什么是版本控制系统（VCS）

很多人认为 Git 难以理解的第一个门槛在于：所谓的「Git 是一个分布式版本控制系统」这句话的具体含义不够清楚。其实分布式版本控制系统（Distributed Version Control System – DVCS）这个定义并不难，不过一步一步来，我先告诉你，什么是版本控制系统（Version Control System – VCS）。

版本控制：最基本功能

> 版本控制系统（VCS）最基本的功能是版本控制。所谓版本控制，意思就是在文件的修改历程中保留修改历史，让你可以方便地撤销之前对文件的修改操作。

最简化的版本控制模型，是大多数主流文本编辑器都有的「撤销（Undo）」功能：你本来想删除一个字符，却在按删除键之前不小心选中了全文，结果一下子整篇文档都被删光了，没关系，按一下「撤销」(Ctrl + Z 或 ⌘ + Z 或 U 等等，具体和你的操作系统以及编辑器有关），删掉的文字就都回来了。这其实是文本编辑器帮你自动保存了之前的内容，当你按下「撤销」的时候，它就帮你把内容回退到上一个状态；同理，按一次是会退到上一个版本，按两次就是回退到上上一个版本。

写程序的时候同样也难免会遇到「写错」的情况，所以程序的 VCS，当然也会需要版本控制功能，这样当你发现「昨天有一行代码写错了」，你就不用凭着记忆把那段代码背出来，而只需要在 VCS 中选择撤回到昨天的那个版本。

主动提交：程序代码和普通文本的区别

VCS 和文本编辑器的撤销功能比起来，有一个很重要的区别是：程序代码的修改的生命周期非常长。一次代码的修改，在几天后、几个月后、几年后都有可能需要被翻出来。如果依然采用「每次改动自动保存」的形式来保留修改历史，将会导致改动历史非常频繁和无章可循，这样，历史代码的查找、阅读和回退就会很困难了。所以，和文本编辑器的撤销功能不同，VCS 保存修改历史，使用的是**主动提交改动**的机制。

在你写了一段完整的代码（例如修复了一个 bug）之后，使用 commit 命令把改动和对改动的描述信息提交，这次改动就被记录到版本历史中了。之后如果你希望回退到这个版本，就可以从 VCS 的历史日志中方便地找到它。

多人合作的同步需求：中央仓库

代码可以一个人写，但更多的时候会是多个人共同开发。那么自然地，就需要有一个中央仓库作为代码的存储中心：所有人的改动都会上传到这里，所有人都能也都能看到和下载到别人上传的改动。

这样，解决了同步的需求，多个人在不同的机器上开发同一个程序就成了可能。

版本控制、**主动提交**、**中央仓库**这三个要素，共同构成了版本控制系统（VCS）的核心：开发团队中的每个人向中央仓库主动提交自己的改动和同步别人的改动，并在需要的时候查看和操作历史版本，这就是版本控制系统。

