# Git之美术操作简书 v1.1

## ▲删除缓存区add的反向操作(restore\reset)

git reset HEAD <files> #可以使用通配符“\*”

git restore --staged <file>… #删除缓存区

git reset HEAD \*Pro\* #unstage包含“Pro”的文件

git reset --keep #清空缓存区

## ▲撤销工作目录修改(restore)

git restore <file>… #回到最后一次add的版本，如果缓存区中没有则回到最后一次commit的版本#

## 停止跟踪文件(rm --cached)

git rm --cached <file> #停止跟踪但不删除文件

之后直接commit提交文件既不跟踪状态，这之前不能add

## ▲查看操作历史(reflog)

git reflog

可以用git show HEAD@{5}通过指针号查看提交记录

可以使用git reset --hard <SHA-1>还原历史到指定提交对象（注意当前工作目录的备份）

## ▲查看commit历史(log --pretty)

git log --pretty=oneline [<branch>|留空-当前分支]

返回：SHA-1 commits

## ▲查看提交的文件(show --raw、log --stat)

查看指定提交的文件列表：

git show --raw <SHA1> #也可指定显示条数-2

git show --raw #省略后面id查看最后一次提交

查看最近3条更新日志，简单显示涉及文件：

git log -3 –stat#列出文件，显示修改添加删除文件数

git log --name-only -1 #查看最近1条提交的文件

## 查看HEAD等快照目录列表(cat-file)

git cat-file（底层命令）

|  |  |
| --- | --- |
| -t | 显示对象类型 |
| -s | 显示对象大小 |
| -e | 如对象存在且有效则不输出，否则返回0 |
| -p | 根据对象类型，显示对象内容，包括文件内容 |

git cat-file -p HEAD

## ▲查看各分支指向的对象(--decorate)

git log --oneline --decorate

输出提交历史、各分支指向及项目的分支分叉情况：

git log --oneline --decorate --graph --all

## 查看单词层面的对比(log --word-diff)

git log -U<num> --word-diff #-U后设置显示行数，默认3行

我们常用 ​​-p​​​ 选项展开显示每次提交的内容差异，用 ​​-2​​ 则仅显示最近的两次更新

git log -p --word-diff -2

## ▲查看远程仓库的名称与连接地址(remote)

可以添加多个远程连接地址，指定分支与远端的跟踪。

git remote -v

添加完仓库连接后，可以将本地分支与远程仓库建立链接，之后就可以拉取推送

git branch --set-upstream-to=origin/master master

## 更新远程仓库分支列表(remote)

查看远程分支列表发现远端不符时，可以执行如下命令：

git remote update <origin> --prune

## ▲修改最后一次提交--amend

git commit --amend -m 'new-message'

**有文件需要修改或添加：**

修改添加文件并git add添加到缓存区后，然后再执行命令git commit –amend修改提交信息。

## ▲重置指定commit历史到工作目录(reset)

git reset --hard [<SHA-1>|留空提取最新本地提交]

