Git-GitHub
Getting Started

Git-GitHub Manual of Vision Lab

Jinna Cui

August 2016

摘要

Git 是一种分布式版本控制系统，实验室主要用它来管理文档、代码等资料，所以学弟学妹们要学习 Git 相关知识，并熟练使用 Git 。本文参考孙雪师姐及郑老师《Git-GitHub-howto》文档，结合《git 中文教程》及网上相关教程，围绕“为什么选择 Git-GitHub”和“如何使用 Git-GitHub”两个主题，对 Git-GitHub 做了详细却不冗余的介绍。由于是新手适用的 Git 攻略，本文更侧重基础知识，欢迎对 Git 的其他应用感兴趣的同学与我讨论，也欢迎大家批评指正！

目录

第 1 章

Git & GitHub

1.1 什么是 Git

在理解 Git 之前，首先要理解版本控制系统的概念。版本控制系统是用来记录文件内容变化以便查阅版本修改情况的系统。用大家都经历过的毕业论文举例，写论文要经过很多次的修改，我们惯用的是每次修改之前将前一版本复制到本地，备份好之后，再修改文件名以示区分: 论文 1，论文 2……论文 final，论文 final1……论文 finalfinal……往往一篇论文结束，本地备份几十次，这就是本地版本控制系统。如果文件很大，这样的备份方式对内存来说，也是一种挑战。有的同学会将备份上传到中央服务器，这属于集中化的版本控制系统，而中央服务器一旦宕机，丢失数据的风险很大。除此之外，这两种版本控制系统最大的缺陷在于：无法直观的看到版本与版本之间的不同，系统并没有对修改过的地方清晰地标注。这些问题都可以用 Git 来解决。Git 属于分布式版本控制系统，在 Git 中，本地仓库将远程仓库的文件完整地镜像下来（如图 1），这样任何一处服务器发生故障，事后都可用本地仓库恢复。除此之外，Git 管理文件时，可清晰地看到版本更新时内容的变化，可恢复任意版本。这对文件管理尤其代码管理有重要意义。

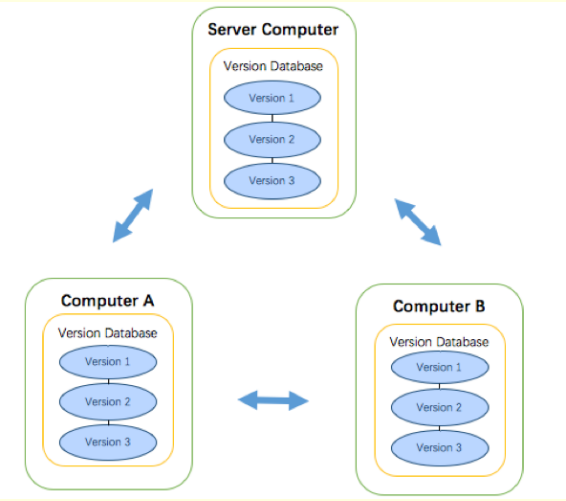

图 1: 分布式版本控制系统

1.2 什么是 GitHub

GitHub 是利用 Git 进行版本控制，用于代码托管的共享虚拟主机服务。Github 上可以创建私人仓库和公共仓库，创建私人仓库需付费，这也是 GitHub 的优点所在，公共仓库可以很方便的进行分享和协作。举个例子，A 新建一个公共仓库，上传自己的代码，其他人都可以看到，大家可以

一起讨论，提问等。如果 B 发现 A 的代码有问题，可以用 **Issues** 选项向 A 提出他代码中的问题，如果 B 对这部分代码有更好的建议，可以写出来，通过 **Pull requests** 选项向 A 提交自己的编码，如果 A 认为 B 的方案确实不错，A 将会把 B 贡献的代码融合到自己的代码中。这为代码分享和分工合作提供了极大的便利。

第 2 章

设置 SSH 密钥

Git 可以通过四种协议来传输数据：本地传输，SSH 协议，Git 协议和 HTTP 协议。下面介绍如何通过 SSH 协议来使用 Git。关于服务器上 Git 协议的问题，请参考：https://git-scm.com/book/zh/v1/4.1 节，在此不再赘述。SSH（Secure Shell）是建立在应用层和传输层基础上的安全协议，为计算机上的 Shell 提供安全的传输和使用环境。简言之，SSH 负责保护本地仓库和远程仓库之间数据传输时不被截获。

