Lecture2 SCM

1. 软件配置管理 Software Configuration Management (SCM)

SCM 的四个方面

- **变更管理 Change Control**
- **版本管理 Version Control**
- **搭建 Building**
- **发布 Releasing**

它们通过某些工具实现，需要专业知识和监督，在大型项目中非常重要

SCM 根据 SCI

一门控制软件系统发展的学科

标识 Identification

- Versions
- Baseline
  - Baseline 是一个经过评审并达成一致的软件配置项，它只能通过正式的变更控制过程进行更改
  - 没有被审查的中间版本是 SCI ，而不是 Baseline
- Release
  - 发布是一个软件配置项，开发人员给其他人
  - 发布是一个 Baseline

控制 Control

谁可以读写配置项?

你如何知道更改是否被允许/正确?

状态统计 Status Accounting

报告组件和更改请求的状态

- 哪个组件在这周有更改
- 哪个组件是 Bob 更改的
- 哪个组件更改的次数最多
- 哪些变更请求超过一个月，优先级为 3 或更高

审计和回顾 Audit and Review

- 我们如何知道现在构建的脚本是可行的
- 我们如何知道只有有权限的人才能更改数据库接口
- 我们可以运行 2015 年 9 月的版本吗?

SCM 正常运作依赖

- 政府官员
- 学科/纪律
- 工具
    - 版本控制：Git，Cvs，SVM
    - 变更控制：Bugzilla，Mantis，Jira
    - 搭建：Make，Ant，Mvn
    - 发布：Maven Central，Nexus

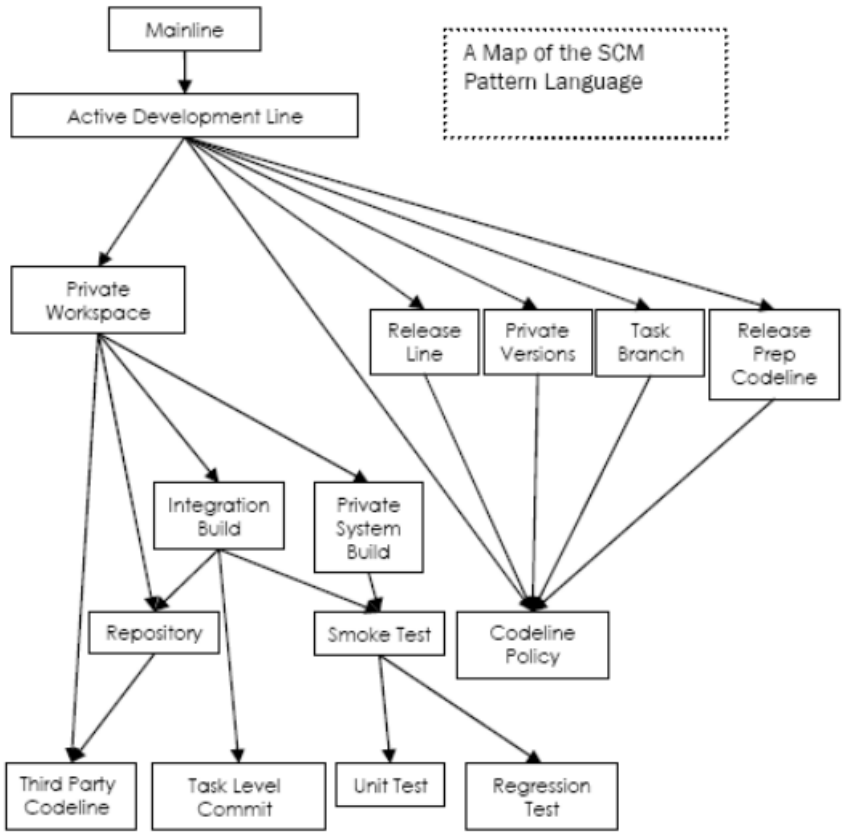
SCM 经理

复杂的工具需要专家来管理

SCM 专家需要

- 维护工具
- 维护配置文件，安排分支
- Merging
- 创建版本控制、变更控制策略

SCM 模式



测试 Test

- Smoke test
  - 确保系统在进行更改后仍然运行
- Unit test
  - 进行更改后，确保模块没有损坏
- Regression test
  - 确保在进行其他改进时，现有代码不会变得更糟

开发者问题 Developer issues

- 私人工作空间
  - 通过在私有工作区中进行开发，避免集成问题分散您的注意力，避免您的更改导致其他问题
- 私人系统构建
  - 在将更改提交到存储库之前，通过进行私有系统构建来避免破坏构建

