链滴

常用 Git 命令清单

作者：panqq

原文链接：https://ld246.com/article/1536675576235

来源网站：链滴

[TOC]

常用 Git 命令清单

Workspace：工作区
Index / Stage：暂存区
Repository：仓库区（或本地仓库）
Remote：远程仓库

一、新建代码库

```
# 在当前目录新建一个Git代码库
$ git init

# 新建一个目录，将其初始化为Git代码库
$ git init [project-name]

# 下载一个项目和它的整个代码历史
$ git clone [url]
```

git提交本地文件到码云

```
#在本地创建一个新的文件夹,进入文件夹,鼠标右键选择git bash here
$ git init                                          #初始化一个git仓库
$ git remote add origin https://git.oschina.net     #码云仓库地址
$  git add .                                        #添加目录下所有项目
$ git commit -m  “添加注释信息"
$ git push -u origin master -f                       #强制push
```

二、配置

Git的设置文件为.gitconfig，它可以在用户主目录下（全局配置），也可以在项目目录下（项目配置。

```
# 显示当前的Git配置
$ git config --list

# 编辑Git配置文件
# 修改当前项目下的git配置文件信息
$ git config -e
# 修改git全局配置文件信息
$ git config -e --global
# 设置提交代码时的用户信息
$ git config [--global] user.name "[name]"
$ git config [--global] user.email "[email address]"
```

三、增加/删除文件

```
# 添加指定文件到暂存区
$ git add [file1] [file2] ...

# 添加指定目录到暂存区，包括子目录
$ git add [dir]

# 添加当前目录的所有文件到暂存区
$ git add .

# 添加每个变化前，都会要求确认
# 对于同一个文件的多处变化，可以实现分次提交
$ git add -p

# 删除工作区文件，并且将这次删除放入暂存区
$ git rm [file1] [file2] ...

# 停止追踪指定文件，但该文件会保留在工作区
$ git rm --cached [file]

# 改名文件，并且将这个改名放入暂存区
$ git mv [file-original] [file-renamed]
```

四、代码提交

```
# 提交暂存区到仓库区
$ git commit -m [message]

# 提交暂存区的指定文件到仓库区
$ git commit [file1] [file2] ... -m [message]

# 提交工作区自上次commit之后的变化，直接到仓库区
$ git commit -a

# 提交时显示所有diff信息
$ git commit -v

# 使用一次新的commit，替代上一次提交
# 如果代码没有任何新变化，则用来改写上一次commit的提交信息
$ git commit --amend -m [message]

# 重做上一次commit，并包括指定文件的新变化
$ git commit --amend [file1] [file2] ...
```

##五、分支

```
# 列出所有本地分支
$ git branch

# 列出所有远程分支
$ git branch -r

# 列出所有本地分支和远程分支
$ git branch -a
```

```
# 新建一个分支，但依然停留在当前分支
$ git branch [branch-name]

# 新建一个分支，并切换到该分支
$ git checkout -b [branch]

# 新建一个分支，指向指定commit
$ git branch [branch] [commit]

# 新建一个分支，与指定的远程分支建立追踪关系
$ git branch --track [branch] [remote-branch]

#在本地新建分支x，并自动切换到该本地分支x,采用此种方法建立的本地分支会和远程分支建立映射系。
$ git checkout -b 本地分支名x origin/远程分支名x

# 切换到指定分支，并更新工作区
$ git checkout [branch-name]

# 切换到上一个分支
$ git checkout -

# 建立追踪关系，在现有分支与指定的远程分支之间
$ git branch --set-upstream [branch] [remote-branch]

# 合并指定分支到当前分支
$ git merge [branch]

# 选择一个commit，合并进当前分支
$ git cherry-pick [commit]

#查看本地分支与远程分支的映射关系
$ git branch -vv

#撤销本地分支与远程分支的映射关系
$ git branch --unset-upstream

# 删除分支
$ git branch -d [branch-name]

# 删除远程分支
$ git push origin --delete [branch-name]
$ git branch -dr [remote/branch]

# ps 本地分支可以与远程不同名的分支建立映射关系
```

六、标签

```
# 列出所有tag
$ git tag

# 新建一个tag在当前commit
$ git tag [tag]
```

```
# 新建一个tag在指定commit
$ git tag [tag] [commit]

# 删除本地tag
$ git tag -d [tag]

# 删除远程tag
$ git push origin :refs/tags/[tagName]

# 查看tag信息
$ git show [tag]

# 提交指定tag
$ git push [remote] [tag]

# 提交所有tag
$ git push [remote] --tags

# 新建一个分支，指向某个tag
$ git checkout -b [branch] [tag]
```

