Git基本原理及其使用

分享人：梨子

GitHub :https://github.com/MollyMmm

- 与SVN区别

1.版本控制方式

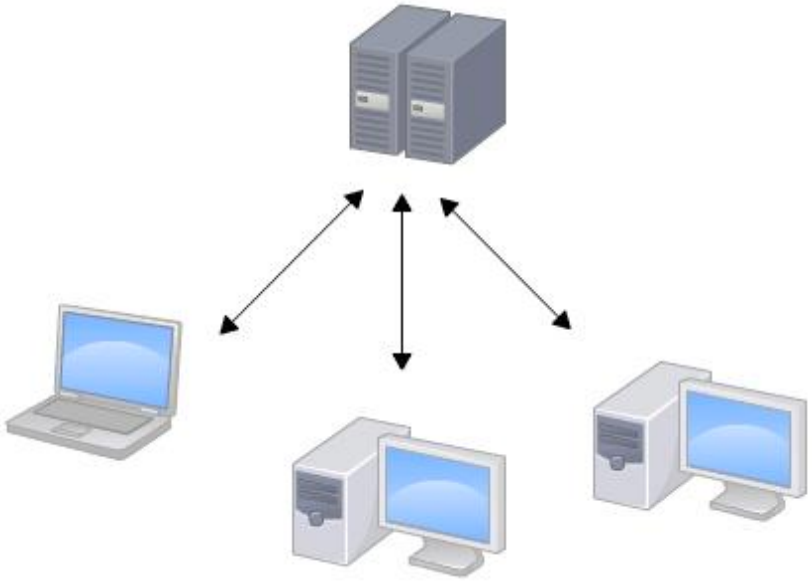
SVN: 集中式

Git: 分布式

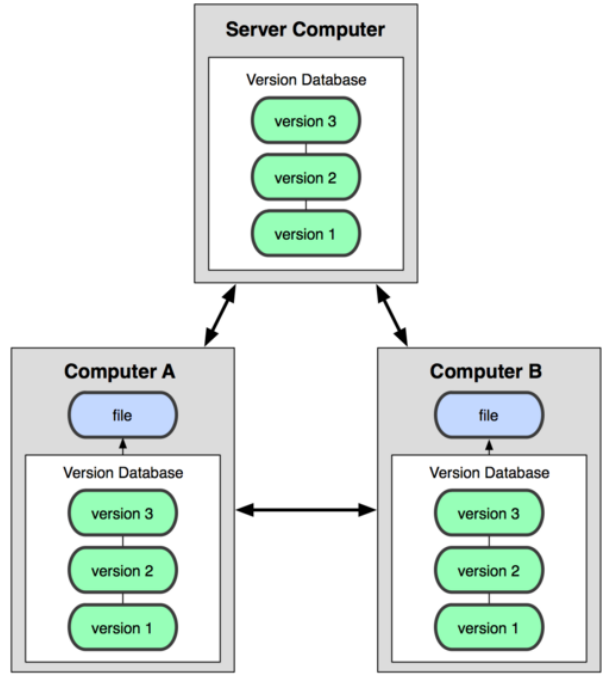
2.文件存储方式

SVN: 按文件

Git: 按元数据



集中式版本控制系统



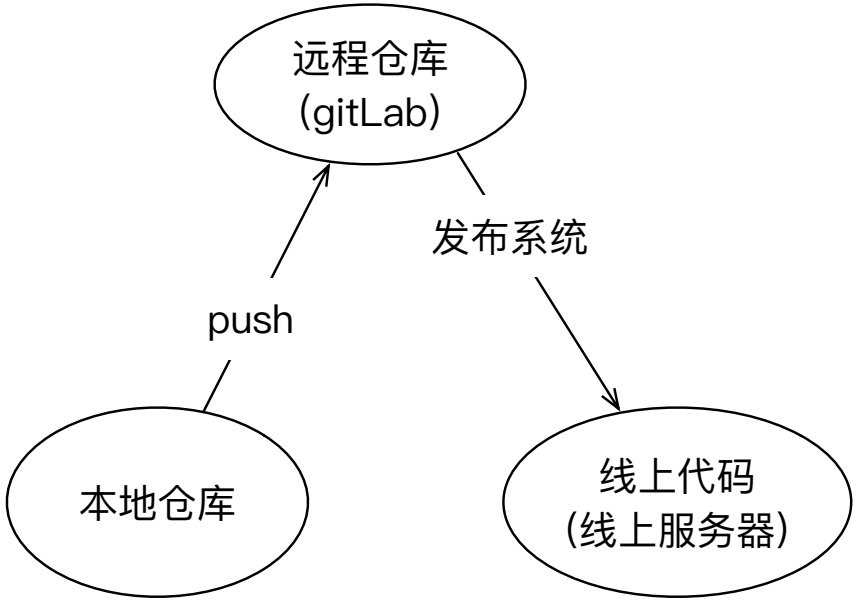
集中式版本控制系统

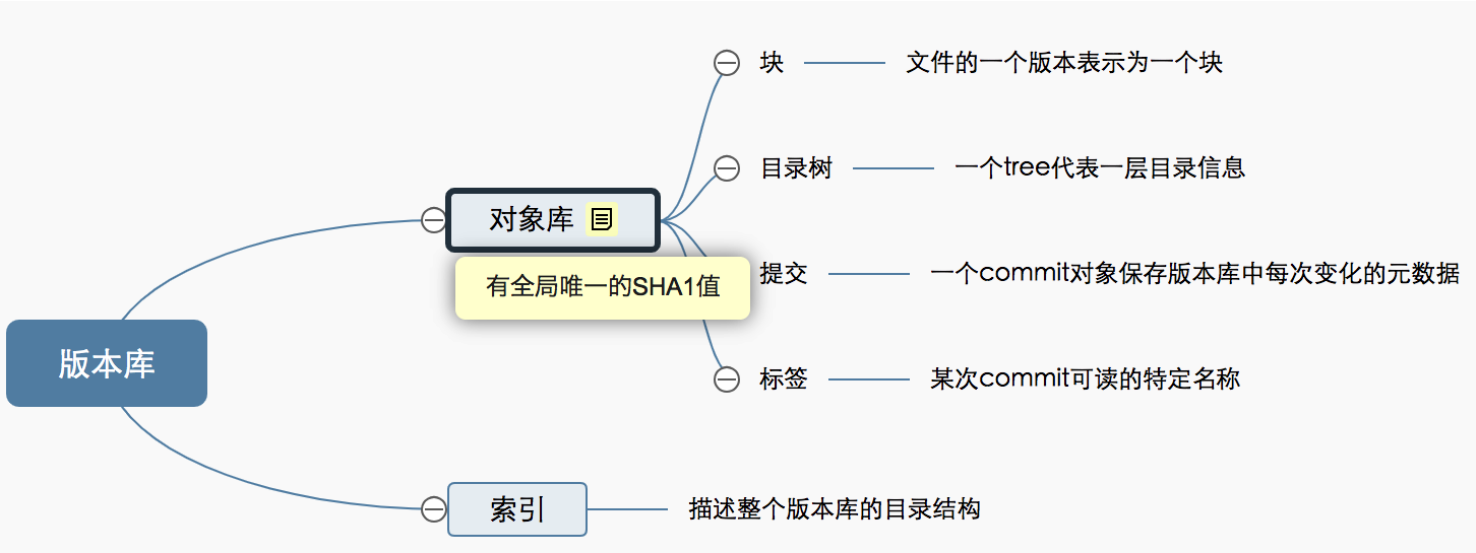
1. SVN :仓库位于中央服务器,必须联网才能提交, log在未联网情况下看不了
Git: 本地仓库, log和commit在本地实现

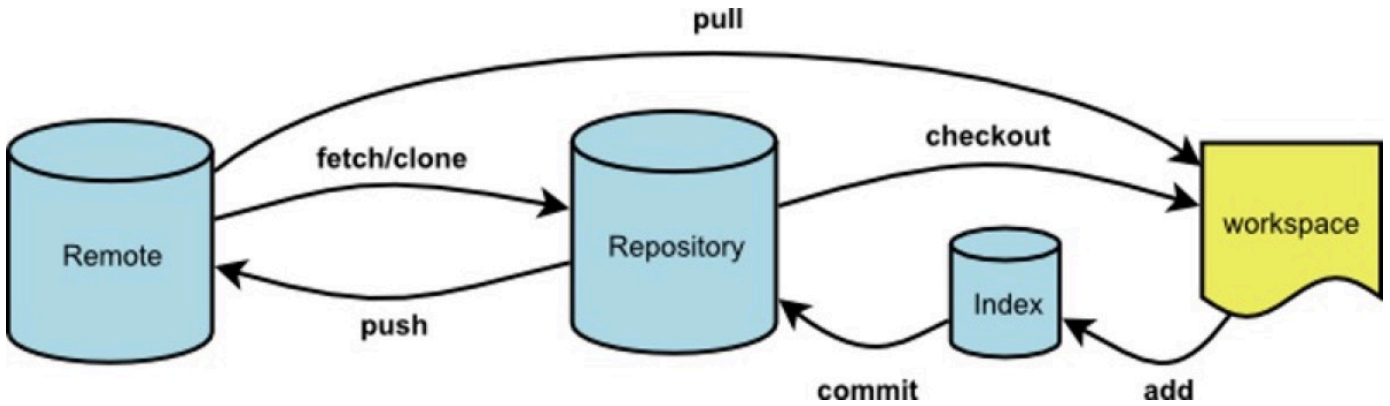
http://git.oschina.net/progit/1-%E8%B5%B7%E6%AD%A5.html#

