# 命令

### 1: git add （工作区-->缓存区） git add [参数] <路径>

 \*将当前目录下修改的所有代码从工作区添加到暂存区

. 代表当前目录

[参数]和 <路径>之间加空格，但是如果是.可以省略空格

git add . 不加参数:修改、添加（不包括删除）

不加参数默认为将修改操作的文件和未跟踪新添加的文件添加到git系统的暂存区，注意不包括删除

git add -u. u:已跟踪的修改删除

表示将已跟踪文件中的修改和删除的文件添加到暂存区，不包括新增加的文件

git add -A. A:所有改动

表示将所有的已跟踪的文件的修改与删除和新增的未跟踪的文件都添加到暂存区

### 2: git commit （缓存区/工作区-->本地） git commit [参数] [描述]

 \*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(commit-id)

commit-id 主要作用 版本回退 相当于一个快照

git commit -m ‘message’ m:后面可以直接加上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

git commit -am ‘message’ am:已跟踪的工作区-/缓存区>本地

-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。

总结:添加文件：1:git add 2:git commit

修改/删除： git commit -am ‘message’

都有： 1:git add -A <路径> 2:git commit -m ‘message’

### 3: git push （本地-->远程）

### git push <远程主机名> <本地分支名>:<远程分支名>

 \*origin代表远程主机

git push origin master:dev

本地的master分支推送到origin主机的dev分支

\*推送并不代表建立了追踪关系

git push origin master 省略远程

如果省略远程分支名，则表示将本地分支推送与之同名的分支，如果不存在，新键远程同名分支

git push origin :master 省略本地

删除指定的远程分支master；推送一个空分支到远程，等同于 git push origin --delete master

git push origin 省略本地和远程

将当前分支推送到origin主机的对应追踪分支

git push 省略本地分支和远程分支和主机

只追踪了一个远程分支时

git push -u origin master 设置默认远程主机

上面命令将本地的master分支推送到origin主机，同时指定origin为默认主机，后面就可以不加任何参数使用git push了

# 回滚命令

### 一:未使用 git add 缓存代码时:

**git checkout -- filepathname**// 放弃所有的文件修改

git checkout .

此命令用来放弃掉所有还没有加入到缓存区（就是 git add 命令）的修改：内容修改与整个文件删除。但是此命令不会删除掉刚新建的文件。因为刚新建的文件还没已有加入到 git 的管理系统中。所以对于git是未知的。自己手动删除就好了。

### 二:已经使用了 git add 缓存了代码:

**git reset HEAD filepathname** //清除缓存

此命令用来清除 git 对于文件修改的缓存。相当于撤销 git add 命令所在的工作。在使用本命令后，本地的修改并不会消失。

### 三:已经用 git commit 提交了代码:

**git reset --hard HEAD^**  //回滚提交版本

参数：

--hard(版本回滚，直接清除本次修改)

--soft(版本回滚，保留更改到缓存区，即修改保留，本地和缓存区一致git add后)

--mixed(版本回滚，只保留工作区修改，缓存区和本地库回到上次git add前)

**回退到上一次commit的状态。  
此命令可以用来回退到任意版本：**git reset --hard commitid \*\*

· --hard：(1) 要放弃目前本地的所有改变時，即去掉所有add到暂存区的文件和工作区的文件，可以执行 git reset -hard HEAD 来强制恢复git管理的文件夹的內容及状态；(2) 真的想抛弃目标节点后的所有commit（可能觉得目标节点到原节点之间的commit提交都是错了，之前所有的commit有问题）。

· · --soft：原节点和reset节点之间的【差异变更集】会放入index暂存区中(Staged files)，所以假如我们之前工作目录没有改过任何文件，也没add到暂存区，那么使用reset --soft后，我们可以直接执行 git commit 將 index暂存区中的內容提交至 repository 中。为什么要这样呢？这样做的使用场景是：假如我们想合并「当前节点」与「reset目标节点」之间不具太大意义的 commit 记录(可能是阶段性地频繁提交,就是开发一个功能的时候，改或者增加一个文件的时候就commit，这样做导致一个完整的功能可能会好多个commit点，这时假如你需要把这些commit整合成一个commit的时候)時，可以考虑使用reset --soft来让 commit 演进线图较为清晰。总而言之，可以使用--soft合并commit节点。

· · --mixed（默认）：(1)使用完reset --mixed后，我們可以直接执行 git add 将這些改变果的文件內容加入 index 暂存区中，再执行 git commit 将 Index暂存区 中的內容提交至Repository中，这样一样可以达到合并commit节点的效果（与上面--soft合并commit节点差不多，只是多了git add添加到暂存区的操作）；(2)移除所有Index暂存区中准备要提交的文件(Staged files)，我们可以执行 git reset HEAD 来 Unstage 所有已列入 Index暂存区 的待提交的文件。(有时候发现add错文件到暂存区，就可以使用命令)。(3)commit提交某些错误代码，或者没有必要的文件也被commit上去，不想再修改错误再commit（因为会留下一个错误commit点），可以回退到正确的commit点上，然后所有原节点和reset节点之间差异会返回工作目录，假如有个没必要的文件的话就可以直接删除了，再commit上去就OK了。

1:commit版本是全局(一个git项目)的不分目录，git log在任何一个目录查看都是一样的

2:所以回滚，不管在哪个目录下都是全局回滚。

3:git只能回滚不能到未来

# 跟踪分支（追踪）

同名分支：

### 1：git checkout --track orgin/branch\_name

在本地创建一个与 branch\_name 同名分支跟踪远程分支

### 2：git push --set-upstream origin branch\_name

远程创建一个与本地branch\_name 分支同名的分支并跟踪

**3:git clone**

默认track 远程master; git clone -b branch\_name 默认track branch\_name

不同名分支：

### 4：git checkout -b new\_branch\_name orgin/branch\_name

这条指令本来是根据一个 branch\_name 分支分出一个本地分支 new\_branch\_name，但是如果所根据的分支 branch\_name 是一个远程分支名，那么本地的分支会自动的 track 远程分支。建议跟踪分支和被跟踪远程分支同名

1: git push origin branch\_name :branch\_name 不是追踪，git pull 也不能

2: 如果远程新键分支，本地要先**git fetch**才能看到/追踪到远程分支

# Merge与rebase(变基)

<https://www.jianshu.com/p/6960811ac89c>

注意：

Merge是往谁身上合并就先切换到谁身上 git merge dev （从dev合并到master）

Rebase是变谁的基，操作谁 git rebase master (dev的基点换成master最新一次提交)

