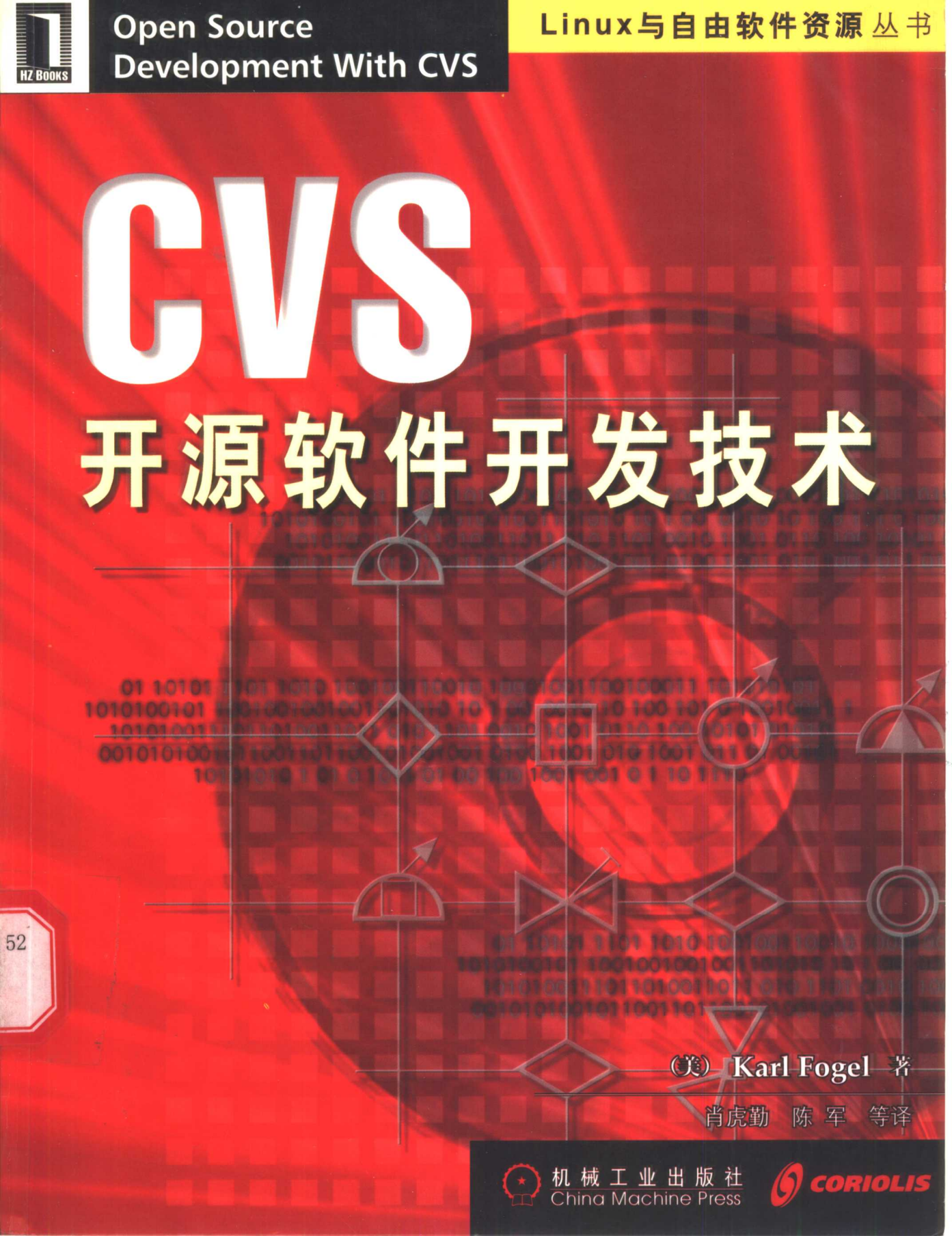
HZ BOOKS
Open Source
Development With CVS
Linux与自由软件资源丛书
CVS
开源软件开发技术
(美) Karl Fogel 著
肖虎勤 陈军 等译
机械工业出版社
China Machine Press
CORIOLIS