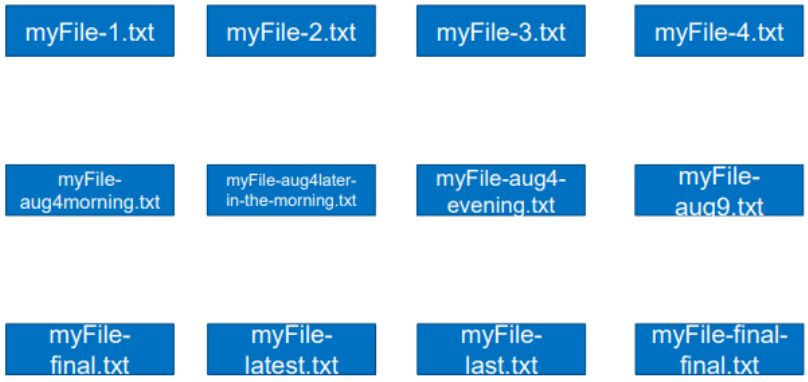
代码管理策略 Codeline Policy

- 活跃的开发线 Active Development Line
- 发布线 Release Line
  - 保存一个版本的错误修复
- 私有版本 Private Versions
- 测试分支 Task Branch
  - 向团队其他成员隐藏破坏性任务
- 准备发布分支 Release Prep Codeline

2. 版本管理 Version Control

传统的方式

但是到后面可能遇到这种情况



所以我们需要版本控制

版本控制系统 Version Control System

版本控制系统是一个跟踪的软件系统，对一组文件所做的更改，以便您可以召回特定的的版本

它可以为你提供

- 在项目中与多个其他开发人员协作，合并更改并解决冲突
- 恢复修改
- 倒回到一个特定的版本

> **folk 和 clone 的区别**
>
> - fork：在 github 页面，点击fork按钮。将别人的仓库复制一份到自己的仓库
> - clone：将 github 中的仓库克隆到自己本地电脑中

> **pull request 的作用**
>
> 比如在仓库的主人（A）没有把我们添加为项目合作者的前提下，我们将 A 的某个仓库名为“a”的仓库 clone 到自己的电脑中，在自己的电脑进行修改，但是我们会发现我们没办法通过 push 将代码贡献到B中
>
> 所以要想将你的代码贡献到 B 中，我们应该：
>
> 1. 在 A 的仓库中 fork 项目 a （此时我们自己的 github 就有一个一模一样的仓库 a，但是URL不同）
> 2. 将我们修改的代码 push 到自己 github 中的仓库B中
> 3. pull request ， 主人就会收到请求，并决定要不要接受你的代码
> 4. 也可以可以申请为项目 a 的 contributor，这样可以直接 push

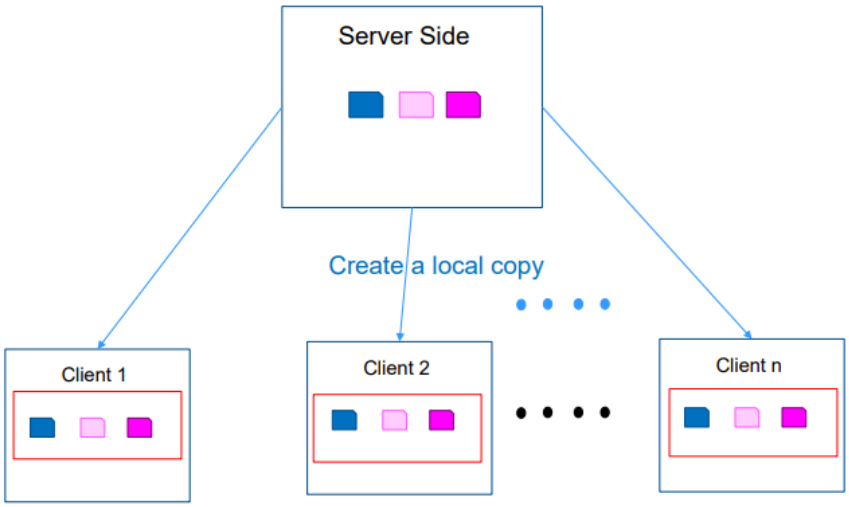
> **fork 了别人的项目到自己的 repository 之后，别人的项目更新了，我们 fork 的项目怎么更新**
>
> 首先 fetch 网上的更新到自己的项目上，然后再判断 merge

