git init：初始化一个Git仓库
git add readme.txt：把文件修改从工作区(*working directory*)添加到暂存区(stage)
git commit -m "wrote a readme file"： 一次性把暂存区的所有内容提交到当前分支(master)
例：第一次修改 -> `git add` -> 第二次修改 -> `git commit`
-- commit的是第一次修改的内容, commit只提交暂存区的修改内容
git status：仓库当前的状态
git diff readme.txt：查看当前修改文件的difference
git rm readme.txt：删除readme.txt
git mv readme.txt newdir：将readme.txt移动到newdir目录下,
相当于三条命令(1.mv readme.txt newdir 2. git rm readme.txt 3. git add readme.txt)
git log：文件修改日志
git reset --hard HEAD^：回退到上一个版本
git reset --hard 38a37f4：回退到38a37f4(版本号的前几位即可)这个版本
git reflog：每一条命令及版本号
git checkout -- readme.txt ：让这个文件回到最近一次`git commit`或`git add`时的状态.
git pull : 从远程仓库拉下代码到本地仓库
git remote -v ：查看远程库信息
git push origin dev : 将本地dev分支上的代码推送到远程dev分支仓库
分3种情况
1. readme.txt自修改后还没有被放到暂存区, 现在,撤销修改就回到和版本库一模一样的状态;
2. readme.txt已经添加到暂存区后, 没有修改, 命令不起作用
3. readme.txt已经添加到暂存区后,又作了修改, 现在, 撤销修改就回到添加到暂存区后的状态。
git reset HEAD readme.txt : 可以把暂存区的修改撤销掉（unstage），重新放回工作区，然后git checkout -- readme.txt 回到最初版本

删除文件后, `git rm test.txt --> git commit -m "remove test.txt"` 删除分支上的文件
误删文件后, `git checkout -- test.txt` 从分支上下载文件到工作区

`git branch` ： 查看分支
`git branch dev` 创建dev分支
`git checkout dev` ： 切换到dev分支
`git checkout -b dev` ： 创建+切换dev分支
`git merge dev` ： 合并dev分支到当前分支
`git branch -d dev` ： 删除dev分支

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的push。

Bug分支(当前在dev分支下,需要在master分支下修改bug)

1 (dev). `git stash` : 把当前工作现场“储藏”起来，等以后恢复现场后继续工作

2 (dev). `git checkout master` : 切换到master分支

3 (mst). `git checkout -b issue-101` : 创建并切换到issue-101分支

4 (iss). 修复bug

5 (iss). `git add bugfile.txt`

6 (iss). `git commit -m "fix bug 101"`

7 (iss). `git checkout master` 修复完成后，切换到master分支

8 (mst). `git merge --no-ff -m "merged bug fix 101" issue-101` 将issue-101分支合并到master分支

9 (mst). `git branch -d issue-101` 删除issue-101分支

10(dev). `git checkout dev` 切换回dev分支

11(dev). git stash list 查看 “储藏”起来的工作现场

12(dev). git stash pop 恢复工作现场被删除"储藏"的(等于git stash apply + git stash drop)

同一分支下多人开发冲突问题

1. A和B都从dev分支下拉取了代码
2. A修改了test.txt文件并提交到了远程dev分支
3. B修改完test.txt试图提交到远程dev分支, 这时提示rejected
4. 此时B应该pull远程dev分支到本地dev分支`git pull`
   (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,则执行5, 否则执行6)
5. B指定本地dev分支与远程origin/dev分支的链接(`git branch --set-upstream dev origin/dev`)
6. 这时git pull成功，但是合并有冲突
7. 手动解决冲突
8. add -> commit -> push

GitHub

把一个已有的本地仓库与github库之关联

git remote add origin git@github.com:changboa66/TestGit.git

把本地仓库master分支的内容推送到origin的master分支上(远程仓库的默认名称为origin)

```
git push origin master
```

所以整个开发流程是先提交到暂存区再到本地分支再到GitHub;

add -> commit -> push

从github库克隆到本地,两种方法

```
https方式: git clone https://github.com/changboa66/MyRepo.git(成功)
ssh方式: git clone git@github.com:changboa66/MyRepo.git(未成功)
```