# Git相关

1. ​[常用Git命令](https://liaoxuefeng.gitee.io/resource.liaoxuefeng.com/git/git-cheat-sheet.pdf)
2. 一、安装
   1. 官网下载后可以一路默认
   2. 1、配置和修改自己的识别（用户名和邮箱）

要和GitHub上的一致

* + 1. git config --global [user.name](http://user.name/) "Your name"
    2. git config --global [user.email](http://user.email/) "Your email"
  1. 2、查看自己的配置
     1. git config [user.name/user.email](http://user.name/user.email)

1. 二、创建版本库
   1. 版本库又名仓库，英文名**repository**，你可以简单理解成一个目录，这个目录里面的所有文件都可以被Git管理起来，每个文件的修改、删除，Git都能跟踪，以便任何时刻都可以追踪历史，或者在将来某个时刻可以“还原”。
   2. 初始化一个版本库
      1. 每个仓库的文件夹尽量都不同，但是都在一个总文件夹里
      2. git init 文件夹名（文件夹尽量是空的）
      3. 要先进入这个文件夹后再输入命令
   3. 添加文件到Git仓库
      1. 1、git add <file>,这个是添加到暂存区（可以多次add多个文件）
      2. 2、git commit -m <message>

其中commit后不必加文件名，而是把之前add的所有文件都添加到当前分支里，并且-m后需要加解释说明的内容，一定加，好知道每次提交（更改）都是改了什么

1. 三、文件更改
   1. 已提交到仓库的文件如果更改之后git自动知道它改动了可以用gitstatus查看状态，而且更改后的文件需要再次提交
   2. 查看状态
      1. git status
   3. 查看更改的内容

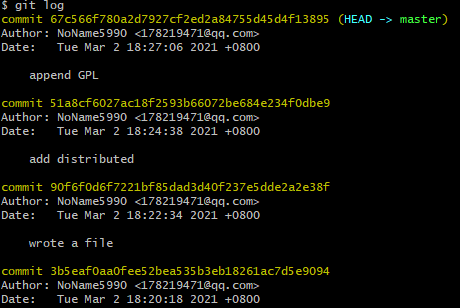
如果时间久了忘了改了什么可以用这个命令行

* + 1. git diff <file>

查看对应的文件更改的内容，即上次提交和这次更改的不同处

* + 1. 提交更改后的文件
       1. 和提交新文件一模一样

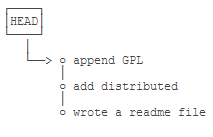
1. 四、版本回退
   1. 因为我们要不停的修改和不停的提交，不然多久的成果都有可能浪费，但是我们时间久了就记不住了，因此我们要选择性的回退到任意版本。
   2. 其实每次提交，Git都会把他们串成一条时间线
   3. 查看版本日志
      1. cat <file>可以直接查看文件里的内容
      2. git log

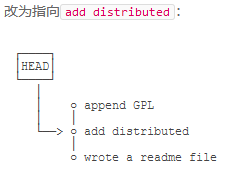


* + - 1. 顺序是从最近的一次修改到第一次创建
      2. 其中类似51a8cf6027ac18f2593b66072be684e234f0dbe9这样的数字就是版本号了，而且每个提交的版本下边都有注释，这就是为什么要commit时加-m <message>的原因了，可以清楚的知道这个版本改了什么。
      3. 其中HEAD表示当前版本
      4. 英文状态下按Q键退出 Git log 状态
  1. 版本回退
     1. HEAD表示当前版本（即最新提交的）
     2. 退回到上一个版本就用HEAD^,上上个用HEAD^^,往上一百个版本用HEAD~100
     3. git reset --hard HEAD^/HEAD~50
     4. 很重要的一点是，版本回退要知道现在版本是哪一个，如果你退了一个版本那么最新版本就变了。
  2. 版本前进
     1. 如果回退了那么之前的版本就不见了，但是如果命令行窗口还没有关闭可以通过版本号回到未来的某个版本。
        1. git reset --hard 版本号