> **pull 和 fetch 有啥区别**
>
> fetch+merge与pull效果一样。但是要多用fetch+merge，这样可以检查fetch下来的更新是否合适
>
> pull直接包含了这两步操作，如果你觉得网上的更新没有问题，那直接pull也是可以的

SVN Apache Subversion

常见命令

local copy

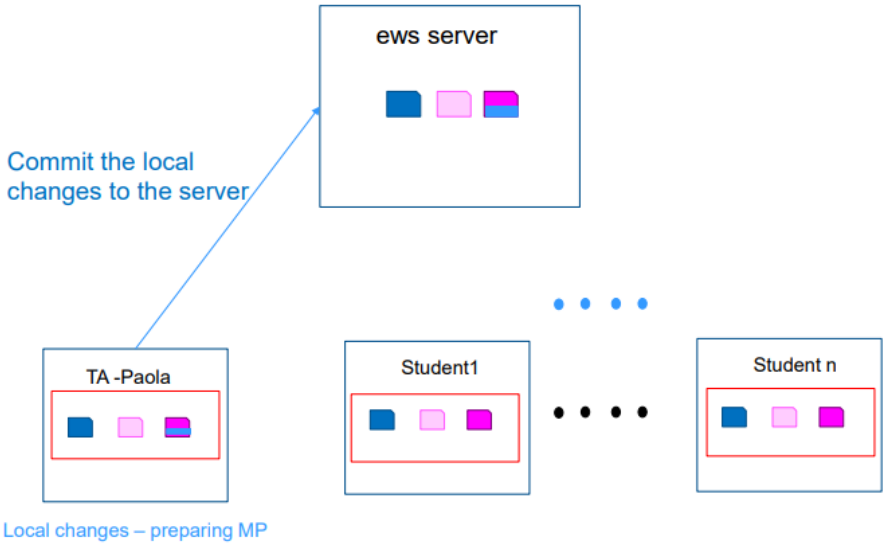


使用 SVN 创建一个 local copy

```
svn checkout <address_to_remote> <name_of_local_dir>
svn checkout https://subversion.ews.illinois.edu/svn/fa15-cs427/<netid> cs427
```

- 创建一个本地的叫做 cs427 的文件夹
- 将 ews 远程地址的数据拷贝到本地目录 cs427
- svn 被设置为跟踪对本地副本所做的任何更改

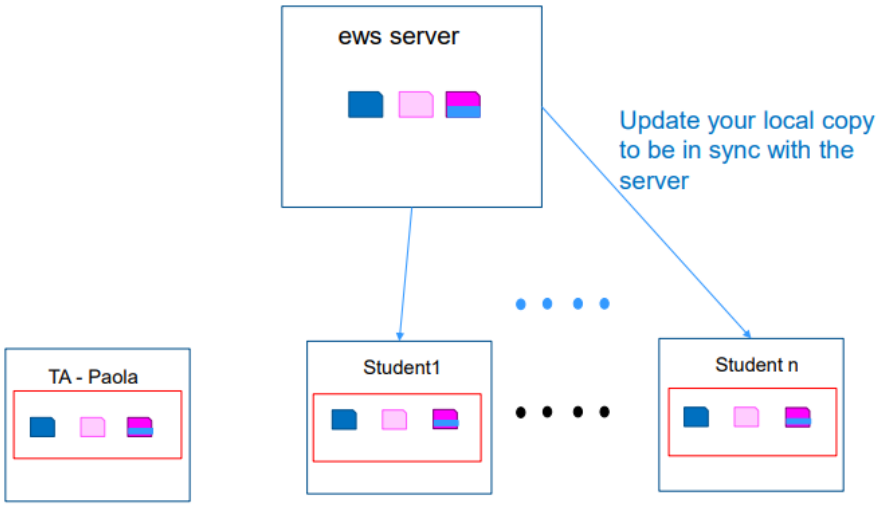
commit



提交本地文件的更改到服务器上

```
svn commit -m "<message>"\
svn commit -m “updated the grades for mp0”
```

update



从服务器上获取更改

```
1   svn up
```

add file

告诉 svn 有一个需要追踪的新文件

```
1   svn add <name_of_new_file>
```

其它命令

- `svn st`：显示当前 svn 目录下的文件状态
- `svn rm`：从跟踪的文件集中删除一个文件（也将在远程服务器上删除）
- `svn mv`：将文件从一个目录移动到另一个目录（如果在同一个目录，则重命名文件）
- `svn diff`：两个修订之间的差异，或差异文件以查看未提交的本地更改

