改善研发管理，从 Topo 开始

2012

配置管理从理论到实践

杭州云图科技

2012-5-4

配置管理从理论到实践

摘要：

配置管理（Configuration Management）工作本身的重要性毋庸置疑，然而在国内一般只有大公司才会有专门的配置管理员（有些又叫配置经理）职位，即使这样，如果你问配置管理员的工作范围是什么？应该具备哪些基本技能？每个公司的配置管理员对此的理解都大不一样，期望本文能够给相关研发工程师全面认识配置管理有所帮助。

一：配置管理关键活动

1.配置项（Software Configuration Item，SCI）识别

Pressman 对于 SCI 给出了一个比较简单的定义："软件过程的输出信息可以分为三个主要类别：（1）计算机程序（源代码和可执行程序），（2）描述 计算机程序的文档（针对技术开发者和用户），以及（3）数据（包含在程序内部或外部）。这些项包含了所有在软件过程中产生的信息，总称为软件配置项。"

由此可见，配置项的识别是配置管理活动的基础，也是制定配置管理计划的重要内容。

软件配置项分类软件的开发过程是一个不断变化着的过程，为了在不严重阻碍合理变化的情况下来控制变化，软件配置管理引入了"基线（Base Line）"这一概念。IEEE 对基线的定义是这样的："已经正式通过复审核批准的某规约或产品，它因此可作为进一步开发的基础，并且只能通过正式的变化 控制过程改变。"

所以，根据这个定义，我们在软件的开发流程中把所有需加以控制的配置项分为基线配置项和非基线配置项两类，例如：基线配置项可能包括所有的设计文档和源程序等；非基线配置

项可能包括项目的各类计划和报告等。

关于配置项的标识和控制,所有配置项都都应按照相关规定统一编号,按照相应的模板生成,并在文档中的规定章节(部分)记录对象的标识信息。在引入软件配置管理工具进行管理后,这些配置项都应以一定的目录结构保存在配置库中。

所有配置项的操作权限应由 CMO 严格管理,基本原则是:基线配置项向软件开发人员开放读取得权限;非基线配置项向 PM、CCB 及相关人员开放。

2. 工作空间管理

在引入了软件配置管理工具之后,所有开发人员都会被要求把工作成果存放到由软件配置管理工具所管理的配置库中去,或是直接工作在软件配置管理工具提供的环境之下。所以为了让每个开发人员和各个开发团队能更好的分工合作,同时又互不干扰,对工作空间的管理和维护也成为了软件配置管理的一个重要的活动。

一般来说,比较理想的情况是把整个配置库视为一个统一的工作空间,然后再根据需要把它划分为个人(私有)、团队(集成)和全组(公共)这三类工作空间(分支),从而更好的支持将来可能出现的并行开发的需求。

每个开发人员按照任务的要求,在不同的开发阶段,工作在不同的工作空间上,例如:对于私有开发空间而言,开发人员根据任务分工获得对相应配置项的操作许可之后,他即在自己的私有开发分支上工作,他的所有工作成果体现为在该配置项的私有分支上的版本的推进,除该开发人员外,其他人员均无权操作该私有空间中的元素;而集成分支对应的是开发团队的公共空间,该开发团队拥有对该集成分支的读写权限,而其他成员只有只读权限,它的管理工作由 SIO 负责;至于公共工作空间,则是用于统一存放各个开发团队的阶段性工作成果,它提供全组统一的标准版本,并作为整个组织的 Knowledge Base。

当然,由于选用的软件配置管理工具的不同,在对于工作空间的配置和维护的实现上有比较大的差异,但对于 CMO 来说,这些工作是他的重要职责,他必须根据各开发阶段的实际

情况来配置工作空间并定制相应的版本选取规则，来保证开发活动的正常运作。在变更发生时，应及时做好基线的推进。

3．版本控制

版本控制是软件配置管理的核心功能。所有置于配置库中的元素都应自动予以版本的标识，并保证版本命名的唯一性。版本在生成过程中，自动依照设定的使用模型 自动分支、演进。除了系统自动记录的版本信息以外，为了配合软件开发流程的各个阶段，我们还需要定义、收集一些元数据（Metadata）来记录版本的辅 助信息和规范开发流程，并为今后对软件过程的度量做好准备。当然如果选用的工具支持的话，这些辅助数据将能直接统计出过程数据，从而方便我们软件过程改进（Software Process Improvement，SPI）活动的进行。

对于配置库中的各个基线控制项，应该根据其基线的位置和状态来设置相应的访问权限。一般来说，对于基线版本之前的各个版本都应处于被锁定的状态，如需要对它们进行变更，则应按照变更控制的流程来进行操作。

