**Github学习小记**

**概况**

CVS及SVN都是集中式的版本控制系统，而Git是分布式版本控制系统。

集中式版本控制系统最大的毛病就是必须联网才能工作；

分布式版本控制系统根本没有“中央服务器”，每个人的电脑上都是一个完整的版本库，这样，你工作的时候，就不需要联网了，因为版本库就在你自己的电脑上。

和集中式版本控制系统相比，分布式版本控制系统的安全性要高很多。

**基本操作**

（1）安装Git

首先，选择一个合适的地方，创建一个空目录：

$ mkdir learngit

$ cd learngit

$ pwd

pwd命令用于显示当前目录。在我的Mac上，这个仓库位于/Users/michael/learngit。

如果你使用Windows系统，为了避免遇到各种莫名其妙的问题，请确保目录名（包括父目录）不包含中文。

第二步，通过git init命令把这个目录变成Git可以管理的仓库：

（2）创建版本库

初始化一个Git仓库，使用git init命令。

添加文件到Git仓库，分两步：

1.使用命令git add <file>，注意，可反复多次使用，添加多个文件；

2.使用命令git commit -m <message>，完成。

**[注]：**

vi readme.txt 进入VIM编辑器,可以新建文件也可以修改文件 如果这个文件，以前是没有的，则为新建，则下方有提示为新文件。

:I 进入编辑模式

按ESC键 跳到命令模式，然后：

:w 保存文件但不退出vi

:w file 将修改另外保存到file中，不退出vi

:w! 强制保存，不推出vi

:wq 保存文件并退出vi

:wq! 强制保存文件，并退出vi

q: 不保存文件，退出vi

:q! 不保存文件，强制退出vi

:e! 放弃所有修改，从上次保存文件开始再编辑

（3）状态与修改内容查看

* 要随时掌握工作区的状态，使用git status命令。
* 如果git status告诉你有文件被修改过，用git diff可以查看修改内容。

（4）版本回退

* HEAD指向的版本就是当前版本，因此，Git允许我们在版本的历史之间穿梭，使用命令git reset --hard commit\_id。在Git中，用HEAD表示当前版本，上一个版本就是HEAD^，上上一个版本就是HEAD^^，当然往上100个版本写100个^比较容易数不过来，所以写成HEAD~100。
* 穿梭前，用git log可以查看提交历史，以便确定要回退到哪个版本。
* 要重返未来，用git reflog查看命令历史，以便确定要回到未来的哪个版本。

（5）撤销修改

场景1：当你改乱了工作区某个文件的内容，想直接丢弃工作区的修改时，用命令git checkout -- file。

场景2：当你不但改乱了工作区某个文件的内容，还添加到了暂存区时，想丢弃修改，分两步，第一步用命令git reset HEAD <file>，就回到了场景1，第二步按场景1操作。

场景3：已经提交了不合适的修改到版本库时，想要撤销本次提交，参考[版本回退](https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/0013744142037508cf42e51debf49668810645e02887691000" \t "_blank)一节，不过前提是没有推送到远程库。

（6）删除文件

* 命令 rm 用于从工作区删除一个文件：rm filename
* 命令git rm用于从版本库删除一个文件，并且git commit。git rm filename；rm filename；git commit –m”message”。
* 如果一个文件已经被提交到版本库，那么你永远不用担心误删，：
* git checkout其实是用版本库里的版本替换工作区的版本，无论工作区是修改还是删除，都可以“一键还原”，但是要小心，你只能恢复工作区文件到最新版本，你会丢失**最近一次提交后你修改的内容**：git checkout -- filename

（7）添加远程库

要关联一个远程库，使用命令git remote add origin git@server-name:path/repo-name.git；

关联后，使用命令git push -u origin master第一次推送master分支的所有内容；

此后，每次本地提交后，只要有必要，就可以使用命令git push origin master推送最新修改；

（8）从远程库克隆

首先，登陆GitHub，创建一个新的仓库；

下一步是用命令git clone克隆一个本地库：

$ git clone git@github.com:michaelliao/gitskills.git

（9）完整的操作代码（push+clone）：

mkdir diskname://foldername

cd diskname://foldername

git init

ls –ah

git add file

git commit –m”reference”

get remote add origin git@server-name:path/repo-name.git

git push origin master

get clone git@server-name:path/repo-name.git

（10）创建与合并分支

Git鼓励大量使用分支：

查看分支：git branch

git branch命令会列出所有分支，当前分支前面会标一个\*号。

创建分支：git branch <name>

切换分支：git checkout <name>

创建+切换分支：git checkout -b <name>

git checkout命令加上-b参数表示创建并切换，相当于以下两条命令：

$ git branch dev

$ git checkout dev

Switched to branch 'dev'