Linux 与自由软件资源丛书

CVS 开源软件开发技术

(美) Karl Fogel 著

肖虎勤 陈 军 等译

机械工业出版社
China Machine Press

本书详细介绍CVS 的发展历史、基本概念、高级应用等内容。本书不仅介绍了CVS的基本知识，而且提供了管理或参与开发开放资源项目的具体建议。本书介绍了许多自由软件开发中常见的问题和分析，还针对用得最广的版本控制系统提供了便利的指南。本书有助于自由软件开发人员利用在线文档进行设计与开发工作。本书适合CVS服务器的管理员、自由软件管理者、自由软件爱好者等参考。

本书版权登记号：图字：01-2001-0683

图书在版编目(CIP)数据

CVS 开源软件开发技术/（美）佛吉（Fogel, K.）著；肖虎勤等译. - 北京：机械工业出版社，2001.6

（Linux 与自由软件资源丛书）

书名原文：Open Source Development with CVS

ISBN 7-111-08891-3

I. C… II. ①佛… ②肖… III. 软件工具，CVS - 软件开发 IV. TP311.56

中国版本图书馆CIP数据核字(2001)第26653号

机械工业出版社 (北京市西城区百万庄大街22号 邮政编码 100037)

责任编辑：宋燕红 张金梅

北京第二外国语学院印刷厂印刷 · 新华书店北京发行所发行

2001年6月第1版第1次印刷

787mm × 1092mm1/16 · 16.25印张

印数：0 001-5 000册

定价：35.00元

译 者 序

今天，自由软件在世界上日益普及。由于它的源代码是公开的；因此，如何更好地管理自由软件的源代码，使其更好地进行开发已成为一个现实问题。于是，为了满足这一需要，CVS（Concurrent Versions System，版本协作控制系统）就应运而生了。

本书就是在这种环境下完成的。作者针对自由软件开发的现状和发展前景，作了大量的研究，并且在许多专业人士的帮助下，完成了此书，它对于自由软件的发展具有很强的实用性。书中不但解答了许多有关自由软件开发的疑问，还针对用得最广的版本控制系统提供便利的使用指南。本书主要为CVS服务器的管理员、自由软件管理者以及自由软件爱好者编写。

另外，本书通过大量的实验介绍了CVS如何管理源代码、Web页以及帮助读者熟悉开发源文件发展的一些协定等等。

参加本书翻译的工作人员主要有肖虎勤、陈军、姚凯、李向荣、龙浩等。由于译者水平有限，疏漏之处在所难免，欢迎读者批评指正。

2000年11月

前　言

如果你像别的程序员一样，不需靠开发自由软件谋生（只要你愿意，一定能赚很多钱，不是吗？），那么至少你可能使用过一些自由软件，并给自由项目捐献了一些代码或文件。假若这样，就要注意到近几年来在自由软件领域所展示出的现象：

- 正在形成一种文化和一组具有共享价值的资源。即：自由软件有维护者，但没有所有者，开发者与用户之间没有明确的分界线。错误（bug）被公布，而不是被隐藏，并且所有的软件开发信息都被公布。
- 许多自由项目，特别是一些大型的、分布式的软件开发小组，将其源代码保存在一个名为CVS的版本控制系统中。

第二种现象和第一种现象紧密相关。CVS能够迅速流行，得益于自由软件支持并鼓励开放、发展的软件开发方式。同时，CVS允许因特网上的任何人及时访问源代码的最新版本，以利于“补丁文件”的建立，及适应人们贡献修正错误的方法和项目的新功能的需要。CVS还允许积极开发者对相同的基代码进行修改，并且不需保持一致的修改步调。事实上，如果没有CVS，我们也必须创造一个这样的版本控制系统。

本书有两个目的：一个是社会目的，另一个是技术目的。社会目的是证明出现的这种新文化，并且对人们管理或参与开发开放资源项目能提供具体的建议。技术目的是当你运用CVS时，告诉你必须了解一些有关CVS的知识，以便能更好地在开放资源项目上运用CVS。

至于在第一个目的中涉及的单词“建议”，许多人认为这可能是我对一个初期且不稳定项目的不太确切的看法。但是在World Wide Web问世不到一年时，就出现了主页设计源程序块。事实证明，许多的看法是可疑的或者根本就是错误的。为避免出现这种问题，我将在进行开放资源的练习中，尽量坚持采用从实际经历中学到的知识为例子。所以，当我的个人看法成为一种观点之后，请不要把它作为我的一种臆断。

尽管你能从开放资源项目的内容中认识到CVS，但如果你能经常运用CVS，则能更好地学习CVS。CVS不仅能管理程序源代码，也能翻译网址、文本文档、配置文件等（不必通过长期的实验，只需运用CVS，你就能判断出CVS是否为给定状态下的恰当程序工具）。

