# GIT命令总结

系统学习移步:http://www.liaoxuefeng.com

安装好之后配置GIT

$ git config --global user.name "Your Name"

$ git config --global user.email "email@example.com"

注意git config命令的--global参数，用了这个参数，表示你这台机器上所有的Git仓库都会使用这个配置，当然也可以对某个仓库指定不同的用户名和Email地址。

命令们：

**1创建一个本地的版本库**

$ mkdir learngit

**2目录转到版本库 或者后退**

$ cd learngit / cd ..

**3查看当前目录**

$ pwd

**4将这个目录变成GIT可以管理的仓库**

$ git init

**5将文件readme.txt添加到仓库**

$ git add readme.txt

**6将工作区的所有文件提交到仓库 –m表示提交信息**

$ git **commit** -m "wrote a readme file"

**7可以查看当前仓库的状态信息**

$ git status

**8可以查看修改的内容**

$ git diff readme.txt

**9显示从最近到最远的提交日志 可以加参数为一行显示**

$ git log

$ git log --pretty=oneline

**10退回上一个版本 或根据commit ID唯一标识滚回ID版本**

$ git re**set** --hard HEAD^ //上上版本 ^^ 上100个版本~100

$ git re**set** --hard 3628164

**11查看文件信息**

$ cat readme.txt

**12查看记录每一次命令(关机重启依然有效)**

$ git reflog

**13丢弃工作区的修改**

$ git checkout -- readme.txt

**14可以把暂存区的修改撤销掉重新放回工作区 再进行13(HEAD：最新版本)**

$ git reset HEAD readme.txt

**15删除一个文件**

$ rm test.txt

$ git rm test.txt

现在你有两个选择，一是确实要从版本库中删除该文件，那就用命令git rm删掉，并且git commit：

$ git rm test.txt

$ git **commit** -m "remove test.txt"

另一种情况是删错了，因为版本库里还有呢，所以可以很轻松地把误删的文件恢复到最新版本：

$ git checkout -- test.txt

**16关联一个远程库github （我的github用户名为cTrloio）**

$ git remote add origin [git@github.com:cTrloio/learngit.git](mailto:git@github.com:cTrloio/learngit.git)

第一次推送到远程库时：

由于远程库是空的，我们第一次推送master分支时，加上了-u参数，Git不但会把本地的master分支内容推送的远程新的master分支，还会把本地的master分支和远程的master分支关联起来，在以后的推送或者拉取时就可以简化命令。

$ git push -u origin master

以后推送到远程库时：

$ git push origin master

**17从远程库克隆一个仓库到本地(使用ssh协议/http协议)**

还可以用https://github.com/cTrloio/gitskills.git这样的地址

$ git clone git@github.com: **cTrloio**/gitskills.git

查看文件列表

$ ls

**18创建一个分支 (名叫dev,** git checkout**命令加-b参数表示创建并切换)**

$ git checkout -b dev

相当于(创建&切换)

$ git branch dev

$ git checkout dev

**19用git branch命令查看当前分支(**列出所有分支，当前分支前面会标一个\*号**)**

$ git branch

**20切换回master分支：**

$ git checkout master

**21git merge命令用于合并指定分支到当前分支**

$ git merge dev

**22删除分支**

$ git branch -d dev

**23冲突解决**

可以看到分支合并图

$ git log --graph --pretty=oneline --abbrev-commit

**24表示禁用Fast forward(删除分支后，会丢掉分支信息)模式来合并分支**

$ git merge --no-ff -m "merge with no-ff" dev

普通模式合并应创建一个新的commit，所以加上-m参数，把commit描述写进去。

### 分支策略

在实际开发中，我们应该按照几个基本原则进行分支管理：

首先，master分支应该是非常稳定的，也就是仅用来发布新版本，平时不能在上面干活；

那在哪干活呢？干活都在dev分支上，也就是说，dev分支是不稳定的，到某个时候，比如1.0版本发布时，再把dev分支合并到master上，在master分支发布1.0版本；

你和你的小伙伴们每个人都在dev分支上干活，每个人都有自己的分支，时不时地往dev分支上合并就可以了。

**25可用git stash保存当前工作现场 在BUG分支上修复BUG后再还原现场**

保存当前现场

$ git stash

查看所有保存的现场列表

$ git stash list

还原现场有两个办法

一是用git stash apply恢复，但是恢复后，stash内容并不删除，你需要用git stash drop来删除；

另一种方式是用git stash pop，恢复的同时把stash内容也删了：

$ git stash pop

你可以多次stash，恢复的时候，先用git stash list查看，然后恢复指定的stash，用命令：

$ git stash apply stash@{0}

删除一个stash

$ git stash drop stash@{0}

**26在开发新功能时 最好新建一个feature分支 在不需要的时候(没有合并时)强行删除用：**

$ git branch -D 分支名

**27要查看远程库的信息**

当你从远程仓库克隆时，实际上Git自动把本地的master分支和远程的master分支对应起来了，并且，远程仓库的默认名称是origin。

$ git remote / $ git remote -v

当你从远程仓库克隆时，实际上Git自动把本地的master分支和远程的master分支对应起来了，并且，远程仓库的默认名称是origin。

28推送分支

$ git push origin master

$ git push origin dev

29抓取分支

$ git pull

git pull也失败了，原因是没有指定本地dev分支与远程origin/dev分支的链接，根据提示，设置dev和origin/dev的链接：

$ git branch *--set-upstream dev origin/dev*

$ git pull

这回git pull成功，但是合并有冲突，需要手动解决，解决的方法和分支管理中的[解决冲突](http://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/001375840202368c74be33fbd884e71b570f2cc3c0d1dcf000" \t "_blank)完全一样。解决后，提交，再push：

