mod_dav_svn的自动版本化完全工作正常，即使一个单独对服务器的写尝试需要LOCK请求作为前提，而这是mod_dav_svn不支持的。此时，还没有数据表明是否可以使用mod_dav_lock解决这个问题。

附录 D. 第三方工具

目录

D.1. 客户端和插件 314
D.2. 语言绑定 315
D.3. 版本库转化 315
D.4. 高级工具 315
D.5. 版本库浏览工具 316

Subversion的模块设计（在第 8.1 节 “分层的库设计”讨论过）和语言绑定的能力（在第 8.2.3 节 “使用C和C++以外的语言”描述过）使的我们可以作为扩展和后端支持来替代软件的某些部分，在这个附录里，我们会简略介绍一些使用Subversion功能的第三方的工具。

关于更新的信息，可以在Subversion的网站（http://subversion.tigris.org/project_links.html）查看。

D.1. 客户端和插件

AnkhSVN (http://ankhsvn.tigris.org/)
Subversion的Microsoft Visual Studio .NET插件

JSVN (http://jsvn.alternatecomputing.com/)
Subversion的Java客户端，包括了IDEA的插件

psvn.el (http://xsteve.nit.at/prg/vc_svn/)
Subversion为emacs的接口

RapidSVN (http://rapidsvn.tigris.org/)
跨平台的Subversion图形化工具，基于WxPython库

Subclipse (http://subclipse.tigris.org/)
Subversion关于Eclipse环境的插件

Subway (http://nidaros.homedns.org/subway/)
Microsoft SCC为Subversion的提供者

sourcecross.org (http://www.sourcecross.org/)
Microsoft SCC为Subversion的提供者

Supervision (http://supervision.tigris.org/)
Subversion的Java/Swing可视化客户端

Sven (http://www.nikwest.de/Software/#SvenOverview)
Subversion在Mac OS X的Cocoa框架的本地图形化界面

Svn4Eclipse (http://svn4eclipse.tigris.org/)
Subversion为Eclipse IDE的插件

Svn-Up (http://svnup.tigris.org/)
Java为基础的Subversion图形化界面和IDEA的插件

TortoiseSVN (http://tortoisesvn.tigris.org/)
Subversion客户端，使用Microsoft Windows的shell扩展实现

WorkBench (http://pysvn.tigris.org/)
跨平台的Python基础建立在Subversion之上的软件开发图形化界面

D.2. 语言绑定

PySVN (http://pysvn.tigris.org/)
Subversion客户端API的面向对象的Python绑定

Subversion (http://subversion.tigris.org/)
Subversion的Python、Perl和Java的绑定，作为核心C的API的映射

SVNCPP (http://rapidsvn.tigris.org/)
Subversion客户端的C++面向对象绑定

D.3. 版本库转化

cvs2svn (http://cvs2svn.tigris.org/)
CVS到Subversion的转化

vss2svn (http://vss2svn.tigris.org/)
Microsoft SourceSafe到Subversion的转化

Subversion VCP Plugin (http://svn.clkao.org/revml/branches/svn-perl/)
VCP的CVS到Subversion插件

D.4. 高级工具

Trac (http://projects.edgewall.com/trac)
最低限要求的web基础的项目管理和bug/问题追踪系统，包括了版本控制界面并集成了Wiki支持。

Scmbug (http://freshmeat.net/projects/scmbug/)
包括bug追踪的软件配置管理软件，支持Subversion

Subissue (http://subissue.tigris.org/)
直接追踪Subversion版本库的问题

Subwiki (http://subwiki.tigris.org/)
使用Subversion作为数据版本库的Wiki

svk (http://svk.elixus.org/)
基于Subversion的分布式版本控制工具

submaster (http://www.rocklinux.org/submaster.html)
基于Subversion的分布式软件开发系统

D.5. 版本库浏览工具

SVN::Web (http://svn.elixus.org/repos/member/clkao/)
Perl基础的Subversion版本库Web界面

ViewCVS (http://viewcvs.sourceforge.net/)
Python基础的CGI脚本，用来浏览CVS和Subversion版本库

WebSVN (http://websvn.tigris.org/)
PHP基础的Subversion版本库浏览器

附录 E. 版权

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 Work.

   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 shall have all rights and obligations of Licensor.

==========

术语表

A

administrative subdirectory （管理目录）	通常是工作目录中的.svn目录，保存了帮助Subversion工作的文件。
administrative area （管理区域）	管理目录的一种更抽象的说法。
authentication challenge （认证请求）	服务器向客户端发出要求提供认证信息的动作叫做认证请求。

B

Branch （分支）	分支是一种特殊的版本形式，我们可以对文件或项目分出一个并行开发的版本，对这个并行开发的版本进行修改，然后将分支版本的修改合并到主干上来。

C

check in （检入）	文件的检入是指将自己的工作拷贝提交到配置管理系统的文件库中，使得他人可以从文件库中获取你的最新修改版本。
check out（检出）	文件的检出是指从配置管理系统的文件库中取出一个副本作为自己的工作拷贝，并在这个工作拷贝上进行自己的工作。

D

delta（增量）	提交时只提交区别，也就是增量的模式。
directive（指示）	Apache配置文件中的配置选项。
dump format（转储格式）	从版本库里导出的文件的一种格式。

P

porting changes（搬运修改）	合并两个分支的行为称作搬运修改。

贝）	Subversion在管理区域为每个文件保留了一个备份，这是上一个版本（叫做“BASE”版本）没有修改的（没有关键字扩展，没有行结束翻译，没有任何其他区别）拷贝。

R

Repository（版本库）	存储所有修订版本历史记录的地方。
Revision（修订版本）	文件的版本和项目的版本是不同的，对于文件来说，每次修改后提交都会产生一个新的修订版本号，从版本控制系统中可以找到从文件创建到最后一次提交的每一个版本，对于文本格式的文件，通过一些比较工具（版本控制系统中的或第三方的）可以比较每一个版本之间的不同。

U

unified diff format（标准区别格式）	标示文件区别的一种格式。
Universal Unique IDentifier（全局唯一标示）	每一个版本库都有唯一的UUID。

V

Vendor branches（卖主分支）	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，为此建立的分支叫做卖主分支。

W

work copy（工作拷贝）	从配置管理系统的文件库中取出的保存在本地的文件副本。