本书的章节之间并非紧密关联：其中，一些是有关开放资源发展的内容，另一些是有关CVS的内容。随着内容的不断深入，你将发现这两个主题之间没有严格的区分，并且是紧密相连的。学习本书后不要求你能马上精通CVS，但你肯定能用在线文档进行自由软件的设计和开发工作。并且，至少能马上熟悉UNIX，因为CVS是一个UNIX环境下的实例。但如果你能迅速研究本书，就能在抛开UNIX环境的情况下学会CVS。

两个术语

“开放资源”（Open Source）和“自由软件”（free software）有何不同？这是一个旷日持

久的辩论话题。在我看来，这两条术语没有本质的区别，并且，我将在本书中交替使用这两条术语。在http://www.gnu.org/philosophy/open-source-or-free.html中可见到Richard Stallman的论文“为什么称‘自由软件’比称‘开放资源’好”。该论文精辟地说明了这两个术语的不同之处。

自由软件中的“自由”，当然不是指价格，而是指软件中的源代码能自由地进行修改和再分发。对于自由软件开发的自由度，并不是指软件的低价位。实际上，这种自由度正是自由软件能够成功的关键。遗憾的是，英语单词本身不能区分“自由”与“免费”这两个不同概念。

CVS在本书中的用法

在本书中，会有许多带有解释文字的命令行实例，并且主要例子中的用户名为Jrandom，它工作在名为floss.red-bean.com的机器上。其命令提示行为：

```
floss$
```

提示行下面的输出，与提示行为同一字体：

```
floss$ whoami
jrandom
floss$
```

偶尔因命令行本身太长而占据标准UNIX终端的多行时，在命令行的末端将出现一个反斜杠。该反斜杠表明下一行继续为命令行。由于可读性的原因，命令行有可能缩排。例如：

```
floss$ cvs diff -c -r prerelease-beta-2_09-19990315 -r \
       postrelease-3_0-19990325 fudgewinkle.c
```

（在本书末尾，介绍所有命令的含义。）

有时，需要从其他位置执行外部命令（例如，当前项目由两个不同的人同时开发时）。在这种情况下，假设另外一个人的名字为qsmith，她工作的机器名为paste：

```
paste$ whoami
qsmith
paste$
```

所有的命令都在一个UNIX（Bourne Shell）环境下运行。如果你基本熟悉UNIX，就会发现本书会有一些特别的内容。例如，你会注意到ls命令的运行有时会有一点异常：

```
floss$ ls
foo.txt   bar.c   myproj/
```

myproj/的后缀“/”并不是文件名的一部分，只表明myproj是一个目录，因为显示的反斜杠是在jrandom的工作环境下。ls命令相当于运行ls -CF命令，即把文件按行列显示，并显示它们的类型（“/”代表目录，“*”代表可执行文件，“@”代表符号连接等等）。

我决定在本书的例子中都保持这种格式，因为当阅读输出时，这种格式对于区分文件和目

录非常有用。所以，即使你没有在ls命令后附加-CF选项，输出也会默认该选项。

联系与下载

本书有关具体CVS运用的章节（2、4、6、8、9和10章），其版权属于GNU General public License，所以读者能够直接从http://cvsbook.red-bean.com中浏览或下载这些章节。如果你从在线目录版本中发现CVS的bug，请将这些bug报告给bug-cvsbook@red-bean.com。

目　录

第1章 CVS 发展过程

1.1 什么是自由软件

许多人不买软件，并不意味着“盗版软件”是获得软件的常用途径（尽管有可能，但这里只是设想一下）。即使所有的人都抵制盗版，即使所有的制造商都合理地对软件进行标价，仍然不会有太多的人销售或购卖正版软件。

因为软件商店里能得到的不是有用的软件。假如你编写过自由软件，就会明白这一点。许多软件的功能已经是被大大削弱了。这并不是说它不能像广告中所说的那样工作，而是因为它们的源代码被隐藏起来了，造成维护和开发潜力的削弱。如果用户没有源代码，则将无力改变软件的运行方式，即使软件里充满了bug也是这样。这使得软件在使用过程中的维护和升级更加困难。而对于软件来说，要改变其功能，就必须改变可执行文件，而要改变其可执行文件，就必须能编辑软件的源代码。面对这一切，只能做出一个无奈的选择：等待下一个版本出现，并希望制造商能够清除bug。

