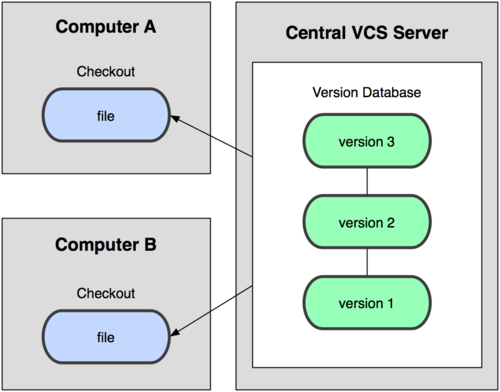
Git Share

包括核心原理、命令行使用、结合eclipse的使用。使用主要就是常规的pull push merge rebase等。

集中版本控制SVN



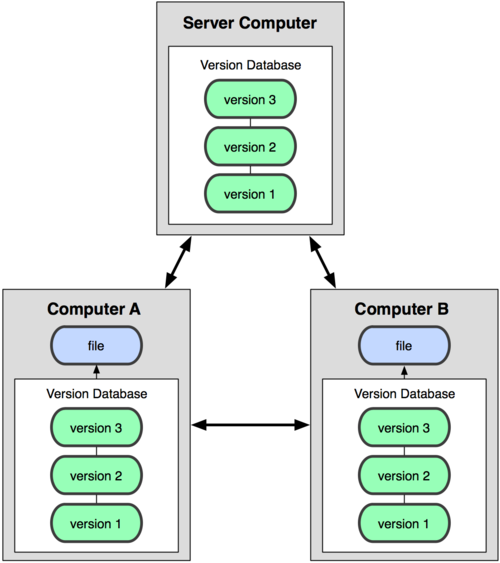
所有的版本数据都保存在服务器上，一起工作的人从服务器上同步更新或上传自己的修改。

但是，所有的版本数据都存在服务器上，用户的本地设备就只有自己以前所同步的版本，如果不连网的话，用户就看不到历史版本，也无法切换版本验证问题，或在不同分支工作。

而且，所有数据都保存在单一的服务器上，有很大的风险这个服务器会损坏，这样就会丢失所有的数据，当然可以定期备份。

分布式版本控制Git

不是复制指定版本的快照，而是把所有的版本信息仓库全部同步到本地，这样就可以在本地查看所有版本历史，可以离线在本地提交，只需在连网时push到相应的服务器或其他用户那里。由于每个用户那里保存的都是所有的版本数据，所以，只要有一个用户的设备没有问题就可以恢复所有的数据。



缺点：当然，这增加了本地存储空间的占用。

优点：

* 速度
* 简单的设计
* 对非线性开发模式的强力支持（允许上千个并行开发的分支）
* 完全分布式
* 有能力高效管理类似 Linux 内核一样的超大规模项目

[Git的思想和基本工作原理](http://www.nowamagic.net/academy/detail/48160210)

**直接记录快照，而非差异比较**

Git 和其他版本控制系统的主要差别在于，Git 只关心文件数据的整体是否发生变化，而大多数其他系统则只关心文件内容的具体差异。这类系统（CVS，Subversion，Perforce，Bazaar 等等）每次记录有哪些文件作了更新，以及都更新了哪些行的什么内容：



其他系统在每个版本中记录着各个文件的具体差异

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链接。Git 的工作方式就如下图所示。



Git 保存每次更新时的文件快照

这是 Git 同其他系统的重要区别。它完全颠覆了传统版本控制的套路，并对各个环节的实现方式作了新的设计。Git 更像是个小型的文件系统，但它同时还提供了许多以此为基础的超强工具，而不只是一个简单的 VCS。稍后会讨论 Git 分支管理的时候，我们会再看看这样的设计究竟会带来哪些好处。

**近乎所有操作都是本地执行**

在 Git 中的绝大多数操作都只需要访问本地文件和资源，不用连网。但如果用 CVCS 的话，差不多所有操作都需要连接网络。因为 Git 在本地磁盘上就保存着所有当前项目的历史更新，所以处理起来速度飞快。

举个例子，如果要浏览项目的历史更新摘要，Git 不用跑到外面的服务器上去取数据回来，而直接从本地数据库读取后展示给你看。所以任何时候你都可以马上翻阅，无需等待。如果想要看当前版本的文件和一个月前的版本之间有何差异，Git 会取出一个月前的快照和当前文件作一次差异运算，而不用请求远程服务器来做这件事，或是把老版本的文件拉到本地来作比较。

