# Git版本管理学习笔记

## 概述

版本控制是一种软件工程技巧，籍以在开发的过程中，确保由不同人所编辑的同一档案都得到更新。通俗地说，就是更好地管理在工作过程中逐步积累的文档、知识、经验等经常更新的东西，并方便多人之间的交流。常用的版本管理工具有RCS，CVS，GIT，Mercurial，Bazzar等，其中git，Mercurial和Bazzar属于分布式版本控制系统。分布式与集中式的比较在后文进行阐述。

## 参考内容

Git官方地址：<http://git.or.cz>

Git总结：<http://zh-tw.whygitisbetterthanx.com>

《看日记学git》系列

## 基本命令

对git的使用涉及到三类模式，独立开发者、合作开发者和仓库管理员。不论在哪种模式下，大部分git命令都需要掌握。这里首先给出git最基本的命令，疑难之处会给出相应的解释，然后针对不同模式下使用的特定指令进行介绍。整个过程中会附带介绍一些git原理性的东西，其实现细节在后面描述。

1. 配置个人信息

git config –global user.name “litao”

git config –global user.email [\*@qq.com](mailto:*@qq.com)

全局配置信息可在~/.gitconfig中查看，如果去掉—global，那么局部信息可在.git/config文件中查看；

系统全局配置信息在/etc/gitconfig中；

2. 初始化仓库，记录和提交

这里首先要明确git中以下三者之间的关系：working tree，index file和git repository。工作树是当前用户的工作目录中的所有内容；index file是git维护的一个对当前工作树的一个快照，在git仓库和工作树之间作为桥梁。

git init，在当前目录下建立.git目录，内容放在objects中；

git add，在objects下新建git对应的对象，同时更新index file；

git commit，将在index file中注册的对象提交到git repository；

至于各命令的选项，可通过man来查阅，在此不述；

3. 查看状态

git diff，git diff –cached，git diff HEAD，分别用于查看以上三者之间的差别；

git status，查看当前已修改需要add和commit的文件；

git log [-p]，查看提交历史；

这几个指令需要常用，以查看项目当前的状态，哪些已被修改，哪些需要提交，修改的地方都在哪里等；

4. 撤销操作

git checkout -- <file>，放弃对working tree中某个文件（未add）的修改；

git reset -- <file>，从index file中删除注册记录，也就是add记录；

git reset –soft <commit>，只撤销commit，保留index file和working tree；

git reset –hard <commit>，撤销所有；

git reset –mixed <commit>，只保留working tree；

git reset 和 git reset –mixed一样；注意，这里是撤销到某一次提交；

5. 分支管理

git branch [options]，根据commit来建立分支，显示，删除分支；

git checkout，切换分支，未提交状态不能切换；

git merge，合并（已提交）分支，有冲突的文件需手动改，未发生冲突的文件则被登记在index file里；

代码和log信息都是向下继承的，但不会自动向上复制，只有merge之后才能看到下层分支的代码和log信息；

6. 合作开发

git clone，git clone /home/fffu/gitexp /home/shark/exp，处于哪个分支，就clone那个分支对应的最新commit的内容；

git pull，从远端取出内容，合并到当前分支；git pull .相当于git merge，具体的参数格式和默认参数参考man page；

git fetch，git fetch /home/fffu/gitexp master:bob，取他的master分支到我的bob分支下；然后通过git whatchanged –p master..bob，看一下我的master和bob分支的区别；