版本号没必要写全，前几位就可以了，Git会自动去找。当然也不能只写前一两位，因为Git可能会找到多个版本号，就无法确定是哪一个了。

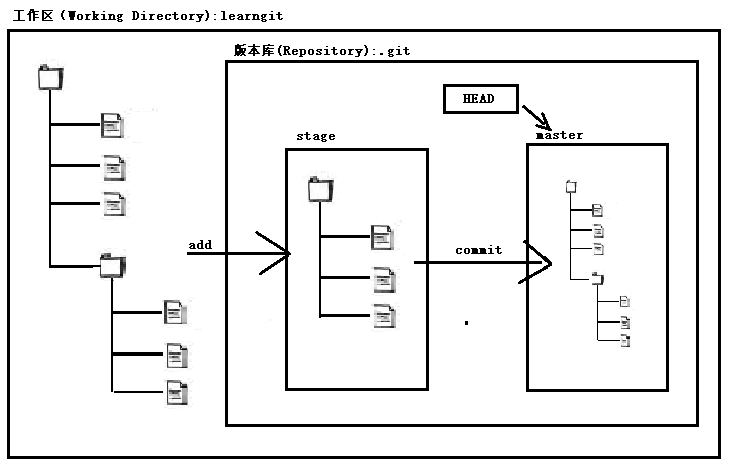
* 1. git版本回退非常的快是应为内部有个指向当前版本的HEAD指针，回退的时候就是把指针指到对应的版本而已。然后顺便把工作区的文件更新了。





* 1. 后悔药
     1. 如果回到某个版本后关掉了电脑第二天又后悔了可以git reflog（记录你的每一次commit命令）
  2. 小结
     1. 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。
     2. 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。
     3. 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。

1. 五、工作区和暂存区



* 1. 工作区（Working DIrectory）
     1. 工作区就是自己初始化的仓库，可以显示的
  2. 暂存区（stage）
     1. .git就是Git的版本库
     2. 暂存区是Git的版本库（.gti隐藏目录)下的一个文件
     3. <https://www.liaoxuefeng.com/wiki/896043488029600/897271968352576>
     4. 所有东西都是先add到暂存区，然后才能commit到master分支中
     5. 当所有东西都提交了后，那么stage就边空了，此时状态git status(查的就是暂存区working tree的状态)为clean,nothing.

1. 六、管理、撤销修改、删除文件

注意多使用git status命令，会有相应的提示

* 1. git diff HEAD -- readme.txt命令可以查看工作区和版本库里面最新版本的区别
  2. 撤销工作区修改
     1. 1、git restore <file>，建议使用

gitbash上有提示,使用这个

* + 1. 2、git checkout -- <file>，只做了解

git checkout -- file命令中的--很重要，没有--，就变成了“切换到另一个分支”的命令，我们在后面的分支管理中会再次遇到git checkout命令

* 1. 撤销暂存区
     1. 1、git restore --stage <file>（建议使用）回到工作区，撤销工作区参看上边的

GitBash上有提示

* + 1. 2、git reset HEAD <file>（只做了解）
  1. 撤销本地版本库
     1. 这个就是直接的回退版本
  2. 删除文件的恢复
     1. 删除 rm <file>
     2. 1、已经提交到暂存区或版本库的文件
        1. 可以恢复
        2. gti restore <file>
     3. 2、未add或者commit的
        1. 删除就完蛋了

1. 七、远程仓库
   1. SSHkey与Github连接
      1. 1、看用户主目录下（C:\Users\admin\.ssh）有id\_rsa和id\_[rsa.pub](http://rsa.pub/)
      2. 2、没有的话输入$ ssh-keygen -t rsa -C "youremail@example.com"然后一路回车
      3. 3、把公匙id\_[rsa.pub](http://rsa.pub/)中的内容复制，添加到自己的GitHUb连接上就行，标题尽量是具有辨别性的
      4. 为什么GitHub需要SSH Key呢？因为GitHub需要识别出你推送的提交确实是你推送的，而不是别人冒充的，而Git支持SSH协议，所以，GitHub只要知道了你的公钥，就可以确认只有你自己才能推送
   2. 关联远程库
      1. 1、首先要在GitHub上有一个远程库，那么名字呢尽量和本地的一样
      2. 2、进入到本地的想要与远程库关联的库的目录（在对应版本库中操作）
      3. 3、输入命令行git remote add origin git@github.com:NoName5990/learngit.git

