**软件产品与代码版本管理指南**

**目 录**

[1 版本管理 1](#_Toc50320247)

[1.1 代码版本库：Trunk Branch Tag使用 1](#_Toc50320248)

[1.2 版本命名规范 1](#_Toc50320249)

[1.2.1 版本号格式 1](#_Toc50320250)

[1.2.2 版本号修改规则 1](#_Toc50320251)

[1.2.3 先行版本号 2](#_Toc50320252)

[2 代码管理原则 3](#_Toc50320253)

[2.1 先更新，再提交（强制） 3](#_Toc50320254)

[2.2 多提交（建议） 3](#_Toc50320255)

[2.3 修改一个问题或者一类问题提交一次，不同类问题分开提交（建议） 3](#_Toc50320256)

[2.4 不要提交不能通过编译的代码,必须本地通过测试后才可提交（强制） 4](#_Toc50320257)

[2.5 每次提交必须书写明晰的标注（强制） 4](#_Toc50320258)

[2.6 提交时注意不要提交本地自动生成的文件（强制） 4](#_Toc50320259)

[2.7 不要提交自己不明白的代码（建议） 4](#_Toc50320260)

[2.8 慎用锁定功能（建议） 5](#_Toc50320261)

[附录 6](#_Toc50320262)

# 版本管理

代码版本库：Trunk Branch Tag使用

**Trunk :** 主开发目录，开发环境，没有版本号和发布名称，是项目最新进度的开发版本，如1.0.0开始开发，trunk此时为1.0.0的开发版。

**Branch :** 分支开发目录，环境基于目的来配置，基于任意进度版本（Trunk或者Tag），为某一具体目的进行copy并开发，一般有三类：准备发布的分支（进行生产环境的测试、准备）Release Branch 、Bug修复的分支（进行某编号的bug修复）Fix Branch 、新技术实验性分支（将某个新技术引进项目）Experimental Branch ，如fix-1.0.0\_898 (copy from tag/tag- release-1.0.0，bug版本号为898)，release-1.0.0 (1.0.0版本的Release Branch)，experimental-1.0.0\_netty(copy from tag/tag\_release\_1.0.0，Netty技术验证)，这些都要根据需要最终merge到Trunk里面。

**Tag** : Tag存档/备份目录（不允许修改），生产环境，基于已经冻结的Release Branches，进行copy并为版本打tag，如tag-release-1.0.0

版本命名规范

版本号格式

**主版本号[MAJOR].次版本号[MINOR].修订号[PATCH]**

例如：1.0​​.0-alpha、1.0.0-release

版本号修改规则

**主版本号**：当功能模块有较大的变动（不向下兼容），比如增加多个模块或者整体架构发生变化。此版本号由项目决定是否修改。

**子版本号**：当功能有一定的增加或变化（向下兼容），比如增加了对权限控制、增加自定义视图等功能。此版本号由项目决定是否修改。

**修订号**：一般是 Bug 修复或是一些小的变动（向下兼容），要经常发布修订版，时间间隔不限，修复一个严重的bug即可发布一个修订版。此版本号由项目经理决定是否修改。

**先行版本号**：此版本号用于标注当前版本的软件处于哪个开发阶段，当软件进入到另一个阶段时需要修改此版本号。此版本号由项目决定是否修改。

说明：先行版本号及版本编译信息可以加到“主版本号.次版本号.修订号”的后面，作为延伸；当主版本号升级后，次版本号和修订号需要重置为0，次版本号进行升级后，修订版本需要重置为0。

