**Git的奇技淫巧🙈**

Git常用命令集合，Fork于[tips](https://github.com/git-tips/tips)项目

Git是一个“分布式版本管理工具”，简单的理解版本管理工具：大家在写东西的时候都用过“回撤”这个功能，但是回撤只能回撤几步，假如想要找回我三天之前的修改，光用“回撤”是找不回来的。而“版本管理工具”能记录每次的修改，只要提交到版本仓库，你就可以找到之前任何时刻的状态（文本状态）。

下面的内容就是列举了常用的Git命令和一些小技巧，可以通过"页面内查找"的方式进行快速查询：Ctrl/Command+f。

**开卷必读**

*如果之前未使用过Git，可以学习[Git小白教程](http://rogerdudler.github.io/git-guide/index.zh.html)入门*

1. **一定要先测试命令的效果后**，再用于工作环境中，以防造成不能弥补的后果！**到时候别拿着砍刀来找我**
2. 所有的命令都在git version 2.7.4 (Apple Git-66)下测试通过
3. 统一概念：
   * 工作区：改动（增删文件和内容）
   * 暂存区：输入命令：git add 改动的文件名，此次改动就放到了‘暂存区’
   * 本地仓库(简称：本地)：输入命令：git commit 此次修改的描述，此次改动就放到了’本地仓库’，每个commit，我叫它为一个‘版本’。
   * 远程仓库(简称：远程)：输入命令：git push 远程仓库，此次改动就放到了‘远程仓库’（GitHub等)
   * commit-id：输出命令：git log，最上面那行commit xxxxxx，后面的字符串就是commit-id
4. 如果喜欢这个项目，欢迎Star、提交Pr、[反馈问题](https://github.com/521xueweihan/git-tips/issues)😊

**展示帮助信息**

git help -g

**回到远程仓库的状态**

抛弃本地所有的修改，回到远程仓库的状态。

git fetch --all && git reset --hard origin/master

**重设第一个commit**

也就是把所有的改动都重新放回工作区，并**清空所有的commit**，这样就可以重新提交第一个commit了

git update-ref -d HEAD

**展示工作区和暂存区的不同**

输出**工作区**和**暂存区**的different(不同)。

git diff

还可以展示本地仓库中任意两个commit之间的文件变动：

git diff <commit-id> <commit-id>

**展示暂存区和最近版本的不同**

输出**暂存区**和本地最近的版本(commit)的different(不同)。

git diff --cached

**展示暂存区、工作区和最近版本的不同**

输出**工作区**、**暂存区** 和本地最近的版本(commit)的different(不同)。

git diff HEAD

**快速切换分支**

git checkout -

**删除已经合并到master的分支**

git branch --merged master | grep -v '^\\*\| master' | xargs -n 1 git branch -d

**展示本地分支关联远程仓库的情况**

git branch -vv

**关联远程分支**

关联之后，git branch -vv就可以展示关联的远程分支名了，同时推送到远程仓库直接：git push，不需要指定远程仓库了。

git branch -u origin/mybranch

或者在push时加上-u参数

git push origin/mybranch -u

**列出所有远程分支**

-r参数相当于：remote

git branch -r

**列出本地和远程分支**

-a参数相当于：all

git branch -a

**创建并切换到本地分支**

git checkout -b <branch-name>

**创建并切换到远程分支**

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

**删除本地分支**

git branch -d <local-branchname>

**删除远程分支**

git push origin --delete <remote-branchname>

或者

git push origin :<remote-branchname>

**重命名本地分支**

git branch -m <new-branch-name>

**查看标签**

git tag

展示当前分支的最近的tag

git describe --tags --abbrev=0

**本地创建标签**

git tag <version-number>

默认tag是打在最近的一次commit上，如果需要指定commit打tag：

$ git tag -a <version-number> -m "v1.0 发布(描述)" <commit-id>

**推送标签到远程仓库**

首先要保证本地创建好了标签才可以推送标签到远程仓库：

git push origin <local-version-number>

一次性推送所有标签，同步到远程仓库：

git push origin --tags

**删除本地标签**

git tag -d <tag-name>

**删除远程标签**

删除远程标签需要**先删除本地标签**，再执行下面的命令：

git push origin :refs/tags/<tag-name>

**切回到某个标签**

一般上线之前都会打tag，就是为了防止上线后出现问题，方便快速回退到上一版本。下面的命令是回到某一标签下的状态：

git checkout -b branch\_name tag\_name

**放弃工作区的修改**

git checkout <file-name>

放弃所有修改：

git checkout .