注意事项

- 不要 commit 自动生成的文件
  - 例如，.class 文件可以被忽略
- 命令 `svn status` 显示所在的目录子树中的文件的状态（而不是整个repo）
- `svn commit`、`svn up` 等命令只对你当前所在的目录生效

SVN 目录布局

svn 目录的总体布局由 3 个目录组成

- **Trunk**
  - 最新的开发版本
- **Branches**
  - 发布，bug修复，实验
  - 不要因为下面的几种情况而分支

        - 支持不同的硬件
        - 支持不同的客户
- **Tags**
    - 标记代码的状态（例如发布）

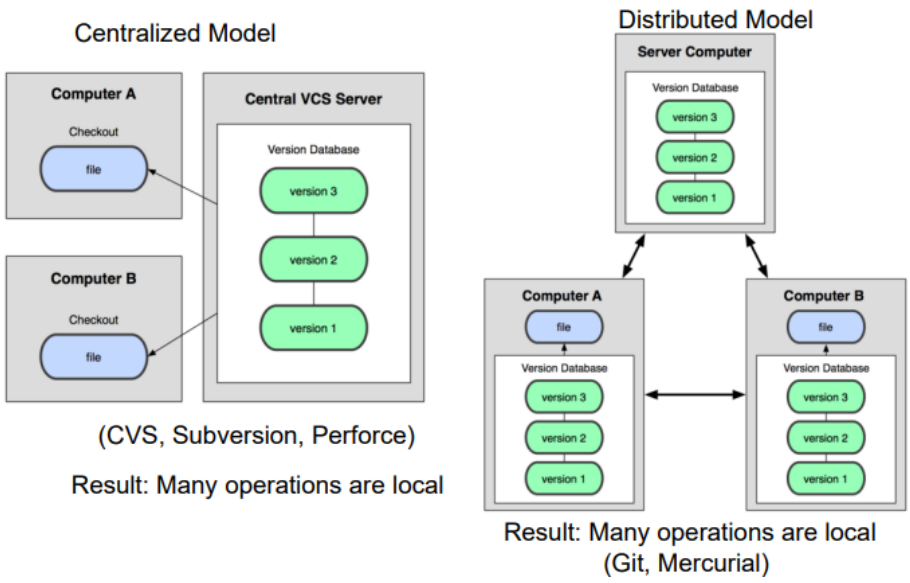
Git

Git 的历史

来自 Linux 开发社区，由 Linus Torvalds 开发出来

初始的愿景是

- 速度
- 支持非线性开发（数以千计的并行分支）
- 完全分布式的
- 可以高效承载大的项目，如 Linux



checksum

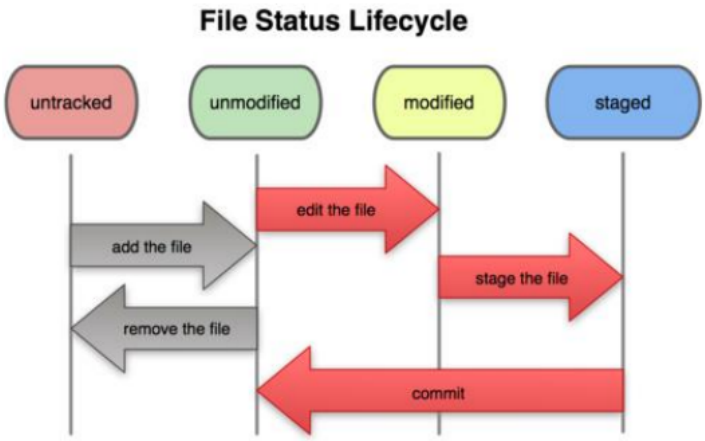
在 SVN 中，对中央 repo 的每一次修改都会增加整个 repo 的版本号

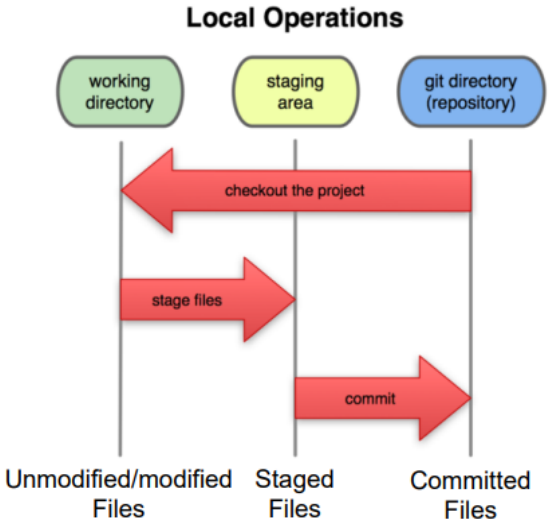
当每个用户都有自己的 repo 副本，并在推送到中央服务器之前将更改提交到他们的本地 repo 副本时，这个编号方案将如何工作?

