Git基础教程

版本号	作者	联系方式	日期	详情
1.0	周航	QQ 893144519/email: 213162574@seu.edu.cn	18-9-23	git培训教程

- 0.前言
- 1. 版本控制与团队开发
- 2. Git环境布置
  - 2.1 图形界面or命令行？
  - 2.2 安装git
- 3. Git基础——版本管理，远程服务器

前言

本文档的主要目的是为了从实际使用角度来帮助理解git的概念，力求纲举目张，着重概念的阐述，具体细节不表。

个人认为学习git的方法论如下：
1. 了解git这个软件本身的设计目标。
2. 逐渐了解git的一些特性
3. 实践这些特性
4. 反思这些特性在实际开发中的用处
5. 回顾设计目标

git在实际使用的角度主要有两个方面：
1. 个人的代码版本库。
2. 团队工作流。

本文档要求和Pro Git同时进行学习。

由于github和git概念容易混淆，开始学习的时候注意辨别。
学习资料

任何时候都要记得，学习新技术时，论坛(csdn,stackoverflow等)是第二个应该想到的地方，你第一个应该去找的地方是**官网**。
git官方文档
Git官网里有一本书 Pro Git，这本书的前三章看完应付算法组这一年的开发过程足够了。如果有浓厚兴趣，可以去后几章了解更多关于git团队开发，服务器架设的内容。

在觉得官网内容难以理解的前提下，可以去论坛等地方找教程（推荐廖雪峰的博客）

最后，最高效的方法，是找已经会的同学，拿实例进行讲解。

版本控制与团队开发

在实际的软件开发中，版本控制是最基本的问题之一。
以robomaster2018的算法组开发为例：我们的总项目构架中包含了（部分）
装甲板识别（Armor子项目）
大小符（Rune子项目）
位姿解算（Pose子项目）

数字识别神经网络（Darknet子项目）
这些子项目是分别由四到五位不同的开发者负责的。

下面先看一个版本控制的实例：

实例1

负责开发装甲板识别的同学在本地建立了一个vs工程，并完成了1.0稳定版本算法的开发。然而这时他突然决定优化一部分代码，为了防止修改代码之后出了bug后无法恢复到稳定版本1.0，他决定，将自己的Armor.cpp文件复制一份存起来，然后才进行代码的修改。

一次两次使用这种方法是能凑合的，但是试想一下如果经常需要修改代码，那你的文件夹里可能就会出现大量的各种花式命名的Armor.cpp的副本，而且你需要花很大的代价才能搞清楚每一个文件对应的版本是什么。

对于软件开发来说，这种低效粗暴的方法low得不行且不说，其效率对开发流程来说也是一场灾难。

这时候，如果有一个软件能够记录你代码文件的每次修改。这样，就算你乱来一气把整个项目中的文件改的改删的删,你也照样可以轻松恢复到原先的样子。但额外增加的工作量却微乎其微。

这就是**“版本控制”**的一个基本概念。

再来看第二个例子：

实例2

第一个例子中，所有的开发都没有涉及到和其他开发者的合作。假设到了后期，算法组组长开始对所有人的代码进行整合，将其放入同一个大项目Robomaster2018中。

这时每个人虽然还是负责自己的算法开发，但是拿到手上的却是综合过所有开发者代码的大项目。

这时的代码管理就会变得更加棘手，一旦一个人修改了一行代码，组里所有其他人都得拿着U盘拷一份新代码，然后整合到自己电脑中的项目里。除此之外，每个人还要时刻面临自己的部分和别人的部分发生冲突的危险。自己想象一下，一旦动起工来，整个场面会变得多么混乱不堪。

这时候，如果有一个软件，不仅能记录每个人的修改记录，还能让每个人都能安全快速地获得最新版本的Robomaster2018，并且通过一些机制，简化并且约束每个开发者的行为，以保证软件版本的安全。

