Git 和 SVN的主要差别：

* **Git 只关心文件数据的整体是否发生变化，而SVN这类版本控制系统则只关心文件内容的具体差异**。这类系统（如SVN）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，然而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。

          -----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**个人理解**：Git 并没有像 SVN一样在每一次提交的时候都将更新的详细信息都分析并提交，这是很科学的，因为我们有90%的时候不会闲着要查看每一个文件都做了哪些更新，特别是模块化开发时，那些不是我们开发的模块，因为Git 会在本地保存完整的历史版本库，有十个用户共同开发就会有十份这样的库，这样做的话会会出现非常多冗余的数据。但是在SVN 上这样还是科学的，因为只有SVN 服务器上保存着唯一的一份历史版本库，当然是越详细越好了。但是当遇到那10%的时候，我们想要查看一下某个文件的与昨天的版本相比的更新变动信息，**Git 会取出一天前的快照和当前文件作一次差异运算，显示出更新变动信息。**

          ------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

* **在Git 中的绝大多数操作都只需要访问本地文件和资源，不必联网就可以看到所有的历史版本记录，而SVN 却需要联网。**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，但我们需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**。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。**
* **SVN 断开网络或者断开VPN就无法commit代码，但是Git 可以先commit到本地仓库。**用SVN的话，没有网络或者断开VPN时，你当然也可以继续在本地开发，但是无法commit代码，因为SVN 每次commit都必须联网，长时间不commit代码会丢失大量开发进程的历史纪录。有个比喻是：不能commit就像用word写文档不能save一样危险。而且有网络的情况下每一次commit都会花上数秒甚至更长时间。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，因为是在本地仓库commit所以几乎不需要时间，而且**commit一定要频繁，不然无法记录你的改动，**如果你一天commit一次，中间的修改你就找不回来，然后等到了有网络的时候再**将版本纪录和代码一起上传到远程仓库。**
* **Git 的内容完整性要优于SVN。**因为Git 在commit（存储在本地）或者push（上传到远程仓库）之前，通过对**文件的内容**或**目录的结构**计算出一个 SHA-1哈希值，作为指纹字符串进行内容的校验，并将此结果作为数据的唯一标识和索引，在远处仓库接受到commit的文件之后，会再计算一遍哈希值然后跟传递过来的哈希值做比较，如果不一致，说明**文件在传输时变得不完整**，或者**磁盘损坏导致文件数据损坏。另外在 Git 数据库中的东西都是用此哈希值来作索引，而不是靠文件名。**
* **Git 克隆一个完整项目的速度非常快，SVN 非常慢。**我们以克隆一份拥有五个分支的完整项目以及版本库来说，SVN是同时复制5个版本的文件,也就是说重复五次同样的动作。而Git 只是获取文件的每个版本的元素，然后只载入主要的分支(master)在我的经验，克隆一个拥有将近一万个提交(commit)，五个分支，每个分支有大约1500个文件的 SVN，耗了将近一个小时！而Git只用了区区的1分钟。
* **其中最重要的区别是在于Git 上的分支远比SVN上的强大。下面具体介绍分支的概念。**

分支是什么：

* 在 **SVN** 这类的版本控制系统上，**分支（branch）是一个完整的目录，且这个目录拥有完整的实际文件**。如果工作成员想要开启新的分支，那将会影响“全世界”！每个人都会拥有和你一样的分支。如果你的分支是用来对系统模块进行**安全检查测试**的，那将会像传染病一样，你改一个分支，还得让其他人重新切分支重新下载，而且这些代码很可能对稳定版本还是具有破坏性的。
* 在 **Git**上，每个工作成员可以任意在自己的**本地版本库**开启无限个分支。举例：当我想尝试破坏自己的程序（安检测试），并且想保留这些被修改的文件供日后使用， 我可以开一个分支，做我喜欢的事。完全不需担心妨碍其他工作成员。**只要我不合并及提交到主要版本库，没有一个工作成员会被影响**。等到我不需要这个分支时， 我只要把它从我的本地版本库删除即可，无痛无痒。

项目开发中什么时候需要创建一个分支：

     举个例子：我们需要开发一个新的网站，我们已经在主分支（master分支）上开发出了1.0发布版本，这个时候我们需要开发某个新的功能模块，那就需要创建一个分支（dev分支），而不是在主分支上继续开发，这样做有两个好处：

* 我们在开发新的功能模块时，可能会遇到各种bug或者冲突，如果我们还在**主分支**上开发，万一冲突很严重，造成当前稳定版本的分支出问题，就会很麻烦。如果主分支始终保留着最新的稳定版本，在新的分支上开发，冲突严重时，最多也就是把当前分支删掉，从那个稳定分支重新分一支出来，这样处理起来就方便了，**而且分支还可以保留开发中可能出现的各种bug方便修复但不影响主分支多的使用。**
* 当我们需要切换分支，例如切换到主分支（master）时候，会保存当前分支（dev）的状态，以便日后继续开发，防止丢失开发进度。举个例子：你突然接到一个电话说1.0发布版本有个很严重的问题需要紧急修补，而我们正在**dev分支**上开发新的功能模块，这时我们先返回到主分支，为这次紧急修补建立一个新分支（**repair分支**），并在其中修复问题。通过测试后，回到**主分支**，将**repair分支合并进来**，然后push到远程仓库。最后，我们切换到之前开发新需求的**dev分支**，继续工作而不会丢失掉已经开发的进度。

     我可以在Git的任意一个提交点（commit point）开启分支！（其中一个方法是使用gitk –all 可观察整个提交记录，然后在任意点开啟分支。）

Git具有以下特点：

* Git 中每个克隆(clone)的版本库都是平等的。可以从任何一个版本库的克隆来创建属于自己的版本库，同时你的版本库也可以作为源提供给他人，只要你愿意。
* Git 的每一次提取操作，实际上都是一次**对代码仓库的完整备份**。
* 提交完全在本地完成，无须别人给你授权，你的版本库你作主，并且提交总是会成功。
* Git 的提交不会被打断，直到你的工作完全满意了，PUSH给他人或者他人PULL你的版本库，**合并会发生在PULL和PUSH过程中**，不能自动解决的冲突会提示你手工完成。

git 的缺点：

* Git 没有严格的权限管理控制，一般通过系统设置文件读写权限的方式来做权限控制。
* 工作目录只能是整个项目。比如 checkout，建分支，都是基于整个项目的。而 svn 可以基于项目中的某一个目录

1. Import URL
2. SVN checkout
3. 增删改->svn commit
4. Commit之前先update一下以免别人的commit没有收到
5. 通过SVN revert进行还原