git remote add origin git@server-name:path/repo-name.git

* + 1. 4、关联后使用命令git push -u origin master第一次推送master分支的所有内容

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。

* + 1. 5、此后每次本地提交后只要有必要，就可以使用命令git push origin master推送最新内容
    2. SSH警告
       1. 当你第一次使用Git的clone或者push命令连接GitHub时，会得到一个警告：The authenticity of host '[github.com](http://github.com/) (xx.xx.xx.xx)' can't be established.RSA key fingerprint is xx.xx.xx.xx.xx.Are you sure you want to continue connecting (yes/no)?这是因为Git使用SSH连接，而SSH连接在第一次验证GitHub服务器的Key时，需要你确认GitHub的Key的指纹信息是否真的来自GitHub的服务器，输入yes回车即可。Git会输出一个警告，告诉你已经把GitHub的Key添加到本机的一个信任列表里了：Warning: Permanently added '[github.com](http://github.com/)' (RSA) to the list of known hosts.这个警告只会出现一次，后面的操作就不会有任何警告了。如果你实在担心有人冒充GitHub服务器，输入yes前可以对照[GitHub的RSA Key的指纹信息](https://help.github.com/articles/what-are-github-s-ssh-key-fingerprints/)是否与SSH连接给出的一致。
    3. 删除远程库（关联）
       1. 如果添加的时候地址写错了，或者就是想删除远程库，可以用git remote rm <name>命令。使用前，建议先用git remote -v查看远程库信息：
       2. $ git remote -v
       3. origin git@github.com:michaelliao/learn-git.git (fetch)
       4. origin git@github.com:michaelliao/learn-git.git (push)
       5. 然后，根据名字删除，比如删除origin：
       6. $ git remote rm origin
       7. 此处的“删除”其实是解除了本地和远程的绑定关系，并不是物理上删除了远程库。远程库本身并没有任何改动。
  1. 从远程仓库克隆
     1. git clone git@github.com:NoName5990/gitskill.git

git clone git@server-name:path/repo-name.git

* + 1. Git支持多种协议，包括https，但ssh协议速度最快，其中点击code下载样式即可复制对应的协议方式

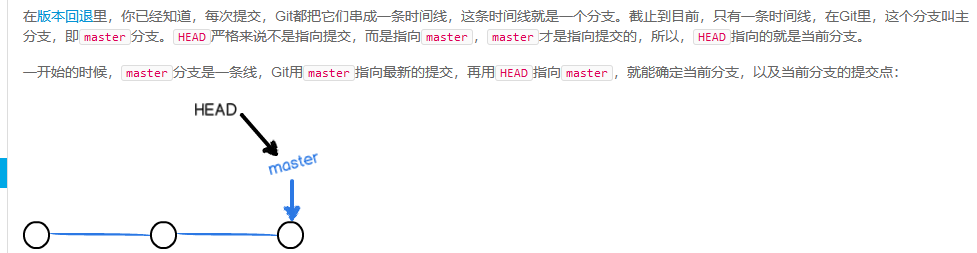
1. 八、分支管理

分支就是从一个原来的分支上开创一个自己的分支，这样的话自己就可以在自己的分支上工作而不影响主分支其他人的正常共工作了

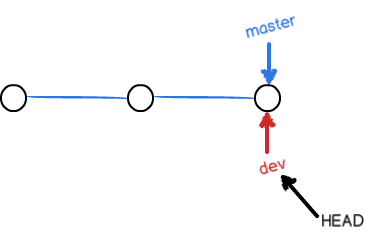
* 1. 创建与合并分支
     1. 命令行
        1. 查看分支：git branch
        2. 创建分支：git branch <name>
        3. 切换分支：git switch <name> 或者 git checkout <name>
        4. 创建+切换分支：git switch -c <name>或者 git checkout -b <name>
        5. 合并分支到当前分支：git merge <name>

如果需要进行新的提交可 git merge -m<message> <name>

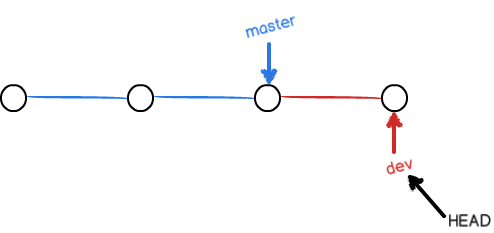
* + - 1. 删除分支：git branch -d <name>
    1. Git会把每次提交都串成一条时间线，这条时间线就是一个分支。