30创建标签

$ git tag v1.0

还可以创建带有说明的标签，用-a指定标签名，-m指定说明文字：

$ git tag -a v0.1 -m "version 0.1 released" 3628164

还可以通过-s用私钥签名一个标签：

$ git tag -s v0.2 -m "signed version 0.2 released" fec145a

签名采用PGP签名，因此，必须首先安装gpg（GnuPG），如果没有找到gpg，或者没有gpg密钥对，就会报错：

31查看标签

$ git tag

32为以前版本添加标签用commit ID添加

$ git tag v0.9 6224937

33查看标签信息

$ git **show** v0.9

34删除标签

$ git tag -d v0.1

35标签推送

$ git push origin v1.0

或者，一次性推送全部尚未推送到远程的本地标签：

$ git push origin --tags

如果标签已经推送到远程，要删除远程标签就麻烦一点，先从本地删除：

$ git tag -d v0.9

Deleted tag 'v0.9' (was 6224937)

然后，从远程删除。删除命令也是push，但是格式如下：

$ git push origin :refs/tags/v0.9

36使GIT显示颜色

$ git config --global color.ui **true**

忽略文件的原则是：

1. 忽略操作系统自动生成的文件，比如缩略图等；
2. 忽略编译生成的中间文件、可执行文件等，也就是如果一个文件是通过另一个文件自动生成的，那自动生成的文件就没必要放进版本库，比如Java编译产生的.class文件；
3. 忽略你自己的带有敏感信息的配置文件，比如存放口令的配置文件。

**37 有些时候，你想添加一个文件到Git，但发现添加不了，原因是这个文件被.gitignore忽略了：**

$ git add App.**class**

**The following paths are ignored by one of your .gitignore files:**

App.**class**

**Use -f if you really want to add them.**

**38如果你确实想添加该文件，可以用-f强制添加到Git：**

$ git add -f App.**class**

**39或者你发现，可能是.gitignore写得有问题，需要找出来到底哪个规则写错了，可以用git check-ignore命令检查：**

$ git check-ignore -v App.**class**

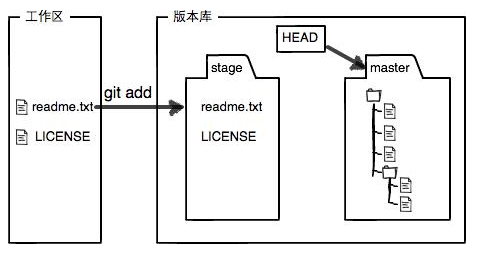
**.gitignore:3:\*.class App.class**

Git会告诉我们，.gitignore的第3行规则忽略了该文件，于是我们就可以知道应该修订哪个规则。

**40我们只需要敲一行命令，告诉Git，以后st就表示status：**

$ git config --global **alias**.st status

* 当然还有别的命令可以简写，很多人都用co表示checkout，ci表示commit，br表示branch：
* $ git config --global **alias**.co checkout
* $ git config --global **alias**.ci commit
* $ git config --global **alias**.br branch
* 以后提交就可以简写成：
* $ git ci -m "bala bala bala..."
* --global参数是全局参数，也就是这些命令在这台电脑的所有Git仓库下都有用。
* 在[撤销修改](http://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/001374831943254ee90db11b13d4ba9a73b9047f4fb968d000)一节中，我们知道，命令git reset HEAD file可以把暂存区的修改撤销掉（unstage），重新放回工作区。既然是一个unstage操作，就可以配置一个unstage别名：
* $ git config --global **alias**.unstage 'reset HEAD'
* 当你敲入命令：
* $ git unstage test.py
* 实际上Git执行的是：
* $ git re**set** HEAD test.py
* 配置一个git last，让其显示最后一次提交信息：
* $ git config --global **alias**.last 'log -1'
* 这样，用git last就能显示最近一次的提交：
* $ git last
* commit adca45d317e6d8a4b23f9811c3d7b7f0f180bfe2
* Merge: bd6ae48 291bea8
* Author: Michael Liao <askxuefeng@gmail.com>
* Date: Thu Aug 22 22:49:22 2013 +0800
* merge & fix hello.py
* 甚至还有人丧心病狂地把lg配置成了：
* git config --global **alias**.lg "log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit"

****

git add命令实际上就是把要提交的所有修改放到暂存区（Stage），然后，执行git commit就可以一次性把暂存区的所有修改提交到分支。