2.1 生成密钥

打开终端，在～目录下执行 **ssh-keygen -t rsa**。中间会出现 enter passphrase 的指令，此处一直按 enter 键即可。执行完之后，在.ssh 文件中已生成了 id _ rsa.pub 文档。此时，执行 **ls -a**，可见名为.ssh 的隐藏文件，执行 **cd .ssh**，转到.ssh 文件夹，再执行 **ls** 指令便可看到该目录下已生成的名为 id _ rsa.pub 文件。

2.2 把公共密钥保存到 GitHub 网站上

浏览器打开网页http://www.github.com, 注册（sign up）自己的 github 账号之后登陆（sign in），点击 settings, 选择 SSH and GPG keys 选项，点击 New SSH key, title 写为：your name@iplouc(如图 2), 打开 id_rsa.pub（在.ssh 文件夹展开看到 id_rsa.pub，执行 vim id_rsa.pub 指令，便可读取 id_rsa.pub 中的内容），将里面的内容复制到 key 中，然后保存即可。

2.3 将密钥加载到 SSH 里

执行 **ssh-add** 即可。

＊ SSH 协议和 HTTP 协议的区别：通过 SSH 协议和 HTTP 协议使用 Git，二者下载速度所差无几，在通过 SSH 协议向 GitHub 推送内容时，无需输入用户名和密码，而通过 HTTP 协议则每次推送都需输入帐号和密码。因此，SSH 协议更为方便，推荐大家使用。在设置好 SSH 密钥之后，就可以通过 SSH 协议来使用 Git 了。在下面克隆部分将讲到如何切换 HTTP 协议和 SSH 协议。

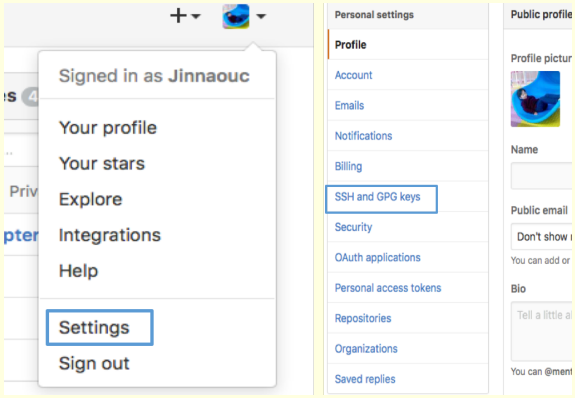

图 2: 进入 SSH keys 设置界面

第 3 章

Git 的安装与配置

本章的主要内容是：基于 Ubuntu 16.04 系统，Git 的安装与配置。

3.1 下载与安装

执行 **sudo apt-get install git**, 便可完成 Git 的下载，这个过程有点慢，耗时多少一定程度上取决于网速和人品。

3.2 初次运行 Git 前的配置

在初次运行 Git 前，要设置个人的用户名和电子邮件地址，Git 提交时会引用用户名和电子邮箱地址来说明是谁提交了更新，git config 是专门用来对 Git 进行配置的指令，我们用 git config 来设置用户名和密码。执行以下指令：

git config --global user.name “your name”

git config --global user.email “your email”

注：双引号部分填写个人用户名和 email 地址，不要忘记双引号。设置完成后，执行 **git config --list** 指令，便可查看是否设置成功。

第 4 章

Git 的基础使用

本章主要对 Git 基本使用做了详细的讲解，其中＊号部分在第一次练习使用 github 时可先不练习，因为其中涉及到一些删除，撤销等问题，刚开始使用 Git 可能不太熟悉。待学会基础 Git 的基础使用之后，可练习＊部分。

4.1 克隆远程仓库

首先，在你的 GitHub 上新建一个 repository, 命名为 gittest（如图 3）, 设置好之后，点击 Create repository。此时，便已新建一个名为 gittest 的仓库。