4．变更控制

在对 SCI 的描述中，我们引入了基线的概念。从 IEEE 对于基线的定义中我们可以发现，基线是和变更控制紧密相连的。也就是说在对各个 SCI 做出了识别， 并且利用工具对它们进行了版本管理之后，如何保证它们在复杂多变得开发过程中真正的处于受控的状态，并在任何情况下都能迅速的恢复到任一历史状态就成为了 软件配置管理的另一重要任务。因此，变更控制就是通过结合人的规程和自动化工具，以提供一个变化控制的机制。

在本文的前面的部分中，已经把 SCI 分为基线配置项和非基线配置项两大类，所以这里所涉及的变更控制的对象主要指配置库中的各基线配置项。

变更管理的一般流程是：

A) （获得）提出变更请求；

B) 由 CCB 审核并决定是否批准；

C) （被接受）修改请求分配人员为，提取 SCI，进行修改；

D) 复审变化；

E) 提交修改后的 SCI；

F) 建立测试基线并测试；

G) 重建软件的适当版本；

H) 复审（审计）所有 SCI 的变化；

I) 发布新版本。

在这样的流程中，CMO 通过软件配置管理工具来进行访问控制和同步控制，而这两种控制则是建立在前文所描述的版本控制和分支策略的基础上的。

5．状态报告

配置状态报告就是根据配置项操作数据库中的记录来向管理者报告软件开发活动的进展情况。这样的报告应该是定期进行，并尽量通过 CASE 工具自动生成，用数据库中的客观数据来真实的反映各配置项的情况。

配置状态报告应根据报告应着重反映当前基线配置项的状态，以作为对开发进度报告的参照。同时也能从中根据开发人员对配置项的操作记录来对开发团队的工作关系作一定的分析。

配置状态报告应该包括下列主要内容：

A) 配置库结构和相关说明；

B) 开发起始基线的构成；

C) 当前基线位置及状态；

D) 各基线配置项集成分支的情况；

E) 各私有开发分支类型的分布情况；

F) 关键元素的版本演进记录；

G) 其它应予报告的事项。

6．配置审计

配置审计的主要作用是作为变更控制的补充手段，来确保某一变更需求已被切实实现。在某些情况下，它被作为正式的技术复审的一部分，但当软件配置管理是一个正式的活动时，该活动由 SQA 人员单独执行。

总之，软件配置管理的对象是软件研发活动中的全部开发资产。所有这一切都应作为配置项纳入管理计划统一进行管理，从而能够保证及时的对所有软件开发资源进 行维护和集成。因此，软件配置管理的主要任务也就归结为以下几条：（1）制定项目的配置计划；（2）对配置项进行标识；（3）对配置项进行版本控制； （4）对配置项进行变更控制；（5）定期进行配置审计；（6）向相关人员报告配置的状态。

在此，我想特别指出的是：由于软件配置管理覆盖了整个软件的开发过程，因此它是改进我们的软件过程、提高过程能力成熟度的理想的切入点。希望本文所描述的这个软件配置管理的角色分配和工作流程能在实践中不断地得到完善 ,从而使我们的软件开发活动能够更加有序、高效的进行！

二：如何做好软件项目的配置管理

在软件项目实施过程中正确、有效地进行配置管理，需要进行科学合理的规划工作，并确定相应的执行策略。下面针对软件项目工作的特点，介绍了配置管理工作的一般步骤和注意事项。

当软件开发团队发展到一定规模时，会越来越强调开发过程规范化和成熟度。软件项目的成败在很大程度上取决于对其开 发过程的控制，这包括对质量、源代码、进度、资金、人员等的控制。软件配置管理可以帮助开发团队对软件开发过程进行有效地变更控制，高效地开发高质量的软件。在质量体系的诸多支持活动中，配置管理处在支持活动的中心位置，它有

机地把其他支持活动结合起来，形成一个整体，相互促进，相互影响，有力地保证了质 量体系的实施。

1. 配置项的分类

在开展配置管理工作之前，首先应根据项目特点，对项目实施过程中涉及到的配置项进行分类工作。一般来说，一个完整的软件项目会包括项目管理文档、软件开发文档、程序代码、集成文档、维护文档等五类配置项，配置项分类结构如图所示。

