Git/github笔记

# 前言

（1）git管理的对象

git管理的对象是整个git仓库，每次提交会生成一个新的版本快照(对整个仓库进行一次快照)，伴有commit-id. git会对你所建立的git仓库的所有文件进行跟踪管理（如果你新创建了一个文件，但没有add，使用git status,会显示文件未跟踪；然后add了但没有commit，再git commit,此时会显示文件有待提交；只要没有commit,没有产生commit-id,就等于没有存到git仓库，也不能进行版本回溯）

（2）如何理解git的快照流原理？

每次提交，git会对仓库内的所有文件进行快照。比如你共进行了两次操作，第二次你提交了你修改的文件，git会将你提交的文件与上一版文件对比，如果有差异，则存下你提交的文件；如果没差异，则在第二份快照中申城一个链接指向上一版文件，对于那些你没提交的文件，也是生成链接指向上一版文件，这样你任意一个时刻的快照都保存了版本中所有文件的的信息。

则你回溯到任意时刻，则都能拥有该时刻的所有文件。

# 安装git

1. windows上有专门的软件包，可以在git的官网上下载，不需要选什么，一路yes、next就可以完成安装。
2. Linux直接输入命令 sudo apt-get install git

## （1）git init

(在仓库中)git init

(不在仓库中)git init (仓库目录)

## （2）git clone

克隆仓库的命令格式是

git clone [url] ——url是该仓库的网址

git clone https://github.com/libgit2/libgit2 mylibgit

这将执行与上一个命令相同的操作,不过在本地创建的仓库名字变为 mylibgit。

注:声明仓库的主人

Git cinfig --global user.name “your name”

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

--global，官方原文：如果使用了该选项,那么该命令只需要运行一次,因为之后无论你在该系统上做任何事情, Git 都会使用那些信息。 当你想针对特定项目使用不同的用户名称与邮件地址时,可以在那个项目目录下运行没有 --global 选项的命令来配置。

个人理解：用了global选项就像你在你的电脑上登录qq时勾选记录账号与密码，以后再登录时都不用再输入账号和密码了；也可以再次使用带global的命令来永久修改账号和密码，修改后的账号和密码也会长期记录。

如果别人在你的电脑上登qq，要登录他的账号，怎么办呢？

此时可以先创建一个文件夹，再使用不带global的命令来登录账号，那么这个目录下提交的文件的账号会是别人的了。

# 理论基础

三棵树：

1. 当前工作目录
2. 暂存目录
3. 仓库

# git add

（在没有进行提交前，只有工作目录有文件，暂存区，仓库都没有文件，head指向工作目录的文件）

将文件提交至暂存区

Git add file1,file2,.... ——将文件从工作目录提交至暂存区（此时工作区与暂存区文件版本相同，仓库中还没有文件，head指向暂存区的文件）

git add 文件夹 ——则会暂存该文件夹下的所有文件

# 4.git commit

 git commit 主要是将暂存区里的改动给提交到本地的版本库。每次使用git commit 命令我们都会在本地版本库生成一个40位的哈希值，这个哈希值也叫commit-id，   
 commit-id 在版本回退的时候是非常有用的，它相当于一个快照,可以在未来的任何时候通过与git reset的组合命令回到这里。

git commit -m ‘message’   
-m 参数表示可以直接输入后面的“message”，如果不加 -m参数，那么是不能直接输入message的，而是会调用一个编辑器一般是vim来让你输入这个message，   
message即是我们用来简要说明这次提交的语句。  
 git commit -am ‘message’ -am等同于-a -m   
-a参数可以将所有已跟踪文件中的执行修改或删除操作的文件都提交到本地仓库，即使它们没有经过git add添加到暂存区，   
注意: 新加的文件（即没有被git系统管理的文件）是不能被提交到本地仓库的。

（此时工作目录，暂存区，仓库的文件版本统一，head指向仓库的文件）

--amend:用以修改已提交的备注信息（-m）

使用方法有二：

1. 刚commit以后发现备注信息写错了，此时，直接git commit --amend

便可修改备注信息。而不会生成新的快照。

1. commit以后发现还有文件忘暂存提交了，

$ git commit -m 'initial commit'