2. Git 原子集变更,将暂存区的内容一起提交,要么都成功,要么都失败.两个提交快照之间的变更及合代表一个完整的树到树的转换

- 远程仓库&本地仓库

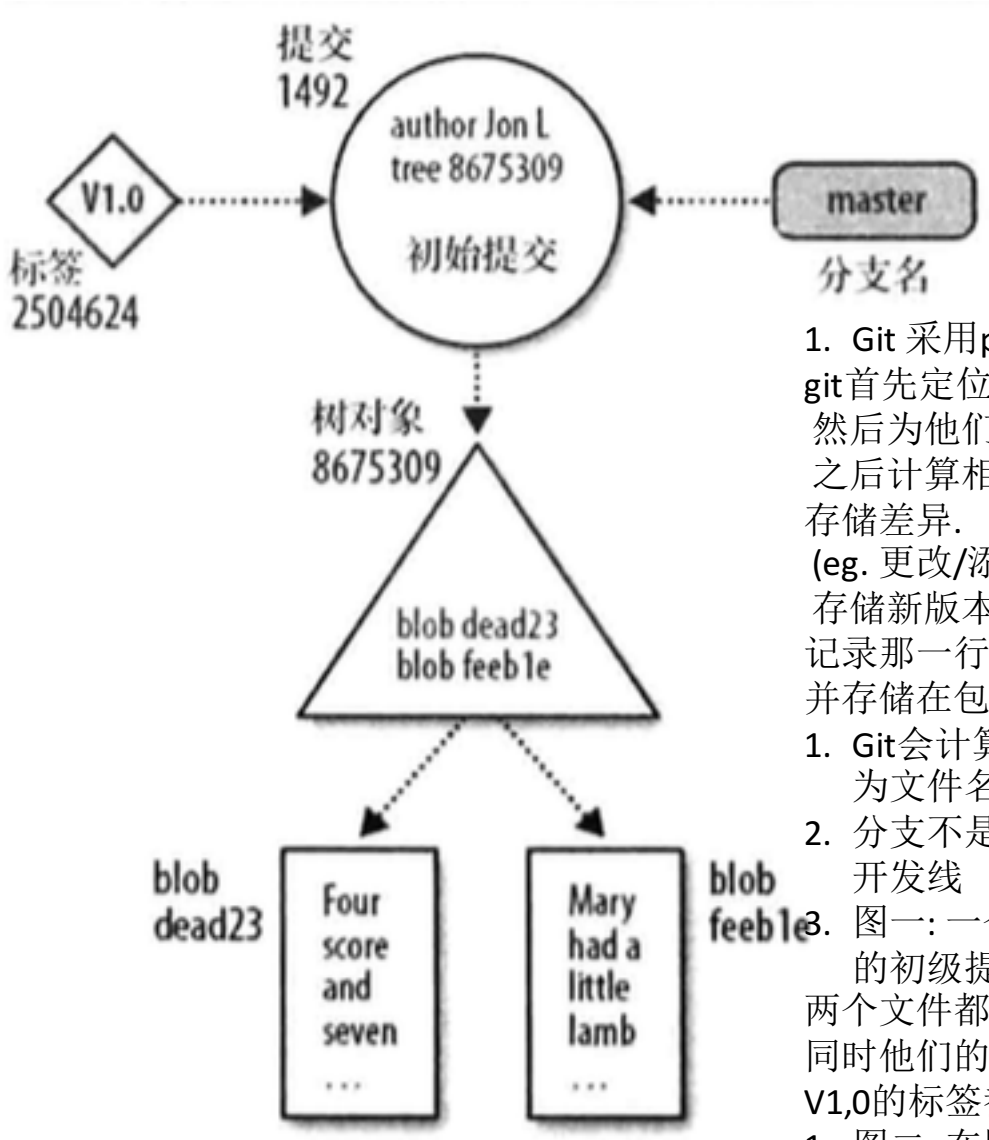






1.索引 – index – stage 暂存区：不包含任何文件内容, 仅仅追踪你想要提交的那些内容

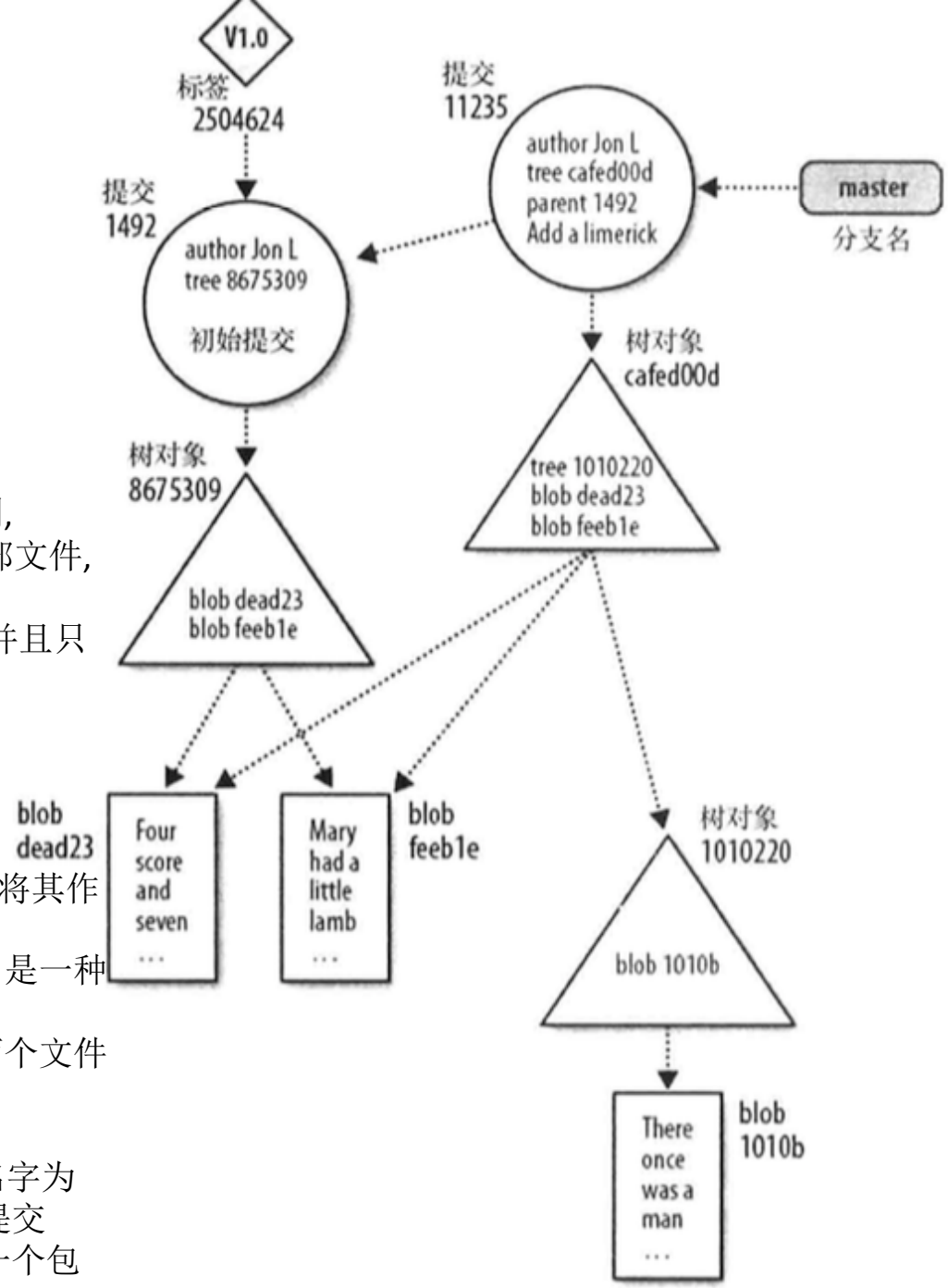
- Git对象



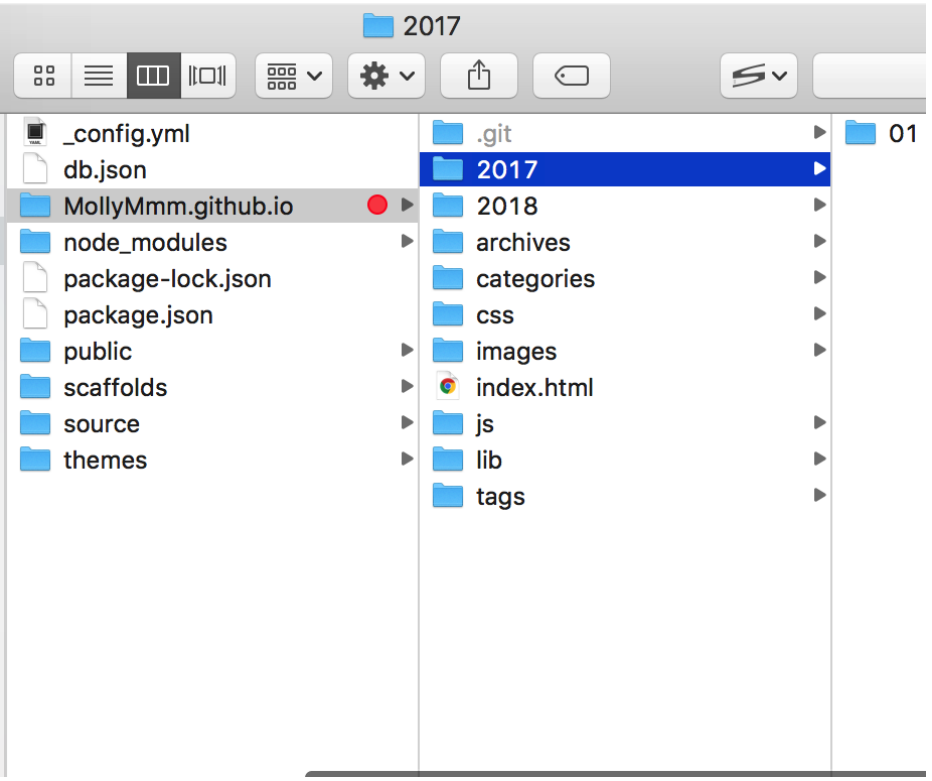
1. Git 采用packe file 的存储机制,