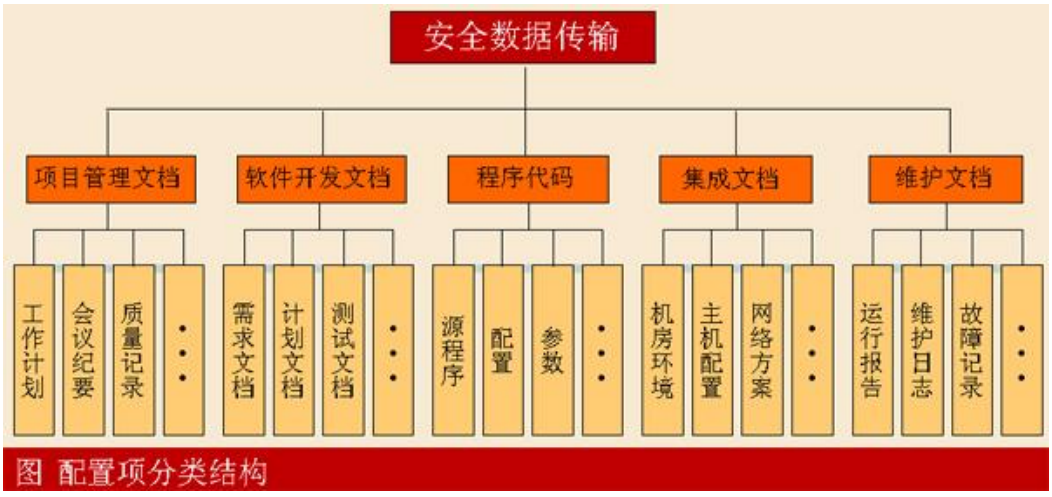


图 配置项分类结构

对于每一类配置项，又可以划分为若干细类，具体的分类方法如下：在项目实施过程中制订的工程总体计划、阶段计划、周计划， 定期召开的项目例会或技术专题会议的纪要，质量评审记录、配置管理报告等与质量相关的工作记录等文档，都属于项目管理方面的范畴，因此均可以划分到项目管 理类文档进行管理。

作为一个软件项目，必然会产生贯穿软件工程标准定义的需求调研、需求分析、概要设计、详细设计、单元测试、系统测试、用户测试等各个阶段的软件文档，均可以归入到软件开发类文档之中。

在软件开发过程中产生的各模块程序代码 ,软件系统运行所需的各类参数以及配置文件等内容，由于其技术和管理特征与文档有很大的不同，而且相互之间的关联性比较强，为了对其版本进行有效地控制，建议单独划为一个类别进行管理。

软件系统的设计、开发和运行离不开硬件环境的支持，因此在软件项目的实施过程中，通常都会涉及到机房设计、主机安装、网络规划等方面的工作内容，因此系统集成类的文档应作为单独的一个类别，纳入到软件项目的配置管理之中。

在软件系统投入运行之后，需要进行相应的日常维护工作，在维护过程中产生阶段性运行总结报告、定期产生的维护日志、系统运行中出现的故障现象及问题解决情况等维护记录，都需要纳入维护类文档进行管理。

在软件项目实施过程中产生的各类文档、程序代码纷繁复杂、数量众多，通过对各类配置项的归类工作，形成逻辑清晰的配置管理结构，便于对文档和程序代码进行日常管理，使项目实施中产生的各类配置记录始终处于可控状态。

2. 建立配置库

在软件项目的启动阶段，应指定一名专职或兼职的配置管理员，建立一台专用的配置服务器，安装相应的配置工具软件， 并根据配置项分类方法，对程序代码和文档的目录结构进行规划工作，在配置工具软件中，建立起相应的配置目录结构，同时根据使用者角色的不同，设定相应的目录访问和存取权限。配置项及工作角色的对应关系如表所示。

表 配置项及工作角色的对应关系

配置项类别	工作角色
项目管理文档	项目经理、QA、配置管理员
软件开发文档	技术经理、软件开发人员
程序代码	技术经理、软件开发人员
集成文档	主机工程师、网络工程师、数据库工程师
维护文档	软件维护人员

对于每一个具体的配置项，都需要标识出其作者、时间、版本号、当前状态等基本信息，以便对配置项的版本进行实时监控，方便项目成员对配置项的检索和更新工作。

配置管理员负责整个配置库的安全管理工作，应妥善保管好系统管理员的口令，并进行定期

的变更工作，以保证配置库的安全性。

3.建立执行机制

在配置库建立起来以后，配置管理员应将配置目录结构和权限分配表在项目组内部进行公布，并根据应用行业特点，对 CMM/ISO9001 的配置管理过程进行合理裁减，制订适用于本项目的配置工作流程，明确项目组中的每位成员在配置管理方面的分工职责，并对项目组成员进行相应的职责和流程培训工作。

配置管理员除了负责对各类配置项进行管理之外，还应对项目配置状况进行分析，定期提供配置报告，发布最新的配置项状态，提出改进建议并跟踪执行情况，避免出现因为文档或程序代码版本更新的不一致，而导致系统故障的情况发生。

在配置管理工作中，为了保证配置项的可靠性，应制订相应的备份策略，对配置库中的不同类型的配置项进行定期备份。 在软件系统的设计开发阶段，程序代码类的配置项由于变更频繁，建议每天备份一次，在正式发布之后，可以改为每周备份一次。文档类的配置项变更机率相对较 小，建议每周备份一次。具体的备份方法，可以采用手工方式执行备份操作，也可以在工具软件或操作系统中设定备份策略，定期自动执行备份操作，同时配置管理 员应做好相应的备份记录工作。