$ git add forgotten\_file

$ git commit --amend

git commit --amend是会将暂存区的文件提交的，只需要在git commit --amend前将为暂存的文件add上，便可以补上文件并修改备注信息，但不会产生新的快照。

# 忽略文件（一般是自动生成的文件）

用于忽略比如自动生成的文件，这些文件你并不想提交，百度吧。

# git reset

git reset --soft HEAD～

--mixed

--hard

其中HEAD～代表移动指针，改变版本，一个～代表回到上一个版本，10个代表回到上是个版本，也可以写为HEAD～10

--soft HEAD～其实就是取消git commit 的作用，此时工作目录与暂存区都是最新版，仓库是前一版的

--mixed HEAD～ ，工作目录是最新版的，但暂存区个仓库都是上一版的，等于既没有add也没有commit

--hard HEAD～ ，三棵树都回复到上一版，当前工作区的内容会直接覆盖掉，慎用

也可以通过版本快照的id号来进行版本的切换:

git reset 版本id号

如果不加参数的话，只会恢复到暂存区，license不会回来的，可以加上hard选项，也可以用checkout命令。

# checkout命令

# git log

git log:什么参数都不加，会显示出当前分支的提交记录

-p：会显示每一次改动的内容，显示比较详细

--stat:会显示出文件的修改记录（比如增加了几句，删除了几句，但不会显示具体修改内容）

--oneline参数可以将每条日志的输出为一行，如果日志比较多的话，用这个参数能够使结果看起来比较醒目，可以节约日志的篇幅。

## git log –graph

--graph参数会绘制提交的线索，如果有合并的话，也会更清晰地显示出来

## git log --decorate

git log --pretty=format：可以自定义日志输出的格式（用法自行百度）

## git log --author yourname

git log --author=simonyao——找出作者是simonyao的提交记录

## --committer

git log --committer=simonyao——仅显示指定提交者相关的提交。

## git log --grep keywords

git log --grep=xxx

--all-match：如需要满足多个搜索项（比如作者与指定的关键字都满足），则需要加上该选项

## git log -[length]

-[length]参数用于指定显示多少条日志

## git log –skip=[skip]

--skip=[skip]参数用来指定跳过前几条日志。

# git status

git status：要查看哪些文件处于什么状态（暂存？提交？）,可以用 git status 命令

-s :以简短的形式显示仓库中所有文件的状态

？？：文件未被跟踪

左边的M（绿色）：文件被修改且已放入暂存区

右边的M（红色）：文件被修改但未放入暂存区

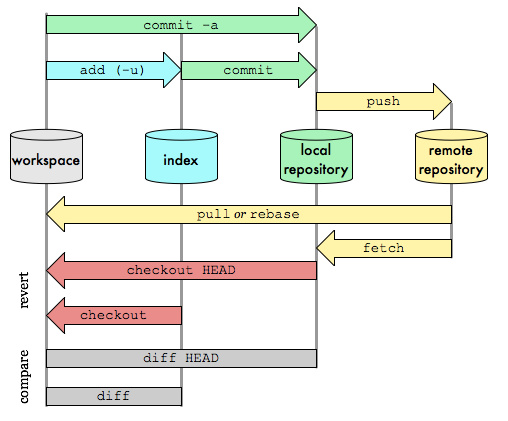
MM：修改了将文件放入暂存区后，又在工作区进

修改，但第二次的修改未放入暂存区

A：新添加到暂存区的文件（由未跟踪刚刚开始跟踪）

注：commit后未修改的文件则不会显示。

# git diff



git diff 后跟两个参数，如果只写一个参数，表示默认跟 workspace中的代码作比较。git diff 显示的结果为 第二个参数所指的代码在第一个参数所指代码基础上的修改。如， git diff HEAD 表示 workspace 在 最新commit的基础上所做的修改。