git首先定位内容非常相似的全部文件,
然后为他们之一存储整个内容,
之后计算相似文件之间的差异并且只存储差异.
(eg. 更改/添加文件中某某行,
存储新版本的全部内容,
记录那一行的更改作为差异,
并存储在包里)
1. Git会计算文件内容的sha1值,将其作为文件名放到对象库中
2. 分支不是一个基本的git对象, 是一种开发线
3. 图一: 一个版本库在添加了两个文件的初级提交状态,

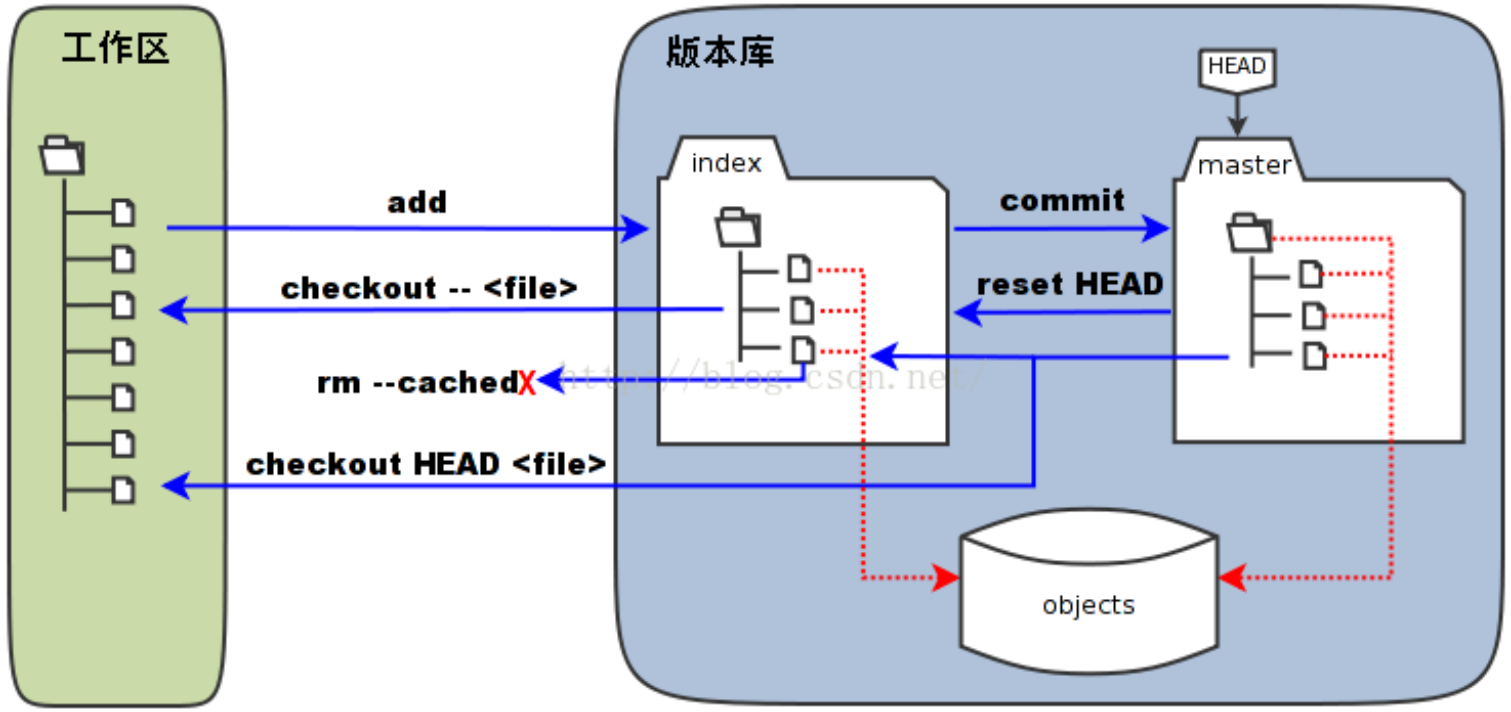
两个文件都在顶级目录中,
同时他们的master分支和一个名字为V1,0的标签都指向 ID为1492的提交