**恢复删除的文件**

git rev-list -n 1 HEAD -- <file\_path> #得到 deleting\_commit

git checkout <deleting\_commit>^ -- <file\_path> #回到删除文件 deleting\_commit 之前的状态

**回到某一个commit的状态，并重新增添一个commit**

git revert <commit-id>

**回到某个commit的状态，并删除后面的commit**

和revert的区别：reset命令会抹去某个commit id之后的所有commit

git reset <commit-id> #默认就是-mixed参数。

git reset –mixed HEAD^ #回退至上个版本，它将重置HEAD到另外一个commit,并且重置暂存区以便和HEAD相匹配，但是也到此为止。工作区不会被更改。

git reset –soft HEAD~3 #回退至三个版本之前，只回退了commit的信息，暂存区和工作区与回退之前保持一致。如果还要提交，直接commit即可

git reset –hard <commit-id> #彻底回退到指定commit-id的状态，暂存区和工作区也会变为指定commit-id版本的内容

**修改上一个commit的描述**

git commit --amend

**查看commit历史**

git log

**查看某段代码是谁写的**

blame的意思为‘责怪’，你懂的。

git blame <file-name>

**显示本地执行过git命令**

就像shell的history一样

git reflog

**修改作者名**

git commit --amend --author='Author Name <email@address.com>'

**修改远程仓库的url**

git remote set-url origin <URL>

**增加远程仓库**

git remote add origin <remote-url>

**列出所有远程仓库**

git remote

**查看两个星期内的改动**

git whatchanged --since='2 weeks ago'

**把A分支的某一个commit，放到B分支上**

这个过程需要cherry-pick命令，[参考](http://sg552.iteye.com/blog/1300713" \l "bc2367928)

git checkout <branch-name> && git cherry-pick <commit-id>

**给git命令起别名**

简化命令

git config --global alias.<handle> <command>

比如：git status 改成 git st，这样可以简化命令

git config --global alias.st status

**存储当前的修改，但不用提交commit**

详解可以参考[廖雪峰老师的git教程](http://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/00137602359178794d966923e5c4134bc8bf98dfb03aea3000)

git stash

**保存当前状态，包括untracked的文件**

untracked文件：新建的文件

git stash -u

**展示所有stashes**

git stash list

**回到某个stash的状态**

git stash apply <stash@{n}>

**回到最后一个stash的状态，并删除这个stash**

git stash pop

**删除所有的stash**

git stash clear

**从stash中拿出某个文件的修改**

git checkout <stash@{n}> -- <file-path>

**展示所有tracked的文件**

git ls-files -t

**展示所有untracked的文件**

git ls-files --others

**展示所有忽略的文件**

git ls-files --others -i --exclude-standard

**强制删除untracked的文件**

可以用来删除新建的文件。如果不指定文件文件名，则清空所有工作的untracked文件。clean命令，**注意两点**：

1. clean后，删除的文件无法找回
2. 不会影响tracked的文件的改动，只会删除untracked的文件

git clean <file-name> -f

**强制删除untracked的目录**

可以用来删除新建的目录，**注意**:这个命令也可以用来删除untracked的文件。详情见上一条

git clean <directory-name> -df

**展示简化的commit历史**

git log --pretty=oneline --graph --decorate --all

**把某一个分支到导出成一个文件**

git bundle create <file> <branch-name>

**从包中导入分支**

新建一个分支，分支内容就是上面git bundle create命令导出的内容

git clone repo.bundle <repo-dir> -b <branch-name>

**执行rebase之前自动stash**

git rebase --autostash

**从远程仓库根据ID，拉下某一状态，到本地分支**

git fetch origin pull/<id>/head:<branch-name>

**详细展示一行中的修改**

git diff --word-diff

**清除gitignore文件中记录的文件**

git clean -X -f

**展示所有alias和configs**

**注意：** config分为：当前目录（local）和全局（golbal）的config，默认为当前目录的config

git config --local --list (当前目录)

git config --global --list (全局)

**展示忽略的文件**

git status --ignored

**commit历史中显示Branch1有的，但是Branch2没有commit**

git log Branch1 ^Branch2

**在commit log中显示GPG签名**

git log --show-signature

**删除全局设置**

git config --global --unset <entry-name>

**新建并切换到新分支上，同时这个分支没有任何commit**

相当于保存修改，但是重写commit历史

git checkout --orphan <branch-name>

**展示任意分支某一文件的内容**

git show <branch-name>:<file-name>

**clone下来指定的单一分支**

git clone -b <branch-name> --single-branch https://github.com/user/repo.git

**忽略某个文件的改动**

关闭 track 指定文件的改动，也就是 Git 将不会在记录这个文件的改动

git update-index --assume-unchanged path/to/file

恢复 track 指定文件的改动

git update-index --no-assume-unchanged path/to/file

**忽略文件的权限变化**

不再将文件的权限变化视作改动

git config core.fileMode false

**以最后提交的顺序列出所有Git分支**

最新的放在最上面

git for-each-ref --sort=-committerdate --format='%(refname:short)' refs/heads/

**在commit log中查找相关内容**

通过grep查找，given-text：所需要查找的字段

git log --all --grep='<given-text>'

**把暂存区的指定file放到工作区中**

不添加参数，默认是-mixed

git reset <file-name>

**强制推送**

git push -f <remote-name> <branch-name>