相反，Git 为每次提交生成一个唯一的 **SHA-1哈希值** - 40个十六进制数字字符串，通过这个ID而不是版本号来引用提交，通常我们只看到前7个字符

- 1677b2d Edited first line of readme
- 258efa7 Added line to readme
- 0e52da7 Initial commit

Git 生命周期





基本的 Git 工作流

1. Modify 修改你现在工作目录的文件
2. Stage 暂存文件，将它们的快照添加到暂存区域
3. Commit 获取暂存区中的文件，并将快照永久存储到 Git 目录中

- Notes:
  - If a particular version of a file is in the **git directory**, it's considered **committed**.
  - If it's modified but has been added to the **staging area**, it is **staged**.
  - If it was **changed** since it was checked out but has not been staged, it is **modified**.

merge

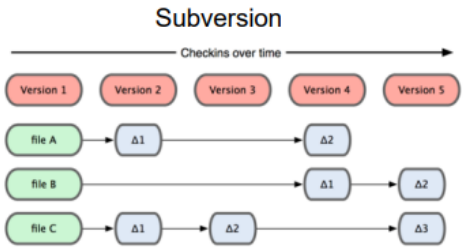
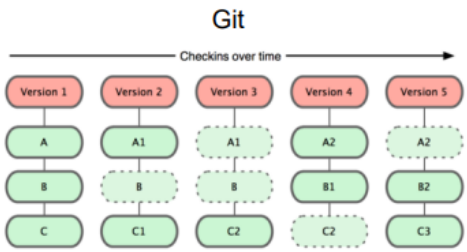
> 如果两个人同时更改相同的软件会发生什么?
>
> - Jack checks out V23, changes it, and checks in V24
> - Jill checks out V23, changes it, and checks in ???

check 软件的第二个人必须 merge 变更，自动合并在大多数情况下都是可行的（但是可能会悄无声息地引入错误）

merge 可能不好，因为它有时会产生错误

- 两个人更改了同一行，系统将强制 merge 更改的一方手动执行 merge
- 更改看起来并不冲突，但是 merge 会导致错误

SVN 和 Git 的区别

Subversion	Git
	
中央 repo 的方法，是唯一“真正”的源代码，	分布式存储库方法 每次签出的 repo 都是一个成熟的 repo ，具有完整的历史记录
用户使用当前版本的 local copy	更大的冗余和速度，分支和合并存储库被大量使用

