如果是第一次安装git并且上传代码，请在VS Code的命令终端（ctrl+`）中进行一下操作，以使自己的电脑识别你的github账户

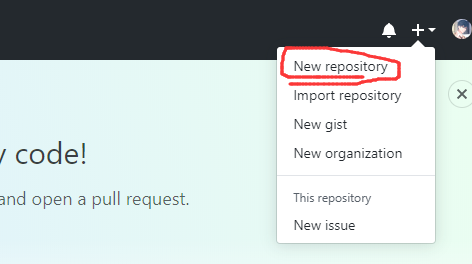
## 一、Git 全局设置

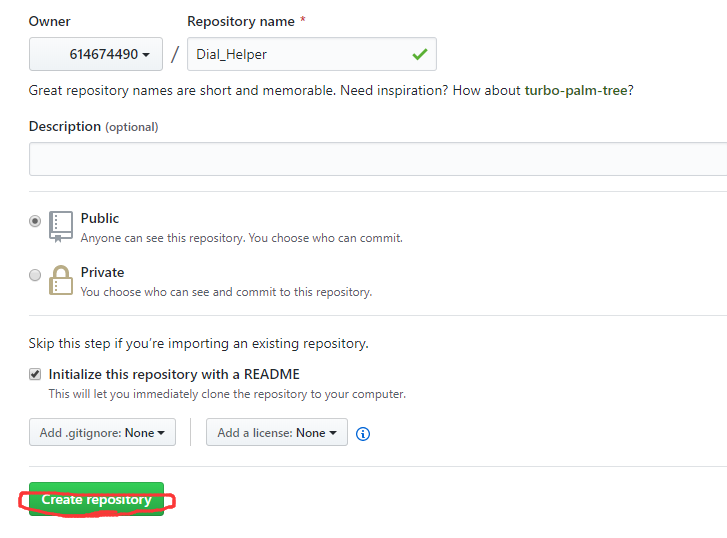
git config --global user.name "用户名"

git config --global user.email "邮箱"