1. 图二: 在图一的基础上 添加一个包含一个文件的新子目录



一次 commit -> 40字符, 前几个可以代表他的唯一前缀, 参照sourcetree



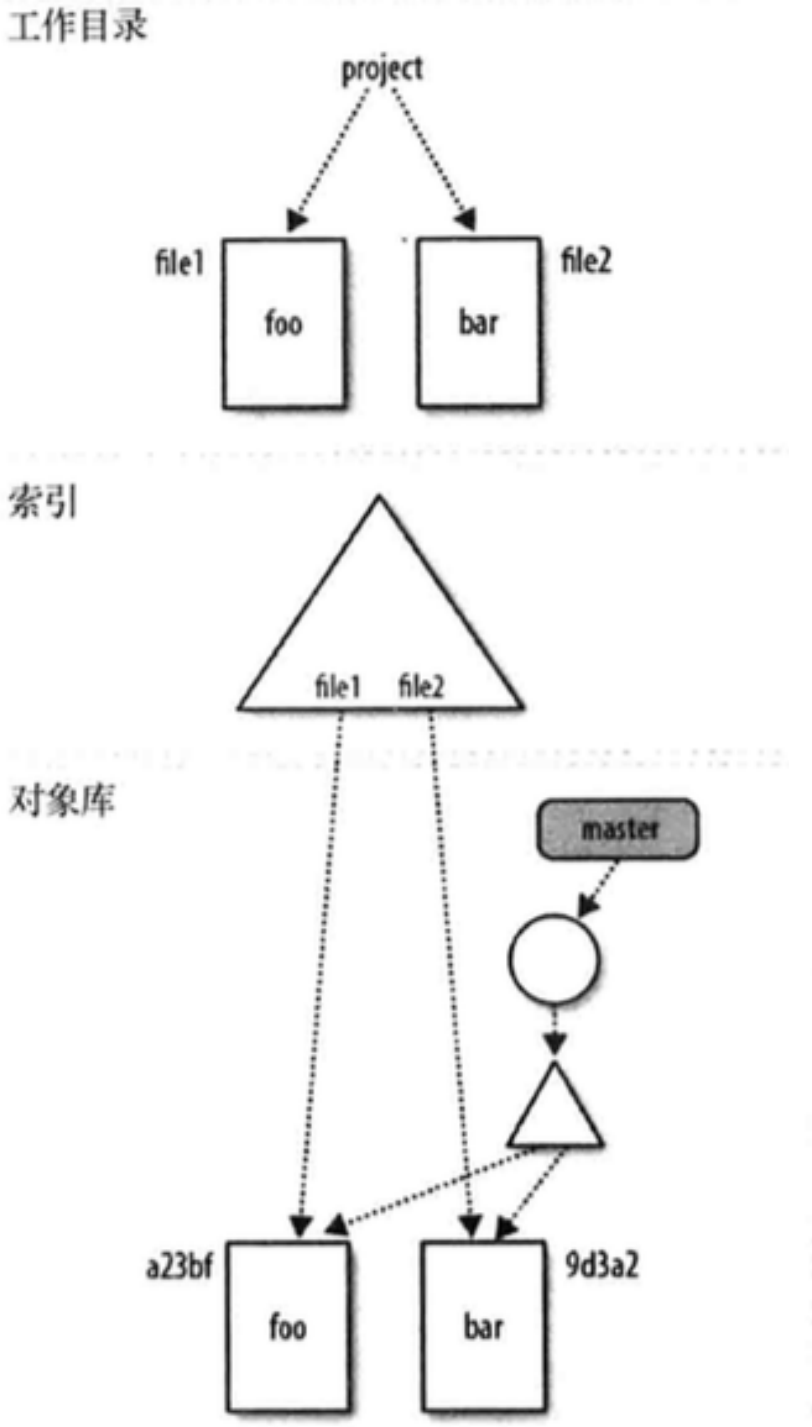


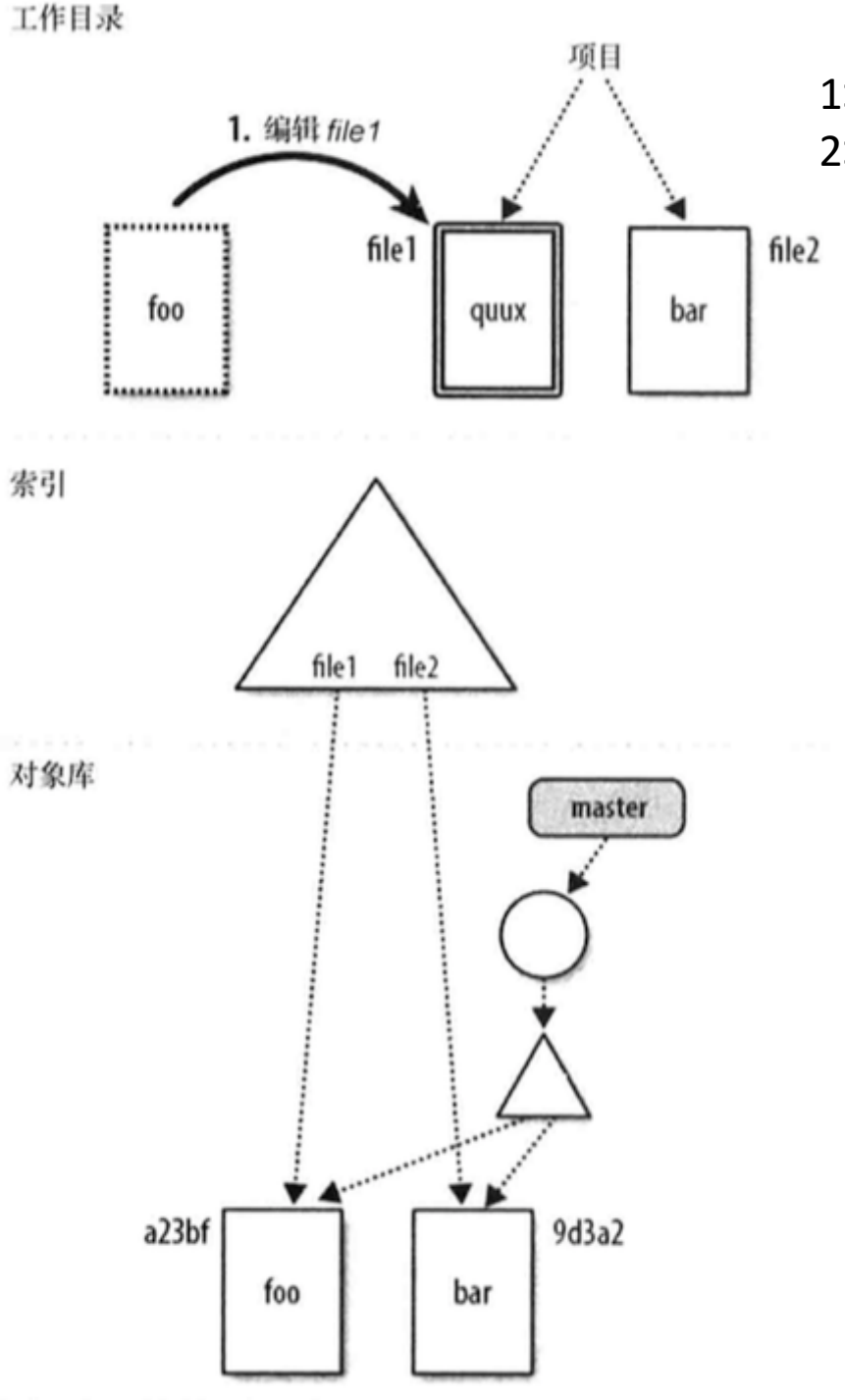
https://www.cnblogs.com/lianghe01/p/5846525.html

1. 本地普通文件, git init 初始化一个空的版本库, 生成 .git
2. .git 版本库
3. 索引 – index – stage 暂存区 : 不包含任何文件内容, 仅仅追踪你想要提交的那些内容

工作区和暂存区

https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/0013745374151782eb658c5a5ca454eaa451661275886c6000