由于配置工具软件本身一般都提供对每一个配置项历史版本的追溯机制，因此对配置库的备份操作，一般只需对当前配置 库的内容进行备份即可。这里需要注意的一点是，在执行配置库的备份操作之前，应对配置库目录中的数据是否正常进行检查，以避免因库文件损坏而使错误数据覆 盖正常备份库，从而导致配置项丢失的情况出现。

4.经验总结

如果配置工作流程制订得过于复杂，不具备可操作性，反而起不到应有的管理作用，因此在

开展配置管理工作时，应以简单、有效、适合应用行业特点为基本准则，推进软件项目实施过程中的配置管理工作。

配置管理对象不仅仅限于 CMM/ISO9001 体系规定的内容，凡是与项目实施有关的文档、代码或数据均应纳入配置管理，这样可以实现对项目实施中的每一项工作进行追溯，及时处理项目实施过程中出现的各类问题。

在软件系统投入运行之时，应对配置库进行整理和提炼，形成从项目启动到系统运行阶段，涵盖项目管理、软件开发、系 统集成等领域的一套完整的项目档案，随同软件系统正式交付给用户，并给予适当的培训和辅导，使用户能够快速有效地开展系统维护工作，为生产系统的稳定与可靠运行提供了保证。

三：国内外常见的 10 种配置管理工具

配置管理工具是配置管理相关理论的实践载体，工具的功能范围在某种程度上可以直接影响一个组织中配置管理水平的高低。所以，选择一个符合自己组织的配置管理工具，是一些企业建立配置管理规程，实施配置管理实践过程中的重要一 环。接下来通过将目前国内外的一些常用的配置管理工具的简介，希望能启到一个抛砖引玉的作用。

1：VSS-- Visual Source Safe

此工具是 Microsoft 提供的，是使用的相当普遍的工具之一，他可以与 VS.net 进行无缝集成，适合独立项目代码规模较小，基本上 Window 平 台上开发的中小型企业，当规模较大后，其性能通常是无法忍受的，对分支与并行开发支持的比较有限。其相关的外挂支持工具为 SAW,SOS.详细请见:
http://msdn.microsoft.com/zh-cn/library/ms181038(en-us).aspx

2：CVS--Concurrent Versions System

此工具是一个开源工具，与后面提到的 SVN 是同一个厂家：Collab.Net 提供的。此工具是相当著名，使用得相当广泛的版本控制工具之 一，使用成熟的"Copy-Modify-Merge"开发模型，可以大大的提高开发效率，适合于项目比较大，产品发布频繁，分支活动频繁的中大型项目。 可以与 Eclipse 等流行工具进行集成开发。详细请见：http://ximbiot.com/

3：SVN --CollabNet Subversion

此工具是在 CVS 的基础上，由 CollabNet 提供开发的，也是开源工具，目前越来越受到大家的欢迎，估计将来可能会成为最著名，使用最广泛的工具。他修正 cvs 的一些 局限性，适用范围同 cvs，目前有一些基于 SVN 的第三方工具，如 TortoiseSVN,是其客户端程序，使用的也相当广泛。在权限管理，分支合并等方 面做的很出色，他可以与 Apache 集成在一起进行用户认证。不过在权限管理方面目前还没有个很好用的界面化工具，SVNManger 对于已经使用 SVN 进行配置的项目来说，基本上是无法应用的，但对于从头开始的项目是可以的，功能比较强大，但是搭建 svnManger 比较麻烦。大家可以通过：http://www.collab.net/products/subversion/或 http://www.subversion.cn/ 来进行进一步的了解。

4、ClearCase

CC 是由 IBM Rational Software 提供，此软件是配置管理方面的高端软件，功能强大，属于阳春白雪级的产品，价格比较高。但是如果公司实力雄厚，采用此软件进行配置管理，相信一定受益匪浅。可以通过:http://www-306.ibm.com/software/awdtools/clearcase/support/index.html 来进一步了解。

5、PVCS

此工具由 Serena 公司提供，包括 PVCS Version Manager，PVCS Professional 系列商用软件，它们是非常完备的 SCM 软件，不仅包括了版本管理，而且包含了变更管理和过程管理，在性能上要优于 Clear case，价格上也比较高。HP 等大型公司给此工具以很高的评价。可以通过 http://www.serena.com/products/pvcs/index.html 来进一步了解。

6:SI

MK Source Integrity 是由 MKS 公司提供的。在操作上与 PVCS Version Manager 非常相似。SI 最强有力的特征之一是"变更包"，它可以用来保存单项变更任务或一组变更，在主干与分支的开发路径之间来回移动。您可以通过 http://www.mks.com/products/sie/ 来进一步的了解。

7:BitKeeper

它是由 BitMover 公司提供的，BitKeeper 自称是"分布式"可扩缩 SCM 系统。不是采用 C/S 结构，而是采用 P2P 结构来实现的，同样支持变更任务，所有变更集的操作都是原子的，与 svn,cvs 一致。您可以通过 http://www.bitkeeper.com/ 进一步了解。