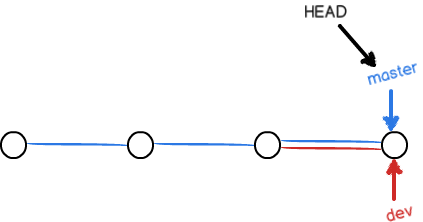
* + 1. 每一次提交，master分支都会向前移动一步，不断提交，master分支的线也越来越长。
    2. HEAD指向的是当前分支，而非主分支，也可能是新创建的分支，当我们创建一个新的分支dev时，Git创建一个指针叫dev，指向master相同的提交（即当前的master而非之前版本的），再把HEAD指向dev，就表示当前分支在dev上。



* + 1. Git创建一个分支只增加了一个dev指针，更改了HEAD指向，工作区的文件都没有任何变化。
    2. 不过对工作区的修改和提交就是针对dev分支的了，比如新提交一次后，dev指针往前移动了一步，而master指针不变。



* + 1. 如果dev上的工作完成了，就可以把dev合并到master上，合并的方式就是直接把master指向dev当前提交。因此Git合并分支也是非常的快的。就是改指针，工作区内容不变。合并后可以删除dev分支。



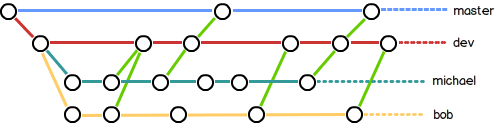
* 1. 解决冲突 #冲突
     1. 当master分支和dev分支都做了修改和提交后，合并的时候会试着把都改变了的进行修改，但是有可能会冲突，用git status查看冲突文件然后手动修改为我们想要的结果，再提交。
     2. 查看冲突时可以用status和diff看具体的情况
     3. 用带参的git log可以看到分支的合并情况。
     4. git log --graph --pretty=online --abbrev-commit

查看分支情况

* 1. 分支管理策略
     1. 一般都是在次分支上进行工作，而master主分支是比较稳定的，当次分支上的东西进行的比较稳定的时候再合并到主分支master上
     2. 发布新版本：master、比较稳定
     3. 工作干活：dev（自己创建的分支，即自己的分支）

每个人都在自己的分支上干活，最后完成工作后再进行合并

* + 1. 团队合作分支



* + 1. 这也是Git分支强大之处，在团队中应该充分应用
    2. 注意：合并分支的时候，加上--no-ff参数就可以用普通模式进行合并，合并后的历史有分支，能看出来曾经做过的合并，而fast forward合并就看不出来曾经做过合并。
    3. 附：Git会用Fastr forward模式，但这种模式下，删除分支后，会丢掉分支信息，而--no-ff就是普通模式，记得区分这两种模式的不同。
  1. Bug分支
     1. 命令行
        1. git stash保存未提交的工作区
        2. git stash list 查看保存的列表
        3. git stash apply <>恢复保存的
        4. git stash drop <>删除保存的
        5. git stash pop恢复并删除
        6. git cherry-pick <commitid>选择性改动
     2. 处理Bug时需要保护现在的工作区但是现在的分支工作区还没完成工作不能提交
  2. Feature分支
     1. 开发一个新feature（功能），最好新建一个分支；
     2. 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。
  3. 多人协作 #冲突
     1. 远程库的默认名称是origin
     2. 命令行
        1. git push origin dev：推送对应的分支到远程分支

git push origin <name>
git push origin master把本地库master分支推送到远程库对应分支

* + 1. 推送分支
       1. 推送分支就是把本地所有的提交推送到远程库。推送时，要指定本地分支，这样，Git就会把该分支推送到远程库对应的远程分支上。
       2. 分支是否推送取决于自己
          1. master是主分支，因此要时刻与远程同步
          2. dev分支是开发分支，团队成员都要在上面工作，所以也需要远程同步。
          3. bug分支只用于在本地修复bug，没必要推到远程
          4. 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。
    2. 抓取分支
       1. 当进行克隆的时候一般克隆的都是远程仓库的master分支，对于其他的分支需要进行单独的抓取
          1. 1、git clone git@github.com:NoName5990/learn.git
          2. 2、git branch 只有master
          3. 3、git switch -c dev origin/dev:本地创建dev分支并与远程仓库的dev分支关联
          4. 4、可以在dev上继续修改，提交到本地
          5. 5、推送回远程git push origin dev
       2. 从抓取分支到推送分支的流程
          1. 1、首先，可以试图用git push origin <branch-name> 推送自己的修改
          2. 2、若果推送失败，这因为远程分支比你的本地更新，需要先用git pull 试图合并
          3. 3、如果Git pull不成功（no tracking information）根据提示进行设置再进行合并git pull

