配置管理工作分享

# 认识配置管理

A discipline applying technical and administrative direction and surveillance to identify and document the functional and physical characteristics of a configuration item , control changes to those characteristics,record and report change processing and implementation status, and verify compliance with specified requirements.

一套应用技术上和管理上的指导和监督的方法，用来识别和记录配置项的功能特征和物理特征；控制这些特征的变更；记录和报告变更的处理和执行的状态，以及验证其是否符合特定的需求；

## 配置管理6个职能

### 源代码管理

涉及控制每一段计算机代码，包括源代码、配置文件、二进制文件和所有编译及运行时依赖关系。我们称这些工件为配置项—CI configuration Iteam

主要目的是有效的保护所有的项目资源。

还有一个关键配置管理功能，创建一个特定里程碑永久记录—代码基线。

也包括创建代码变体来管理并行开发，错误修正。

源代码管理原则：

* 代码受控，不能丢失；
* 在特定的里程碑或者时间点及时的建立基线；
* 通过合适的分支管理代码变体；
* 分支上的代码变化可以合并回主线（或其他分支）；
* 源代码管理的过程须可重复、轻量化和精益；
* 源代码管理具备回溯能力，可跟踪所有的变化；

### 构建工程

构建工程实践应该是高效、可靠和可重复的。构建工程也包括嵌入配置识别所需，必不可少的版本ID到程序中。

涉及选择一个特定的代码变体（例如基线）来可靠的编译，链接和打包代码组件。构建工程提供了一个可重复的过程来管理编译依赖关系。持续集成CI—continuous integration

构建工程是高效的吧源代码 生成二进制文件的学科。

构建工程的原则：

* 构建易理解，可重现；
* 构建速度快，过程可靠；
* 每个配置项都可识别；
* 代码和编译依赖易确定；
* 代码仅需构建一次，就可部署到任何地方；
* 构建异常易识别和管理；
* 可以快速定位构建失败的原因（并解决）；

### 环境配置

涉及编译和运行时变化的处理。

涉及管理那些可能随着代码从开发到测试，再到生产的推进而改变的编译和运行时的依赖关系。环境配置还涉及管理开发环境、测试环境、集成环境和生产环境本身。

目标是始终指明正确的运行时资源。

环境控制原则：

* 正确识别和理解环境配置依赖关系；
* 环境当前状态可查询（例如：通过开发的端口查询到状态）；
* 产品部署之前，如果环境配置发生变化，应重新构建源代码；
* 应在可控和可预测的方式下进行环境配置的修改；
* 有相应的文档记录环境配置且所有相关方已理解其内容；

### 变更控制

防止未经授权的版本被推进到生产环境（或者是QA环境）；

优先变更控制：拟对代码做出的任何变动先经过审查（在实际改动发生前）然后再经授权（或拒绝）实施建议的修改。

变更控制的原则：

* 变更需计划，不应最后一分钟才去更改；
* 变更应易理解，包括其后续影响；
* 建立和实行针对变更的授权和批准流程；
* 建立针对紧急变更的流程；
* 评估变更控制流程，使其符合所有的配置管理流程；

### 发布工程

按照需要包装配置项，使之成为能够可靠提测和部署的组件。

在构建工程功能中构建的所有组件的打包和标识。

软件产品，保留独立的QA、集成和生产环境；互联网产品，直接交付成品给最终用户（或者通过一个自动安装过程推送变更给用户）

发布管理职能包括打包和配置标识过程，如不可变的版本ID和可在生产环境中验证的发布清单；

发布管理的原则：

* 用一个不可变的，易识别的版本ID标识发布的版本；
* 所有依赖项都应该打包到发布版本中；
* 自动化发布版本打包过程，避免人为犯错；
* 发布管理应快速，可靠；
* 建立一种可以审计发布安装包的机制，核实所有包含的配置项；
* 发布版本的内容（包括涉及的需求）应易理解；
* 发布管理是所有发布状态的信息源；

### 部署

按照需要发布预先安装包到QA或生产环境。

部署的主要目标是在不出现任何问题的情况将发布版本部署到生产环境上去；如果发布版本有问题，运营团队应该按照部署过程将版本回滚。所有更改可控，版本部署和回滚简单，可靠和可预期；

部署的原则：

* 部署版本和现场恢复的过程都应可靠且尽可能地简单；
* 部署版本和现场恢复的过程都应完全可追溯，所有变更都有审计日志；
* 只有授权人员才能参与部署；
* 开发人员和部署版本的团队之间应该职责明确；
* 任何未经授权的更改都应立刻被检测到；
* 有既定的程序检查生产（或QA）环境中的版本；
* 根据需要不断审查和改进部署过程；