上面的描述对新手来说可能过于抽象，简而言之，这款软件能给团队提供一个安全稳定的**“工作流”**（本质上来说还是一种版本控制）

Git就是这个利器，这款软件的很多特性与功能，让个人，团队开发的版本控制实现了“现代化”。

Git环境布置

图形界面or命令行?

git为用户提供了图形界面（主要是在windows下），而一些IDE如visual studio，Qt等内部都内置了git版本控制。

我个人是倾向于在一开始学习的时候就抛弃对于图形界面的依赖，理由如下：
1. 无论是图形界面还是命令行都需要同等程度地理解git。
2. 命令行对使用者对于git的掌握程度要求相对更高
3. 在linux开发环境下，命令行还是最实际的解决方案。

前期不熟悉命令可以慢慢适应，git的使用到后期才会慢慢体现出来。

安装Git

Git的安装十分简单，请参考官网或者自行百度。

Git基础

这一部分说明了git的基本工作模式。
请务必仔细阅读Pro Git的第二章”Git基础“，写的非常清楚，此处只提取了重要的内容进行介绍。

1.仓库的概念

git的功能实现是通过”仓库“实现的。在任意路经下加入.git的隐藏文件，这个路经就会变成一个git仓库，仓库里所有的文件受git的版本监控。

git分本地仓库和远程仓库。关于仓库有如下几点
1. 首先无论是远程还是本地，所有仓库记录更改的功能都是一样的。git设置远程仓库功能的意义在于实现分布式代码管理，开始理解不了的话可以简单类比为”云端“的功能。这个”云端“能够对团队开发人员共同开放，以便所有人协同开发。
2. 远程仓库可以安放在自己的服务器上，可以放在公司的服务器上，甚至可以放在本机的另一个位置上（但是这样就失去意义了）。
3. 承接上一条，最常用的远程仓库放置的位置就是大名鼎鼎的**github**，没错，github本质上来说就是一个代码托管的服务器网站（当然其功能）。除此之外，还有别的代码托管平台如Gitee（码云，国产平台）, Gitlab等。

2.本地仓库的版本控制

git 记录的是文件的更改信息。
1. 本地文件更改信息首先需要添加(**add**)到缓存区**Stage**中，才能进行更改信息的提交(**commit**)
2. 文件的本身状态分为”**已跟踪**“和”**未跟踪**“，顾名思义，新添加或者刚删除的文件会处于未受git跟踪的状态，而已经受到git追踪的文件经过修改则会变成”**已修改**“，正常未经修改的文件就处于”**未修改**“状态。对于以上文件状态了解即可。

3.本地仓库与远程仓库的交互

远程仓库默认为github上的远程仓库。回想前面例子中我们的Robomaster项目，这个项目就是存在github上面的远程仓库中。
1. 由于远程仓库一般是多人协作，所以为了安全问题，这些仓库会对本地仓库有相应的读写，提交更改的权限。当然如果是你自己建的仓库，你自然会拥有最高的权限。
2. 最简单的情况下，你可以修改本地的仓库，然后**推送**(“**push**“)到远程仓库中。这样远程仓库中的代码就和本地保持同步了。
3. 如果另一个开发者更改了远程仓库的内容，你就可以**获取**(**“fetch”**)最新的代码。
4. 如果本地代码自从上一次提交后没有更改过，你就可以再**合并**(**“merge”**)fetch到的代码到本地仓库中，这样你的本地仓库就和远程仓库保持一定。
如果自从上次推送之后本地仓库又进行了修改，这时候git是没有办法直接merge代码的，而是需要你手动处理git无法处理的冲突（一般是根据git diff命令的提示手动修改对应文件）后再进行合并。这时候你本地的仓库中的代码就是最新的版本（综合了上次push后你的更改和别的开发者的更改），比远程仓库更新，由于原则是保持远程仓库为最新的版本，所以你需要将这些代码再次push到远程仓库。