**工作区 -- 暂存区 -- 本地仓库 -- 远程仓库**

工作区就是.git文件夹所在的目录

暂存区和本地仓库都在.git文件里

本地仓库也叫做版本库

master是分支名, orgin/master指远程仓库origin的master分支

# git init 创建一个版本库

# git remote add origin git@github.com:ericdd/python.git 把一个已有的本地仓库关联到GitHub仓库

# git push -u origin master 把本地仓库的内容推送到上面关联的GitHub仓库

# git remote rm origin 取消本地仓库与GitHub仓库的关联

不一定必须在空目录下创建Git仓库，选择一个已经有东西的目录也是可以的。

git init创建版本库 当前目录下多了一个.git的目录，这个目录是Git来跟踪管理版本库的

版本库里存了很多东西，其中最重要的就是称为stage（或者叫index）的暂存区，还有Git为我们自动创建的第一个分支master，以及指向master的一个指针叫HEAD。

# git add readme.txt 文件修改添加到暂存区

# git add . 工作区的全部修改一起提交到暂存区

# git commit -m "wrote a readme file" 把文件提交到仓库

# git status 查看仓库的当前状态

# git rm a.txt 相当于 rm a.txt和git add a.txt

修改还没add到暂存区时,撤销工作区的修改

# git checkout -- readme.txt 撤销文件在工作区做的修改

# git checkout -- . 恢复工作区的全部修改

修改已经add到暂存区，撤销暂存区的修改

# git reset HEAD file.txt 撤销暂存区的修改，重新放回工作区

# git reset HEAD 撤销暂存区的全部修改 等于git reset HEAD .

# git reset --hard HEAD 撤销未提交的全部修改

修改已经commit了，版本回退

# git reset --hard HEAD^ 回退到上个版本，修改已经commit

# git reset --hard f343e24 回退到指定版本

# git diff readme.txt 查想看文件到底改了什么内容

# git reflog | grep commit

**远程push**

# git remote add origin git@github.com:[ericdd](https://github.com/ericdd)/[testgit](https://github.com/ericdd/testgit2).git 将远程仓库与本地仓库关联

# git push -u origin master 把本地库的内容推送到远程

# git push -f origin master 加入-f参数，强制提交，远程端将强制更新到本地reset的版本

远程库的名字就是origin，这是Git默认的叫法，也可以改成别的

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。

第一次要用-u 以后不需要

**撤销修改**

场景1：当你改乱了工作区某个文件的内容，想直接丢弃工作区的修改时，用命令git checkout -- file。

场景2：当你不但改乱了工作区某个文件的内容，还添加到了暂存区时，想丢弃修改，分两步，第一步用命令git reset HEAD file，就回到了场景1，第二步按场景1操作。

# git reset HEAD file 如果file没加到暂存区，这行可以不用执行

# git checkout -- readme.txt 撤销文件在工作区做的修改

**版本回退**

HEAD指向的版本就是当前版本，因此，Git允许我们在版本的历史之间穿梭，使用命令git reset --hard commit\_id。

穿梭前，用git log可以查看提交历史，以便确定要回退到哪个版本。

要重返未来，用git reflog查看命令历史，以便确定要回到未来的哪个版本。

# git reflog  获取到版本号

# git log --pretty=oneline 查看提交的历史

# git reset --hard 查看当前HEAD指针指向的版本号

# git reset --hard 版本号 回退到指定的版本

# git reset --hard HEAD^ 回退到上一个版本

# git reset --hard HEAD~100  回退到前100个版本

# git remote show origin 显示远程信息

# git remote –v 要查看远程库的详细信息

# git remote rm 远程仓库名称 删除远程仓库

**push**

git push push所有分支

git push origin master 将本地主分支推到远程主分支

git push -u origin master 将本地主分支推到远程(如无远程主分支则创建，用于初始化远程仓库)