## 二、新建github仓库 并用vs code+git 拉取、推送、同步、克隆

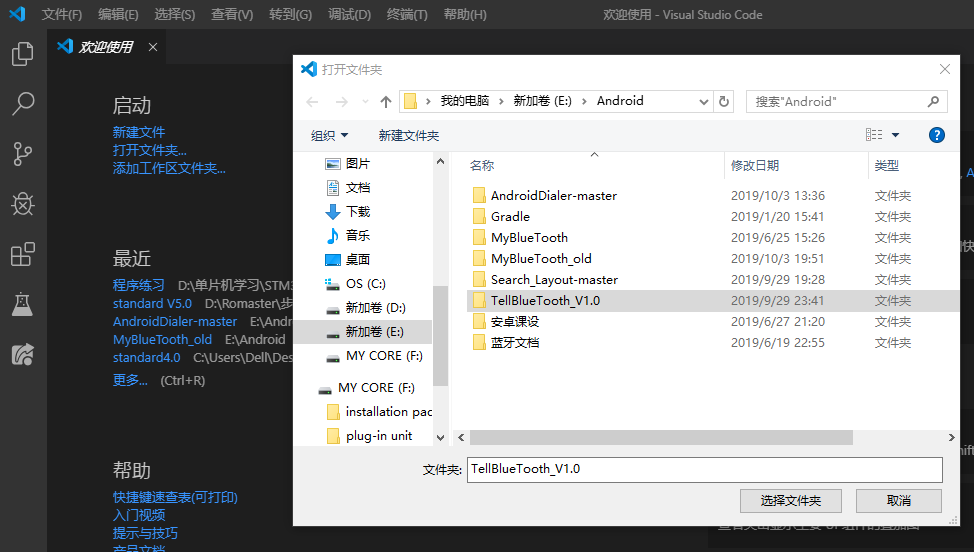
1. GitHub新建仓库



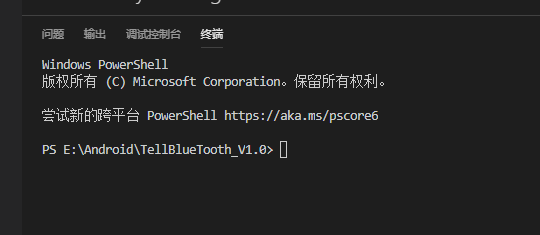


1. 在VS Code上获取远程仓库，并进行关联

打开要上传工程所在的文件夹🡪ctrl+K 然后ctrl+O



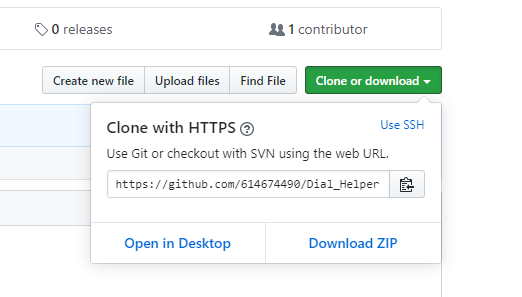
打开命令终端（ctrl+`）



注意所有的git指令操作要在工程文件夹下进行

git init 🡪初始化仓库 就是建立仓库 生成.git文件夹

1. 通过Https的方法进行关联



获取图中的仓库连接 即获取远程存储库 后面的操作都是基于该存储库的

git remote add origin <https://github.com/614674490/Dial_Helper.git>

1. 通过SSH的方式进行关联

通过密钥进行关联，首先要生成密钥

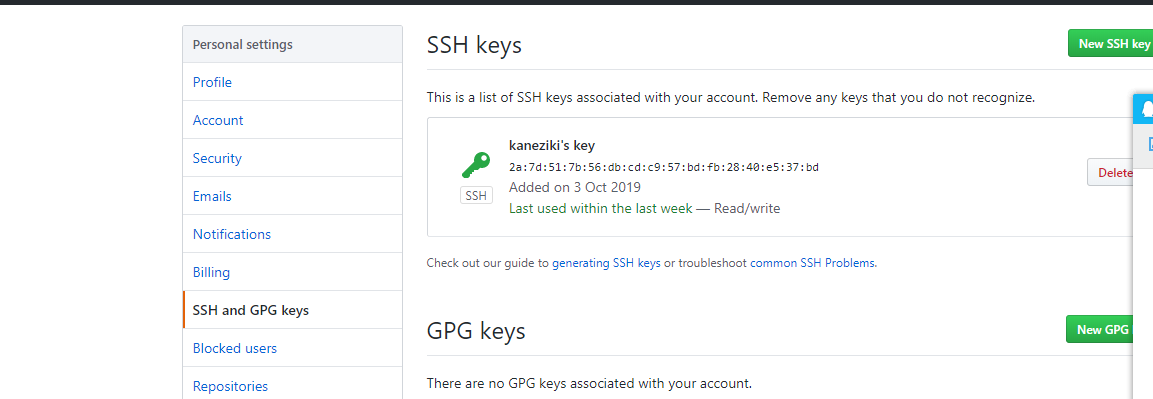
ssh-keygen -t rsa -C “[614674490@qq.com](mailto:614674490@qq.com)” //此处为github的个人注册邮箱

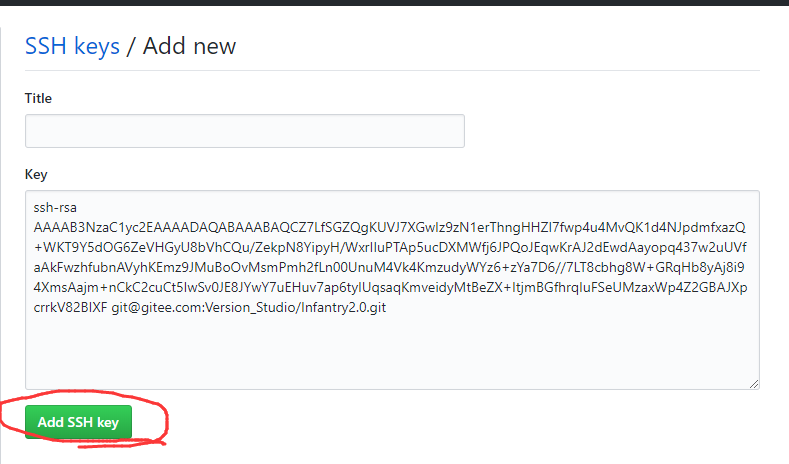
生成的密钥默认保存在C:\Users\Dell\.ssh



打开is\_rsa.pub，复制里面的内容，然后粘贴到github的生成密钥的页面中

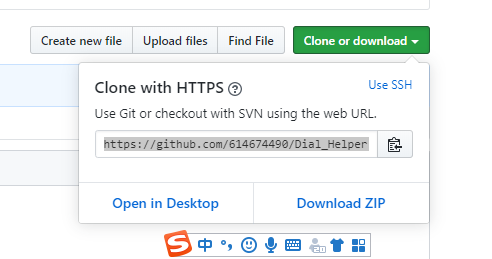
ssh的添加在用户的设置页面 头像🡪Setting🡪SSH/GPG Keys





由于我之前已经生成的密钥，在此处不再生成密钥

然后在VS Code的命令终端输入



git remote add origin [git@github.com:614674490/Dial\_Helper.git](mailto:git@github.com:614674490/Dial_Helper.git)