配置管理3个基本目标：

能够轻松识别已部署到生产或（QA）环境中的代码（配置审计）

可以检索用于创建一个发布版本的全部源代码（以及其他配置项）的确切版本

必须能够创建一个工作区来完成一个小的“漏洞补丁”而不会因为错误版本的头文件（其他依赖关系）而造成回滚整个发布。

从项目的计划到最后运维中的一切代码和文档都应该在配置管理库中存放。

一个良好的配置管理体系，可以让我的开发过程回滚到任何一段时间之前的状态。

配置管理不仅会标记出软件开发过程的各个阶段和状态记录，

配置管理出现后会成为所有软件工程活动运行的基础载体的核心部分。

## 角色及分工

对于任何一个管理流程来说，保证该流程正常运转的前提条件就是要有明确的角色、职责和权限的定义。特别是在引入了软件配置管理的工具之后，比较理想的状态就是：组织内的所有人员按照不同的角色的要求、根据系统赋予的权限来执行相应的动作。因此，在本文所介绍的这个软件配置管理过程中主要涉及下列的角色和分工：

### 项目经理（Project Manager，PM）：

项目经理是整个软件研发活动的负责人，他根据软件配置控制委员会的建议批准配置管理的各项活动并控制它们的进程。其具体职责为以下几项：

指定项目CM工程师。

* 参与制订和批准配置管理计划。
* 审批配置库目录权限申请。
* 评审和确定基线的发布。
* 受理变更，确定配置项变更策略，组织变更的实施和跟踪。
* 配合公司CM管理员完成项目外部的配置审计。

### 配置控制委员会（Configuration Control Board，CCB）：

CCB一般由项目经理、客户代表、技术代表、业务代表和市场代表等组成，中小型项目中，CCB人员可裁剪，由项目经理兼任。

负责指导和控制配置管理的各项具体活动的进行，为项目经理的决策提供建议。其具体职责为以下几项：

* 定制开发子系统；
* 定制访问控制；
* 制定常用策略；
* 建立、更改基线的设置，审核变更申请；
* 根据配置管理员的报告决定相应的对策。

### [配置管理员](http://baike.baidu.com/view/3917295.htm)（Configuration Management Officer，CMO）：

根据配置管理计划执行各项管理任务，定期向CCB提交报告，并列席CCB的例会。其具体职责为以下几项：

* 软件配置管理工具的日常管理与维护；
* 提交配置管理计划；
* 各[配置项](http://baike.baidu.com/view/729243.htm)的管理与维护；
* 执行[版本控制](http://baike.baidu.com/view/183136.htm" \t "_blank)和变更控制方案；
* 完成配置审计并提交报告；
* 对开发人员进行相关的培训；
* 识别软件开发过程中存在的问题并拟就解决方案。

### 系统集成员（System Integration Officer，SIO）：

系统集成员负责生成和管理项目的内部和外部发布版本，其具体职责为以下几项：

* 集成修改；
* 构建系统；
* 完成对版本的日常维护；
* 建立外部发布版本。

### 开发人员（Developer，DEV）：

开发人员的职责就是根据组织内确定的软件配置管理计划和相关规定，按照软件配置管理工具的使用模型来完成开发任务。

 配置项的日常操作，执行配置项变更。

 配合项目CM工程师实施配置活动。

## 公司对配置管理相关制度规定

### 配置库建立

解项目情况，征询项目经理意见，决定是否建立新的配置库。产品升级立项及基于已有产品的合同项目，可使用原产品的配置库。配置库类型为VSS/CVS/SVN，项目组需采用其它类型配置库时，需向CM管理员申请。

项目长期在外开发时，项目经理可向CM管理员申请在外地建立配置库，但目录结构使用公司统一标准，并每月同步回公司。人力外包项目、纯工程实施项目和客户有特殊要求时，可申请剪裁。

目录结构

### 配置库权限设置：

一般项目经理拥有全部配置项的读写权限，开发工程师拥有开发文档配置项的读写权限，以及代码配置项的读写权限；测试工程师拥有测试目录的读写权限，发布包负责人拥有发布目录下的读写权限，新员工的权限可逐步开放。

### 基线管理

 项目策划阶段，项目经理应确定基线发布计划，可在《配置管理计划》中反映，开发阶段一般在里程碑点处建立基线，如需求基线、设计基线、测试基线等。维护阶段，可根据变更情况进行基线发布，每年至少应发布一次基线。