1>.初始文件和对象
2>.编辑文件1的内容, 工作区是脏的

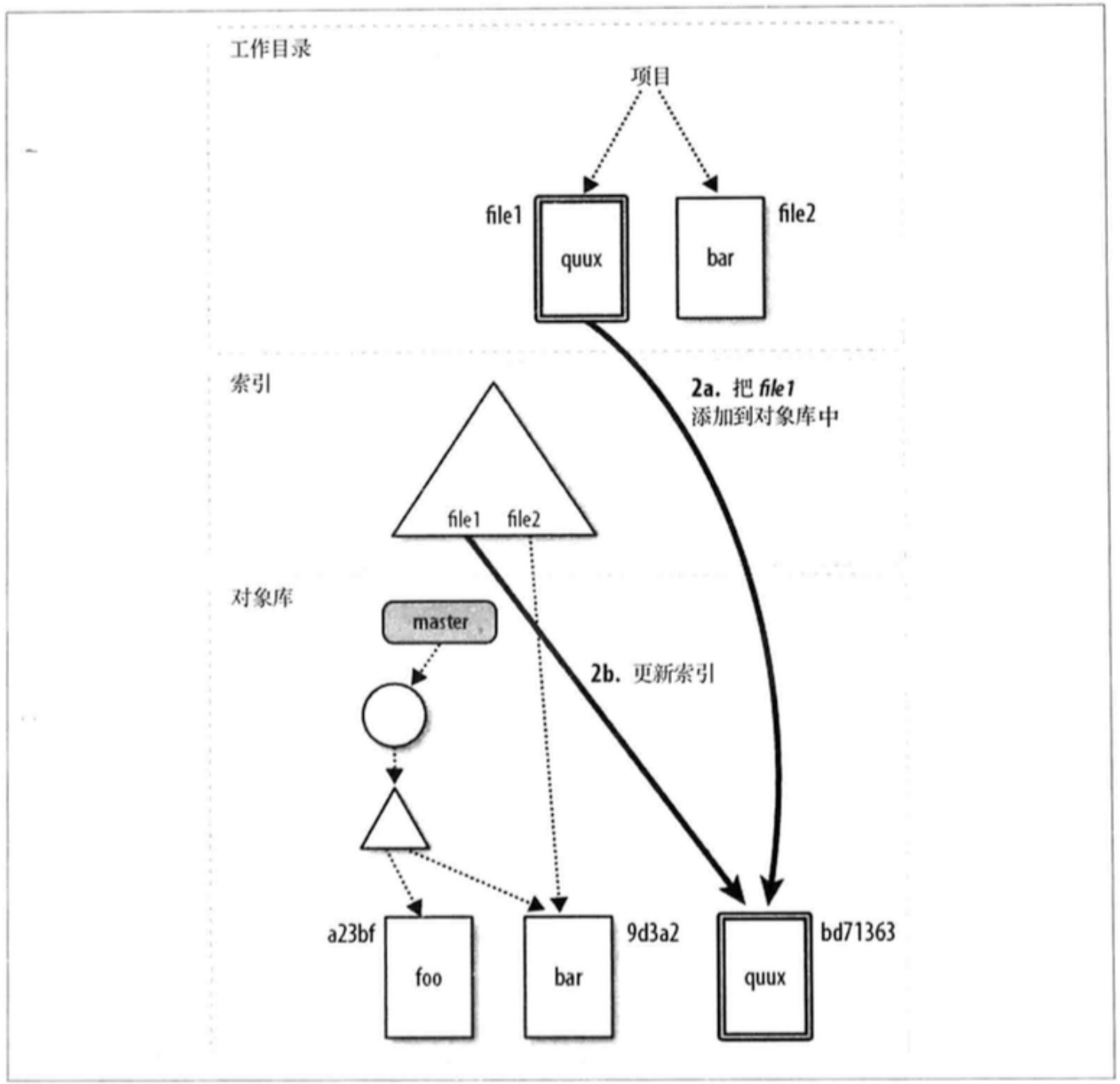


图 5-3 在 git add 之后

git add 为工作区中file1的内容计算sha1 ID (bd71363) , -> 把ID存入对象库 -> git 将索引中file的路径名更新为新的 sha1 ID (bd71363)
此时工作区 = 暂存区, 但是 就HEAD而言, 索引时脏的

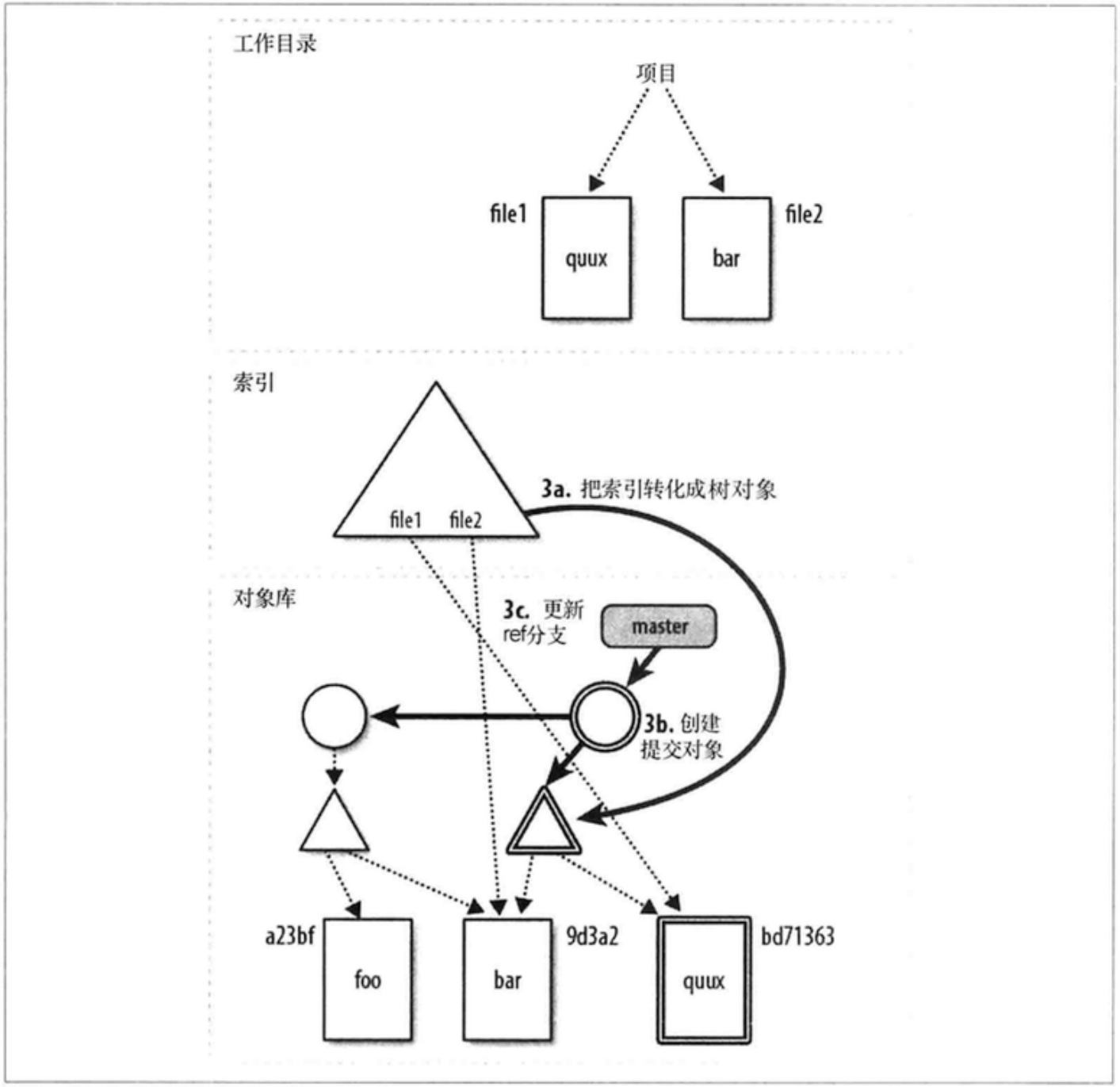
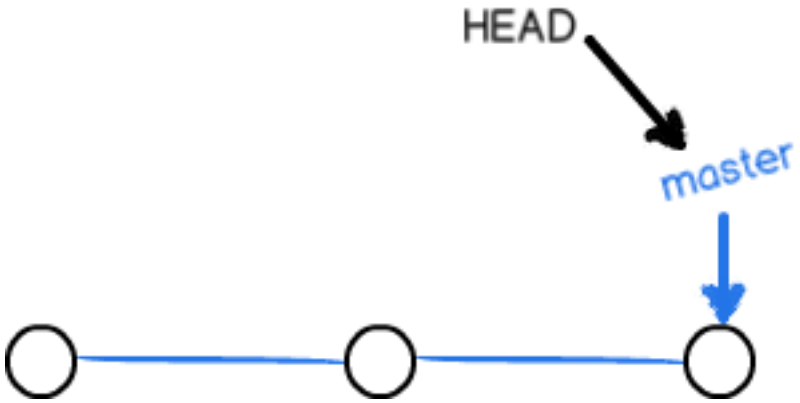
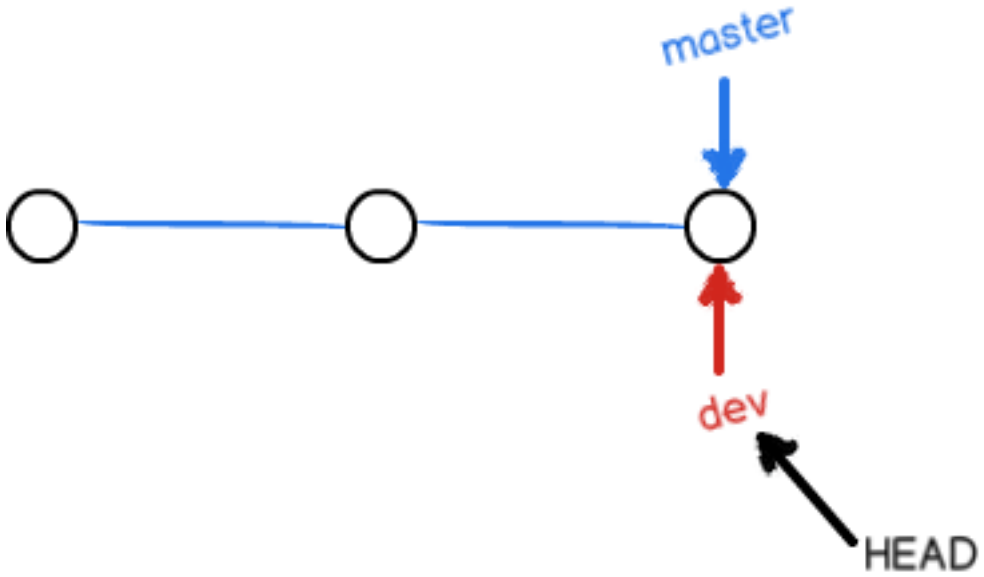


图 5-4　在 git commit 之后

git commit：索引转换成真正的树对象,生成一个sha1, 放入对象库中 -> 你的日志消息穿件一个新的提交对象, 新的提交指向新创建的树对象以及前一个或者父提交 -> master分支的引用从最近的提交移动到新创建的提交对象 , 成为新的master HEAD
此时, 工作区 = 暂存区 = 对象库一致, 同图一