注意：用Https的办法的缺点是每次push的时候，都需要输入用户名和密码，如果不想输入，可以通过命令git config --global credential.helper store 记住密码

而用SSH的优点是不用输入用户名和密码，因为密钥是用来识别每一个github用户的。

1. 直接修改config文件

初始化仓库后，.git文件中有一个config文件



然后添加如下内容

[remote "origin"]

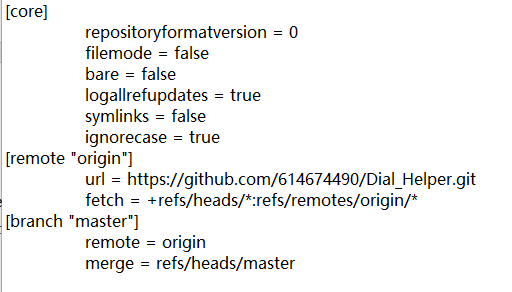
url = https://github.com/614674490/Dial\_Helper.git

fetch = +refs/heads/\*:refs/remotes/origin/\*

[branch "master"]

remote = origin

merge = refs/heads/master



1. Pull push

在上传代码前，先要点击√（commit）并输入版本说明



如果是第一次上传该工程，需要pull，将远程存储库拉取到本地，然后push

在git pull可能会出现fatal: refusing to merge unrelated histories(拒绝合并无关历史)

用该命令解决：git pull origin master --allow-unrelated-histories

Push完成后出现以下页面：

对于刚创建的仓库，没有任何内容，可通过以下命令进行操作

echo "# STM32F1\_Example" >> README.md

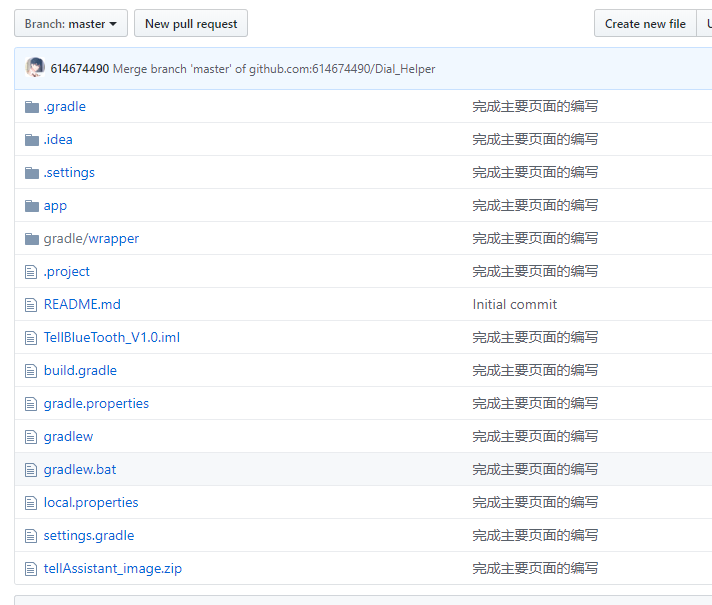
git init

git add README.md

git commit -m "first commit"

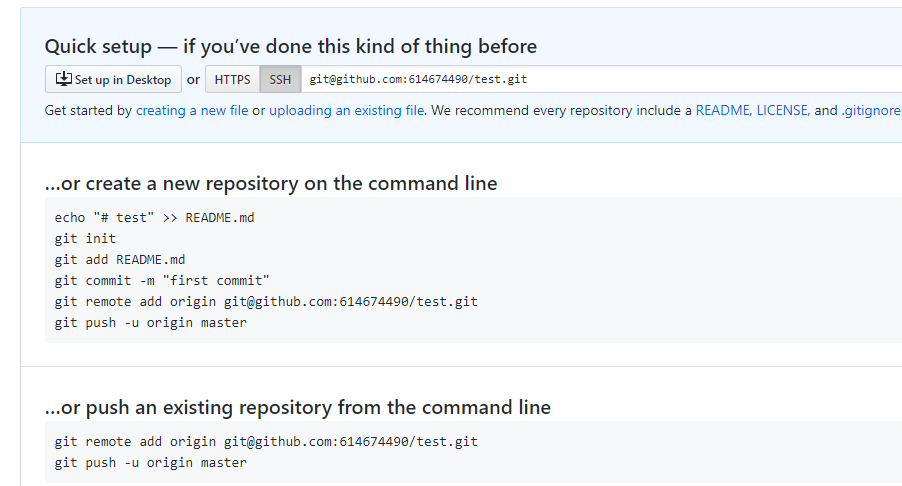
git remote add origin https://github.com/614674490/STM32F1\_Example.git