先行版本号

**alpha版**：此版本表示该软件在此阶段主要是以实现软件功能为主，通常只在软件开发者内部交流，一般而言，该版本软件的Bug较多，需要继续修改。

**beta版**：该版本相对于Alpha版已有了很大的改进，消除了严重的错误，但还是存在着一些缺陷，需要经过多次测试来进一步消除，此版本主要的修改对像是软件的UI。

**rc版**：发行候选版本。RC版不会再加入新的功能了，主要着重于除错，与即将发行的正式版相差无几。

**release版**：该版本意味“最终版本”，在前面版本的一系列测试版之后，终归会有一个正式版本，是最终交付用户使用的一个版本。该版本有时也称为标准版。一般情况下，Release不会以单词形式出现在软件封面上，取而代之的是符号(R)。

# 代码管理原则

先更新，再提交（强制）

代码更新的原则是要随时更新，随时提交。

当完成了一个小功能，能够通过编译并且自己测试之后，谨慎地提交。

如果在修改的期间别人也更改了svn/git的对应文件，那么commit就可能会失败。如果别人和自己更改的是同一个文件，那么update时会自动进行合并，如果修改的是同一行，那么合并时会产生冲突，这种情况就需要同之前的开发人员联系，两个人一起协商解决冲突，解决冲突之后，需要两人一起测试保证解决冲突之后，程序不会影响其他功能。

在更新时注意所更新文件的列表，如果提交过程中产生了更新，则也是需要重新编译并且完成自己的一些必要测试，再进行提交。

这样既能了解别人修改了哪些文件，同时也能避免SVN合并错误导致代码有错。

多提交（建议）

每次提交的间歇尽可能地短，以几个小时的开发工作为宜。

例如在更改UI界面的时候，可以每完成一个UI界面的修改或者设计，就提交一次。在开发功能模块的时候，可以每完成一个小细节功能的测试，就提交一次，在修改bug的时候，每修改掉一个bug并且确认修改了这个bug，也就提交一次。我们提倡多提交，也就能多为代码添加上保险。

修改一个问题或者一类问题提交一次，不同类问题分开提交（建议）

比如：修改了EPG菜单UI，又修改了伴音曲线，这种情况就要分开上传，不是一类的问题修改的分开上传；

不要提交不能通过编译的代码,必须本地通过测试后才可提交（强制）

代码在提交之前，首先要确认自己能够在本地编译。如果在代码中使用了第三方类库，要考虑到项目组成员中有些成员可能没有安装相应的第三方类库。项目经理在准备项目工作区域的时候，需要考虑到这样的情况，确保开发小组成员在签出代码之后能够在统一的环境中进行编译。

每次提交必须书写明晰的标注（强制）

在一个项目组中使用SVN/Git，如果提交空的标注或者不确切的标注将会让项目组中其他的成员感到很无奈，项目经理无法很清晰的掌握工作进度，无法清晰的把握此次提交的概要信息。在发现错误后也无法准确的定位引起错误的文件。所以，在提交工作时，要填写明晰的标注，能够概要的描述所提交文件的信息，让项目组其他成员在看到标注后不用详细看代码就能了解你所做的修改。

提交时注意不要提交本地自动生成的文件（强制）

例如eclipse中的.classpath文件，Windows生成的缩略图Thumbs.db，项目编译生成的临时文件.obj, .class等等。如果项目中没有进行这方面的配置来强行禁止提交这样的文件，请自觉不要提交这样的文件。提交了这样的文件后，别人在更新后就可能与本地的环境冲突从而影响大家的工作。

不要提交自己不明白的代码（建议）

代码在提交入SVN/Git之后，你的代码将被项目成员所分享。如果提交了你不明白的代码，你看不懂，别人也看不懂，如果在以后出现了问题将会成为项目质量的隐患。因此在引入任何第三方代码之前，确保你对这个代码有一个很清晰的了解。

慎用锁定功能（建议）

在项目中要慎用锁定的功能，在你锁定了一个文件之后别人就无法继续修改提交该文件，虽然可以减少冲突的发生率，但是可能会影响项目组中其他人员的工作。平时只有在编辑那些无法合并的文件（例如图片文件，flash文件等）时，才适当的采用锁定操作。

# 附录