8:AccuRev

一个相对较晚出现的工具，由 AccuRev 公司提供，它不太著名，其最大特征之一是，它是"时间安全"，它里面有强有力的"流"的概念。你可以用这个"流"作为码线，工作区，tag 等。您可以通过：http://www.accurev.com/ 进一步了解。

9:Perforce

很常用的，简单而功能很强的商业 SCM，对分支，合并有着特别的支持，由 Perforces Software 提供，自称是"快速软件配置管理系统"。可以通过

http://www.perforce.com 进一步了解。

10：Synergy

CM Synergy ，原名是 Continuus 由 Telelogic 提供，是非常强有力的、以过程为中心的 SCM 工具，具有配置能力很强的工作流，可以到 http://www.telelogic.com/products/synergy/ 进一步了解。

四：版本控制工具的应用分为六个层次

近年来,我们对版本控制工具的关注点似乎正在改变.起初,主要也是唯一的目的就是对代码进行监控,使我们能够安全的返回到旧的版本,以便能够诊断代码中的问题.后来,我们的关注点更侧重于如何使人与人之间的合作更为顺畅. 这个关注点并不是要取代对代码的监控,而是以代码监控为基础,并建立于其上的.现在,我们又越来越关注使用这些工具来描述代码的变更,因此就出现了对于重写代码历史命令(history rewriting command)的需求.当然,对代码变更的描述也同样需要建立在前两种关注点之上.我们可以把版本控制工具的应用分为如下几个层次:

Level-0. 没有版本控制

完全没有版本控制的解决方案,或者就使用一个共享的文件系统,并对其做定期的备份.一个开发人员,或者最多几个开发人员在没有工具的情况下共享代码,其面临的风险可想而知:

- 在任何时刻,代码可能都是不兼容的.
- 代码可能会由于开发人员的错误面丢失.
- 如果开发人员想要改写覆盖他人所做的修改,那再容易不过了.

Level-1. 初步的探索

这时的工具有了基本的版本控制功能,如 checkout,版本记录和锁文件.通常这就意 味着开发人员在同一份代码上工作,而代码的同步就会依赖于每个人代码文件的锁状态.这种工具在扩展和长期工作上都会有问题.对资源的重命名难到几乎不可能 完成.Branching 和 Tagging 操作则会需要同时操作三份代码的权限,而且可能还需要一个宰好的或者两柱香.如, VSS.

- 开发人员拥有了网络上的工作空间,他们无法在线下工作.
- 运行一次代码构建也许就意味着有时间可以去好好吃一顿了.
- 重构即使能够进行,也慢的要死.
- checkout 代码可能需要一整夜.
- checkin 代码也很慢.
- 没有原子提交.
- Branching 和 Tagging 操作的代价昂贵.
- 个人或是本地建立 branch 就意味着再次的 checkout.
- 集中式的,而不是公布式的.
- 合并点的跟踪很慢或是根本无法使用.
- 这时的工具还没有办法理解重命名的合并.
- 代码库有时会崩溃,需要较高的专家/开发人员比例,如 1:10.

Level-2. 笨拙

- 开发人员有了本地的拷贝,并且可以在线下工作.
- 本地的文件系统意味着构建的速度大大提升.
- 重构的时间只够喝杯茶了.
- checkout 的速度已经非常快了.

- checkin 也许还是慢一点.
- 仍然没有原子提交.
- Branching 和 Tagging 代价仍然昂贵.
- 集中式的,不是分布式的.
- 合并点的跟踪很慢或是根本无法使用.
- 没有办法合并重命名文件,需要在提交前使用一些扩展的跟进冲突解决机制.
- 代码库有时会崩溃,专家/开发人员比例已经得到优化,如 1:20.

如 CVS 和 TFS.

Level-3. 基本成型

- 开发人员拥有本地拷贝并且可以在线下工作.
- 在本地文件系统上可以进行快速的构建.
- 可以快速的重构.
- checkout 和 checkin 的速度都会非常的快.
- 终于有了原子提交.
- 轻量级的 Branching 和 Tagging 操作.
- 基本的合并操作.
- 个人/本地的 branching 操作仍然需要再次 checkout.
- 因为仍然是集中式的,而不是分布式的.
- 基本的合并点追踪.
- 没有办法合并重命名文件,需要在提交前使用一些扩展的跟进冲突解决机制.
- 代码库有时会崩溃,专家/开发人员比例已经很低,如 1:100.

如 Subversion.