git branch --set-upstream-to=origin/<branch> dev

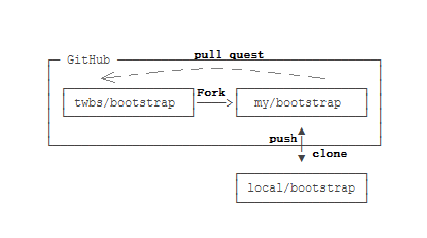
* + - * 1. 4、如果合并有冲突，则解决掉冲突，其方式与前边解决冲突的方式一样git status 然后git diff查看冲突的地方
        2. 5、手动解决掉冲突之后再进行push 即可。
      1. 命令行
         1. git remote -v ：查看远程库信息
         2. git push origin branch-name：推送本地的分支到远程对应的分支
         3. git pull：推送失败后就尝试合并，如果失败要解决冲突
         4. git switch -c branch-name origin/branch-name：在本地创建一个分支并获取远程分支对应的内用（此时只是拉取并未关联）
         5. git branch --set-upstream branch-name origin/branch-name：建立本地分支与远程分支的关联
  1. Rebase
     1. 多人在同一个分支上协作时，很容易出现冲突。即使没有冲突，后push的童鞋不得不先pull，在本地合并，然后才能push成功。
     2. 操作特点：把分叉的提交历史“整理”成一条直线，看上去更直观。缺点是本地的分叉提交已经被修改过了。
     3. 小结
        1. rebase操作可以把本地未push的分叉提交历史整理成直线
        2. rebase的目的是使得我们在查看历史提交的变化时更容易，因为分叉的提交需要三方对比。

1. 九、标签管理

发布一个版本时，通常现在版本库中打一个标签（tag）这样就唯一确定了打标签时刻的版本。这样将来就可以直接取这个标签时刻的历史版本出来即可，标签也是版本库的一个快照。

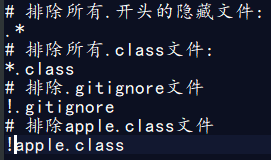
* 1. 标签
     1. Git的标签虽然是版本库的快照，但其实它就是指向某个commit的指针（跟分支很像对不对？但是分支可以移动，标签不能移动），所以，创建和删除标签都是瞬间完成的。
     2. 有commit还引入tag的目的是因为commit号太长不好找，而标签就是版本号tag v1.2即可。
     3. tag是一个容易让人记住的名字，它跟某个commit绑在一起。
  2. 创建标签
     1. 注意
        1. 标签是默认打在最新提交<HEAD>的commit上的（tag与commit绑定一起）
           1. 如果这个commit即出现在master分支又出现在dev分支，那么在这两个分支上都可以看到这个标签
     2. 1、切换到需要打标签的分支上
     3. 2、git tag v1.0.0：打上标签
     4. 3、git tag:查看标签
     5. 4、git tag -a <tagname> -m "message"：可以指定标签信息
     6. 指定版本打标签
        1. 1、git log --pretty=oneline --abbrev-commit：查看以往commit历史记录
        2. 2、git tag v1.2.1 <commitid>前几位即可
        3. 3、git tag 查看标签
  3. 操作标签
     1. 删除
        1. 本地
           1. git tag -d v0.1
           2. 标签都只能存储在本地，不会自动推送到远程。
        2. 远程（手动推送到远程的）
           1. 1、先从本地删除
           2. 2、从远程删除:git push origin :refs/tags/v0.9
           3. 3、登录GitHub查看是否删除
     2. 推送到远程
        1. git push origin <tagname>：推送指定标签
        2. git push origin --tags：推送全部本地的标签
     3. 命令行
        1. git push origin <tagname>推送指定标签
        2. git push origin --tags推送全部标签
        3. git tag -d <tagname>删除本地指定标签
        4. git push origin :refs/tags/<tagname>删除远程指定标签