### 配置审计

|  |  |  |
| --- | --- | --- |
| **配置审计级别** | **方式和内容** | **备注** |
| 初级 | 在不编译的情况下，检查备份内容是否完整。 | 处于各阶段的在研项目、成熟产品。 |
| 中级 | 检查备份内容是否完整，部分或全部源代码是否通过编译，主功能验证是否正确，用户手册是否及时更新。 | 适用于技术支持类的产品项目。 |
| 高级 | 全部源代码进行编译，并对编译生成的执行码进行功能性测试，以检查备份内容是否完整、源程序是否可编译，源程序与需求和设计等文档是否保持一致，编译说明是否完整，执行码是否与手册类文档（如用户手册）保持一致。 | 研发项目在办理结项时必须进行结项高级配置审计。 |

高级配置审计的交付物标准和清单：

  项目全套源代码（存放于公司指定的配置库）

  配置库中具有完整的结项基线

  满足协同立项申请的验收标准

  需求规格说明书

  概要设计、数据库设计

  集成编译说明

  系统测试报告

  安装手册

  用户手册

### 配置项命名规则

1.1   文档类命名规则：

[项目简称-] 文档名称 [-模块/主题简称]

1.2   代码类配置项的命名应反映产品对象特性，如业务名称。

2    配置项版本编号规则

版本编号组成：

**m．n．j．k**

说明：

|  |  |  |
| --- | --- | --- |
| 编号 | 名称 | 说明 |
| m | 主版本号 | 初始值在立项时确定；版本升级时加1，m初定或变动时，n、j、k均为0。 |
| n | 次版本号 | 初始值在立项时确定；版本升级时加1，n初定或变动时，j、k均为0。 |
| j | 文档批准  或代码发布次数 | 初始时为0；每次批准后加1；j初定或变动时，k为0。 |
| k | 文档修改次数  或代码测试次数 | 即文档批准后到下一次批准前的修改次数，或代码发布后到下一次发布前的提交测试的次数，初始时为0；每次改动时加1。 |

注：配置管理系统不支持“**．**”分隔时，可改用短横线“-”。

3    配置项标签命名规则

**CI-版本编号-配置项状态简称**

各项以英文横线分隔，其中配置项状态为：

|  |  |  |
| --- | --- | --- |
| 类别 | 状态 | 简称 |
| 文档类 | 草稿Draft | D |
| 发布Release | R |
| 代码类 | 草稿Draft | D |
| 评审通过Pass | P |
| 测试Test | T |
| 发布Release | R |