- 分支





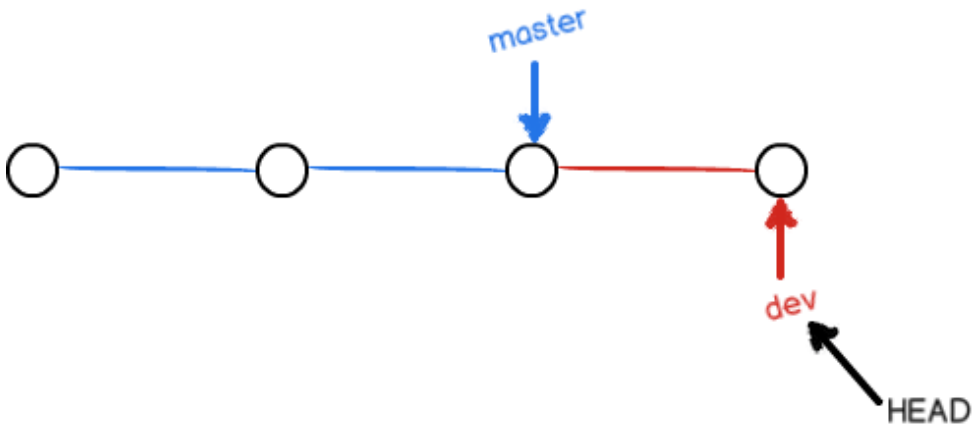
新建分支两种方式：

1、git checkout -b dev

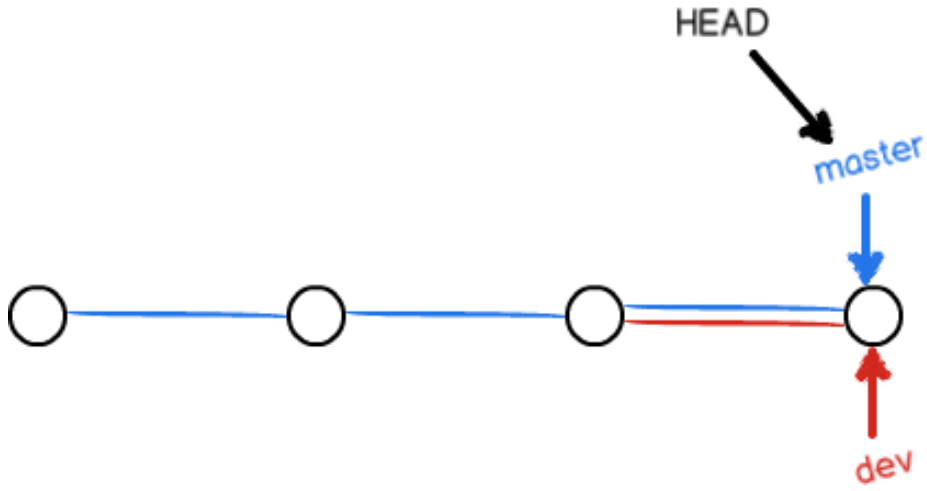
2、git branch dev

git checkout dev

- 分支



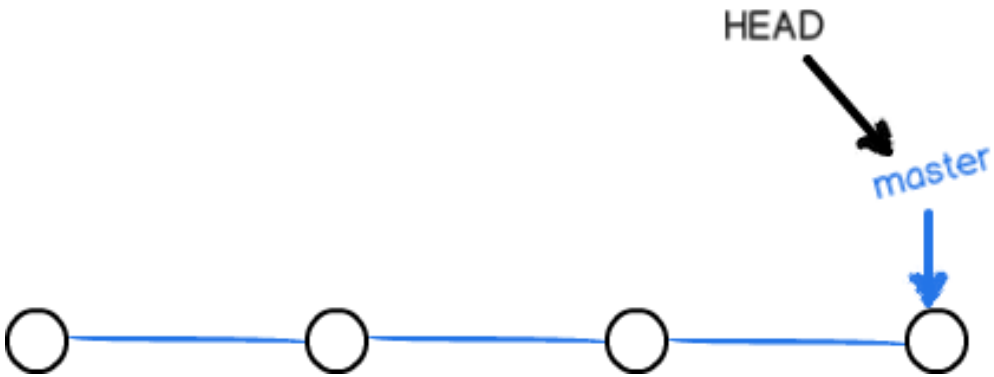
修改一部分代码
dev分支领先于master分支

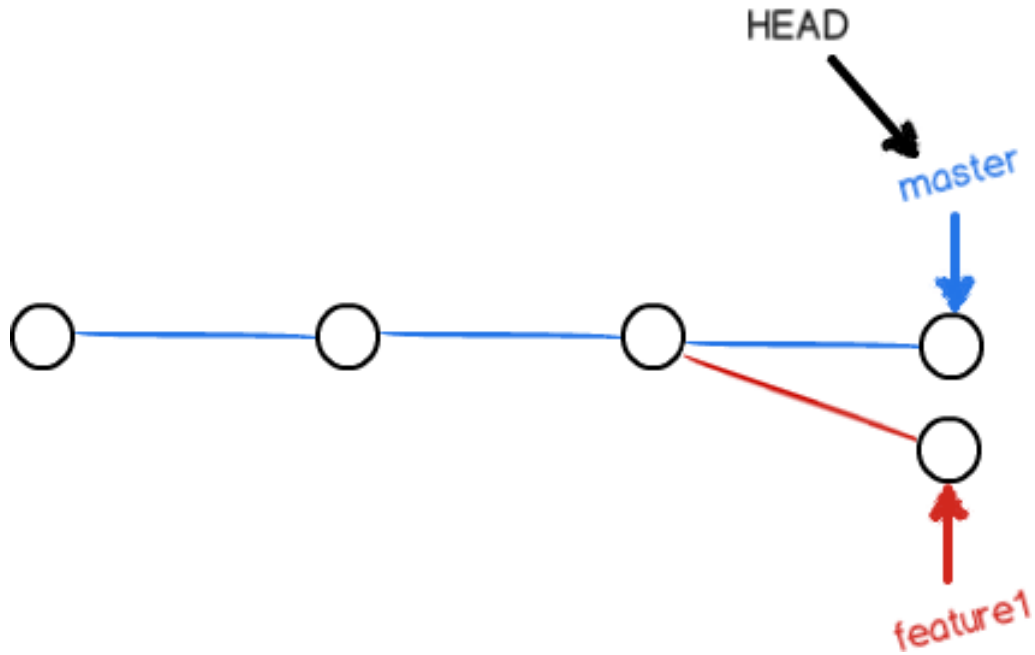


git checkout master
git merge dev

所以git合并分支也很快！就改改指针，工作区内容也不变！

- 分支





git branch -d dev
git branch -D dev

<<<<<<< HEAD
当前分支
=======
合并分支
>>>>>>> feature1

- merge

1.git diff

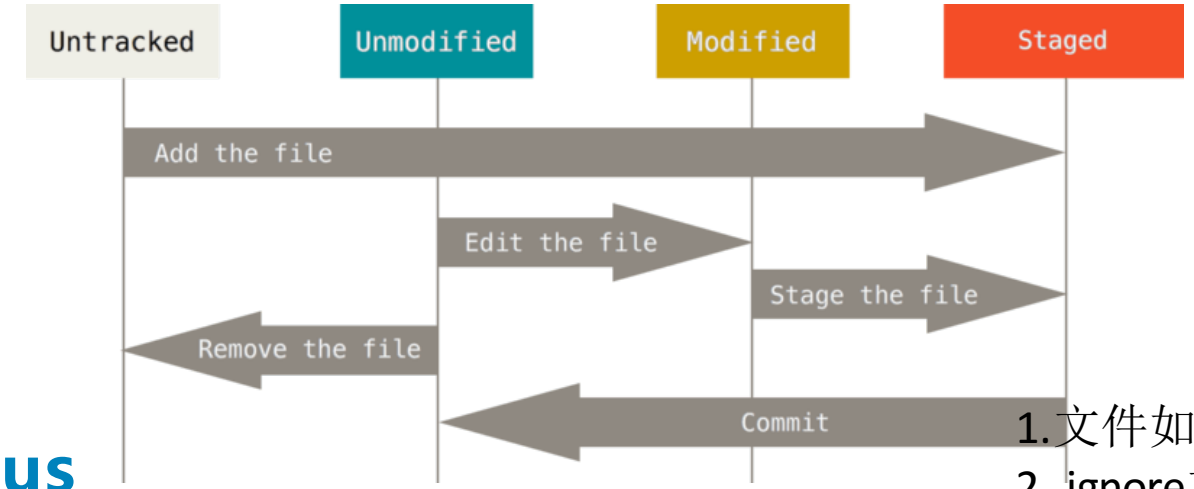
<<<<<<< HEAD
当前分支
=======
合并分支
>>>>>>> feature1