Git diff HEAD~ HEAD表示commit至仓库中的最新版本相比与前一版本的差异。

git diff：查看 以暂存区为基础，工作区与暂存区中文件的差异。

git diff --cached（--staged）：查看 以仓库为基础，暂存区与仓库中文件的差异。

git diff HEAD：查看 以仓库为基础，工作区与仓库中文件的差异。（所有的HEAD也可以用commit-id来代替）

注：

（1）基础都是靠后的树

（2）git diff HEAD 表示目前工作区内容与仓库中最新版文件的内容对比差，

Git diff HEAD~n 表示目前工作区内容与仓库中前n版文件的内容对比差，

涉及到分支，则跟^有关，后面再补充 ）

关于HEAD后的～与^的意义区别，见网址

https://blog.csdn.net/GW569453350game/article/details/46998395

1. 现在也可以比较word的内容区别了

# 10.git rm

Git rm 文件名

只能删除工作目录和暂存区的记录（下一次快照中不会有该文件），如果存到了仓库（已经生成快照），这个命令无法删除仓库中的记录；这时只能用git reset了，将仓库中的版本快照切掉。

如果第一次修改文件存入了暂存区没提交，又第二次修改了文件，此时工作区与暂存区的文件内容是不一样的，此时使用git rm 则要考虑到一下选项。

-f 强制删除，工作区和仓库中所有关于该文件的内容个全部删除

git rm --cached filename ——删除指定文件在暂存区中的暂存，而不删除工作区中的内容。

# 11.git中改名

如果直接在图形界面的工作区改名，则git status会显示你删除了一个文件有增加了一个文件，这与我们想要的是不一样的，此时就要使用

git mv oldname newname

此后用git status,会看到已改名的提示

# 11.分支管理

## 如何理解branch与HEAD？

branch:branch为一个指针，它总指向当前分支上最新的commit版本。

比如：master分支上现在最新的快照版本为a1，此时分支master指向的就是a1；你又创建了一个分支apple，此时你就有两个分支指针了，此时他们都指向a1；你切换至apple分支工作并提交了一个版本快照b1，此时指针apple指向了b1，而指针master依然指向a1。

HEAD：其总是指向你当前工作目录的最新版本。

## （1）git branch

git branch:显示目前所有已有分支，并以\*表示当前工作目录所在分支。

git branch <new-branch-name>:创建新分支

git branch -u:

git branch -d <branch-name>:删除分支

git branch -D <branch-name>:强制删除分支（对于那些你创建了但没合并的分支）

git branch -m <old=branch-name> <new-branch-name>:改名

git branch --merged:显示已合并的分支（删除以后的分支不会显示）

git branch --no-merged:显示未合并的分支

## （2）git checkout

git checkout <branch-name>:切换分支

git checkout -b branchname:创建并切换至新分支

git checkout -b [branch1] [remotename]/[branch2]：在本地创建新分支branch1（此本地分支称为跟踪分支）去自动跟踪远程仓库remotename上的branch2分支。

作用：如果在一个跟踪分支上输入 git pull,Git 能自动地识别去哪个服务器上抓取、合并到哪个分支。比如你和你的同事都从master分支上分出了两个分支= =这有什么用啊

注：1.匿名分支：如果你使用 git checkout HEAD~进行分支切换，因为没有指定名字，所以进入的是匿名分支，后面再切回master分支时，匿名分支可能会被丢弃，等于没做。但也可以用匿名分支来做些实验。

2.和reset 类似的作用

切换文件版本时（head的指向是不会变的，和reset一样）

checkout 只是把指针的指向切换会指定的快照，而之前的快照依然存在

Reset把版本快照和指针都切换回过去，先前的指针没了

## (3)git merge

git merge <branch-name>:将指定分支合并入当前分支

## （4）变基

含义：将分支b的修改，复制到分支a上，或者说在a上重放一遍（指针b也会指向a上新复制的内容，原本的分支b作废），那么分支a上便有了分支b的内容，从而不用将分支a，b合并（分支b可删除），在往远程仓库提交的分支也会更清晰（因为每次向远程仓库只提交一个分支，不会显示你这次提交的分支是由那些分支合并的；等于你只需要给远程仓库提交重要的结果，而不用提交如何得到这个结果的过程）

《1》变基实现步骤

1）git rebase <branch1>:将当前所在分支（test01）的修改变基到指定分支branch1上（或者说，将当前所在分支的求改在指定分支上重放一遍）