如：CI-1.0.0.1-D

配置项标签无状态说明时，默认为发布状态。

4    基线标签命名规则

**BL-YYYYMMDD-项目阶段-基线状态简称**

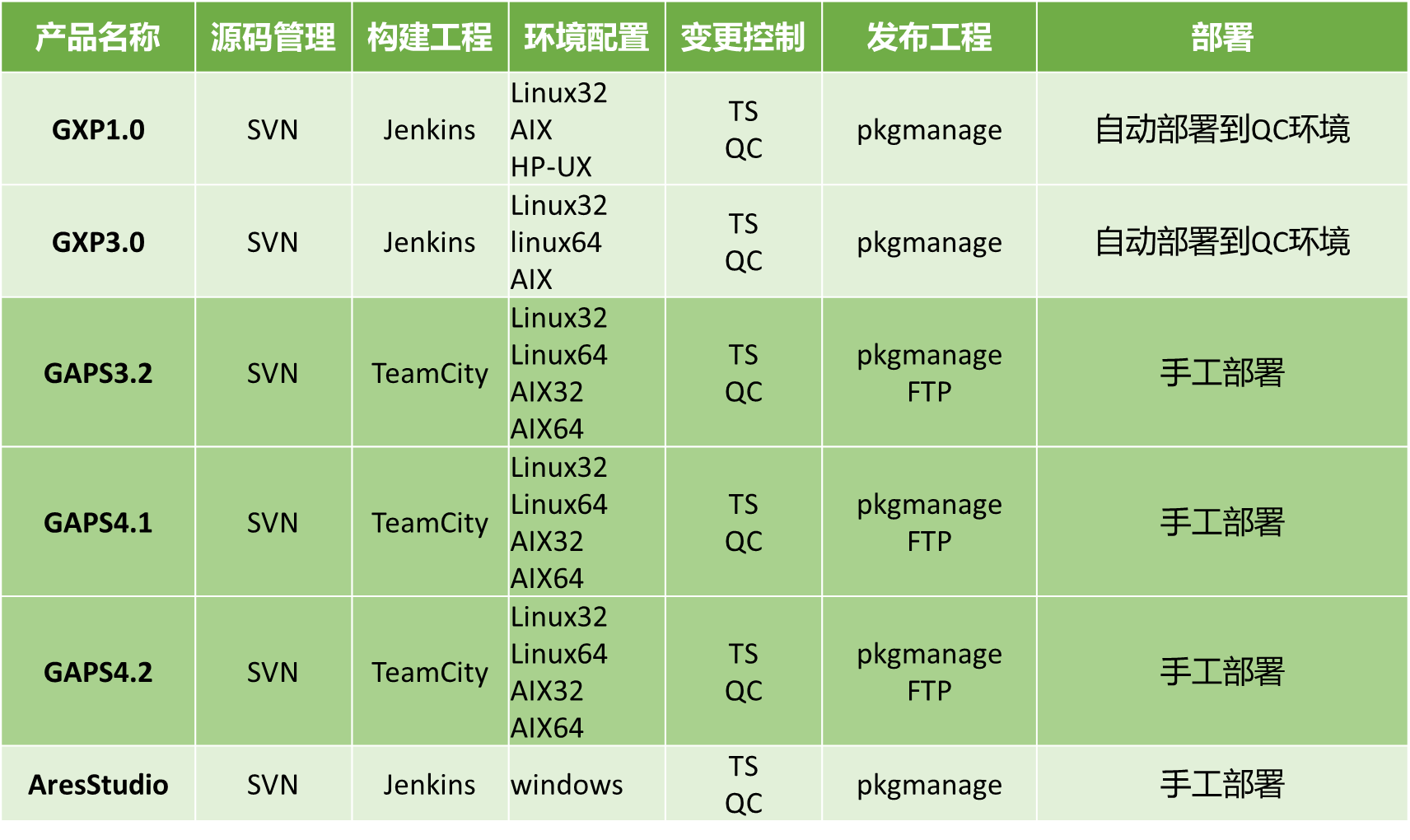
各项以英文横线分隔，其中YYYYMMDD为年月日，项目阶段分：

|  |  |  |
| --- | --- | --- |
| **类别** | **说明** | **简称** |
| 项目阶段 | 需求阶段 | Requirement |
| 设计阶段 | Design |
| 测试阶段 | Test |
| 稳定阶段 | Stable |
| 维护阶段 | Maintain |
| 基线状态 | 建立Establish | E |
| 发布Release | R |

如：BL-20071218-Design-R

# 项目配置管理

## 项目配置管理工作情况介绍



# 配置管理工具

## 源代码管理工具

## SVN

## GIT

https://github.com/danlanqianlan0/git-demo.git

软件配置管理关心：这个文件的各个历史版本是否记录下来，以便今后翻阅；各次修改的修改者、修改单原因是否记录下来了，以便将来可以理解当时的情形，理解为什么做出这样的 改动。

一是程序修改前，从哪儿拿到最新版本；二是修改之后，把修改结果提交到哪儿；

用变更集来记录软件开发活动，便于明确做了哪些工作，谁完成的，什么时候完成的等团队协作必然遇到的问题。【补丁说明】

[GIT](http://git-scm.com/)不仅仅是个版本控制系统，它也是个内容管理系统(CMS),工作管理系统等。如果你是一个具有使用SVN背景的人，你需要做一定的思想转换，来适应GIT提供的一些概念和特征。所以，这篇文章的主要目的就是通过介绍GIT能做什么、它和[SVN](http://subversion.tigris.org/" \t "_new)在深层次上究竟有什么不同来帮助你认识它。

那好，这就开始吧…

**1.GIT是分布式的，SVN不是：**

这是GIT和其它非分布式的版本控制系统，例如SVN，CVS等，最核心的区别。如果你能理解这个概念，那么你就已经上手一半了。需要做一点声明，GIT并不是目前第一个或唯一的分布式版本控制系统。还有一些系统，例如[Bitkeeper](http://www.bitkeeper.com/" \t "_new), [Mercurial](http://mercurial.selenic.com/)等，也是运行在分布式模式上的。但GIT在这方面做的更好，而且有更多强大的功能特征。

GIT跟SVN一样有自己的集中式版本库或服务器。但，GIT更倾向于被使用于分布式模式，也就是每个开发人员从中心版本库/服务器上chect out代码后会在自己的机器上克隆一个自己的版本库。可以这样说，如果你被困在一个不能连接网络的地方时，就像在飞机上，地下室，电梯里等，你仍然能够提交文件，查看历史版本记录，创建项目分支，等。对一些人来说，这好像没多大用处，但当你突然遇到没有网络的环境时，这个将解决你的大麻烦。