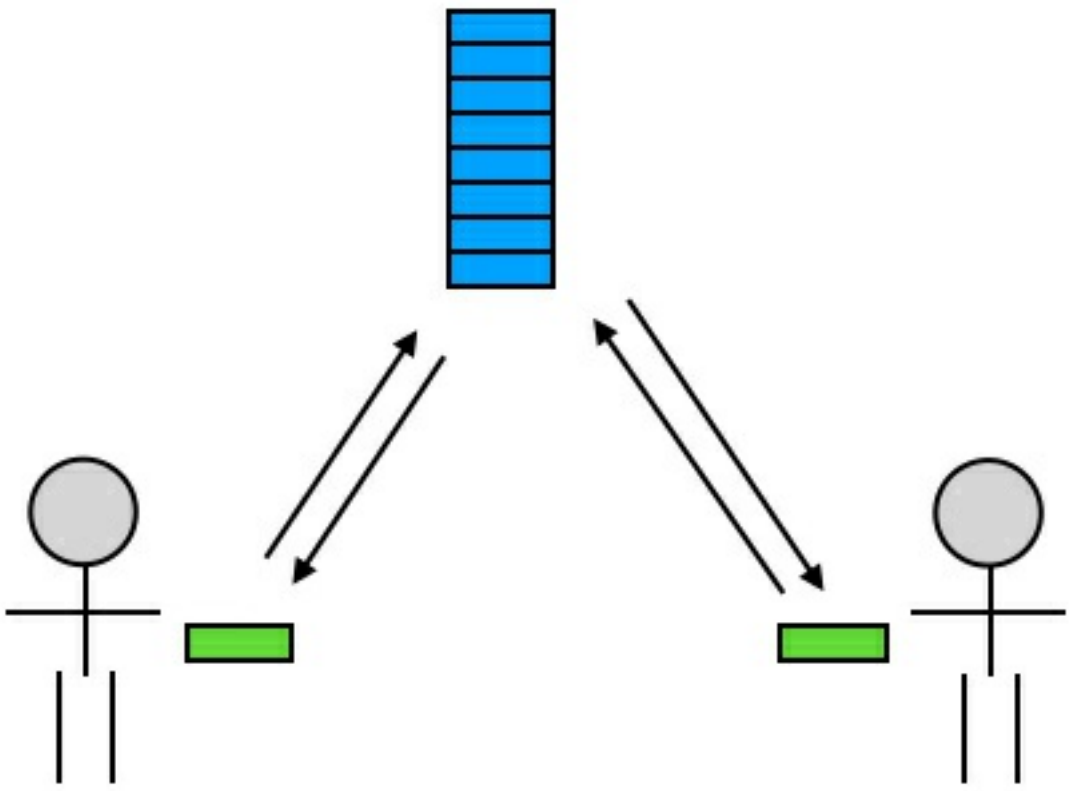
中央式版本控制系统

最初的版本控制系统，是中央式版本控制系统（Centralized VCS），也就是前面我讲的这种。Git 是分布式的版本控制系统（Distributed VCS），它和中央式的区别我在下节说，现在先说一下中央式版本控制系统的工作模型。

工作模型

假设你在一个三人团队，你们计划开发一个软件或者系统，并决定使用中央式 VCS 来管理代码。于是：

1. 作为项目的主工程师，你独自一人花两天时间搭建了项目的框架；
2. 然后，你在公司的服务器（这个服务器可以是公司内的设备，也可以是你们买的云服务）上**创建了一个中央仓库，并把你的代码提交到了中央仓库上**；
3. 你的两个队友**从中央仓库取到了你的初始代码**，从此刻开始，你们三人开始**并行开发**；
4. 在之后的开发过程中，你们三人为了工作方便，总是每人独立负责开发一个功能，在这个功能开发完成后，这个人就把他的这些**新代码提交到中央仓库**；
5. 每次当有人把代码提交到中央仓库的时候，另外两个人就可以选择**把这些代码同步到自己的机器上**，保持自己的本地代码总是最新的。



而对于团队中的每个人来说，就会更简单一点：

1. 第一次加入团队时，把中央仓库的代码取下来；
2. 写完的新功能提交到中央仓库；
3. 同事提交到中央仓库的新代码，及时同步下来。

这样，一个三人的团队就成功做到了各自在自己的电脑上开发同一个项目，并且互不影响，就好像你们三个人是在同一台电脑上操作一样。

这就是中央式 VCS 最基本的工作模型。当然，实际的开发工作并没有简单到这种程度，因为你时常会需要处理代码冲突、查看版本历史、回退代码版本等；另外，Git 属于分布式 VCS，它的概念也比中央式 VCS 要复杂一些。但这些概念你需要一步步地理解和吸收，你现在只需要先知道中央式 VCS 的这个基本工作模型，其他的内容我会在后面慢慢地全部讲清楚。