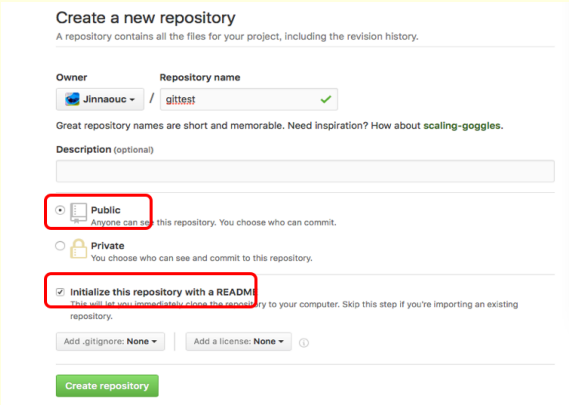

图 3: 新建仓库

进入 gittest 仓库之后，在红框部分可看到 gittest 仓库的 ssh 地址（如图 4）。

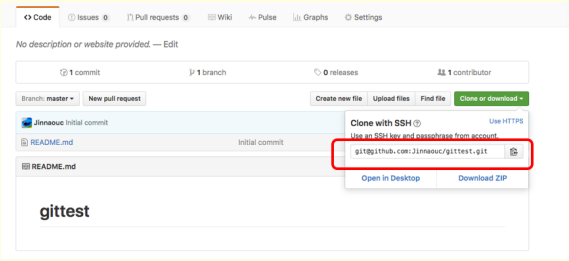

图 4: 复制仓库地址

执行 **git clone git@github.com:Jinnaouc/gittest.git**，记得改成自己的 ssh 地址（注意检查是否时 Clone with SSH, 如果此时显示的是 Clone with HTTPS，点击右上方，蓝色字体：Use SSH 来切换）。此时，在当前目录下用 ls 指令查看，便可看到本地目录有了一个名为 gittest 的文件夹，进入这个文件夹, 用 ls 命令可查看，此文件夹中只有一个 README.md 文件。

＊删除仓库：如果我们不想让此次创建的 gittest 仓库继续留在我们的 GitHub 中，我们可将该仓库删除。在自己的 GitHub 主页上，进入 gittest 仓库后点击 **Settings** 选项（如图 5），将该页面拉至最下方，找到 **Delete this repository**，点击后并输入当前仓库名确认后，便可将该仓库删除。

＊修改仓库名称：修改仓库名称也是在仓库的 Settings 选项下，如图 5，点击 **rename** 即可。

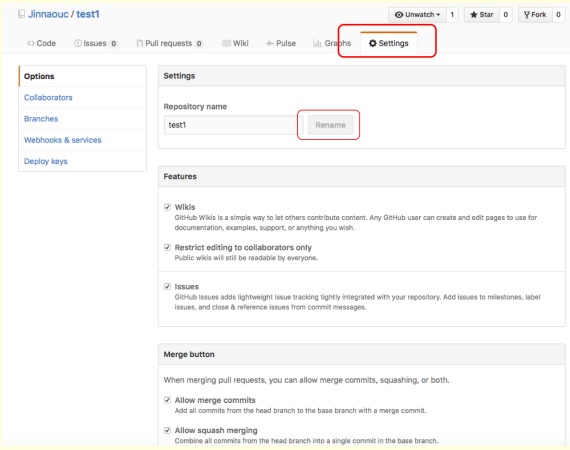

图 5: 进入仓库设置

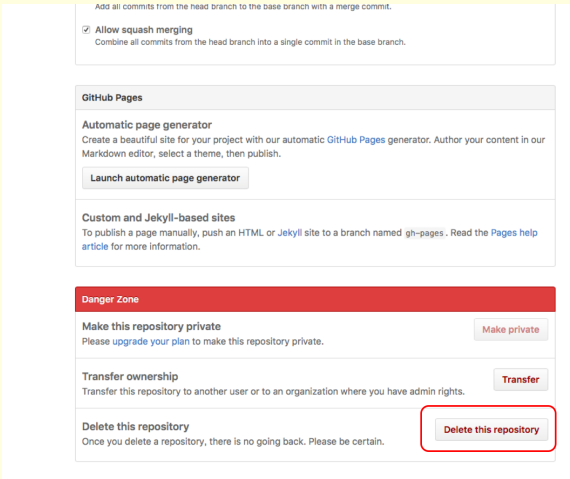

图 6: 点击删除仓库

4.2 初始化仓库

进入 gittest 文件夹后，在此目录下初始化仓库。执行 **git init** 初始化仓库之后，在当前 Git Test 目录下，已生成名为.git 的隐藏工作目录，通过指令 ls -a 查看，一定要确认.git 文件已生成。这一步非常重要，以后所有 Git 需要的数据和资源都存放在这个目录中。

4.3 文件提交

工作目录中的文件的两种状态：已跟踪或未跟踪。已跟踪的文件是指已经纳入版本控制管理的文件，它们可以进行修改，暂存，提交等操作。接下来，以文档“hello”为例，详细说明文件提交的过程。