用 CVCS 的话，没有网络或者断开 VPN 你就无法做任何事情。但用 Git 的话，就算你在飞机或者火车上，都可以非常愉快地频繁提交更新，等到了有网络的时候再上传到远程仓库。同样，在回家的路上，不用连接 VPN 你也可以继续工作。换作其他版本控制系统，这么做几乎不可能，抑或非常麻烦。比如 Perforce，如果不连到服务器，几乎什么都做不了（默认无法发出命令 p4 edit file 开始编辑文件，因为 Perforce 需要联网通知系统声明该文件正在被谁修订。但实际上手工修改文件权限可以绕过这个限制，只是完成后还是无法提交更新。）；如果是 Subversion 或 CVS，虽然可以编辑文件，但无法提交更新，因为数据库在网络上。看上去好像这些都不是什么大问题，但实际体验过之后，你就会惊喜地发现，这其实是会带来很大不同的。

**时刻保持数据完整性**

在保存到 Git 之前，所有数据都要进行内容的校验和（checksum）计算，并将此结果作为数据的唯一标识和索引。换句话说，不可能在你修改了文件或目录之后，Git 一无所知。这项特性作为 Git 的设计哲学，建在整体架构的最底层。所以如果文件在传输时变得不完整，或者磁盘损坏导致文件数据缺失，Git 都能立即察觉。

Git 使用 SHA-1 算法计算数据的校验和，通过对文件的内容或目录的结构计算出一个 SHA-1 哈希值，作为指纹字符串。该字串由 40 个十六进制字符（0-9 及 a-f）组成，看起来就像是：

|  |  |
| --- | --- |
| 1 | 24b9da6552252987aa493b52f8696cd6d3b00373 |

Git 的工作完全依赖于这类指纹字串，所以你会经常看到这样的哈希值。实际上，所有保存在 Git 数据库中的东西都是用此哈希值来作索引的，而不是靠文件名。

**文件的三种状态**

好，现在请注意，接下来要讲的概念非常重要。对于任何一个文件，在 Git 内都只有三种状态：已提交（committed），已修改（modified）和已暂存（staged）。已提交表示该文件已经被安全地保存在本地数据库中了；已修改表示修改了某个文件，但还没有提交保存；已暂存表示把已修改的文件放在下次提交时要保存的清单中。

由此我们看到 Git 管理项目时，文件流转的三个工作区域：Git 的工作目录，暂存区域，以及本地仓库。



工作目录，暂存区域，以及本地仓库

每个项目都有一个 Git 目录（如果 git clone 出来的话，就是其中 .git 的目录；如果 git clone --bare 的话，新建的目录本身就是 Git 目录。），它是 Git 用来保存元数据和对象数据库的地方。该目录非常重要，每次克隆镜像仓库的时候，实际拷贝的就是这个目录里面的数据。

从项目中取出某个版本的所有文件和目录，用以开始后续工作的叫做[工作目录](http://www.nowamagic.net/academy/tag/%E5%B7%A5%E4%BD%9C%E7%9B%AE%E5%BD%95" \t "_blank)。这些文件实际上都是从 Git 目录中的压缩对象数据库中提取出来的，接下来就可以在工作目录中对这些文件进行编辑。

所谓的暂存区域只不过是个简单的文件，一般都放在 Git 目录中。有时候人们会把这个文件叫做索引文件，不过标准说法还是叫暂存区域。

基本的 Git 工作流程如下：

1. 在工作目录中修改某些文件。
2. 对修改后的文件进行快照，然后保存到暂存区域。
3. 提交更新，将保存在暂存区域的文件快照永久转储到 Git 目录中。

所以，我们可以从文件所处的位置来判断状态：如果是 Git 目录中保存着的特定版本文件，就属于已提交状态；如果作了修改并已放入暂存区域，就属于已暂存状态；如果自上次取出后，作了修改但还没有放到暂存区域，就是已修改状态。

git clone

git add

git commit –m

git commit –a

git branch

git branch –d

git branch –D

git checkout

git checkout –b

git reset

git revert

git merge

git rebase

git push

git fetch

git pull

git pull –rebase

基础命令

git init

git log

git diff

git status

git stash

git stash list

git stash apply

git stash drop

git stash pop

Git Merge和Git Rebase

user1



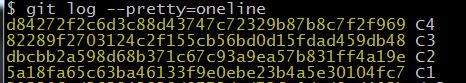
user2



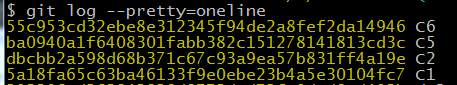
master



user1



user2



master