同样，这种分布式的操作模式对于开源软件社区的开发来说也是个巨大的恩赐，你不必再像以前那样做出补丁包，通过email方式发送出去，你只需要创建一个分支，向项目团队发送一个推请求。这能让你的代码保持最新，而且不会在传输过程中丢失。[GitHub.com](http://www.github.com/)就是一个这样的优秀案例。

有些谣言传出来说subversion将来的版本也会基于分布式模式。但至少目前还看不出来。

**2.GIT把内容按元数据方式存储，而SVN是按文件：**

所有的资源控制系统都是把文件的元信息隐藏在一个类似.svn,.cvs等的文件夹里。如果你把.git目录的体积大小跟.svn比较，你会发现它们差距很大。因为,.git目录是处于你的机器上的一个克隆版的版本库，它拥有中心版本库上所有的东西，例如标签，分支，版本记录等。

**3.GIT分支和SVN的分支不同：**

分支在SVN中一点不特别，就是版本库中的另外的一个目录。如果你想知道是否合并了一个分支，你需要手工运行像这样的命令[*svn propget svn:mergeinfo*](http://jan.baresovi.cz/dr/en/subversion-mergeinfo)，来确认代码是否被合并。感谢Ben同学指出这个特征。所以，经常会发生有些分支被遗漏的情况。

然而，处理GIT的分支却是相当的简单和有趣。你可以从同一个工作目录下快速的在几个分支间切换。你很容易发现未被合并的分支，你能简单而快捷的合并这些文件。

[](http://jbcdn2.b0.upaiyun.com/2012/07/Git-branch-management-strategy1.png)

**4.GIT没有一个全局的版本号，而SVN有：**

目前为止这是跟SVN相比GIT缺少的最大的一个特征。你也知道，SVN的版本号实际是任何一个相应时间的源代码快照。我认为它是从CVS进化到SVN的最大的一个突破。因为GIT和SVN从概念上就不同，我不知道GIT里是什么特征与之对应。如果你有任何的线索，请在评论里奉献出来与大家共享。

更新：有些读者指出，我们可以使用GIT的SHA-1来唯一的标识一个代码快照。这个并不能完全的代替SVN里容易阅读的数字版本号。但，用途应该是相同的。

**5.GIT的内容完整性要优于SVN：**

GIT的内容存储使用的是[SHA-1](http://en.wikipedia.org/wiki/SHA-1" \t "_new)哈希算法。这能确保代码内容的完整性，确保在遇到磁盘故障和网络问题时降低对版本库的破坏。这里有一个很好的关于GIT内容完整性的讨论 –<http://stackoverflow.com/questions/964331/git-file-integrity>

# 话说Svn与Git的区别

字数1492 阅读11337 评论26 喜欢109

这篇主要是谈谈两者的区别，至于谁优谁劣看官自己思考吧！

把第一条理解到位思想到位了做起来才会有的放矢，其他几条都是用的时候才能体会到

   1) 最核心的区别Git是分布式的，而Svn不是分布的。能理解这点，上手会很容易，声明一点Git并不是目前唯一的分布式版本控制系统，还有比如Mercurial等，所以说它们差不许多。话说回来Git跟Svn一样有自己的集中式版本库和Server端，但Git更倾向于分布式开发，因为每一个开发人员的电脑上都有一个Local Repository,所以即使没有网络也一样可以Commit，查看历史版本记录，创建项 目分支等操作，等网络再次连接上Push到Server端。

从上面看GIt真的很棒，但是GIt adds Complexity,刚开始使用会有些疑惑，因为需要建两个Repositories(Local Repositories & Remote Repositories),指令很多，除此之外你需要知道哪些指令在Local Repository，哪些指令在Remote Repository。

   2)Git把内容按元数据方式存储，而SVN是按文件：因为,.git目录是处于你的机器上的一个克隆版的版本库，它拥有中心版本库上所有的东西，例如标签，分支，版本记录等。.git目录的体积大小跟.svn比较，你会发现它们差距很大。

   3) Git没有一个全局版本号，而SVN有：目前为止这是跟SVN相比Git缺少的最大的一个特征。

   4) Git的内容的完整性要优于SVN: GIT的内容存储使用的是SHA-1哈希算法。这能确保代码内容的完整性，确保在遇到磁盘故障和网络问题时降低对版本库的破坏。