商业软件公司利用对源代码的保密来对其产品加以额外的控制，所以可以理解公司把源代码作为最宝贵的秘密来守护的做法。如果源代码开放，则有人就可能会修改程序、清除bug并添加一些与销售策略和买卖项目无关的特征。这样，商业竞争者就会秘密地复制代码的某些部分，并将之运用到自己的可执行程序中。另外，将源代码保持为机密的一个更重要的原因就是其一旦公布，就会受到别人的审阅，竞争者就可能公然地指出源代码中低效率及编码不好的地方。

因此商业软件生产者没有理由将源代码公之于众，相反，有一些很好的理由促使他们不这样做。然而，从用户观点来看，生产者的这种做法有一些不妥之处，例如，在程序的修改上，用户只能依靠单一的卖家。此外，如果卖家由于某些原因停止支持该软件系列，则用户学习该软件的时间和智力投资将得不到任何回报。一个软件得不到商家的支持就不能长时间存在。当得不到维护时，操作系统、计算机硬件和交互协作能力标准得到持续发展的同时，而软件程序将停滞不前。最后软件或者由于太陈旧而不能运行，或者不能与新的软硬件兼容。

和其他类型的公司一样，软件公司也会倒闭，或者停止向顾客提供满意的服务。而顾客对于这样的产品，即使不愿意使用，也不得不坚持使用，这是因为若改用另一类产品，花费会相当高（另外，很难保证新的卖主会对用户的长远需求作出更积极一些的响应）。因此，所有的用户都会有对软件不满意却又无可奈何的时候。

并不一定非要这样，尤其是随着用户人数的增多，事情会有很大改变。所以一个公布源代码、鼓励资源共享（对于源代码和可执行程序都鼓励）的系统出现了。更重要的是，用户被邀请进来以完善和优化源代码。这并不需要使每个人都成为程序员：只要一些用户掌握修改源代

码的技术，而其他人则只需知道怎样找到会修改源代码的人就可以了。全球有权使用该软件的用户群集体开发使得软件具有生存且流行起来的活力。

今天，许多因特网的基础服务设备（包括E-mail路由器，Web服务器以及基本地址查询系统）是运行在这种得到协作维护的软件基础之上的。原因之一是向用户提供的这种软件比其他商业软件更加合适可靠；另一个原因是现在太多的人在选择商业软件时，经常发现软件难以维护，并且不能满足他们的需要。当源代码能被所有人修改和重新发布时，某地的某人将有极大的可能性会在程序中清除bug并加入有用的功能。如果这样的话，每个人都能从中受益，因为，对程序所做的改进将很快进入每个人的计算机，而那些糟糕的修改一般来说却做不到。

1.1.1 CVS的产生

从某种意义上来说，CVS并不是从因特网上自然出现的，尽管随着因特网的发展它最终必将产生。但无论如何，Richard Stallman的努力大大加速了这一进程，非正规的代码共享在此之前已经存在了很长时间，但在Stallman给这一现象命名并提出参与的原因之前，参与者们一般并不知道自己行为的结果。

在20世纪70年代，Stallman在MIT的人工智能实验室工作。用他本人的话说（www.gnu.org/gnu/thegnuproject.html），这个实验室是一个“共享软件团体”。在这里，源程序代码如同房间里的空气一样，被人们所共享。如果你改善了自己使用的系统，那么你的改进应该被所有使用类似软件的人所共享，即所有的使用者都能从这种改进中得到好处。实际上，“你的改进”这个词存在着某种误导：从欣赏的角度来说，这部分工作是属于个人的，但从拥有的角度来看则并非如此。这种共享不会使你损失什么，从长远来看，你会从中获益。因为别人在你改进的基础上所做的改进同样能让你分享。

Stallman所处的这种理想环境在20世纪80年代就不复存在了。此时，一家计算机公司雇佣了人工智能实验室的许多程序员，这家公司付给这些程序员丰厚的报酬。在公司里，这些程序员干的工作同他们在实验室里大致相同，不过共享的环境已不复存在。他们的工作是排它性的。和大多数公司一样，这家公司信奉如下的经营模式：写出一个真正好的程序（在此是一个操作系统），然后将源代码保密，以免别人从中获得好处，这样就能从别人对该软件的拷贝使用中收取费用。当然，这种拷贝是二进制拷贝。这样，公司以外的人可以使用这种软件，但他们就不能看到产生可执行文件的源代码或对源代码做出任何修改。