1. 十、使用GitHub&Gitee
   1. 使用GitHub
      1. 1、在GitHub上，可以任意Fork开源仓库到自己的账号下（Fork就是在自己的账号下克隆了一个bootstrap仓库）
      2. 2、然后一定要从自己的账号下clone仓库这样才可以推送修改（如果直接从开源项目处clone无权限则无法推送）
      3. 3、可以推送pull request给官方仓库来贡献代码（人家可能也不接受，哈哈哈）
      4. Bootstrap的官方仓库twbs/bootstrap、你在GitHub上克隆的仓库my/bootstrap，以及你自己克隆到本地电脑的仓库，他们的关系就像下图显示的那样：



* 1. 使用Gitee
     1. [具体教程](https://www.liaoxuefeng.com/wiki/896043488029600/1163625339727712)

1. 十一、自定义Git
   1. Git配置
      1. 除了git config --global [user.name](http://user.name/) "YourName"的配置以外
      2. 还有git config --global color.ui true设置界面的颜色
   2. 忽略特殊文件
      1. 如果有些文件（比如数据库密码的配置文件）放到了工作目录中，但是又不能提交，每次git status都会显示Untracked files..，因此特殊文件可以忽略是非常有必要的。
      2. 方法：在Git工作区根目录下创建一个特殊的.gitignore（不要加其他别的的东西）文件，然后把要忽略的文件名（或者一类文件）填进去，Git就会自动忽略这些文件。
      3. 不需要从头写.gitignore文件，GItHubUI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。
      4. [配置文件在线浏览](https://github.com/github/gitignore)
      5. 忽略文件原则是
         1. 1、忽略操作系统自动生成的文件
         2. 2、忽略编译器生成的中间文件、可执行文件等，也就是如果一文件还是通过另一个文件生成的，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. 3、忽略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
         4. 小结
            1. 忽略某些文件时，需要编写.gitignore
            2. .gitignore文件本身要放到版本库里，并且可以对.gitignore做版本管理！
      6. 小结
         1. 忽略格式如下，可以直接写文件全名



* + - 1. 1、在工作区根目录下创建.gitignore文件
      2. 2、尽量把.\*和!.gitignore写上
      3. 3、加上自己想要忽略的一类或者一个文件
         1. “#+空格”后为注释
         2. !<filename>为排除指定文件
      4. 4、通过GitBash把.gitignore文件提交到版本库.git
      5. 5、这样对应的文件就不显示了
      6. 6、强制添加提交
         1. git add -f <filename>
      7. 7、查看文件是否被忽略
         1. git check-ignore -v <filename>
  1. 配置别名
     1. 其实就是偷懒，把一些命令改成比较简单的
     2. 使用命令行增加配置
        1. git config [alias.st](http://alias.st/) status：针对当前仓库

把status命令改成st简成模式

* + - 1. git config --global [alias.st](http://alias.st/) status：针对当前用户

所有该用户（NoName5990）或者本Git电脑都有效

* + - 1. git config --global [alias.co](http://alias.co/) checkout
      2. git config alias.ct commit
      3. git config --global [alias.br](http://alias.br/) branch
      4. 注意：其中[alias.name](http://alias.name/)中的name是自定义的后边的是需要表示的命令，但是即使这样原来的命令依然是可以使用的。
    1. 配置文件
       1. 注意：global是全局参数，对这台电脑的所有Git仓库都有用，如果不加就是针对当前仓库的
       2. 位置
          1. 单独仓库：.git/config中
          2. 全局：用户(user)/admin/.gitconfig
    2. 修改、删除配置
       1. 在对应的配置文件中进行删除
       2. 在对应的配置文件中进行修改
  1. 搭建Git服务器
     1. [详细教程（需要Linux机器并有sudo权限账号）](https://www.liaoxuefeng.com/wiki/896043488029600/899998870925664#0)

1. 十二、使用SourceTree
   1. 是一个图形界面工具而已
   2. 等以后真的常用的话可以看下这个使用方法
      1. [常规操作介绍](https://www.liaoxuefeng.com/wiki/896043488029600/1317161920364578#0)
2. ☆期末总结
   1. 需要经常回顾和使用一些命令行才能记得更加牢固
   2. 当然有了一定的基础，再回顾就会容易很多的