Level-4. 有效并且可靠

- 开发人员拥有本地拷贝并且可以在线下工作.
- 在本地文件系统上可以进行快速的构建.
- 可以快速的重构.
- checkout 和 checkin 的速度都会非常的快.
- 无操作的代码同步和更新非常的快速.
- 终于有了原子提交.
- 轻量级的 Branching 和 Tagging 操作.
- 高级的 Branching 和合并操作.
- 个人/本地的 branching 操作仍然需要再次 checkout.
- 因为仍然是集中式的,而不是分布式的.
- 完善的合并点追踪.
- 合并重命名文件只能通过配置好的 branch 映射来实现,否则就需要在提交前进行修订.
- 代码库很少会崩溃,专家/开发人员比例非常低,如 1:1000.

如 Perforce.

Level-5. 高速,无形,高度可用

- 开发人员拥有本地拷贝并且可以在线下工作.
- 在本地文件系统上可以进行快速的构建.
- 可以快速的重构.
- checkout 和 checkin 的速度都会非常的快.
- 无操作的代码同步和更新非常的快速.
- 终于有了原子提交.

- 轻量级的 Branching 和 Tagging 操作.
- 高级的 Branching 和合并操作.
- 非常高效的个人/本地 branching 操作.
- 分布式的,而不是集中式的.
- 完善的合并点追踪.
- 无缝合并重命名文件,无需任何配置.
- 代码库很少会崩溃,专家/开发人员比例几乎为零,如 1:10000.

如 Git 和 Mercurial.

五：集中式 SVN 还是分布式 GIT？

通过前面版本控制工具的演化过程,我们基本上可以看到分布式工具的特点和优势了.相对于以往的客户-服务器端的集中式系统,它所采用的是一种 P2P 的方式.客户端不再需要从一个单一的中央代码库同步代码,每一个端点的代码的拷贝都是真正的代码库.分布式的版本控制系统是通过端点之间交换补丁（patch)的方式来同步代码的,而这种方式就决定了分布式系统与集中式系统的几个重要的区别:

- 默认情况下,没有标准的代码库参照;只有工作代码的拷贝.
- 由于不需要与中央服务器进行通信,因此一般的操作(如提交,查看历史和还原修改等)的执行速度非常快.只有在向其它端点 push 代码更改或者从其它端点 pull 代码更改的时候才会需要进行通信.
- 每一份代码拷贝都可以作为代码库及其更改历史的一份远程备份,这就为数据丢失提供了天然的保护.
- 鼓励测试性的 branch - 创建或者销毁 branch 的操作简单而且快速.

- 同伴之间的合作变得非常容易.

现在的项目用的是 Subversion 做版本控制,但是我们自己用的是 git-svn,这里主要通过这两种有代表性的工具来比较一下集中式和分布式工具的优势.

Subversion 提倡单一的中央代码库模型,不提倡大规模的 branching.在一个使用持续集成的环境中,其实也就是我们每天工作的环境,这个模型是非常合适的.这也是 Subversion 为什么这么流行,应用范围这么广的原因之一.

虽然分布式的系统使你拥有足够的灵活性来自己安排自己的工作流程,但是其实大多数人的工作模式仍然是使用持续集成,这也就意味着需要一条可以共享的 代码库主线.现在的版本控制系统已经有了神奇的合并工具,但这些合并仍然只限于文本.所以对于语义上的一致,仍然需要持续集成来保证.结果就是即使一个团 队在使用分布式的版本控制系统,他们仍然会需要一个主版本库.

即使如此,分布式系统仍拥有一些 SVN 无法提供的体验.

- 在分布式管理系统中,你可以在自己本地磁盘上拥有代码库的完整拷贝,对代码库的操作不需要通过网络向中央服务器进行请求,因此速度会非常的快.特 别是你在进行查看日志,与旧版本代码进行比较或者其它需要完整代码库的操作时,这种速度上的改善会非常明显.对于集中式的系统,在局域网内你也许只会觉得 有点慢,但如果当你工作在一个分布式的项目中,你的代码库在另一个大洲的时候,这就会是非常大的问题了.
- 如果你经常在四处奔走,无法随时与代码库建立网络连接,那么一个分布式的管理系统会使你可以随时与代码库一起工作.你可以随时随地提交你的工作,浏览历史,并且在比较版本间的差异.
- 还有一项体验也许并不能说是一个工具问题,而更多的是一个社会问题.分布式版本控制工具鼓励快速的 branching 和试验.在 SVN 中,你当然 也可以进行 branching 操作,

但是你所做的操作对于其它在这个代码库上工作的人员也是可见的，这也许并不是什么大问题,但确实会降低人们进行一些实验性工作的欲望.分布式系统则会鼓励你为工作代码进行记录:你可以向你的本地代码库提交未完成的修 改,甚至是无法通过测试和无法编译的代码.你同样可以在 SVN 中进行这些操作,但是在公共空间中创建这些 branches 总是让人望而却步.