切换后的操作不要忘记add以及commit

合并某分支到当前分支：git merge <name>

Fast-forward信息，表示此次次合并是“快进模式”，也就是直接把master指向name的当前提交，所以合并速度非常快。

删除分支：git branch -d <name>

（11）解决冲突

用git status也可以告诉我们冲突的文件，Git用<<<<<<<，=======，>>>>>>>标记出不同分支的内容。

用git log --graph命令可以看到分支合并图。

$ git log --graph --pretty=oneline --abbrev-commit

（12）分支管理策略

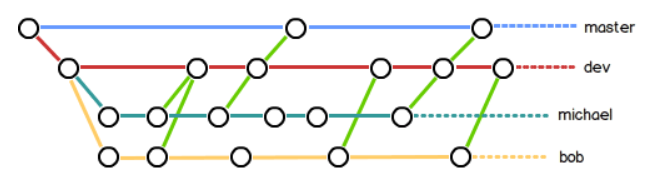
强制禁用Fast forward模式，Git就会在merge时生成一个新的commit，这样，从分支历史上就可以看出分支信息。

git merge --no-ff方式合并：

$ git merge --no-ff -m "merge with no-ff" dev

用git log --graph –pretty=oneline –abbrev-commit看看分支历史

master分支应保持稳定，只用于最终版本发布，平时的修改可以在新的分支上进行，随后与master分支合并。



（13）bug分支

用get stash功能，可以把当前工作现场（如dev）“储藏”起来，等以后恢复现场后继续工作。

git checkout master，调整分支到master上，在master分支上新建临时分支消除bug后，add+commit，切换到master分之后，与消除bug的临时分支合并git merge --no-ff -m "merged bug fix 101" issue-101，删除临时分支，随后恢复工作现场继续工作git checkout dev。

用git stash list命令查看工作现场存放位置。

恢复工作现场：

一是用git stash apply恢复，但是恢复后，stash内容并不删除，你需要用git stash drop来删除；

另一种方式是用git stash pop，恢复的同时把stash内容也删了。

可以多次stash，恢复的时候，先用git stash list查看，然后恢复指定的stash，用命令：

$ git stash apply stash@{0}

（14）feature分支

如果要丢弃一个没有被合并过的分支，可以通过git branch -D <name>强行删除。

（15）多人协作

当你从远程仓库克隆时，实际上Git自动把本地的master分支和远程的master分支对应起来了，并且，远程仓库的默认名称是origin。要查看远程库的信息，用git remote，或者，用git remote -v显示更详细的信息。

推送分支：

* master分支是主分支，因此要时刻与远程同步；
* dev分支是开发分支，团队所有成员都需要在上面工作，所以也需要与远程同步；
* bug分支只用于在本地修复bug，就没必要推到远程了，除非老板要看看你每周到底修复了几个bug；
* feature分支是否推到远程，取决于你是否和你的小伙伴合作在上面开发。

多人协作的工作模式通常是这样：

1. 首先，可以试图用git push origin <branch-name>推送自己的修改；
2. 如果推送失败，则因为远程分支比你的本地更新，需要先用git pull试图合并；
3. 如果合并有冲突，则解决冲突，并在本地提交；
4. 没有冲突或者解决掉冲突后，再用git push origin <branch-name>推送就能成功！

如果git pull提示no tracking information，则说明本地分支和远程分支的链接关系没有创建，用命令git branch --set-upstream-to <branch-name> origin/<branch-name>，再用git pull把最新的提交从origin/dev抓下来，然后，在本地合并，解决冲突，再推送。

（16）rebase

git rebase可以把本地未push的分叉提交历史整理成直线。

rebase操作的特点：把分叉的提交历史“整理”成一条直线，看上去更直观，使得我们在查看历史提交的变化时更容易，因为分叉的提交需要三方对比。

rebase的缺点是本地的分叉提交已经被修改过了。

（17）标签管理：创建与操作

1.首先，切换到需要打标签的分支上：git branch+git checkout <name>

2.打标签：git tag <name>

* 命令git tag <tagname>用于新建一个标签，默认打在HEAD上，也可以指定一个commit id：git tag <tagname> commitid；
* 命令git tag可以查看所有标签。标签不是按时间顺序列出，而是按字母排序的。可以用git show <tagname>查看标签信息。
* 命令git tag -a <tagname> -m "blablabla..."可以指定标签信息，用-a指定标签名，-m指定说明文字。用命令git show <tagname>可以看到说明文字。

$ git tag -a v0.1 -m "version 0.1 released" 1094adb

 【注】标签总是和某个commit挂钩。如果这个commit既出现在master分支，又出现在dev分支，那么在这两个分支上都可以看到这个标签。

3.标签的删除：git tag –d <tagname>。创建的标签都只存储在本地，不会自动推送到远程。

4.标签的远程推送：git push origin <tagname>

一次性推送全部尚未推送到远程的本地标签：git push origin --tags

如果标签已经推送到远程，要删除远程标签就麻烦一点，先从本地删除： git tag –d <tagname>；然后，从远程删除：git push origin :refs/tags/tagname

（18）配置别名

--global参数是全局参数，也就是这些命令在这台电脑的所有Git仓库下都有用。

命令git config –globla alias.abbrname originalname

$ git config --global **alias**.st status

$ git config --global **alias**.co checkout

$ git config --global **alias**.ci commit

$ git config --global **alias**.br branch

$ git config --global **alias**.lg "log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit"