《开源软件设计与开发》课程总结

姓名：岳中伟　　学号：52195100010

1、 开源理解

这里介绍本人关于 git 工具的理解与实践。

1.1 git 基础知识

红色表示未加入版本控制。绿色表示已经加入版本控制暂未提交。白色表示加入版本控制，已提交，无改动。蓝色表示加入版本控制，已提交，有改动。灰色表示版本控制已忽略文件。一个总文件内只有一个.git 文件，如果有多个的话，就不会往仓库里推。一定要有一个.gitignore 文件。如果远程分支的版本高于本地分支的版本，并且是两个人修改的同一份文件，但是修改的行数是不一样的，这里的行数不是修改了多少行，而是修改的具体行数，比如同事修改了第 5 行和第 10 行，自己修改了第 100 行，虽然远程分支版本高于本地分支，但是依然是没有任何冲突的。当使用 git clone 下来的时候，一定要切换成想要的分支，不然就是原来的 master 分支。或者 git clone 的时候，就指定自己想要的具体分支。git 工作中常规操作流程：(a) git add filename。(b) git commit -m “我的提交”。(c) git pull 如果有冲突就会提示，然后解决冲突。(d) git push

1.2 git 使用的疑问与回答

为什么要先 add，其次在 commit，然后在 pull，最后再 push？如果我 pull 了，岂不是把自己修改的代码都给覆盖掉了嘛，因为远程没有我改的代码，我 pull，岂不是覆盖了我本地改动好的地方了？那我还怎么 push?

答:先 add,在 commit 再 pull 再 push 的情况就是为了应对多人合并开发的情况：a) add 是为了把修改的文件或者新添加的文件，添加到本地仓库中。b) commit 是为了告诉 Git，我这次提交改了哪些东西,不然只是改了但是 git 不知道我修改了,也就无从判断比较。c) pull 是为了本地 commit 和远程 commit 的对比记录,git 是按照文件的行数操作进行对比的,如果同时操作了某文件的同一行那么就会产生冲突,git 也会把这个冲突给标记出来,这个时候就需要先把和自己冲突的那同事进行讨论，最终确定保留谁的代码,然后在 git commit && git pull ,再次 pull 是为了防止再自己与冲突同事协商的时候另同事又提交了一版东西,如果真发生了，流程重复一遍即可,通常没有冲突的时候就直接把文件合并了,不会把代码给覆盖掉 。d)出现代码覆盖或者丢失的情况:比如 Alice 和 Bob 两人的代码 pull 的时候，文件版本都是 1,Alic 在本地提交了 2,3 并且推送到远程了,Bob 修改文件完成后没有 commit 操作,直接 git pull 这个时候 Bob 本地版本已经到 3 了。Bob 在本地版本 3 的时候改了 A 写过的代码,再进行了 git commit && git push 那么在远程版本中就是 4,而且 A 的代码被覆盖了,所以说都要先 commit 再 pull,最后在 push,不然会覆盖代码的。

1.3 Tag 使用记录

1 查看 taggit tag

2 在某个 commit 上打 tag　　git tag v1.0.0 c809ddbf83939a89659e51dc2a5fe183af384233

3 本地 tag 推送到远程　git push origin v1.0.0

4 删除本地 tag　　git tag -d fc-V1.0.0

5 删除线上 tag　　git push origin :refs/tags/ fc-V1.0.0(注意冒号前有一个空格)

1.4 如何在 GitHub 上协同工作？

1. 建立关系

(a)查看现有远程仓库：

```
$ git remote -v
origin  https://github.com/YOUR_USERNAME/YOUR_FORK.git (fetch)
origin  https://github.com/YOUR_USERNAME/YOUR_FORK.git (push)
```

(b)添加指向原仓库的 upstream

```
$ git remote add upstream
https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git
```

(c)查看 origin 和 upstream

```
$ git remote -v
origin    https://github.com/YOUR_USERNAME/YOUR_FORK.git (fetch)
origin    https://github.com/YOUR_USERNAME/YOUR_FORK.git (push)
upstream  https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git (fetch)
upstream  https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git (push)
```

(d)直接从原仓库的 master 分支拉取代码并直接合并代码，其中 pull=fetch+merge.

Git pull upstream master

2. 协同工作流程理解和梳理

a)自己本地仓库，自己 GitHub 远程仓库，对方 GitHub 远程仓库。

b)我可以先在本地写好代码。提交到我本地仓库。

c)然后，我在 pull request 的时候应该先去合并对方仓库的代码。

d)然后，我需要把本地仓库的代码，push 到我 GitHub 的仓库上。

e)最后，我在 New pull request。

3. 如何同步对方仓库的代码。

git pull upstream master(这行代码只能把对方仓库中的代码拉取到本地仓库而已)

1.5 自己对 GitHub 协同工作理解

自己本地仓库肯定是和自己的 GitHub 远程仓库建立关系。然后本地又需要和源 GitHub（远程别人的 GitHub 仓库）建立关系。然后理解它就很简单了，自己的 GitHub 远程仓库和源 GitHub 远程仓库想象成两个分支，中间桥梁就是自己本地分支。由自己本地分支去交互自己的 GitHub 远程仓库和源 GitHub 远程仓库。合并了最新的代码后，就直接去 GitHub 网站，New pull request 就行了。它的工作原理其实跟自己玩的时候一样的，我在自己的远程仓库中建立了无数个分支。假如我本地的分支跟远程分支是一一对应的。然后我本地仓库作为

中间桥梁，不停地去合并别的分支的代码到本地，然后 push 到远程仓库分支（push 前是需要先进行 pull 操作的）。他们唯一的区别就是，一个直接在本地就可以完成分支合并，但是合并的结果也还是在本地，需要 push 一下。GitHub 的工作模式是，我本地仓库作为中间桥梁去 pull 源 GitHub 上的最新代码并合并到本地分支，然后在把最新代码 push 到自己的远程仓库中，然后最后一步就是 New pull request 即可。

2、开源贡献

本人虽然对相关开源项目进行了解与学习，但是目前并没有做出相应贡献。

3、课程反馈

该课程加深了自己对开源的认识，了解到开源的好处：（1）不用重复造轮子，（2）不用闭门造车，（3）不用成为商业软件的奴隶，（4）激发自己的创造力。本人认为开源是以后的发展趋势。非常感谢老师在开源课程的精彩演讲。

4、参考文献

书籍《Git 入门指南》

书籍《Git 权威指南》

书籍《Git 团队协作》

书籍《GitHub 入门与实践》

期刊《站在潮头看开源：趋势已定，未来已来》