A Proposal for the Usage of Git Version Control in Our Studio **1** (1): 1–4. Published in China.

一份有关使用 *Git* 版本管理的提案

KnowsCount Chen

knowscount@gmail.com

github.com/knowscount

(*Received 2021 年 4 月 18 日*)

Abstract

本文将讨论为何要在编年史制作组（Chronicles Studio，下文简称制作组）中使用版本控制系统；并更进一步地讨论为何要使用 git 作为版本控制系统，最终给出实现版本控制的有效解决方案。

1 什么是版本控制 | What is Version Control

版本控制是一种记录一个或若干文件内容变化，以便将来查阅特定版本修订情况的系统。在本文将讨论的例子中，我们对保存着软件源代码的文件作版本控制，但实际上，你可以对任何类型的文件进行版本控制。

如果你是位图形或网页设计师，可能会需要保存某一幅图片或页面布局文件的所有修订版本(这或许是你非常渴望拥有的功能),采用版本控制系统(VCS)是个明智的选择。有了它你就可以将选定的文件回溯到之前的状态，甚至将整个项目都回退到过去某个时间点的状态，你可以比较文件的变化细节，查出最后是谁修改了哪个地方，从而找出导致怪异问题出现的原因，又是谁在何时报告了某个功能缺陷等等。使用版本控制系统通常还意味着，就算你乱来一气把整个项目中的文件改的改删的删，你也照样可以轻松恢复到原先的样子。但额外增加的工作量却微乎其微。

1.1 为什么重要? | Why Important?

如果说什么是软件开发项目一定要使用的基础工具，那么版本控制系统应该算最重要的部分。不管是个人开发或是团队协作开发，都可以通过版本控制系统获得巨大的好处。

没有版本控制系统的话，代码可能被别人或自己不小心覆盖或遗失、也不

知道是谁因为什么原因改了这段代码、也没办法可以复原回前几天的修改。有了版本控制系统，开发人员只要将每次程式码的变更都纪录（Commit）起来，并且透过版本控制系统中进行更新。

有了版本控制系统，我们可以浏览所有开发的历史纪录，掌握团队的开发进度，而且作任何修改都不再害怕，因为你可以轻易的复原回之前正常的版本。我们也可以透过分支和标签的功能来进行软件发行的不同版本，例如稳定版本、维护版本和开发中版本。

2 不同种类的版本控制系统 | Different Sort of VCS

2.1 本地版本控制系统 / *Local VCS*

许多人习惯用复制整个项目目录的方式来保存不同的版本，或许还会改名加上备份时间以示区别。这么做唯一的好处就是简单，但是特别容易犯错。有时候会混淆所在的工作目录，一不小心会写错文件或者覆盖意想外的文件。

为了解决这个问题，人们很久以前就开发了许多种本地版本控制系统，大多都是采用某种简单的数据库来记录文件的历次更新差异。

其中最流行的一种叫做 RCS，现今许多计算机系统上都还看得到它的踪影。RCS 的工作原理是在硬盘上保存补丁集（补丁是指文件修订前后的变化）；通过应用所有的补丁，可以重新计算出各个版本的文件内容。

2.2 集中化的版本控制系统 / *Centralised VCS*

接下来人们又遇到一个问题，如何让在不同系统上的开发者协同工作？于是，集中化的版本控制系统（Centralized Version Control Systems，简称 CVCS）应运而生。这类系统，诸如 CVS、Subversion 以及 Perforce 等，都有一个单一的集中管理的服务器，保存所有文件的修订版本，而协同工作的人们都通过客户端连到这台服务器，取出最新的文件或者提交更新。多年以来，这已成为版本控制系统的标准做法。

这种做法带来了许多好处，特别是相较于老式的本地 VCS 来说。现在，每个人都可以在一定程度上看到项目中的其他人正在做些什么。而管理员也可以轻松掌控每个开发者的权限，并且管理一个 CVCS 要远比在各个客户端上维护本地数据库来得轻松容易。

事分两面，有好有坏。这么做最显而易见的缺点是中央服务器的单点故障。如果宕机一小时，那么在这一小时内，谁都无法提交更新，也就无法协同工作。如果中心数据库所在的磁盘发生损坏，又没有做恰当备份，毫无疑问你将丢失所有数据——包括项目的整个变更历史，只剩下人们在各自机器上保留的单独

快照。本地版本控制系统也存在类似问题，只要整个项目的历史记录被保存在单一位置，就有丢失所有历史更新记录的风险。

2.3 分布式版本控制系统 / Distributed VCS

于是分布式版本控制系统（Distributed Version Control System，简称 DVCS）面世了。在这类系统中，像 Git、Mercurial、Bazaar 以及 Darcs 等，客户端并不只提取最新版本的文件快照，而是把代码仓库完整地镜像下来，包括完整的历史记录。这么一来，任何一处协同工作用的服务器发生故障，事后都可以用任何一个镜像出来的本地仓库恢复。因为每一次的克隆操作，实际上都是一次对代码仓库的完整备份。

更进一步，许多这类系统都可以指定和若干不同的远端代码仓库进行交互。籍此，你就可以在同一个项目中，分别和不同工作小组的人相互协作。你可以根据需要设定不同的协作流程，比如层次模型式的工作流，而这在以前的集中式系统中是无法实现的。

3 什么是 git | What is git

Git 和其它版本控制系统（包括 Subversion 和近似工具）的主要差别在于 Git 对待数据的方式。从概念上来说，其它大部分系统以文件变更列表的方式存储信息，这类系统（CVS、Subversion、Perforce、Bazaar 等等）将它们存储的信息看作是一组基本文件和每个文件随时间逐步累积的差异（它们通常称作基于差异（delta-based）的版本控制）。

Git 不按照以上方式对待或保存数据。反之，Git 更像是把数据看作是对小型文件系统的一系列快照。在 Git 中，每当你提交更新或保存项目状态时，它基本上就会对当时的全部文件创建一个快照并保存这个快照的索引。为了效率，如果文件没有修改，Git 不再重新存储该文件，而是只保留一个链接指向之前存储的文件。Git 对待数据更像是一个快照流。

基本的 Git 工作流程如下：

1. 在工作区中修改文件。

2. 将你想要下次提交的更改选择性地暂存，这样只会将更改的部分添加到暂存区。

3. 提交更新，找到暂存区的文件，将快照永久性存储到 Git 目录。

3.1 为何使用 git / Why git

见 2.3；同时，git 是分布式 VCS 的统领者。

4 平台选择及解决方案 | Choice of Platform and Solutions

4.1 GitHub

GitHub 是最大的 Git 版本库托管商，是成千上万的开发者和项目能够合作进行的中心。大部分 Git 版本库都托管在 GitHub，很多开源项目使用 GitHub 实现 Git 托管、问题追踪、代码审查以及其它事情。所以，尽管这不是 Git 开源项目的直接部分，但如果想要专业地使用 Git，你将不可避免地与 GitHub 打交道。

4.2 GitLab

Git 大家族的 GitLab ，它提供了完善的用户权限管理，除了涵盖 Git 所有的功能，同时又提供方便的后台管理。分别有 CE（社区版）、EE（企业版）、OM（RPM 包完整版）三个版本，目前市面上开发者们的普遍选择是使用 CE 版的源码，成本较低且相对方便。

很多公司考虑到安全问题，核心代码库不允许走公网，于是会搭建 GitLab 自用；但在开源方面，GitHub 仍是唯一正确选择。

在中国的特殊环境下，自建 GitLab 有一个不可避免的优势：速度。