七、查看信息

```
# 显示有变更的文件
$ git status

# 显示当前分支的版本历史
$ git log
# ps:输入 ‘q’ 退出log
# 显示commit历史，以及每次commit发生变更的文件
$ git log --stat

# 搜索提交历史，根据关键词
$ git log -S [keyword]

# 显示某个commit之后的所有变动，每个commit占据一行
$ git log [tag] HEAD --pretty=format:%s

# 显示某个commit之后的所有变动，其"提交说明"必须符合搜索条件
$ git log [tag] HEAD --grep feature

# 显示某个文件的版本历史，包括文件改名
$ git log --follow [file]
$ git whatchanged [file]

# 显示指定文件相关的每一次diff
$ git log -p [file]

# 显示过去5次提交
$ git log -5 --pretty --oneline

# 显示所有提交过的用户，按提交次数排序
$ git shortlog -sn
```

```
# 显示指定文件是什么人在什么时间修改过
$ git blame [file]

# 显示暂存区和工作区的差异
$ git diff

# 显示暂存区和上一个commit的差异
$ git diff --cached [file]

# 显示工作区与当前分支最新commit之间的差异
$ git diff HEAD

# 显示两次提交之间的差异
$ git diff [first-branch]...[second-branch]

# 显示今天你写了多少行代码
$ git diff --shortstat "@{0 day ago}"

# 显示某次提交的元数据和内容变化
$ git show [commit]

# 显示某次提交发生变化的文件
$ git show --name-only [commit]

# 显示某次提交时，某个文件的内容
$ git show [commit]:[filename]

# 显示当前分支的最近几次提交
$ git reflog
```

八、远程同步

```
# 下载远程仓库的所有变动
$ git fetch [remote]

# 显示所有远程仓库
$ git remote -v

# 显示某个远程仓库的信息
$ git remote show [remote]

# 删除一个远程仓库
$ git remote rm [shortname]

# 增加一个新的远程仓库，并命名
$ git remote add [shortname] [url]

# 修改远程地址
$ git remote set-url [shortname] [url]


# 取回远程仓库的变化，并与本地分支合并
$ git pull [remote] [branch]
```

pl : git pull origin master

# 上传本地指定分支到远程仓库
$ git push [remote] [branch]

# 强行推送当前分支到远程仓库，即使有冲突
$ git push [remote] --force

# 推送所有分支到远程仓库
$ git push [remote] --all

九、撤销

# 恢复暂存区的指定文件到工作区
$ git checkout [file]

# 恢复某个commit的指定文件到暂存区和工作区
$ git checkout [commit] [file]

# 恢复暂存区的所有文件到工作区
$ git checkout .

# 重置暂存区的指定文件，与上一次commit保持一致，但工作区不变
$ git reset [file]

# 重置暂存区与工作区，与上一次commit保持一致
$ git reset --hard

# 重置当前分支的指针为指定commit，同时重置暂存区，但工作区不变
$ git reset [commit_id]

# 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
$ git reset --hard [commit_id]

# 重置当前HEAD为指定commit，但保持暂存区和工作区不变
$ git reset --keep [commit_id]

# 新建一个commit，用来撤销指定commit
# 后者的所有变化都将被前者抵消，并且应用到当前分支
$ git revert [commit_id]

#git 放弃本地修改强制更新# git 放弃本地修改强制更新
$git fetch --all #只是下载远程的库的内容，不做任何的合并
$git reset --hard origin/master #把HEAD指向刚刚下载的最新的版本

十、储藏（Stashing）

# 往堆栈推送一个新的储藏
$ git stash
#查看现有的储藏
$ git stash list
# 重新应用刚刚实施的储藏,apply 选项只尝试应用储藏的工作——储藏的内容仍然在栈上

```
$ git stash apply
# 应用更早的储藏
$ git stash apply stash@{2}
# 移除贮藏
$ git stash drop
$ git stash drop stash@{0}
# 重新应用储藏，同时立刻将其从堆栈中移走
$ git stash pop

# 取消储藏(Un-applying a Stash)

# 从储藏中创建分支;创建一个新的分支，检出你储藏工作时的所处的提交，重新应用你的工作，如果功，将会丢弃储藏。
$ git stash branch 新分支名
# *这是一个很棒的捷径来恢复储藏的工作然后在新的分支上继续当时的工作。
```

十一、.gitignore的配置：

```
$ touch .gitignore
#.gitignore文件中的用法规则和语义：
# 此为注释 – 将被 Git 忽略
*.a # 忽略所有 .a 结尾的文件
!lib.a # 但 lib.a 除外
/TODO # 仅仅忽略项目根目录下的 TODO 文件，不包括 subdir/TODO
build/ # 忽略 build/ 目录下的所有文件
doc/*.txt # 会忽略 doc/notes.txt 但不包括 doc/server/arch.txt
#注意：如果你是新加的，这里需要注意的是.gitignore只能作用于没有被track的文件，#也就是工作的文件，对于add，commit操作后的文件是没有作用的，这个时候需要#先把本地的缓存删除，在去交，就可以实现忽略整个仓库文件了。
git rm -r --cached .
git add .
git commit -m 'update .gitignore'
git push
```

十二、其他

```
#生成一个可供发布的压缩包
$ git archive

# 生成 SSH 公钥
$ ssh-keygen
$ ssh-keygen -t rsa -C "example@example.com"
#验证ssh-key是否配置成功
ssh -T git@test.yunping.com
```

原文链接：常用 Git 命令清单