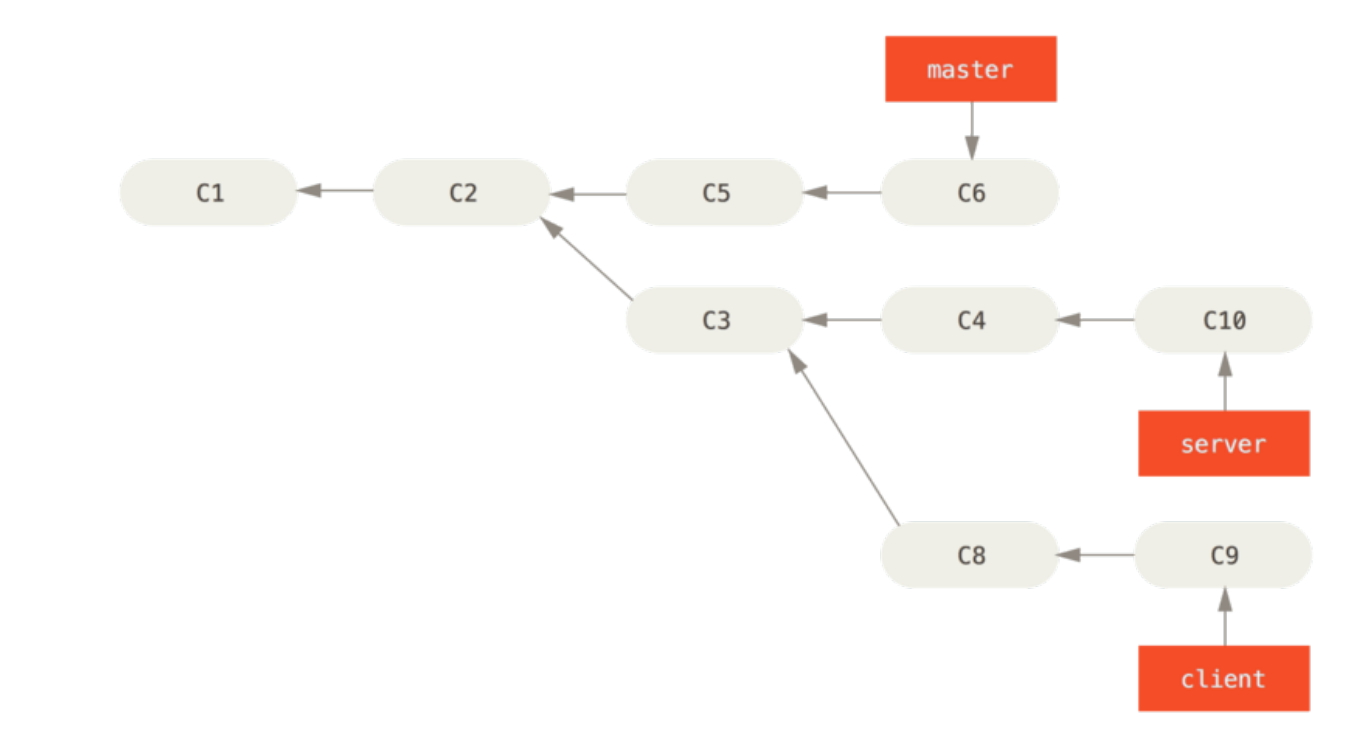
2）git checkout branch1:切换至分支branch1上

3)git merge test01:将分支test01合并入当前分支（其实在变基后，这一步只是实现master指针的移动）

注：变基的使用更可以使你的提交历史更加清晰明了，但千万注意变基与合并的使用场合。

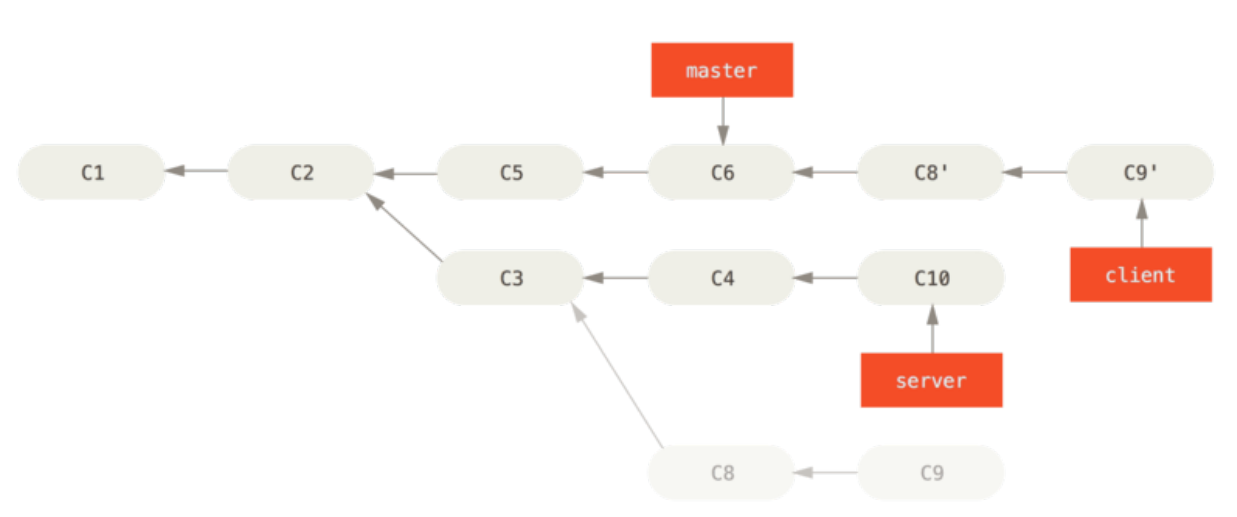
总的原则是,只对尚未推送或分享给别人的本地修改执行变基操作清理历史,从不对已推送至别处的提交执行变基操作,这样,你才能享受到两种方式带来的便利。

《2》多分支变基方法



git rebase master servic client:

取出 client 分支,找出处于 client 分支和 server 分支的共同祖先之后的修改,然后把它们在 master 分支上重放一遍,变为：



# 13.远程仓库

## 如何理解远程仓库？

原文：远程仓库是指托管在因特网或其他网络中的你的项目的版本库。 你可以有好几个远程仓库,通常有些仓库对你只读,有些则可以读写。

个人理解：可以理解为远程仓库为个人在互联网中的虚拟磁盘，你的虚拟磁盘是你的，这是你的远程仓库；如果别人的虚拟磁盘对你给了权限，你可以从中获取资源，那么可以认为他的虚拟磁盘也是你的远程仓库。

## （1）git remote :查看你的远程仓库

-v：更详细地显示你的远程仓库的信息

## （2）git remote add <shortname> <url>：添加一个远程仓库

此后就可以在命令行中用shortname来代替你的远程仓库的url