4.3.1 新建 hello 文档

执行 **touch hello**。

4.3.2 在文档中写入内容

执行 **vim hello** 指令，打开 vim 编辑界面，按 i,o,a,r(不区分大小写) 四个字母在进入编辑界面后输入 Hello! ，按 esc 键，输入:wq（保存退出指令）即可。也可再次在目录终端下输入 vim hello 确认是否编辑成功。

4.3.3 查看当前文件状态

git status 命令可以查看文件在任何时候的状态。执行 **git status** 可以看到，此时跟踪状态显示为存在未跟踪的文件。

4.3.4 开始跟踪

执行 **git add hello**，执行完此条指令之后，hello 便已被跟踪且切换为暂存（staged）状态，正式纳入 Git 的控制系统。此时可再次使用 git status 指令查看。

＊也可以尝试使用 **git add .** 和 **git add *** 指令，二者都可自动将当前目录**所有**修改过的文件跟踪, 但有时会有一些自动生成的文件，无需纳入 Git 的管理，这会将本无需提交的文件也一起纳入 Git 管理。

4.3.5 查看修改文件

再次执行 **vim hello**，将其内容改为 hello world! 保存后退出。此时，执行 **git diff**，便可以查看文档的具体修改内容 (添加和删除的行)。

＊取消修改: 执行 **git checkout --hello** 指令，执行之后可再次使用 git diff 指令查看，便可发现已没有修改情况，即对 hello 的修改已撤消。

4.3.6 跟踪修改文件

此时在目录终端下执行 **git status** 查看 hello 文档的状态。可发现，此时出现了两个状态：原 hello 文件依然是已暂存状态，修改后的 hello 文件是未暂存状态。需再次使用 **git add hello** 指令来暂存已更改的 hello 文档。也就是说，每次对文档进行修改之后，都要用 git add 指令提交暂存。此时，若要查看已暂存文档与上次提交的文档中不同的内容，可执行 **git diff –cached** 命令，此时可以查看已暂存的修改文件和原文件的差异。注意：此时文档已暂存，用 git diff 命令看不到任何修改信息，这两条命令要注意区分。

＊取消跟踪：执行 **git rm --cached hello**, 此时再用 git status 指令查看 hello 的状态，此时 hello 的状态则变成了未跟踪即文件便已从暂存区移除。

＊移除已暂存文件：执行 **git rm hello -f** 强制删除已暂存的 hello 文档，这条命令可将 hello 文档彻底删除。

＊忽略未跟踪文件：在编译过程中，可能会产生一些不需要纳入 Git 管理的文件（通常为隐藏文件），此时我们可以将这些文件忽略，被忽略的文件将不会提交到 Git。首先确保需要忽略的文件是未跟踪状态，如果已跟踪，请用取消指令取消跟踪。然后在项目仓库下创建.gitignore 文件：

touch .gitignore，写入要忽略的文件的扩展名，例如，想要忽略所有.DS_Store 文件，则写入 ***.DS_Store** ，保存退出即可。

4.3.7 提交文件

我们平时更多采用-m 参数来直接提交文件及文件说明。执行 **git commit -m 'message'**，即可直接提交文档 hello。在这个语句中，git commit 是提交文件的指令，'message' 是对本次提交的说明。(message 在 Git-GitHub 的使用中非常重要，每次做更改的时候，都要在 message 部分概述自己的修改。如果一次性将所有文件跟踪，那么在提交时候，这些文件的 message 部分都是一样的，在我们需要不同的文件有不同的 message 的时候，可分别进行跟踪和提交，所以要谨慎使用 git add . 及 git add * 指令。）提交之后，再次查看跟踪状态可发现已没有需要更新的文件。提交前要检查文件的跟踪状态，尤其修改的，新建的文件，要检查是已暂存了。此时，已经完成了第一步：将文件更新到本地仓库。

*跳过使用暂存区域：使用暂存区域的方式可以精心准备和检查待提交的文件，但是有时候却会非常繁琐，Git 提供了一个跳过使用暂存区域的方式，直接执行 **git commit -a -m 'message'** 即可直接提交。

*查看提交历史：执行 **git log** 指令，可以看到以提交时间为顺序的所有更新，若执行 **git log -p -2** 指令可以显示最近两次的更新；**git log –stat** 指令可以简要显示增加和删除的行数。执行 **git log --since=2.weeks** 指令可以显示两周内的提交情况。除此之外，git log 指令还有其他有趣的应用，具体可参考《git 中文教程》第 27 页。