   5) Git下载下来后，在OffLine状态下可以看到所有的Log,SVN不可以。

   6) 刚开始用时很狗血的一点，SVN必须先Update才能Commit,忘记了合并时就会出现一些错误，git还是比较少的出现这种情况。

   7) 克隆一份全新的目录以同样拥有五个分支来说，SVN是同时复製5个版本的文件,也就是说重复五次同样的动作。而Git只是获取文件的每个版本的 元素，然后只载入主要的分支(master)在我的经验,克隆一个拥有将近一万个提交(commit),五个分支,每个分支有大约1500个文件的 SVN,耗了将近一个小时！而Git只用了区区的1分钟！

   8) 版本库（repository):SVN只能有一个指定中央版本库。当这个中央版本库有问题时，所有工作成员都一起瘫痪直到版本库维修完毕或者新的版本库设立完成。而 Git可以有无限个版本库。或者，更正确的说法，每一个Git都是一个版本库，区别是它们是否拥有活跃目录（Git Working Tree）。如果主要版本库（例如：置於GitHub的版本库）发生了什麼事，工作成员仍然可以在自己的本地版本库（local repository）提交，等待主要版本库恢复即可。工作成员也可以提交到其他的版本库！

   9)分支（Branch）在SVN，分支是一个完整的目录。且这个目录拥有完整的实际文件。如果工作成员想要开啟新的分支，那将会影响“全世界”！每个人都会拥有和你一样的分支。如果你的分支是用来进行破坏工作（安检测试），那将会像传染病一样,你改一个分支，还得让其他人重新切分支重新下载，十分狗血。而 Git，每个工作成员可以任意在自己的本地版本库开啟无限个分支。举例：当我想尝试破坏自己的程序（安检测试），并且想保留这些被修改的文件供日后使用， 我可以开一个分支，做我喜欢的事。完全不需担心妨碍其他工作成员。只要我不合并及提交到主要版本库，没有一个工作成员会被影响。等到我不需要这个分支时， 我只要把它从我的本地版本库删除即可。无痛无痒。

    Git的分支名是可以使用不同名字的。例如：我的本地分支名为OK，而在主要版本库的名字其实是master。

    最值得一提，我可以在Git的任意一个提交点（commit point）开启分支！（其中一个方法是使用gitk –all 可观察整个提交记录，然后在任意点开啟分支。）

    10)提交（Commit）在SVN，当你提交你的完成品时，它将直接记录到中央版本库。当你发现你的完成品存在严重问题时，你已经无法阻止事情的发生了。如果网路中断，你根本没办法提交！而Git的提交完全属於本地版本库的活动。而你只需“推”（git push）到主要版本库即可。Git的“推”其实是在执行“同步”（Sync）。

最后总结一下：

SVN的特点是简单，只是需要一个放代码的地方时用是OK的。

Git的特点版本控制可以不依赖网络做任何事情，对分支和合并有更好的支持(当然这是开发者最关心的地方)，不过想各位能更好使用它，需要花点时间尝试下。

参考:

[git和SVN的区别 - 生命在于折腾 - 博客频道 - CSDN.NET](http://blog.csdn.net/bruce_6/article/details/38299677)

[svn - Why is Git better than Subversion? - Stack Overflow](http://stackoverflow.com/questions/871/why-is-git-better-than-subversion" \t "_blank)

## 构建工程工具

## JENKINS

## Teamcity

## 变更管理

## TS

## 发布包管理

## PKGManage

包权限管理

包路径设置

包命名规范

SVN路径权限说明

## 配置管理工具对比

软件配置管理(Configuration Management)是通过技术或行政手段对软件产品及其开发过程和生命周期进行控制、规范的一系列措施。  
  
初读上述定义，感觉不是一点点的有点绕口，枯燥和眩晕了，实际上如果仔细去研究软件工程的理论书籍，有一大堆关于配置管理的概念，定义，理论，工具，总之就是一些把初学者搞晕的东东。所以初学者最好避开这些理论，先去实践一个最基本的代码管理工具，如果公司有Clearcase环境当然是学习Clearcase比较好，如果没有，下载一些共享的代码管理，如CVS(现在已经升级到SVN了)，安装在自己的电脑上，把自己的HelloWorld代码全部用这些代码管理工具管理起来，就是基本的配置管理了，如果再改进一下HelloWorld，做一个2.0版本，那就已经初窥门径了。当然，路漫漫其修远兮，推开一扇门，后面是汪洋大海，慢慢的遨游吧。  
  
笔者有幸接触过以下几种常用的配置管理工具：VSS、SVN、Clearcase，在此做一个小小的总结，并Ctrl+C了以前一些网友的对比评论，不一定准确，只是希望通过这些总结对自己和初学者有所帮助。如果想进一步了解这些工具，请baidu和google，如果想深入了解，敬请到图书馆借书并实践。  
  
