Git 使用

Git 是一个分布式的版本控制系统。分布顾名思义就是不存在一个中心，这也是 Git 和集中式的系统最大的区别。

以前在 Windows 上配置 Git 需要先有一套仿 Linux 环境，很复杂就不详细说明了，有过不少一键安装包也不太好用。后来 GitHub 出了 Windows 客户端就方便多了，小项目直接使用漂亮的图形界面，大项目实在卡的不行也能换回 PowerShell 命令行。至于 Linux 直接从官方源安装就好，填上用户名和邮箱直接就能用。

使用 Git 一般是从自建 repository 或远程 clone 一个开始，自此我们就获得了一个本地分支，这个分支就独立下来不会再受任何外部的影响，除非手动 merge 其他分支。之后我们就可以对工作区内的代码任意进行修改，不需要再进行旧代码注释之类繁琐的过程，Git 已经将所有 commit 过的阶段都保存了下来，需要撤销修改时只需要 reset –hard 即可。当修改到一定阶段时就可以进行 commit 了，用 add 可以将新文件纳入管理，rm 移除，可以使用-a 提交所有管理中文件变化也可选择文件单独提交。上述的提交不同于集中式系统，变更只是提交到了当前本地的分支，你依然可以使用 reset –soft 撤销修改。当需要真正提交到远程分支时就使用 push 命令，当然如果在之前已经有人对远程分支进行了修改，我们还需要将变化 pull 下来进行 merge，存在冲突的代码段 Git 都已经进行了标明。如果需要知道两个版本间的区别则可以使用 diff 命令，它可以对比当前项目和最后一次提交的区别，两个历史版本的区别甚至两个分支任意阶段的区别，告诉你这期间大家究竟做了什么。

截至目前，Git 已经涵盖了大部分传统版本控制工具的功能，但 Git 真正强大的地方在于它的分支控制，很多其他工具难以处理的问题 Git 都可以通过其强大的分支与合并功能予以解决。当你需要修复 Bug 或者引入新功能时，使用 checkout –b 切换到一个新建的分支，埋头写代码，像使用传统工具那样不断对 Git 新分支进行提交，完成后 checkout 回 master，

更新，merge 新分支，解决冲突，commit 结果。在此期间其他人对 master 的修改完全不会受到影响，你在中途的零碎提交也不会被其他人对 master 的修改打断。多么完美的并行化作业机制！当你需要发布一个新版本时，从 master 分离出一个 release 分支，这个分支将不再引入新功能，但是依然会修复 Bug，所有在 release 中修复的问题都可以在解决少量冲突后原封不动地 merge 进 master 里，尽管 master 已经加入了不少稀奇古怪的新功能。

那么 Git 对于 SVN 等现代集中式版本控制系统的优点又是什么呢?答案是可以离线工作，并且它比 SVN 快得多。在去年的暑期实习期间，我深刻体会到了使用 SVN 的痛苦，每天早上先得花半小时更新，下班之前又得等半小时更新再提交，如果在你的更新和提交期间有人修改过相关文件，改代码解决冲突倒是小事，头疼的是你又得花半小时再更新一次……离开了公司还需要使用 VPN 进入内网才能继续工作，一切都依赖那个小小的内部服务器。相比之下 Git 就痛快多了，没有集中式服务器的限制，任何人都可以自立门户，多个 Git 远程分支也可互相 merge 共同发展。

GitHub

说到自立门户，GitHub 倒是为广大穷人提供了巨大的帮助，只要你的仓库开放权限每个人都能查看，服务器就可以靠他们提供，大大降低了自立门户的成本，真正的社区化编程由此开始了。开源项目不再需要自付经费搭建一个服务器了，过去的那种邀请制的开源项目开发也失去了市场。想参与一个开源项目，只要将他 fork 下来，改完代码发给主持者 pull request，他觉得这个分支有用就 merge，没用就 refuse，丝毫不拖泥带水。颇有市场经济取代计划经济的感觉。

围绕着社区编程，GitHub 进一步开发了许多新功能。原来为了统计代码贡献我还专门写了 shell 脚本，现在全部在 GitHub 上都有了图形化的统计，甚至连代分支的演变过程，

不同贡献者的提交频率等等都一应俱全；冷冰冰的 TODO 列表也全都被扔在了一边，取而代之的是 Issues，大家可以一起讨论需要增加什么功能和如何实现它，像是程序员版的 Facebook。

Gitosis

记得以前某个编程作业需要团队作业，那可是大家第一次进行协作，在大部分人祭出 QQ 传文件功能时我们小组毅然决定使用 Git 进行管理，但是作业这东西实在不方便开源地放到 GitHub 上，于是我们只好自行在校园网内搭建 Git 服务器，经过精挑细选最终使用了 Gitosis。

./configure+./make insatll 没啥好说的，直接跳到服务器配置这个重头戏。Gitosis 的配置方式非常奇特，他自身的配置文件不像其他服务器软件那样放在 root 权限的/etc/下，竟然也是使用 Git 管理的……通过 git clone git@localhost:gitosis-admin.git 命令 clone 出配置 repository 进行配置。权限控制方面则是使用 ssh-key 代表用户，所以每次有新人想要加入都需要叫他先把 rsa.pub 密钥发给我塞进配置 repository 中去，然后再手动编辑配置文件赋予他对应权限。有了权限之后则是通过 git remote add origin git@*hostname*:*repositoryname* 命令在服务器中增加新的 repository。之后的所有日常操作就和第一部分介绍的一样了。发现了那个”git@”吗?之所以每个命令都要加它是因为 Gitosis 的仓库默认位于/srv/gitosis 下，这个目录只有名为 git 的用户开启了读写权限，第一次配置时被这点坑了好久，最后受不了对着目录一发 sudo chmod 777 才勉强解决。