4.4 推送数据到远程仓库

4.4.1 查看当前远程库

执行 **git remote -v** 指令，可查看当前配置的远程库，此时可以看到，默认的库名为 origin，若执行 **git remote** 指令，则只能看到远程库名称，不能看到具体的仓库地址。

*执行 **git remote show origin** 可以查看 origin 这个远程仓库的详细信息。

4.4.2 抓取数据

执行 **git fetch origin**，将仓库最新更新的数据抓取到本地，有的人习惯使用 **git pull** 命令，pull 和 fetch 的区别在于：pull 不仅可以抓取数据，还能合并分支（分支的相关问题可参考https://git-scm.com/book/zh/v2第三章 Git 分支部分，在此不再赘述）。fetch 和 pull 命令与克隆命令的区别在于，它们只到远程仓库拉取所有本地仓库中没有的数据。其重要性将在 3.4 节最后实验室应用部分解释。

*添加远程仓库：在自己的 github 主页新增一个仓库，git remote add [名称] [地址]，执行 **git remote add public git@github.com:Jinnaouc/D.git**，记得改成自己新仓库的地址。其中，public 是新仓库的名字，因为原仓库已默认名称为 origin，再次添加仓库的时候，必须输入仓库名，并且每次添加的仓库名称不能重复，否则无法添加。仓库添加后，可执行 **git fetch public** 来从新的仓库中抓取数据。

＊远程仓库重命名：git remote rename [原名] [新名] 执行 **git remote rename origin report**，这样便将原名为 origin 的仓库更名为 report，可用 git remote -v 再看下远程仓库配置信息。

4.4.3 推送数据至远程仓库

执行 **git push origin master**，此时，你的 hello 文档已经 push 到 github 的 gittest 仓库了，可登录 github 查看。其中，origin 为默认的仓库地址名称，master 为默认的分支名称，在未自行设定仓库地址名称和分支名称前，系统使用默认名称，在 github 操作扩展部分（此处加超链接），会具体讲到如何修改名称，若名称修改后，指令也要对应修改。

4.4.4 恢复历史版本

在对文件进行修改之后，如果要恢复到以前的某个版本，Git 也可以实现。首先执行 **git log** 指令，可查看提交历史，每个提交历史 commit 之后都有其对应的哈希值（由 40 个 16 进制字符 0-9 及 a-f 组成），例如 1d755b5ce673d582dcff067120b347f47ca6a6c3。选定你要回到哪个版本，复制下该版本的哈希值后退出。git reset 命令有很多参数选项，现只介绍用来恢复历史版本的 hard 和 soft 选项。执行 **git reset --hard hash**，其中，hash 要改为刚刚复制的哈希值。执行完该命令后，已将历史版本恢复。注意：选用 hard 选项恢复历史版本会删除此版本之后的所有提交更新的版本。恢复之后，要重新更新到远程 GitHub 上，执行 **git push -f origin master** 即可。记住，-f 参数意为强制执行，必不可少，此时如果不强制执行，会提示使用 git pull 命令将远程项目合并到本地，造成混乱。若要恢复单个文件的历史版本，可以执行 **git log filename** 来查看文件版本的哈希值，仍使用 **git reset --hard hash**

第 5 章

实验室 Git-GitHub 的使用

相信你已经学会了如何将本地仓库的文件上传到了自己的 github 新建的仓库中，接下来对实验室的 Git 的使用，给大家做一个简单的介绍。实验室的 Git 网址：https://github.com/zhenglab，点击进入之后，可以看到，有很多老师创建的仓库，从仓库克隆文件不需要权限，直接执行克隆命令，将你想要的文件克隆到本地即可。若推送数据到仓库，则需要老师授予权限，有了权限之后，才可向仓库中推送数据，push 的方式和向自己 Git 仓库中 push 的方式一样。注意，与推送数据到自己的仓库不同的是，向仓库中推送数据的可能不止自己一个人，如果在你推送数据之前，其他人已经推送了若干更新，你的推送将被驳回，必须先把他们的更新抓取到本地，再推送自己的项目，这便是执行 4.4.2 命令的必要性。总结：向仓库推送数据，首先要获得权限，其次要确保无人更新或已将最新更新抓取到本地。

第 6 章

总结

以上就是对 Git 的基础应用的总结，本文是我整理了孙雪师姐和郑海永老师之前的文档，结合相关教程总结出来的。由于身边人对 Git 的使用均有一定的掌握，因此，没有进行很好的测试。这份攻略是否非常适用毫无经验的新生，还需要接受未来学弟学妹们的考验。同时，非常欢迎大家提出批评意见和补充意见！