在某种特定情况下,SVN 也有其相应的优势,如果你需要对版本控制系统难以合并的二进制文件(如 word 文档或者 ppt)进行管理的话,你就应该回 退到独占式 checkout 的锁机制下,这就需要一个集中式的系统.另外,SVN 更容易上手:你有一个代码库,所有的更改都指向这个代码库,如果你知道如 何创建,提交以及 checkout,那你就可以开始使用它了,而像 branching,更新这些操作在使用过程中自然也就慢慢熟悉了.SVN 拥有一些非常 好用的客户端软件,而且几乎所有的主流 IDE 都有与 SVN 集成的插件,这些都能够为你使用 SVN 提供很大的帮助.

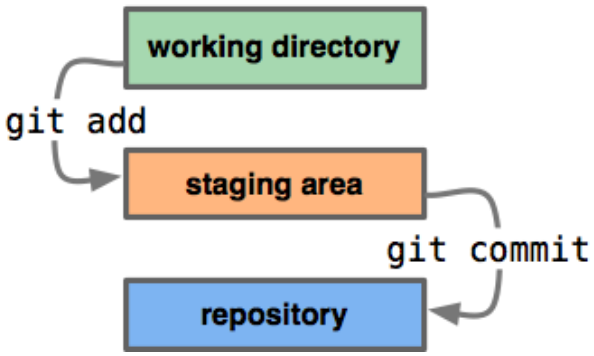
Git 则增加了复杂性,似乎总是有两种模式来进行操作,checkout 和 clone, commit 和 push..... 你得知道哪些命令是针对本地进行操作的,哪些命令是对服务器或者是主代码库进行操作的. 实际上,Git 的命令和思维模式与其它的版本控制系统是有所不同的.Bulter Cole 曾这样形容 Git:"它是一个神奇的功能强大的东西,它几乎可以做任何你让它做的事情,只有你知道如何让它做".Git 的反对者也会抱怨 Git 缺 少可发现性,你很难从它的表面设计推断出它的行为.而 Git 的支持者则认为这只是因为 Git 使用了不同于其它系统的思维模式,你需要先忘掉以前那些关于版 本控制系统的知识才能更好的来欣赏 Git.无论如何,Git 对于那些喜欢研究事物内部工作机制的人来说,还是非常有吸引力的.

通常来讲,Git 相比 Mercurial 在处理 branching 方面表现更好,尤其是用于试验和检查点的短期 branch.Mercurial 提 倡是另一种机制,例如快速的 clone 一份代码库或者是使用补丁,

但是 Git 的 branching 模式更为简单好用.Mercurial 在处理大型二进制 文件时同样有问题.通常的建议是使用 SVN 来管理二进制文件,如果你只有很少的二进制文件需要管理,不值得建立单独的管理机制的话,Mercurial 将 就着也能处理.

此外 Git 之所以能在网络上引起如此多的共鸣,有一个很大的原因是 Git 对于开源项目来说是完善的选择.你可以新建一个项目分支,向你自己的项目分 支提交修改,然后让项目维护人员将你的修改 pull 过去.有了 Git,这一切就变得如此的方便和自然.即使你没有向项目提交修改的权限,你也可以在线上建 立你自己的代码库,将你自己的补丁发布出来,任何喜欢你补丁的人都可以将它们 pull 到他们自己的代码库中,当然也包括项目维护人员.

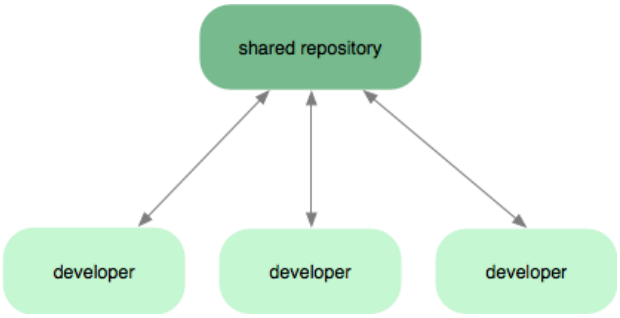
Git 有一个被称为"staging area"的区域.在你向代码库提交之间, 你可以在这个中间区域中构建你的提交.更为重要的是,你可以只提交部分的修改,而不是将所有修改的文件都进行提交.你甚至可以只提交一个文件中修改的一部分.



Git 非常的灵活,非常的 TIMTOWTDI(There is more than one way to do it). 你可以使用任何你喜欢的工作流程,Git 都会对其提供支持.