## （3）git fetch:

抓取远程仓库有而本地没有的数据，但并不会进行分支合并.

如果你fecth 了某仓库一次，隔一段时间再fecth该仓库一次，这次你会得到上次你Fecth以后该仓库的所有更新。

注：这个命令并不会把你抓取过来的资源合并入你的分支，你需要手动进行合并，即你抓取的你同事的修改后你要手动合并入你的分支。

## （4）git pull：

自动的抓取然后合并远程分支到当前分支,相当于git fetch 紧跟一个git merge。这里的合并是将本地分支合并如远程分支。

## （5）git push [remote-name] [branch-name]：

将本地（分支）推送至远程仓库。前提是你对该仓库有写入权限。

1）git push [remote-name] [branch-name1]：[branch-name2] :将本地仓库的分支branchname1推送到远程仓库的分支branchname2上去。

2）git push <remote-name> --delete <branch-name>:从远程服务器上删除一个远程分支。

注：1）当你和其他人在同一时间克隆,他们先推送到上游然后你再推送到上游,你的推送就会毫无疑问地被拒绝。 你必须先将他们的工作拉取下来并将其合并进你的工作后才能推送。

1. 当你第一次推送时，应该
2. 用git fetch <shortname>获取该该仓库的数据，github不允许上传时，有些数据在github有而本地没有，应先进行数据同步，
3. 首次上传某一分支时，应当使用命令