7. git对象查看

Git中总共有四类对象，分别为blob，tree，commit和tag；其中commit对象对应于每一次提交，这个对象指向一个tree对象；而tree对象中一般对应文件blob对象或者是另外的tree对象；blob对象对应具体的文件内容；tag是给commit对象取的别名；通过这四类对象，就可以建立起整个git系统，所有指令的操作实际上就是创建或者删除这些对象；在.git目录下的objects目录中存放着这些对象，这才是git维护的数据，其他的内容都是反应这些对象之间的关系，以及建立git的功能而实现的；

git show，显示commit的内容；git show HEAD；

git tag，给commit取一个别名；git tag V1 7944d；

git grep，git grep “print” V1，在V1中搜索所有包含print的行；

git cat-file，显示git对象内容；git cat-file commit V1；

git ls-tree，显示tree对象的内容；

git ls-files [--stage]，显示index file的内容；

git show-branch，显示branch的结构，相当于gitk中给出的那个分支图；

git whatchanged [-p]，查看分支之间的区别；git whatchanged –p master..bob；

## 常用文件

HEAD，当前分支的最近一次提交；

ORIG\_HEAD，意思还不明确；

[ORIG\_HEAD](https://git-scm.com/docs/gitrevisions) is previous state of HEAD, set by commands that have possibly dangerous behavior, to be easy to revert them. It is less useful now that Git has reflog: HEAD@{1} is roughly equivalent to ORIG\_HEAD (HEAD@{1} is always last value of HEAD, ORIG\_HEAD is last value of HEAD before dangerous operation).

在执行危险操作时，保存HEAD的值；

HEAD@{1}不管在什么时候都可以使用，表示当前HEAD的之后的提交；前面给出了用HEAD得出之前提交的方法（HEAD^）；

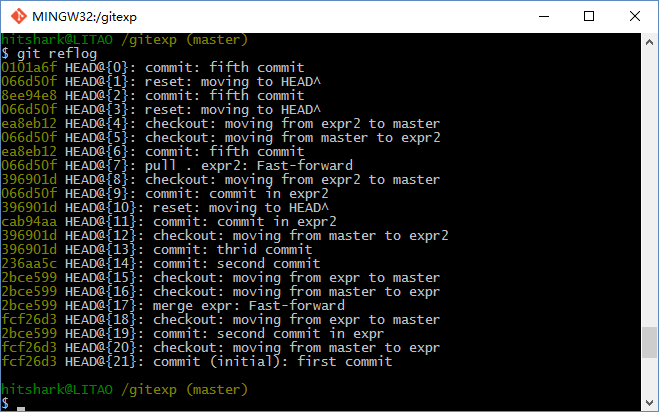
git reset --hard ORIG\_HEAD

Resetting hard to it brings your index file and the working tree back to that state, and resets the tip of the branch to that commit.

git reset --merge ORIG\_HEAD

After inspecting the result of the merge, you may find that the change in the other branch is unsatisfactory. Running "git reset --hard ORIG\_HEAD" will let you go back to where you were, but it will discard your local changes, which you do not want. "git reset --merge" keeps your local changes.

Git在1.8.5版本之后，加入了HEAD@{}功能，它通过一个链表记录HEAD的移动路径。输入$ git reflog，结果类似于：



git help revisions brings up <http://git-scm.com/docs/gitrevisions>, which describes  the most common ways to reference commits:

* HEAD names the commit on which you based the changes in the working tree.
* FETCH\_HEAD records the branch which you fetched from a remote repository with your last git fetch invocation.
* ORIG\_HEAD is created by commands that move your HEAD in a drastic way, to record the position of the HEAD before their operation, so that you can easily change the tip of the branch back to the state before you ran them.
* MERGE\_HEAD records the commit(s) which you are merging into your branch when you run git merge.
* CHERRY\_PICK\_HEAD records the commit which you are cherry-picking when you run git cherry-pick.

From the git source, you can also find out about BISECT\_HEAD, REVERT\_HEAD, REJECT\_NON\_FF\_HEADand several others that you will almost certainly never need.

That reference also explains suffixes (^N, ~N, @{...}), ranges (.. vs ...), and more.

Git,远端仓库的管理：（待补充）

Git仓库常见的使用规则：（待补充）

看log文件：

Git log –all –decorate --graph

本地分支和远端分支的关联：

第一种情况：别人已建好分支，在远端：git checkout –track origin/dev

第二种情况：自己建的分支，第一次发布到远端：

git push –u origin dev

第三种情况：在用的过程中，git branch –u origin/dev