git push origin <local\_branch> 创建远程分支， origin是远程仓库名

git push origin <local\_branch>:<remote\_branch> 创建远程分支

git push origin :<remote\_branch> 先删除本地分支(git br -d <branch>)，然后再push删除远程分支

git fetch：相当于是从远程获取最新版本到本地，不会自动merge，并没更改本地仓库的代码

git pull：相当于是从远程获取最新版本并merge到本地

**pull**

git fetch origin

git pull origin master 抓取远程仓库所有分支更新并合并到本地 git pull = git fetch + git merge

git fetch origin master 抓取远程仓库更新

git merge origin/master 将远程主分支合并到本地当前分支

git pull --no-ff 抓取远程仓库所有分支更新并合并到本地，不要快进合并

**分支**

git branch  查看当前分支

git branch -r 列出远程分支

git branch -a 列出所有分支

git branch -d name 删除分支

git branch branchName  创建分支

git checkout -b branchName  创建并切换到分支

git checkout branchName  切换分支

git checkout   后面不跟任何参数，则就是对工作区进行检查

git checkout --filename  撤销文件在工作区做的修改，前提是文件没加到暂存区

git checkout - 快速切换回上一个使用过的分支

git push origin test         提交本地test分支作为远程的test分支

git push origin test:b1         提交本地test分支作为远程的b1分支

git push -d origin feat   删除远程分支feat

git push [远程repo] [本地分支]:[远程分支]，如果省略 [本地分支]，那就等于是在说“在这里提取空白然后把它变成[远程分支]”。

**检出仓库**

# git clone /d/testgit/ 从本地仓库克隆到当前目录

# git clone [https://github.com/ercidd/testgit](https://github.com/tugenhua0707/testgit)  从远程库中克隆到当前目录

# git clone git@github.com:ercidd/testgit.git

**文件删除**

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

# git mv file newname 改名文件，并且将这个改名放入暂存区

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

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

**[git diff](http://www.cnblogs.com/gbyukg/archive/2011/12/13/2286884.html)**

git diff        显示工作目录与索引区文件之间的差异  
git diff --cached显示索引区与git仓库之间的差异  
git diff HEAD   显示工作目录与git仓库之间的差异  
git diff HEAD^  比较上次提交  
git diff HEAD~2 比较上2次提交

git diff <file> # 比较当前文件和暂存区文件差异 git diff

git diff <id1><id2> # 比较两次提交之间的差异

git diff <branch1>..<branch2> # 在两个分支之间比较

git diff --staged # 比较暂存区和版本库差异

git diff --cached # 比较暂存区和版本库差异

git diff --stat # 仅仅比较统计信息

# git log --graph --pretty=oneline --abbrev-commit 查看分支合并情况

**git标签**

**#** git reset --hard 查看当前HEAD指针指向的版本号

# git show v1.1 查看标签的详细

# git tag v1.0 在当前HEAD创建一个标签，也可以指定一个commit id；

# git tag v0.8 dfbee68 在指定commit id的创建标签

# git tag -a v0.1 -m "version 0.1 released" 3628164 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 <tagname> -m "blablabla..." 可以指定标签信息；

# git push origin v1.0 推送某个标签到远程

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

# git tag -d <tagname> 可以删除一个本地标签；

# git push origin :refs/tags/<tagname> 可以删除一个远程标签。

**git合并**

git merge –no-ff 可以保存你之前的分支历史。能够更好的查看 merge历史，以及branch 状态。

git merge 则不会显示 feature，只保留单条分支记录。

--no-ff：不使用fast-forward方式合并，保留分支的commit历史  
--squash：使用squash方式合并，把多次分支commit历史压缩为一次

**git merge是出现Automatic merge failed; fix conflicts and then commit the result.**

# git co --theirs readme.txt 抛弃我们的修改，保留他们的

# git co --ours readme.txt 抛弃他们的修改，保留我们的

**工作区 -> 缓冲区 -> 版本库**

工作区：就是你在电脑上看到的目录，比如目录下testgit里的文件(.git隐藏目录版本库除外)。或者以后需要再新建的目录文件等等都属于工作区范畴。

版本库(Repository)：工作区有一个隐藏目录.git,这个不属于工作区，这是版本库。其中版本库里面存了很多东西，其中最重要的就是stage(暂存区)，还有Git为我们自动创建了第一个分支master,以及指向master的一个指针HEAD。

使用Git提交文件到版本库有两步：

第一步：是使用 git add 把文件添加进去，实际上就是把文件添加到暂存区。

第二步：使用git commit提交更改，实际上就是把暂存区的所有内容提交到当前分支上。

创建仓库的时候，master 是“默认的”分支

**remote: This repository moved. Please use the new location**

# git remote set-url origin [updated link url https://........git]

错误提示：fatal: remote origin already exists.

解决办法： git remote rm origin