注意命令间的空格和”-”的个数 http://www.liaoxuefeng.com，https://git-scm.com/book/zh/v2

创建版本库：

mkdir learngit 在当前目录下创建一个空目录

cd learngit 指向创建的目录

pwd 显示当前目录路径

git init 把当前目录变成Git仓库

一个文件放到Git仓库只需要两步：

在Git仓库目录下建xxx.xx文件

git add xxx.xx 将xxx.xx文件添加到仓库，实际上就是把文件修改添加到暂存区（stage）

git commit -m "本次提交的说明" 把文件提交到仓库，实际上就是把暂存器的所有内容提交到当前分支

可以简单理解为，需要提交的文件修改通通放到暂存区，然后，一次性提交暂存区的所有修改。

(我觉得嘛，暂存区就像购物车，没到付款的时候你都不确定购物车里的东西全部都是要的。。。每拿一件商品就付一次款。。。那才麻烦大了)

查看仓库当前状态：

git status

查看文件修改内容：

git diff xxx.xx 查看文件xxx.xx修改了哪里

git diff #是工作区(work dict)和暂存区(stage)的比较

git diff --cached #是暂存区(stage)和分支(master)的比较

查看历史记录：

git log 显示从最近到最远的提交日志

git log --pretty=oneline 只显示版本号和提交说明

回退到旧版本：

git reset --hard HEAD^ 退回到上一版本

HEAD表示当前版本，HEAD^表示上一版本，HEAD^^表示上上一个版本，HEAD~100表示往上100个版本

cat xxx.xx 查看文件xxx.xx的内容

git reset --hard xxxxxxx 指定回到xxxxxxx版本

git reflog 显示记录的每一次命令

git checkout -- xxx.xx 放弃文件xxx.xx的修改，不add添加到暂存区（not stage）

git diff HEAD -- xxx.xx查看工作区和版本库里面最新版本的区别

git reset HEAD xxx.xx 可以把暂存区的修改撤销掉（unstage），重新放回工作区（add stage，but not commit，reset to unstage）

rm xxx.xx 删除工作区文件

要从版本库中删除该文件，git rm xxx.xx 在版本库中删除xxx.xx文件，然后git commit

删错了，git checkout -- xxx.xx是用版本库里的版本替换工作区的版本，无论工作区是修改还是删除，都可以“一键还原”。

ssh-keygen -t rsa -C "youremail@example.com"创建SSH KEY，在用户主目录.ssh目录里；登陆GitHub添加SSH KEY，将github\_rsa.pub的内容粘贴到key文本框里，保存。

cat ~/.ssh/id\_rsa.pub cat ~/.ssh/github\_rsa.pub查看公钥

git remote add origin git@ github.com:path/repo-name.git将本地仓库关联到远程库

git push -u origin master 把本地库内容推送到远程库；第一次推送master分支时，加上-u参数，会把本地master分支和远程的master分支关联起来，以后就可以简化命令；本地提交后，就可以通过git push origin master 推送到GitHub

git clone git@github.com: path / repo-name.git将远程仓库克隆到本地

cd ..或cd ../退回到上层目录

git checkout -b dev创建dev分支并切换到dev分支，相当于下面两条命令：git branch dev然后git checkout dev

git branch列出所有分支，当前分支前面会标\*号

git checkout master切换到master分支，然后git merge dev将dev分支的工作合并到master分支，若不合并，master分支是看不到dev分支的工作的

git branch -d dev删除dev分支

查看分支：git branch

创建分支：git branch <name>

切换分支：git checkout <name>

创建+切换分支：git checkout -b <name>

合并某分支到当前分支：git merge <name>

删除分支：git branch -d <name>

git push origin --delete <branchName>删除远程分支，git push origin :[要删除的远程分支名字]

新建分支，修改并提交；返回主分支，修改并提交，然后将分支合并到主分支，可能会产生冲突CONFLICT，需要手动修改产生冲突的文件。

git log --graph --pretty=oneline --abbrev-commit以看到分支的合并情况

当Git无法自动合并分支时，就必须首先解决冲突。解决冲突后，再提交，合并完成。

用git log --graph命令可以看到分支合并图。

pull：本地 <-- 远程

push：本地 --> 远程

本质上都是同步commit

如果你本地落后远程，必然要pull

如果你本地超前远程，必然要push

合并分支时，如果可能，Git会用Fast forward模式，但这种模式下，删除分支后，会丢掉分支信息。

git merge --no-ff -m "merge with no-ff提交说明" otherbranch 合并分支时，加上--no-ff参数就可以用普通模式合并，合并后的历史有分支，能看出来曾经做过合并，而fast forward合并就看不出来曾经做过合并。