Git 和 GitHub 的区别

GitHub.com 是一个在线存储 Git 仓库的网站

许多开源项目都使用它，比如 Linux 内核

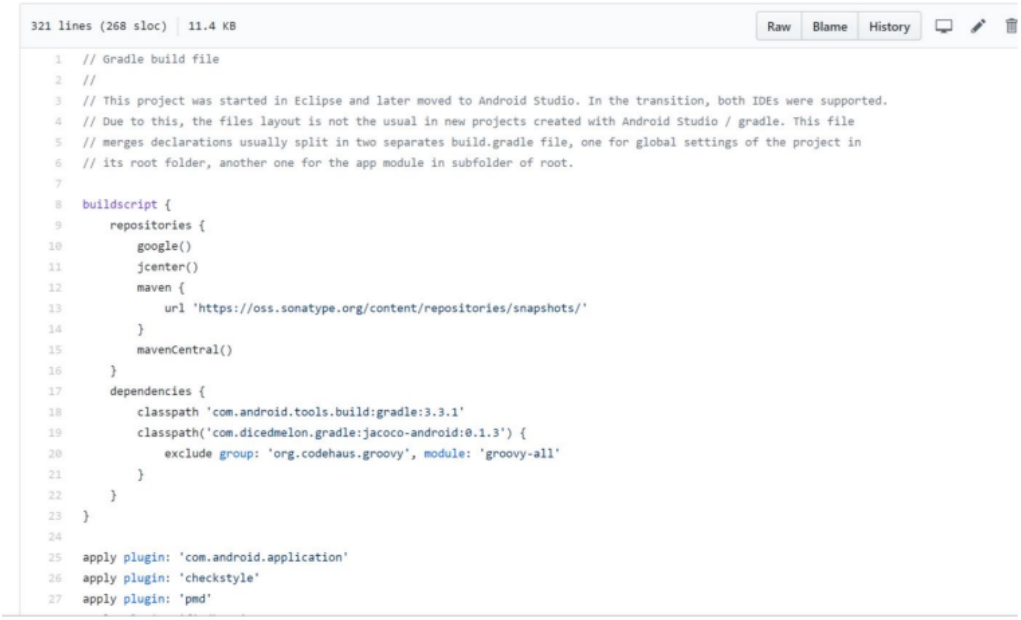
您可以为开源项目获得免费空间，也可以为私人项目支付费用

并不是一定要用 GitHub 才能使用 Git

- 你可以根据自己的需要在本地完全使用 Git
- 你或其他人可以设置服务器来共享文件
- 你可以与同一文件系统上的用户共享 repo

3. 搭建 Build management

Build 介绍


321 lines (268 sloc) | 11.4 KB

Raw Blame History

```
// Gradle build file
//
// This project was started in Eclipse and later moved to Android Studio. In the transition, both IDEs were supported.
// Due to this, the files layout is not the usual in new projects created with Android Studio / gradle. This file
// merges declarations usually split in two separates build.gradle file, one for global settings of the project in
// its root folder, another one for the app module in subfolder of root.

buildscript {
    repositories {
        google()
        jcenter()
        maven {
            url 'https://oss.sonatype.org/content/repositories/snapshots/'
        }
        mavenCentral()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.3.1'
        classpath('com.dicedmelon.gradle:jacoco-android:0.1.3') {
            exclude group: 'org.codehaus.groovy', module: 'groovy-all'
        }
    }
}

apply plugin: 'com.android.application'
apply plugin: 'checkstyle'
apply plugin: 'pmd'
```

当你要构建你的项目时

- 选择哪个编译器?
- 选择哪个源文件?
- 链接哪个库?
- 哪些版本?

构建应该是自动的，需要一些工具

Build 工具

- 知道系统的组成
- 如何编译
- 如何导出最终可执行文件
- 如何 debug
- 如何删除暂存文件
- 如何测试
- 如何制作工具手册

注意事项

破坏 Build 过程的方法

- 写入不好的代码
- 忘记在 makefile 里面记录一些引用的文件
- 移动了库

每天 Build 一次产品的最新版本，并运行简单的测试套件

软件产品 Software Product

产品 = 一系列的组成/文档

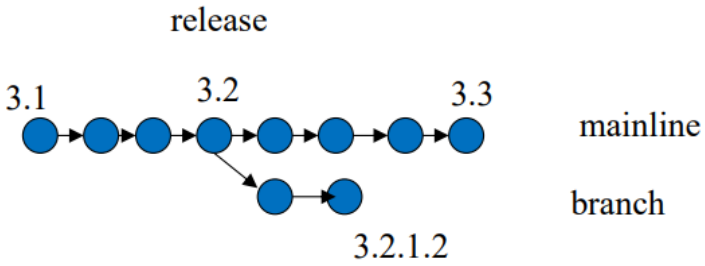
- 代码
- 测试集
- 操作手册（管理员，终端用户）
- 需求
- 说明书
- 设计文档
- 计划/日程安排

需要记录你是如何创建一个产品的

- 构建可执行文件的规则
- 代码的版本
- 库的版本
- 编译器
- 操作系统

配置管理工具 SCM 应该能够跟踪所有这些（以及更多）

版本 Version



开发过程中的版本序列

- 原型 Prototype
- 日常工作
- Alpha/Beta 版本
- 最终版本

产品也有很多不同的版本

分支 Branch

对于版本控制的传统建议

- 避免长时间存在的分支

现在的版本控制（如 Git）

- 鼓励使用（短时间存在的）分支

使用分支的好的情况

- 修复客户版本中的 bug
- 实验性的版本
- 政治问题

使用分支坏的情况

- 支持不同的硬件平台
    - 创建子类/使用条件编译 conditional compilation/创建可移植性库
- 支持不同的客户客户端
    - 将未更改的代码和更改的代码分开
    - 未修改的代码放在库中
    - 更改后的代码创建子类/使用条件编译

4. 变更管理 Change Control

变更 Change

是否所有的东西都变更了

- 测试
- 代码
- 手册
- 说明文档

需要确保变更控制和版本控制保持一致

变更管理 Change Control

有哪些是变更需求

- 新的特性
- Bug

变更控制权限——决定应该执行哪些变更

应该将代码更改与更改的请求相连接