一、 Visual Source Safe( 简称 VSS )  
  
VSS是微软的产品，是配置管理的一种很好的入门级的工具。VSS最初的名字叫Source Safe，是一家小公司的产品，92年曾经获了最佳小型管理工具奖，然后立即被微软收购。但是微软收购的只是source safe的Windows版本，在美国还有另外两家公司分别获得了继续开发和销售source safe的Mac版本和Unix版本的许可，在MS买进vss之后，基本上没有对vss进行任何的研发，MS内部自身也不用vss。  
  
SourceSafe长得很象早先土气的文件管理器，的确难看。但是难看不碍事，SourceSafe的优点可以用8个字来概括“简单易用，一学就会”，这个优点是它老妈Microsoft遗传下来的，是天生的。虽然SourceSafe并不是免费的，但是在国内人们以接近于零的成本得到它，网上到处可以下载啊。当然Microsoft也不在乎这个小不点的软件，它属于“买大件送小件”的角色。如果你合法地得到Visual Studio，你就得到了免费的SourceSafe。  
  
评价如下：  
  
易用性：★★★★★  
  
易学易用是 VSS 的强项， VSS 采用标准的 windows 操作界面，只要对微软的产品熟悉，就能很快上手。 VSS 的安装和配置非常简单，对于该产品，不需要外部的培训（可以为公司省去一笔不菲的费用）。只要参考微软完备的随机文档，就可以很快的用到实际的工程当中。  
  
功能：★★★  
  
VSS 的配置管理的功能比较基本，提供文件的版本跟踪功能，对于 build 和基线的管理， VSS 的打标签的功能可以提供支持。 VSS 提供 share （共享 ) 、 branch( 分支）和合并（ merge) 的功能，对于团队的开发进行支持。 VSS 不提供对流程的管理功能，如对变更的流程进行控制。 VSS 不能提供对异地团队开发的支持。此外 VSS 只能在 windows 平台上运行，不能运行在其他操作系统上。  
  
安全性：★★★  
  
VSS 的安全性不高，对于 VSS 的用户，可以在文件夹上设置不可读，可读，可读 / 写 , 可完全控制四级权限。但由于 VSS 的文件夹是要完全共享给用户后，用户才能进入，所以用户对 VSS 的文件夹都可以删除。这一点也是 VSS 的一个比较大的缺点。  
  
总体成本：★★★★  
  
VSS 没有采用对许可证进行收费的方式，只要安装了 VSS ，对用户的数目是没有限制的。因此使用 VSS 的费用是较低的。  
  
技术支持：★★★★★  
  
由于 VSS 是微软的产品，可以得到稳定的技术支持。  
  
二、 SVN(Subversion) - CVS(Concurrent Version System)的替代和升级版本  
  
先说说CVS，CVS是开源代码的配置管理工具，其源代码和安装文件都可以免费下载。记得在学校读研的时候，学校实验室的代码全部都用CVS管理，为啥？很简单，两个字：免费！它与Eclipse配合，基本上Java的代码编写，代码管理和版本管理都可以“免费”搞定了。  
  
SVN(Subversion)是近年来崛起的版本管理工具，被誉为cvs的接班人。目前，绝大多数开源软件都使用svn作为代码版本管理软件。虽然在 2006年时SVN的使用族群仍然远少于传统的CVS，但已经有许多开放源码团体决定将CVS转换为SVN。已经转换使用SVN的包括了 FreeBSD、Apache Software Foundation、KDE、GNOME、GCC、Python、Samba、Mono 以及许多团体。许多开发团队换用SVN是因为 Trac、SourceForge、CollabNet、CodeBeamer等专案协同作业软件以及Eclipse、NetBeans等IDE提供SVN的支援整合。 除此之外，一些自由软件开发的协作网如SourceForge.net除了提供CVS外，现在也提供专案开发者使用SVN作为原码管理系统，JavaForge、Google Code以及 BountySource 则以SVN作为官方的源码管理系统。2009年，绝大多数CVS服务已经改用SVN。CVS已经停止维护。  
  
易用性 : ★★★★  
  
Subversion支持linux和windows，更多是安装在linux下。svn服务器有2种运行方式：独立服务器和借助apache。2种方式各有利弊。目前业界评价的SVN易用性正在提高。  
  
功能：★★★★  
  
SVN 的功能除具备 VSS 的功能外，还具有：  
  
它的客户机 / 服务器存取方法使得开发者可以从任何因特网的接入点存取最新的代码；它的无限制的版本管理检出 (checkout ：注 1)的模式避免了通常的因为排它检出模式而引起的人工冲突；它的客户端工具可以在绝大多数的平台上使用。同样， SVN 也不提供对变更流程的自动管理功能。  
  