对于那些前人工智能实验室的编程人员来说，实际没有多大的变化。他们彼此仍然能分享代码，因为他们继续在同一个公司里工作。但是不能和公司以外的人共享代码，因为与外人共享代码是违法行为。同样地，他们也不能自由地将别人的代码添加到自己的产品中去。

但是，对Stallman来说，这种将源代码隐藏起来的做法是不可容忍的。他曾经在一个共享源代码团体的氛围中工作过。当此环境不复存在以后，他并不接受这个结果；相反，他决定重新构造一个更不易被损害的团体。因此，他建立了一个叫做自由软件基金会的非盈利性组织，并开始完善一个能彻底与UNIX兼容的操作系统，并将该操作系统称为GNU（GNU并非UNIX）。

更为重要的是，他设计了一种版权许可。这种版权许可保障了软件源代码的永久重分配。与试图保留作者或所有者的专有版权版权不一样的是，公众许可证（即GPL，参见附录B）防止任何人宣称对自己的工作具有排它性的权利。如果你有一份这种类型软件的拷贝，你可以把拷贝分发给其他人，但你不能限制他们继续外传这份拷贝。拷贝源代码的同时必须将权限一起拷贝，并使权限延伸到修改版本上。这样，就没有人能在将GPL下的软件拷贝作出一些修改后，以一种限制更严的协议将它们重新出售。

Stallman的这种想法是切实可行的，别的人开始在GPL的原则下发行他们的程序，偶尔也发明了他们自己类似的许可协议。当前，只要人们选择相互合作，Internet能保证全球的程序员获得彼此的源代码。因此，新的软件共享团体包含全球所有已联网的人，而不论他们实际所处的地理位置。

要指出的是，大概在1990年，仅有一小部分人具有Stallman的这种自信："所有的软件应将源代码公之于众"。即使是一些对GNU事业经常捐款的人也不赞成所有的软件是自由的，虽然他们会为GNU事业所取得的成就而高兴。不久以后，这项运动（如果它能称之为运动的话）获得了巨大的精神支持——这种支持来自于一些完全开放源码操作系统的出现。在芬兰，Linus Torvalds更新并完善了UNIX内核（称为Linux操作系统），并把源代码发布在GPL上，并加入了很多在GNU项目中可得到的UNIX工具，使其成为一种实用的UNIX发行版本。此后不久386BSD发布了。该版本是建立在UNIX的Berkeley软件发行版本基础上的。Berkeley软件发行版本实际上比Linux操作系统更早开发。但很快，该版本就被混乱地称做NetBSD、FreeBSD以及OpenBSD。

完全开放源码操作系统的出现给这项运动带来了真正的好处，这不仅指技术方面。它还可以证明开放源码代码能作出高质量的软件（在很多情形下，免费操作系统比商业操作系统运行更快、更稳定），因为大部分运行在免费操作系统上的应用程序也是自由的，所以自由软件使用者显著增多，同时也有更多的开发者将他们的聪明才智贡献给了自由软件。

1.1.2 两种开发类型

当越来越多的用户将自由操作系统取代商业操作系统时，非编程者注意到一些不曾预料的事情发生了。与此同时，Eric Raymond以其一贯的及时性，发表了一篇文章："教堂和市场"（见www.tuxedo.org/~esr/writings/cathedral-bazaar/），此文部分解释了为什么自由软件能获得技术上的成功。文章比较了两种不同的软件开发方式。首先，"教堂"形式的软件是被紧密组织、中心计划的项目，而且从头到尾都是"一件"创造性的工作，许多商业软件正是以"教堂"形式写出来的。那就是由一个技术权威人士带领一组人，并由其决定发行软件该加入什么特征。

另外一种形式，正如Raymond描叙的那样："一个吵吵嚷嚷的市场，每个人都有不同的安排和想法（像Liunx文档，可以受所有人安排的影响），要想成为一个稳定的体系，除非出现奇迹"。但奇迹确实出现了，Raymond认为奇迹出现的主要原因是，"当有足够多双眼睛盯着时，所有的bug都将被发现"。"教堂"形式软件的缺陷在于，它不能吸引软件最重要的同盟：用户——加入