带--no-ff ，在合并后，Your branch is ahead of 'origin/master' by 2 commits，一个是在分支时的commit，一个是合并时的commit。不带--no-ff，在合并后，Your branch is ahead of 'origin/master' by 1 commit，只有合并时的commit。

git log --graph --pretty=oneline --abbrev-commit查看分支情况：

\* e4530d9 add new branch dev2 /\*Fast forward合并，产生1个commit，看不出曾经有分支做过合并

\* addf439 merge with no-ff /\*合并时产生的commit

|\ /\*带—no-ff合并后保留分支情况

| \* bbfcddd add new branch dev1 /\*分支产生的commit

|/

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。

所以，团队合作的分支看起来就像这样：



git stash可以把当前工作现场“储藏”起来，等以后恢复现场后继续工作

git stash list查看是否有保存工作现场

恢复现场有2种方法：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。

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}

修复bug时，我们会通过创建新的bug分支进行修复，然后合并，最后删除；

当手头工作没有完成时，先把工作现场git stash一下，然后去修复bug，修复后，再git stash pop，回到工作现场。

开发一个新feature，最好新建一个分支；git branch –d <name>删除不了未合并的分支，

如果要丢弃一个没有被合并过的分支，可以通过git branch –D <name>强行删除。

git remote查看远程库信息

git remote –v显示更详细信息

推送分支git push origin branch-name，就是把该分支上的所有本地提交推送到远程库

但是，并不是一定要把本地分支往远程推送，那么，哪些分支需要推送，哪些不需要呢？

master分支是主分支，因此要时刻与远程同步；

dev分支是开发分支，团队所有成员都需要在上面工作，所以也需要与远程同步；

bug分支只用于在本地修复bug，就没必要推到远程了，除非老板要看看你每周到底修复了几个bug；

feature分支是否推到远程，取决于你是否和你的小伙伴合作在上面开发。

总之，就是在Git中，分支完全可以在本地自己藏着玩，是否推送，视你的心情而定！

多人协作的工作模式通常是这样：

首先，可以试图用git push origin branch-name推送自己的修改；

如果推送失败，则因为远程分支比你的本地更新，需要先用git pull试图合并；

如果合并有冲突，则解决冲突，并在本地提交；

没有冲突或者解决掉冲突后，再用git push origin branch-name推送就能成功！

如果git pull提示“no tracking information”，则说明本地分支和远程分支的链接关系没有创建，用命令git branch --set-upstream branch-name origin/branch-name。

这就是多人协作的工作模式，一旦熟悉了，就非常简单。

git branch --set-upstream dev origin/dev指定本地dev分支与远程origin/dev分支的链接

git提示 --set-upstream已被弃用并将被移除

考虑使用 --track 或 --set-upstream-to

标签（tag）是版本库的快照，就是指向某个commit的指针，但是是不能移动的；而分支可以移动

切换到需要打标签的分支上，git tag <name>就可以给最新的commit打一个新标签

用命令git tag可以查看所有标签

git tag <tag-name> <commit id>对某一commit打标签

git show <tagname>查看标签信息

git tag –a <tagname> -m "version description" <commit id>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 –s <tagname> -m "version description" <commit id>可以通过-s用私钥签名一个标签，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；如果报错，请参考GnuPG帮助文档配置Key

git tag -d <tagname>删除标签

创建的标签都只存储在本地，不会自动推送到远程。如果要推送某个标签到远程，使用命令git push origin <tagname>

git push origin --tags一次性推送全部尚未推送到远程的本地标签

删除远程标签，首先要删除本地标签，然后git push origin :refs/tags/<tagname>删除远程标签

* 命令git push origin <tagname>可以推送一个本地标签；
* 命令git push origin --tags可以推送全部未推送过的本地标签；
* 命令git tag -d <tagname>可以删除一个本地标签；
* 命令git push origin :refs/tags/<tagname>可以删除一个远程标签。
* 在GitHub上，可以任意Fork开源仓库；
* 自己拥有Fork后的仓库的读写权限；
* 可以推送pull request给官方仓库来贡献代码。

git config --global color.ui **true 让Git显示颜色**

在Git工作区的根目录下创建一个特殊的.gitignore文件，然后把要忽略的文件名填进去，Git就会自动忽略这些文件。不需要从头写.gitignore文件，GitHub已经为我们准备了各种配置文件，只需要组合一下就可以使用了。所有配置文件可以直接在线浏览：<https://github.com/github/gitignore>

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。

.gitignore文件建好后需要提交到Git

如果添加某个文件，被.gitignore忽略了，git add -f <filename>可以强制添加；

如果发现是.gitignore写错了，git check-ignore -v <filename>可以找到哪里忽略了该文件。

git config --global **alias**.A B可以用此命令为B命令取个别名，以后git A就等同于git B

--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。

配置Git的时候，加上--global是针对当前用户起作用的，如果不加，那只针对当前的仓库起作用。

每个仓库的Git配置文件都放在.git/config文件中，而当前用户的Git配置文件放在用户主目录下的一个隐藏文件.gitconfig中

Git的官方网站：http://git-scm.com