## ▲恢复本地库记录到缓存区(reset)

git reset --soft HARD^ #恢复最近一次commit内容到缓存区（便于重新修改提交）

## ▲干净还原远程版本分支3种方法

1：

git fetch –all #拉取远程到本地库

git reset --hard origin/<branch>

2：

git reset --hard #重置到最顶历史

git pull origin <branch> #拉取远程分支

3：或只需执行一条，直接重置到远程分支：

git reset --hard origin/<branch>

## ▲还原文件为本地或远程(checkout)

在使用cherry-pick解决冲突时会用到

git checkout –3 <file> #还原到远程版本--theirs

git checkout –2 <file> #还原到本地库版本--ours

git checkout <SHA-1> <files> #还原某提交的文件

可以先查看提中包含的文件，

git show --raw <SHA-1> #查看指定提交中的文件

git log --name-only #列出所有提交中的文件

再提取想要的文件：

git checkout -- <file> #还原到缓存区的版本

git checkout <SHA-1> <files> #还原某提交的文件

## ▲查看缓存区中的文件内容(checkout --)

git checkout -- #列出缓存区中的文件

## ▲拉取某分支的某文件(checkout)

可以是本地分支，也可以是远程分支origin/<branch>

git checkout <branch> <file> <…>

## ▲拉取远程分支在本地创建新跟踪分支(ck)

git checkout -b <new-branch> origin/<branch>

git checkout **--orphan** <new-branch> #基于当前分支创建一个干净的没有提交历史的新分支

## 查看两分支间未合并的提交(log)

查看<分支B>中有哪些提交未合并到<分支A>

git log <分支A>..<分支B> --oneline

## 列出所有分支引用的SHA1(for-each-ref)

git for-each-ref

## ▲清理工作目录(clean)

清除未跟踪(Untracked)的文件，直接删除不可还原：

git clean -d #移除未跟踪及空子目录-f强制-n演练

储藏到缓存区stash再清理：

git stash -u #暂存包含未跟踪的文件

## 重命名本地分支(branch -m)

git branch -m <old-name> <new-name>

## 在某个提交基础上创建新分支<branch>

git update-ref refs/heads/<branch> <SHA-1>

git log –pretty=oneline <branch> #查看新分支

## 删除本地、远程分支(branch)

git branch -d <branch> #删除本地分支,强制用-D

git push origin –delete <branch> #删除远程

## 退出变基过程操作 (master|REBASE 1/9)

git rebase --abort

## Git用户名和邮箱(config)

查看

git config user.name

git config user.email

设置

git config --global user.name 'username'

## ！有时查看不到远程部分分支

解决办法：

git fetch

git branch -a #这时应该就能看到所有分支

## 查看所有现有配置设置：

git config –list

git config --list --show-origin #查看config目录

## ！出现Unlink of file ’..’

文件被其他程序暂用Git无法修改，可以关掉暂用文件的相关程序，或重启电脑。

## 设置快捷命令

设置常用命令的简化自定义替代字符

git **dec** #查看各分支指向的对象--decorate

git config --global alias.dec 'log --oneline --decorate'

git **st** #查看当前状态status

git config --global alias.st 'status'

git **cus** #自定义格式查看历史（）

git config --global alias.cus 'log --pretty=format:"%h %s | %ar | %cn"'

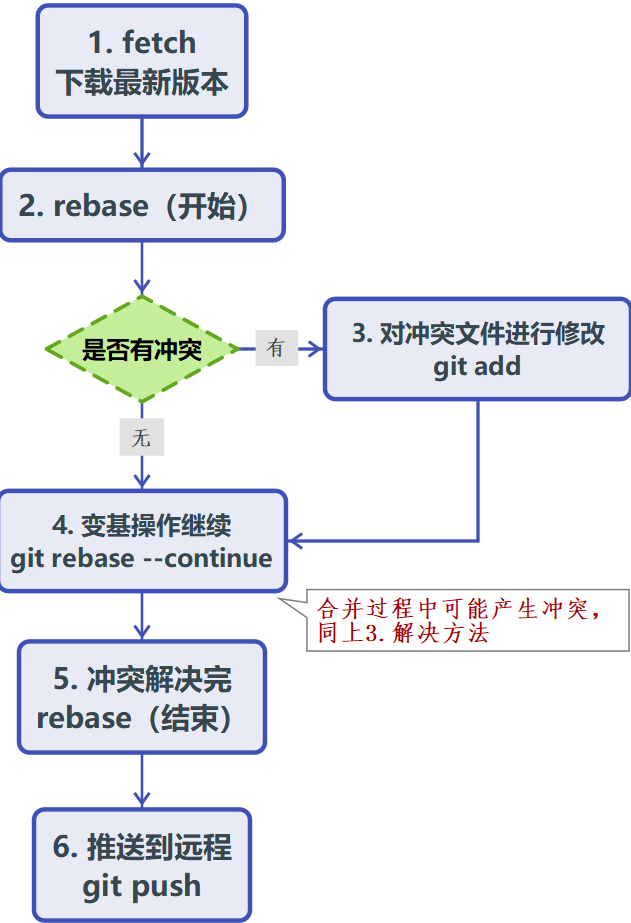
git **lsf** #列出各提交的文件内容，单行显示

git config --global alias.lsf 'log --name-only --oneline'

## 变基rebase

利用变基操作将远程分支作为本地工作分支的基础，实现拉取最新远程的文件的同步，完成变基后就可以直接push推送了。

rebase流程图：



git rebase origin/<远程分支名> //第2步操作

冲突的文件选择使用哪边的来源<本地>或<远程>

git add <冲突的文件> //第3步操作

git rebase --continue //第4步继续变基操作