到开发中来，这种小规模的开发队伍，很快就会因为对bug的报告和对程序新特性的要求等任务弄得一筹莫展，而且他们要用相当的时间去探求当前优先任务是什么？即使他们知道要做什么，也不能预料具体要花费多少时间寻找一个特定的bug。结果是开发人员花费太多的时间来解决这些问题，以至根本无暇顾及其他的工作。

此外，商业开发队伍经常在与软件技术无关的条件下工作，如预算、交货期限和市场战略等。即使决定是否继续维护某一程序，也要考虑到商业因素，软件本身的质量和潜在的作用反而考虑的较少。

反过来讲，用户只需要好的代码。他们需要一个有用的程序、需要除掉Bug和加入新的有用的功能，去除不需要的功能。回顾以上所述，结论非常明显。为什么不把这些问题交给用户自己解决？即使大部分用户不是程序员，不能改变源代码，但那一小部分能做到这一点的人，最终会给每个用户带来利益。

1.2 CVS和自由项目有什么关系

和许多经历过迅速增长的运动一样，自由软件运动很快发现自己出现了逻辑问题。而对于自由软件设计者们，也不再是将代码安装在一个等待别人下载的因特网服务器上就算完了。如果人们下载代码之后再加进许多bug补丁及新特性代码，开发者该怎么办呢？对一个流行程序来说，还没有一个设计者认为能将所有的反馈信息组织并结合起来，并有足够时间来重写源代码。在一个非公开源代码软件开发中，软件公司的指导思想是要求程序员精简高效，并对任务实行优化组合。然而，公开源代码的志愿开发人员，并不知道下一个代码片段来自何处，以及这些代码片段有何作用？如果幸运的话，志愿开发人员可能得到一个共同开发核心组的帮助，该核心组能帮助整理错误（bug）和检查外来代码，以确保开发项目的水准。这种类型的组织也许有很高的人员变换速度，但是，该组织的成员很可能仍是志愿者。

一个地域上分散的志愿者组织显然不可能投入很多的时间来训练其成员彼此合作。这样当该组织有成员变更时，为此付出的投资将损失殆尽。所以需要制定一套基本的项目分配方案，以确保新成员能较容易地适应工作，同时也需要设置一个自动的系统来接收外来代码，并使每个成员能及时得到最新修改的代码。当然，不单是自由软件才会有此要求，但自由软件的要求更加严格，因为自由软件为志愿开发项目，其资源缺乏管理，所以要求制定一个自动的解决方案以节约时间。

其实这种自动系统的基础早已存在。标准的UNIX diff程序能准确辨别两个文件之间的不同之处。如果你用“diff”连接一个修改过的文件和原文件，则diff程序输出的就是它们的不同之处。内行能看懂diff，并且知道文件的大致变化；更重要的是，特定的程序能够解说diff的确切意思。因此，diff程序很快就因派生patch程序而扩大，由Larry Wall作为自由软件编写和发表的patch，正是diff程序完整的派生程序。如果将文件A与文件B的不同之处（B文件即是经修改后的A文件），连同其中的任一文件提供给patch程序，patch程序就能重建另一文件（结果之一就是“diffs”被改称为“patches”，这也正是本书的余下章节里的使用方法）。

如果你怀疑其作用,请设身处地的为那些也需要接收外来代码开发者们想想。一个外来代码在实际项目中包括一系列变化多样的文件。维护人员想确切知道什么样的文件被修改，及其如何被修改。假设需要把这些变化集结起来，开发人员首先应将其在尽可能少出现错误地译成代码。而完成这一过程的理想方式，就是接收一串能用肉眼就能检查的补丁，然后将补丁自动加入到当前源代码中（现实中，维护人员的源代码在那时可能有其他的变化，但“patch”程序能非常巧妙地执行模糊匹配，所以即使文件发生了一定的改变，通常情况下，Patch程序仍能执行正确的指令）。

Diff和patch程序是一种便利、标准的提交捐献资料的途径，但很快出现了更高的要求：有时，提交的捐献资料合成为源代码，随后因为发现存在缺陷而必须将其删除。当然，即使能找到文件的变化内容，并且手工消除掉补丁对源代码的改动，这也是一个乏味且容易出错的过程。解决办法是设立一个保存项目的修改历史记录的系统，并可据此取回以前的版本，与当前的版本进行比较。同样，这个问题也存在于整个商业世界中，而不只限于自由软件事业。不同的系统都有其不同的解决方法。和许多的商业项目一样，自由软件项目选择了Walter Tichy的版本控制系统（RCS）。RCS是一个能解决上述问题的自由且方便的系统。