2. 冲突文件存储在工作区
解决冲突后, 需要重新添加到暂存区, 然后再提交

1. git pull = fetch + merge
拉取并合并

merge时, git会产生冲突, 需要人为解决

2. 直接 git merge 某个分支



git status

展示工作区及暂存区域中不同状态的文件

git add

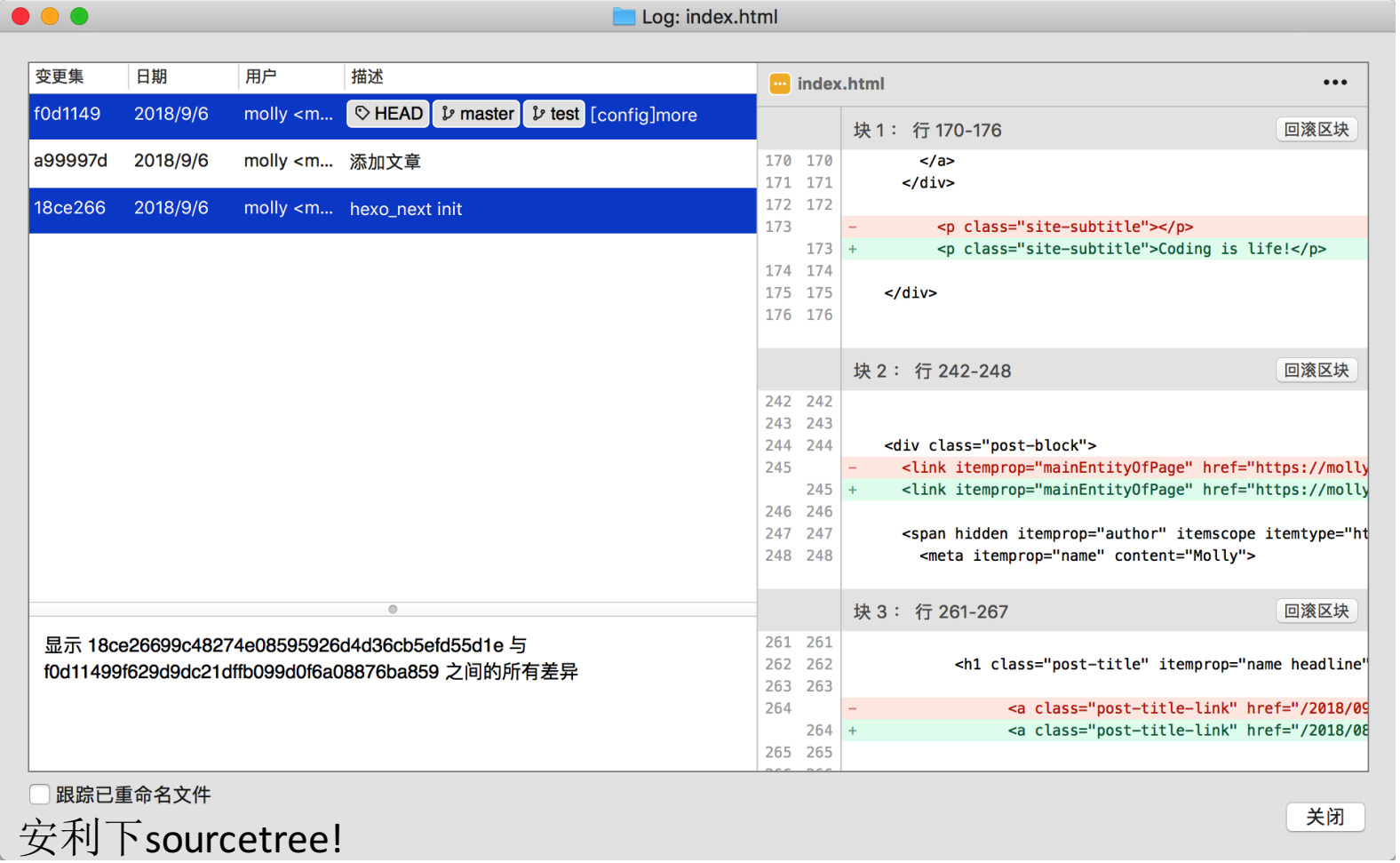
暂存文件，Untracked -> tracked
如果作用于目录，该目录下文件和子目录都会递归暂存起来
暂存一个文件 = 把文件放入索引

git rm

删除工作区和暂存区的文件
对没有添加到版本库或索引中的文件不起作用
git rm --cached 删除暂存区的文件,但工作区仍有

git pull / git push /git log
git show / git blame

1.文件如果添加到忽略中了，git add的时候不会加入到暂存区
2..ignore文件：被忽略的文件可能会在工作目录中出现，跟idea相关的，或个人配置 等;
.git 只在项目根目录中，.ignore文件可以出现在项目中任何文件夹内，只影响该目录及其所有子目录
.gitignor忽略规则
https://www.cnblogs.com/qwertWZ/archive/2013/03/26/2982231.html
3. 暂存 stage 又叫缓存 caching,
Git add 每个文件的全部内容被复制到对象库中，并且按文件的sha1来索引
为了方便 git add . , 将所有的变更/新增暂存起来
4. 工作目录中的文件版本和暂存中的文件版本可能是不同步的，提交的是索引中的文件版本
5. git rm -r - cached .
6. git reflog查看版本切换记录
7.提交后可以 去远程仓库确认



安利下sourcetree!

1.查看单个文件的修改历史

在左下的框框中选中需要diff的文件，呼出菜单，第一个选项叫做查看选中的修改日志，点击进入文件的修改详情列表然后可以看到对这个文件的修改记录，看起来就像是diff命令行的图形化版本。

2. diff神技

在上一个界面，Mac按住command键，再选择一行，直接显示出了这个文件的两次提交的diff，行数和修改都列的很清楚。

感觉还不够方便？

回到提交历史界面，按住command键选择两次提交，直接显示出这两次提交的所有文件的diff，速度相当快。

git diff 比较的是工作区和暂存区的差别

git diff --cached 比较的是暂存区和版本库的差别

git diff HEAD 可以查看工作区和版本库的差别

git diff 5d2b4d5 4dd0879 -- Controller/Web/StuGiftCard.php
对比某两次提交中某个文件的差异

https://www.jianshu.com/p/fe7d60be96d9

- 回退

每个提交都有唯一的commit id
回滚需要指定 HEAD 或者 commit id

- HEAD表示当前版本，上个版本是HEAD^，上上个版本是HEAD^^，上10个版本可以写成HEAD~10
- git checkout -- [file]撤销工作区文件修改，处理工作区修改
- git reset [file]重置暂存区的指定文件，处理git add的文件
- git reset --hard [commit]重置当前分支的HEAD为指定commit，同时重置暂存区和工作区，与指定commit一致
- 回退远程代码

1.> 将本地代码回滚到指定 commit ID

2>. 强制推到远程某分支

git push -f origin 分支名

表 10-1 git reset 选项影响

选　项	HEAD	索　引	工 作 目 录
--soft	是	否	否
--mixed	是	是	否
--hard	是	是	是

- 标签

标签是版本库的一个快照，就是指向某个commit的指针.