安全性：★★★★  
  
一般来说， SVN 的权限设置单一，无法完成复杂的权限控制；但是 SVN 通过 SVN ROOT 目录下的脚本，提供了相应功能扩充的接口，不但可以完成精细的权限控制，还能完成更加个性化的功能。  
  
总体成本：★★★★★  
  
SVN 是开发源码软件，无需支付购买费用。  
  
技术支持：★★★  
  
同样因为 SVN 是开发源码软件，没有生产厂家为其提供技术的支持。如发现问题，通常只能靠自己查找网上的资料进行解决。  
  
三、 ClearCase  
  
Rational公司的ClearCase是软件行业公认的功能最强大、价格最昂贵的配置管理软件。  
  
ClearCase主要应用于复杂产品的并行开发、发布和维护，其功能划分为四个范畴：版本控制、工作空间管理（Workspace Management）、构造管理（Build Management）、过程控制（Process Control）。ClearCase通过TCP/IP来连接客户端和服务器。另外，ClearCase拥有的浮动License可以跨越UNIX和 Windows NT平台被共享。  
  
ClearCase的功能比CVS、SourceSafe强大得多，但是其用户量却远不如CVS、SourceSafe的多。主要原因是：ClearCase价格昂贵，如果没有批量折扣的话，每个License大约5000美元。对于中国用户而言，这无疑是天价。用户只有经过几天的培训后（费用同样很昂贵），才能正常使用ClearCase。如果不参加培训的话，用户基本上不可能无师自通。  
  
易用性：★★★  
  
ClearCase 的安装和维护远比 VSS 复杂，要成为一个合格的 ClearCase 的系统管理员，需要接收专门的培训。 ClearCase 提供命令行和图形界面的操作方式，但从 ClearCase 的图形界面不能实现命令行的所有功能。如果Unix/Linux服务器上安装了Samba服务，可以直接通过Windows下Clearcase的客户端直接访问源代码，相对比较简单。  
  
功能：★★★★★  
  
ClearCase 提供 VSS, SVN所支持的功能，但不提供变更管理的功能。 Rational 另提供了 ClearQuest 工具提供对变更管理的功能，与 VSS不同， ClearCase 后台的数据库是专有的结构。 ClearCase 对于 windows 和 unix 平台都提供支持。 ClearCase 通过多点复制支持多个服务器和多个点的可扩展性，并擅长设置复杂的开发过程。  
  
安全性：★★★★  
  
ClearCase 的权限设置功能与 SVN 相比， SVN 有独立的安全管理机制， ClearCase 没有专用的安全性管理机制，依赖于操作系统。  
  
总体成本：★★  
  
要选用 ClearCase ，需要考虑的费用除购买 license 的费用外，还有必不可少的技术服务费用，没有 Rational 公司的专门的技术服务，很难发挥出 ClearCase 的威力。如现在网上虽有 ClearCase 的破解软件，但尝试应用的公司大多失败的缘故。另外，对于 web 访问的支持，对于变更管理的支持功能都要另行购买相应的软件。  
  
技术支持：★★★★★  
  
Rational 公司已被 IBM 公司收购，所以有可靠的售后服务保证。  
  
  
四、总结  
  
工具对比一览表  
  
特性           VSS           SVN             ClearCase  
  
易用性         ★★★★      ★★★          ★★★  
  
功能           ★★★        ★★★★        ★★★★★  
  
安全性         ★★★        ★★★★        ★★★★  
  
总体成本       ★★★★      ★★            ★★★★★  
  
技术支持       ★★★★      ★★★          ★★★★★  
  
  
以上几种工具的总结如下：   
  
1. VSS 的使用简便易学，但 VSS 的功能和安全性较弱，且只对 windows 平台进行支持，建议作为项目配置管理的入门时采用的工具；  
  
2. SVN 的安全性和版本管理功能较强，可以实现异地开发的支持，但 SVN 安装和使用多采用命令行方式，学习曲线高，同时不提供对变更管理的功能，对于小型团队，可以采用 SVN 进行管理。  
  
3. ClearCase 功能完善，安全性好，可以支持复杂的管理，但学习曲线和学习成本高，需要集成 ClearQuest 才能完成完整的配置管理功能。大公司如果采用异地多研发中心同时开发的模式，一般推荐使用ClearCase。

## 持续集成工具

## http://www.oschina.net/project/tag/344/ci