用RCS系统工作后，可以看出该系统缺乏几条重要的功能。一是采用文件集中方式处理方案，即使在一个目录树上存在各种不同的文件，也不能对它们区别处理；同时采用“上锁——修改——解锁”的开发模式，开发者为防止别人改变文件而希望首先能将文件“上锁”，然后工作，最后对文件“解锁”。如果你想锁住一个已被别人锁住的文件，也许你只能等到它被打开并处理完后再关闭或者去“盗锁”。实际上，即使你在文件的不同的代码区工作，你也有必要在使用同一文件之前与其他开发者进行协商（并且，可推测人们在操作完成后也可能忘记解锁）。最后，由于RCS不是基于网络的机制，所以开发者们不得不在保存有RCS文件历史数据的机器上持续工作，或者采用笨拙的手写脚本来转换工作机器与RCS服务器之间的数据。

因此在这种软件工具的发展中产生了最新的版本控制系统:CVS（Concurrent Versions System，版本协作控制系统）。CVS 对前面所说的在RCS中碰到的每个问题都做了改进。事实上，CVS一开始是由Dick Grune为简化RCS的手稿本而于1986年编写的。然后被送往Vsenet新闻网组comp.sources.unix。Brian Berliner于1989年用C程序语言重写了CVS，随后，Jeff Polk又为CVS增添了一些关键特性。

实际上CVS继续采用了RCS的原始格式来储存历史数据，最初，它需要依靠RCS的实用程序来解析格式，但它增添了一些特别的功能 。CVS是基于目录的，并具有一个能给目录命名的机制（便于存取），所以CVS能将项目当成单一的实体，这也是人们所希望的。同时，CVS不要求对文件进行上锁和解锁操作。相反，开发者同时还能对源代码进行修改，并依次将变化登记到源代码库（该处能保存项目的主要资料和变化的历史记录）中。CVS管理记录改变的结构，在必要的时候可以合并对相同文件所做的编辑，并向开发者通报发生的冲突。

20世纪90年代初期，Jim kingdon最终将CVS设计成基于网络的平台，因此开发者们能从因特网上的任何地方获得程序源代码。这使得基本代码对所有感兴趣的人开放了。由于CVS能巧

妙地对相同文件的变化进行合并，开发者也就无需为很多的人在同一套源代码上工作而担心。从某种意义上来说，CVS对源代码的处理就像是银行对钱的处理：大多数人不必担心钱在银行会出现什么问题、不必到很远的地方取钱、不必记录主要的交易程序、不必在同一家银行取钱，当然也不必超前消费。银行会自动处理前四个问题，而当我们超前消费时会提醒我们。

CVS的工作方式——能通过网络访问源代码，能同步开发，能自动对修改进行合并其功能证明了对源代码保密项目和对自由软件开发项目一样具有吸引力。目前，CVS在两种领域都得到了广泛的应用。然而,CVS只在自由项目中才真正居于主导地位。本书的一个主题就是：CVS得以成为自由软件中版本控制系统的首选，是因为在其促进项目运行的方式和已运行的自由项目运行方式之间，有完全的匹配（即协作）。为了进一步理解，我们需要更进一步了解源代码开放的过程。

1.3 开放资源开发的原则和CVS在其中的用处

开放资源发展的首要原则是能从世界的任何地方取得源代码（与私有软件开发相比，这是个主要的区别）。但马上会出现一个问题：在什么时候以及间隔多少时间应该将源代码开放？一开始，似乎有了最新发布的版本就足够了。但当其他人寻找并清除Bug时，他们也将需要最新开发的源代码，以使维修人员能使用相同的文件。令一个潜在资源捐献者非常气馁的是，花费大量时间捕捉并清除一条Bug，却在提交补丁后竟发现Bug已被找到并清除。一些程序员都知道，发行版本只是一个开发周期中特定时期的瞬态图，有时还没有很好地测试。从源代码的观点来看，发行版本与随时抽取的瞬态图没有本质上的区别。就志愿工作者而言，自由软件项目的开发是处于一种连续的状态之中的。