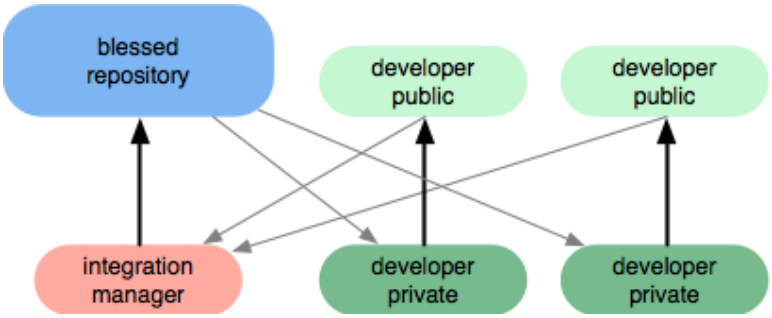
当前主要的工作流程有以下几种:

1. SVN 形式



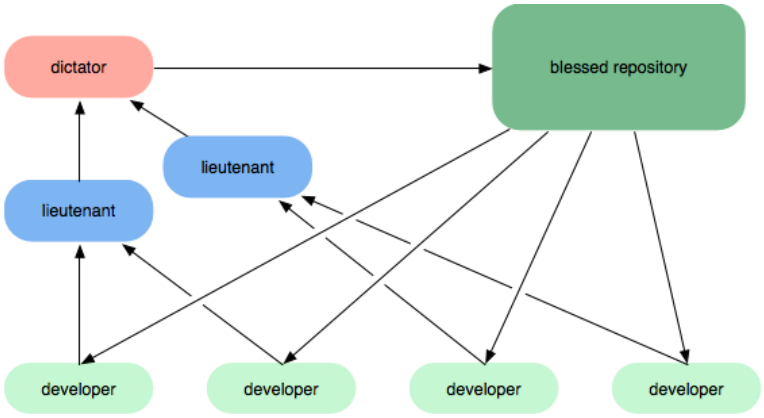
集中式的工作流程,这也是一种非常普遍的 Git 的工作流程.如果在你上一次 fetch 代码之后有其他人进行了提交,那么 Git 将不允许你向主代码库中 push 你的代码.

2. 集成管理形式



在这个工作流程中有一个集成管理人员,他向"blessed"代码库进行提交,其他开发人员从这个代码库 clone 代码,在他们自己的代码库中 push 修改,并让集成管理人员 pull 他们的修改.这其实就是大多数开源项目和 GitHub 所经常使用的开发模式.

3. 独裁者和中尉形式



对于更大规模的项目来说,你可以将开发人员的开发模式设置成类似于 Linux 内核的开发模式.某些人会负责项目的某个特定的子系统(中尉),并且将 所有关于这个子系统的修改都进行合并.另外会有一个集成者(独裁者)可能从他/她的中尉那里 pull 代码的修改,并将"blessed"代码库进行提交. 而所有人都可以从"blessed"代码库进行代码的拷贝.

再次强调,Git 对于工作流程的支持非常的灵活,你可以根据自己的需要来匹配,混合,选择这些工作流程.我们再来看看相对于 SVN,Git 还有哪些优点吧:

- Git 有一个"clean"命令.SVN 急需这个命令.
- Git 有一个"bisect"命令.
- SVN 会在每一个文件夹中创建一个.svn 目录.而 Git 只会创建一个.git 目录.
- 在 SVN 中,每一个文件或文件夹都可能来自于一个不同的版本或是 branch.这很可能会引起混乱.
- 无论你什么时候删除了点东西,你都需要告诉 SVN 一声.Git 会自己发现并处理.
- 在 Git 中,忽略语法很简单,例如*.pyc,它会被应用到所有的子文件夹.当然,如果你只想忽略某个特定文件夹中的内容,也是可以的.在 SVN 中,很难有什么方法可以将

一个忽略模式应用到所有的子文件夹中.

- Git中忽略设置是"private"的,这些设置包含在.git/info/exclude中,并不会影响到其他人.
- Git 跟踪的是内容而不是文件,它对于重命名文件的合并有更好的支持.
- Git 代码库的大小相对于 SVN 来说小很多.
- 在 Git 中,你可以重写历史.在你提交前,这会对准备补丁集以及修改以前的错误提供很大的帮助.
- 另外还有一点是在我们目前这个特殊的环境中所遇到的问题,我们每天都在结对编程,而且我们会经常的更换 pair,甚至是在一个 story 做到一半 的时候.而这里代码并没有完整的提交,而且只存在在一台机器上,而这台机器的主人也许会需要去做其它的 story.这时如果使用 git-svn 的话就会很 方便.因为我们不需要将没有完成,甚至不能编译的代码提交到 SVN 上去.

据说目前 Git 不支持代码库的部分 checkout/clone,但是正在开发中,而且已经有 submodule 方面的支持.SVN 则可以根据需要 只从代码库中 checkout 某个子文件夹.SVN 的版本号更短并且可以预知,而 Git 的版本号则是 40 位的 16 进制数字串.而 Git 在Branch方面 的处理应该是有很大的优势,但是由于我目前为止几乎没有使用过branch,所以这一部分还没有深刻的体会.

参考资料：

《做好软件项目的配置管理》计算机世界报 华国栋

《分布式和集中式版本控制工具-svn,git,mercurial》 CSDN zl728 网友

》