git push --set-upstream <shortname> <branch>

1. 如果你提交时，本地分支的时间比github上还要旧，此时会报错：

提示：更新被拒绝，因为您当前分支的最新提交落后于其对应的远程分支。

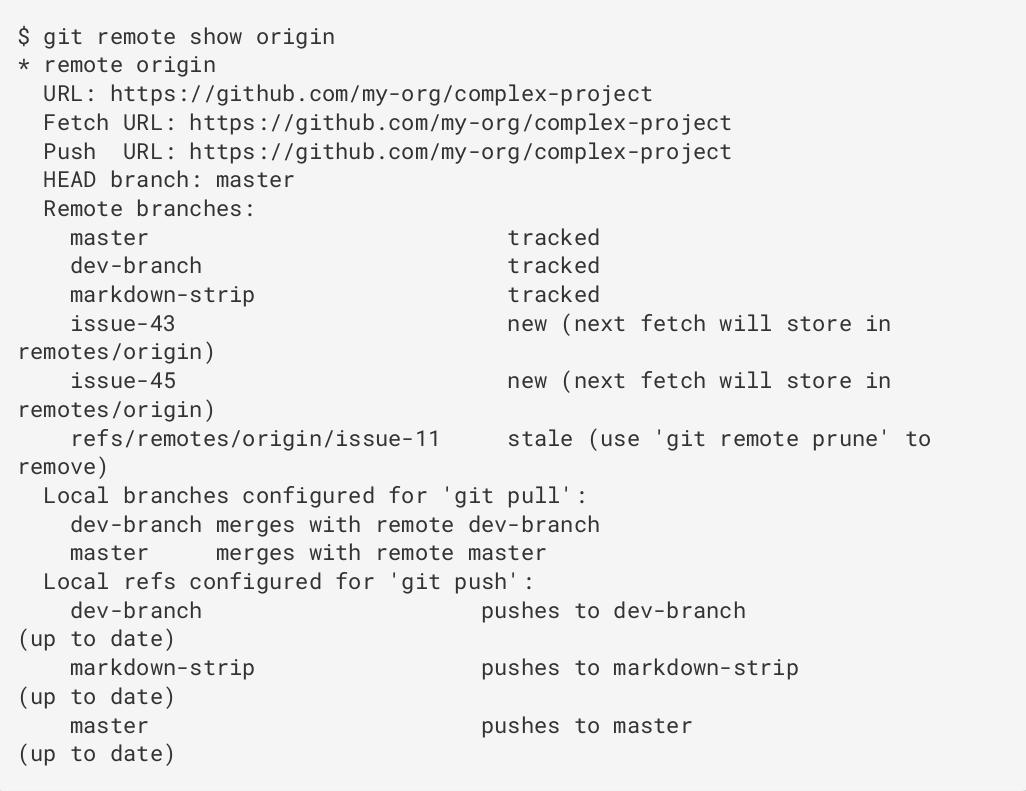
想在应该与远程分支合并git merge.

1. 如果合并分支时出现错误：fatal: 拒绝合并无关的历史

解决办法：git pull repo9 master --allow-unrelated-histories

## （6）git remote show [remote-name] ：

查看某一个远程仓库的更多信息。



1）HEAD branch:master—它告诉你正处于哪一个分支（当前处于master分支）

2）当你在特定的分支上执行 git push 会自动地推送到哪一个远程分支

3）也同样地列出了哪些远程分支不在你的本地

4）哪些远程分支已经从服务器上移除了,

5）当你执行 git pull 时哪些分支会自动合并。

## （7）远程仓库改名

git remote rename oldname newname

注:这同样也会修改你的远程分支名字。 那些过去引用 oldname/master 的现在会引用 newname/master。

## （8）移除远程仓库

git remote rm <shortname>：删除你本地中该网址的联系。让这个虚拟磁盘不再是你的远程仓库。

## （9）查看远程仓库的分支信息

git ls-remote <shortname>

# 14.打标签

（1）

# 15.命令改名或自定义命令

git config --global alias.last 'log -1 HEAD'

添加了一个新的命令git last:查看最近一次提交信息

# 16.github的使用

（1）SSH

SSH连接：如果需要团队里的每个人都对仓库有写权限,又不能给每个人在服务器上建立账户，适用。

SSH私钥&公钥：在仓库的~/.ssh找到一对以 id\_dsa 或 id\_rsa 命名的文件。公钥所在文件 id\_dsa(id\_rsa).pub，私钥所在文件则不带.pub。提交公钥给git仓库管理员，且他将你的公钥加入了授权文件夹中的话，你便可以对该仓库执行读写操作。

如果你的仓库中没有那两个文件，执行ssh-keygen。