注意：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。

git tag v1.0 给当前版本库打标签

git tag v0.9 [commit] 给指定版本库打标签

git show v0.9 查看标签信息

git tag -a v0.1 -m ‘version 0.1 released’ [commit] 创建带有说明的标签

git push origin v1.0 推送标签至远程

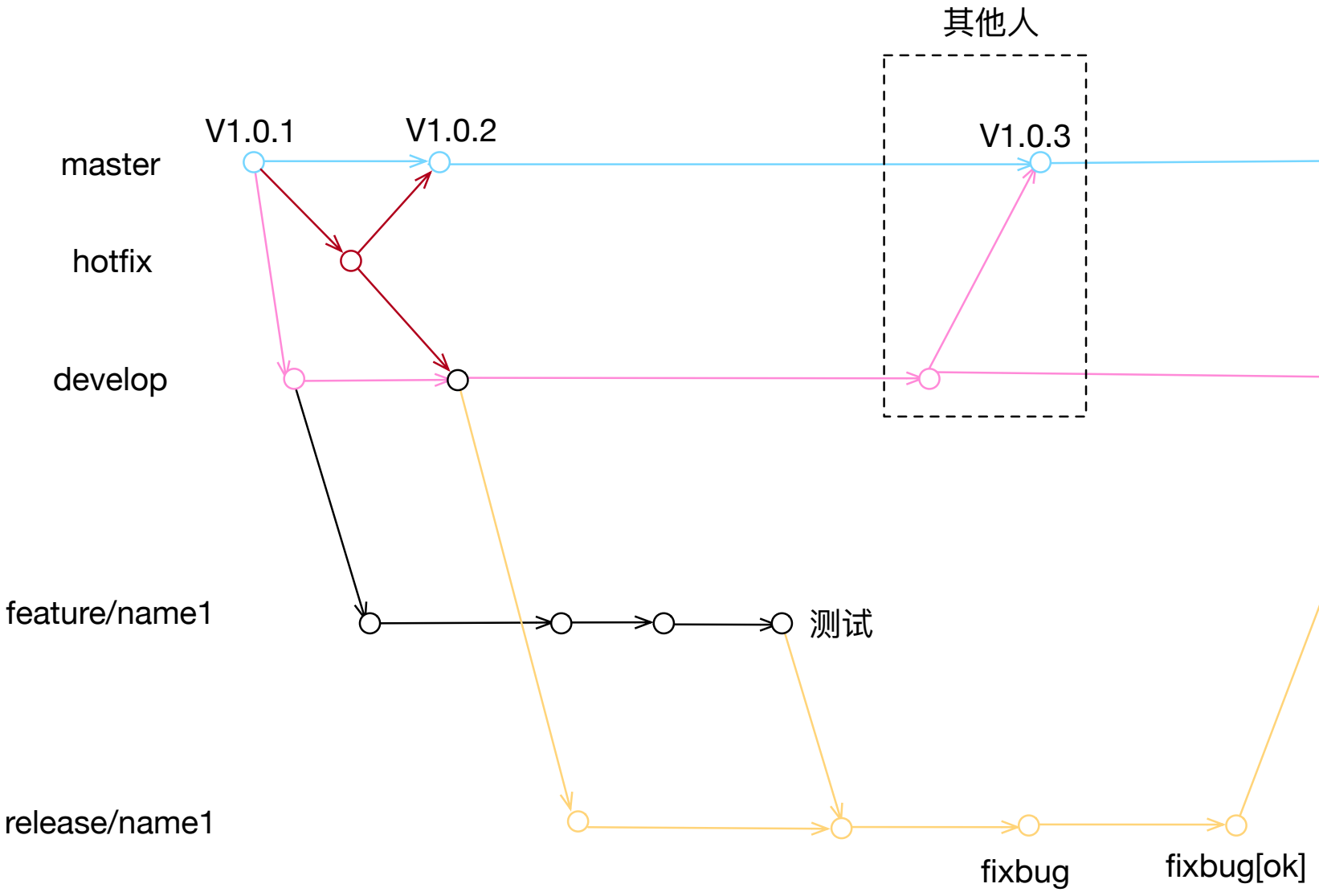
git push origin --tags 一次性推送全部尚未推送到远程的本地标签

git tag -d v0.9 删除标签

git push origin :refs/tags/v0.9 删除远程标签

git push origin v1.0 -> 推送分支到远程也是这个命令,
所以尽量不要将本地分支名与tag名起一样的

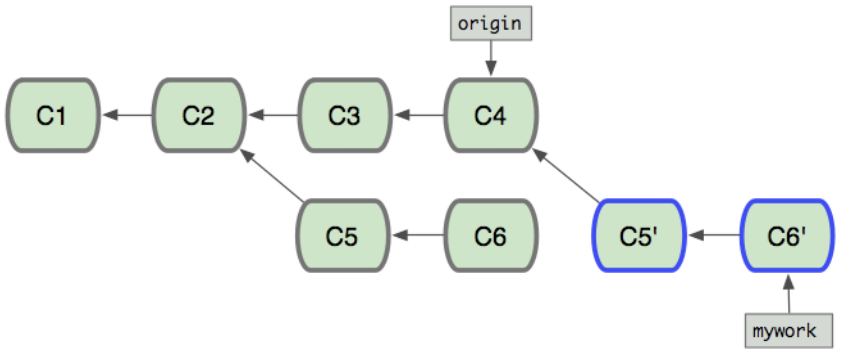
- 常规流程 - gitFlow



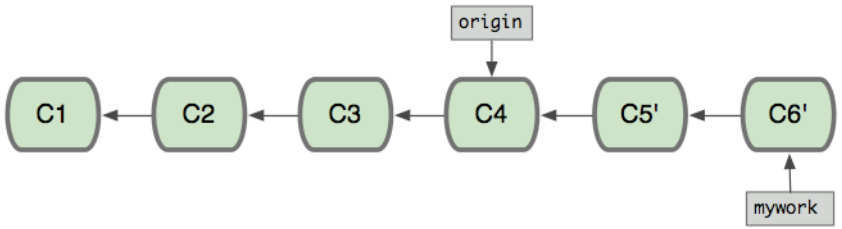
1.远程仓库中始终有master和develop分支, feature/name1 和 release上线后可删除
2.需要多人协作开发时, feature/name1可推送到远程；测试拉取release/name1分支进行测试
3.develop存在的意义:
Master不是所有人有用户合并分支的权限, 仅部分管理员拥有.
Eg: github上fork项目，你push了一些代码, 需要发起merge request才可以合并到其他人的开源库中.

- rebase

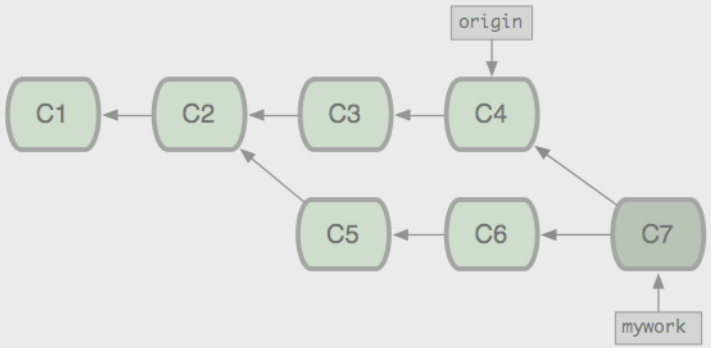
git rebase



git rebase



git merge



http://gitbook.liuhui998.com/4_2.html

1.Track 追踪分支

在Git中‘追踪分支’是用与联系本地分支和远程分支的. 如果你在’追踪分支'(Tracking Branches)上执行推送(push)或拉取(pull)时， 它会自动推送(push)或拉取(pull)到关联的远程分支上.
如果你经常要从远程仓库里拉取(pull)分支到本地,并且不想很麻烦的使用"git pull "这种格式; 那么就应当使用 ‘追踪分支'(Tracking Branches).
‘git clone‘命令会自动在本地建立一个’master’分支，它是‘origin/master’的 ‘追踪分支’. 而‘origin/master’就是被克隆(clone)仓库的‘master’分支.

git branch -vv查看分支与远程分支追踪状态

2.经常有这样的事情发生，当你正在进行项目中某一部分的工作，里面的东西处于一个比较杂乱的状态，而你想转到其他分支上进行一些工作。问题是，你不想提交进行了一半的工作，否则以后你无法回到这个工作点。解决这个问题的办法就是git stash命令。
“‘储藏”“可以获取你工作目录的中间状态——也就是你修改过的被追踪的文件和暂存的变更——并将它保存到一个未完结变更的堆栈中，随时可以重新应用。

- 小技巧

1、git push origin :molly/dev 删远程分支

git branch -r -d origin/molly/dev 删除本地对远程分支的track

2、git remote prune origin 更新远程分支本地track

3、git log --pretty=oneline log信息看着杂乱，可以配置单行显示

4、git reflog 查看版本足迹

5、git rm --cached [file] 移除文件，只从暂存区移除

6、git rm [file] 移除文件，工作区和暂存区都移除

7、git stash 暂存修改

git stash pop 恢复暂存并删除

git stash apply 恢复暂存

git stash drop 丢弃暂存

git stash list 查看所有暂存状态

git stash apply stash@{0} 恢复指定暂存

- 小技巧

8、.gitignore

git add -f App.class 强制添加被ignore忽略的文件

git check-ignore -v App.class 检查哪个规则忽略了某文件

9、git config --global alias.st status 配置别名 ~/.gitconfig

10、git config core.filemode false 忽略文件权限修改

11、git diff 快速预览差异

13、git branch|grep 'branchName'|xargs git branch -D 批量删除本地分支

12、origin/molly/module批量删除远程分支

git branch -r|awk -F '[/]' ’ /molly/ {printf "%s/%s\n",$2,$3}'|xargs -I {} git push origin :{}

13、git clone -b develop ssh://git@git.xesv5.com:10088/BBRD/wx5-web/trade.git 直接克隆分支

14、git push -f origin master 回退远程分支

15 、git commit --amend 1>.修改备注2>.修改提交

16 、从远程仓库拉取指定分支到本地 git checkout -b develop origin/develop ; git pull origin develop

Sourcetree + beyond compare

https://blog.csdn.net/sunnyCheng0121/article/details/78037936