不幸的是，传统的软件发布方式没有将软件的连续更新这一因素考虑进去，其设计是围绕着版本应具有纪念意义、并受到特殊对待的事物这一想法而构思的。在Grand Event的方法中，版本被设计成一个静态的文件集合，把开发项目的历史和未来区分开来，并发布给在下一个版本出现之前继续使用当前版本的用户。源代码的开发在那段时期自然不会停滞不前。在此期间，下一个版本中将出现的有关代码的变化开始慢慢积累在开发者的源代码拷贝上，以便为将来新版本的发行做好准备，而最后代码也处于一种明显与当前版本不同的状态中。所以，即使每个版本都含有完整的源代码，也不会太有用。很快，用户使用的文件便会过期，并且用户根本不能检查那些只有开发人员和维修者才能访问的核心资料。

但不久之后这种情况就可以用工作区来处理。这是一种不太方便但至少还能使用的局部性的解决方法。开发资料的瞬态图以规则的方式被做成可用的在线资料。任何一个想跟踪开发项目发展的用户，都可以在线得到并安装那些资料。对于经常这样做的人，可以通过脚本来使这一过程半自动化，在恰当的时候自动取回并安装样本。但无论怎样，这都不是一个特别令人满意的接收变化的方法。因为即使只有一个文件的一行代码发生了变化，临时版本仍然要被全部收回。

解决这一问题的方法就是CVS。CVS除使当前开发者把代码变化存入其主要源代码库的过

程变得简易之外，还能支持以匿名只读方式访问源代码库。这意味着任何人都能在其机器上保存一个开发树。这样当他们需要在一个特殊的代码区工作时，只要给出一条简单的命令，使开发树能获得更新即可。在确认文件中一些需要尽力解决的问题还没被别人解决后，开发者将马上开始工作。最后当这些问题解决后，CVS会自动产生补丁，并将补丁发送给维护人员以进行检查，最后可能将其并入主资源树中去。

这并不是指CVS能做到以前看来是不能做到的事。如即时取回源代码和生成补丁等很早就在理论上成为可能了。CVS的优点在于它能将一切变得极为便利。CVS的开发主要依靠志愿开发者的力量，而工作是否便利常常是一名开发者是否会向你的开发项目提供捐助的决定因素。开发项目为了争取志愿者,通常要展现各自的优点。这些优点并不仅仅指软件本身的优点，还包括是否更易于使潜在开发者进入项目，以及接收捐助的准备程度。

在CVS中保持它们的主要资源为自由软件项目的百分率，本身就是引人注意的。而更引人注意的是因特网上一些大型（根据捐献资料的数目而言）且非常成功的（根据软件安装基础而言）项目。它们包括Apache WWW服务器、GNOME自由桌面环境、FreeBSD、NetBSD、OpenBSD、PostgreSQL数据库和XEmacs文本编辑器等。在后面的章节，我们将详细介绍项目如何运用CVS管理资源协助志愿开发者。

1.4 怎样获得代码

写到这里，我所讲述的都集中于应用自由软件对于用户的优点。但是，开发者们仍然面临一个有趣的选择。只要版权法以现在的形式存在，那么用私有代码进行工作就可能有利于编程者——当需要购买每一份流行程序的运行拷贝时，利益更为巨大(即使考虑到非法代码共享)。如果你想变得富有，可以这样做：写一份有用且不公开源代码的软件，并让它引起人们的注意，然后等待微软给你的公司开价。

但不知何故，自由软件项目仍能成功地找到编程者。对于这种情况，如果有多少人在编写自由代码，就可能有多少种解释。尽管如此，但若你花足够的时间去观察邮件列表和开发者讨论组，你就会发现一些关键性理由：必要性、团体性和成就感，当然，还有钱的原因，所有的理由都是有机统一的。

原因一：Eric.Remond猜测必要性（如对“搔痒”这样行为的需要）是大部分的自由软件项目得以开发的首要原因。如果你只是想开发一个程序，不管是一次性还是永久性的，并且不希望从程序中获取报酬（除了因为使用该程序而带来的时间上的节省），那么把你的程序以自由软件的方式发布是很有意义的。有时，开发出的一个自由软件能给别人帮助时，那么别人也会帮助你维护该程序。这种情况称之为Kropotkin Factor——有时,相互协作是获得成功的最佳策略。

但我感兴趣的是第二个原因：团体性。作为开发组中的一员，在合作工作时的纯乐趣是进行工作的强大动力，不计报酬更说明了人们对这种程序开发工作的渴望，而且协作方式也证明了工作超越了个人狭窄的意识。同时从一群经验丰富的程序员身上学到的知识也该考虑进来；研究有用的自由代码，并通过在线讨论，询问一些有关代码的知识，比从书本或学校能学到更