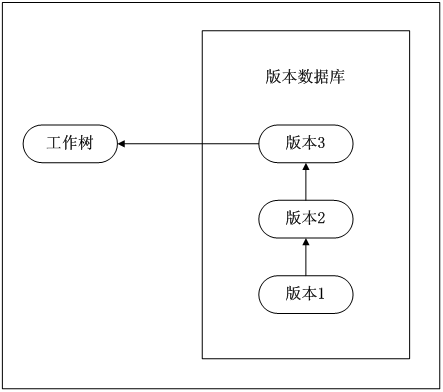
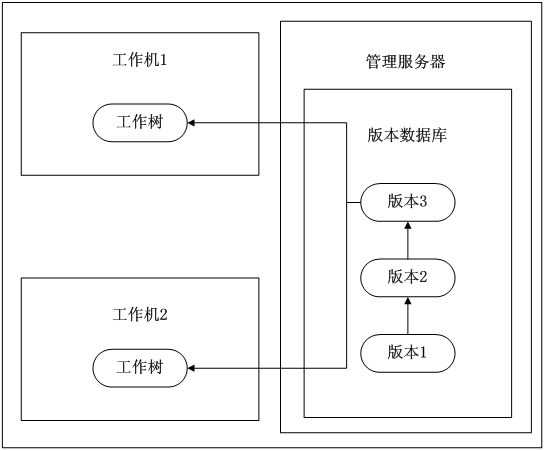
**一、本地版本控制系统**

很久以前人们就开始考虑版本控制的问题，因为简单的通过复制整个项目目录的方式来保存不同的版本虽然操作简单，但是缺点显而易见。为解决此类问题，人们开发出本地版本控制系统，大多是采用简单的数据库方式来记录文件的历史更新差异，如图：

**二、集中化的版本控制系统**

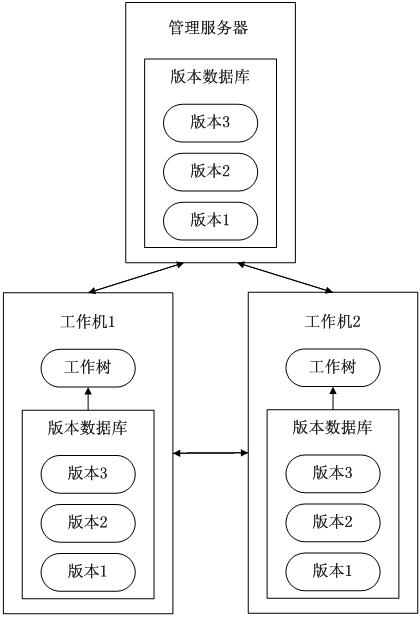
很快人们遇到一个新的问题，即如何让不同系统下的开发者协同工作？于是，集中化的版本控制系统（ Centralized Version Control Systems，简称CVCS ）应运而生，诸如CVS、SVN等，它们的共同点是都有一个单一的管理服务器，保存整个项目的文件历史，而协同工作的开发者通过客户端连接到服务器，取出 最新的文件或者提交自己的更新。



这么做最显而易见的缺点是中央服务器的单点故障。若是宕机一小时，那么在这一小时内，谁都无法提交更新，也就无法协同工作。

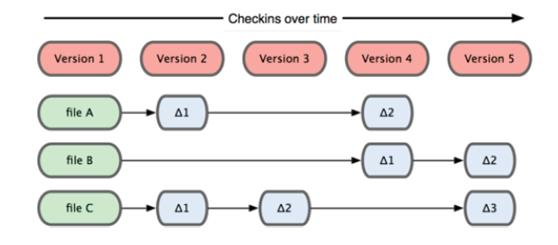
**三、分布式版本控制系统**

分布式版本控制系统（ Distributed Version Control System，简称DVCS ）。在这类系统中，诸如Git，Mercurial，Bazaar 还有Darcs 等，客户端并不只提取最新版本的文件快照，而是把原始的代码仓库完整地镜像下来。这么一来，任何一处协同工作的历史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。因为每一次的提取操作，实际上都是一次对代码仓库的完整备份。

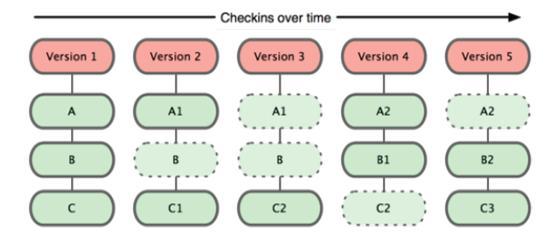
**四、文件差异版本控制**

CVS、SVN等系统进行版本控制的原理为每次都记录有哪些文件作了更新，其控制原理如图所示：

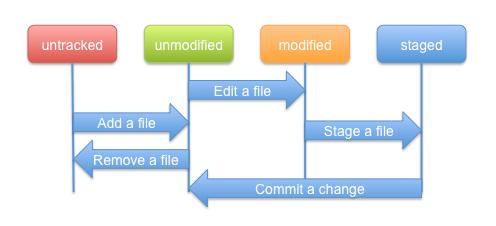
**五、直接快照版本控制**

Git并不保存这些前后变化的差异数据。实际上，Git更像是把变化的文件作快照后，记录在一个微型的文件系统中。每次提交更新时，它会纵览一遍所 有文件的指纹信息并对文件作一快照，然后保存一个指向这次快照的索引。为提高性能，若文件没有变化，Git不会再次保存，而只对上次保存的快照作一连接。

**六、四个状态和三个区域**

Git内部只有三个状态，分别是未修改unmodified、修改modified、暂存staged。对于没有加入Git控制的文件，可以视为第四种状态未跟踪untracked。



       Git文件流转有三个区域，分别是工作区域、索引区域、本地数据区域。工作树中的文件添加到git版本控制索引中，则git开始对文件进行跟踪监控。索引区域也可以理解为数据暂存区域，当提交操作时，暂存区域的数据被记录到本地数据仓库中。