git push -u origin master



此时，基本的仓库关联和pull push完成，之后可以进行sync远程和本地仓库的同步

注意：如果在GitHub上创建仓库后没有勾选自动添加README.MD，不在仓库中添加任何内容会出现以下页面



此时若在VS Code中进行pull和push可能会出现出现not found any branch

因为此时没有任何分支可言，但是有仓库连接，所以最好勾选自动生成README.MD

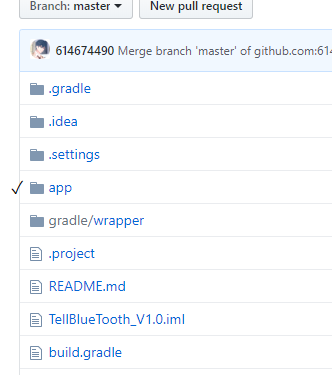
## 二、 克隆远程仓库

复制仓库的Https 或者SSH,在VS Code上输入以下命令，即可自动克隆远程仓库

git clone https/ssh 由于网络原因，大型仓库可能时间比较长，也可以直接把代码以压缩包的形式下载到本地

## 三、GitHub上单个文件的下载

在Google浏览器chorme中的应用商店下载扩展应用“GitZip for github”，然后在仓库中双击某个文件即可下载



## 四、搜索框快捷键

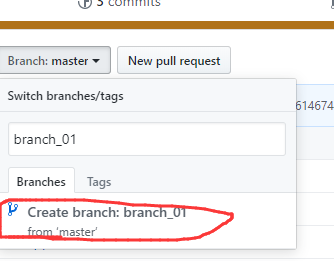
ctrl+shift+P

## 五、分支的添加与删除

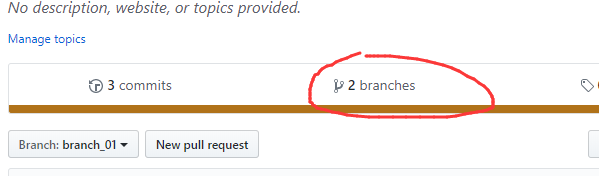
1. 分支的作用

对一个项目同时进行多个版本的控制编写，并且最后可以进行分支合并

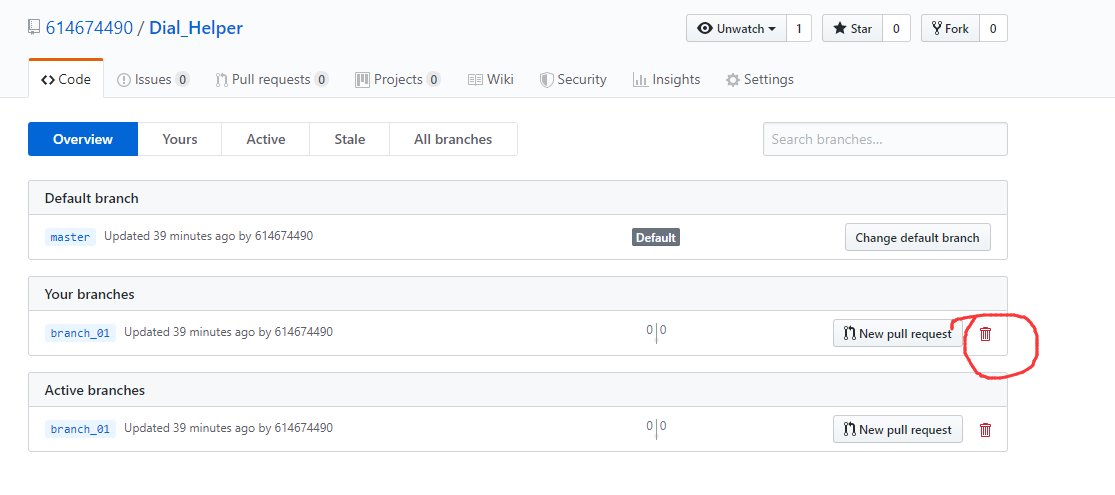
2. GitHub上的添加与删除（即远程仓库分支）



输入你要创建的分支名字，自动会弹出创建选项，进行创建即可

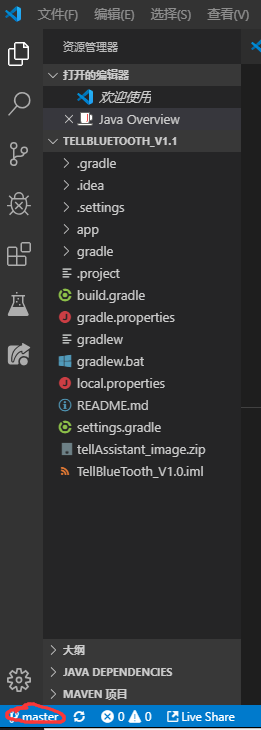


点击branches进入如下页面

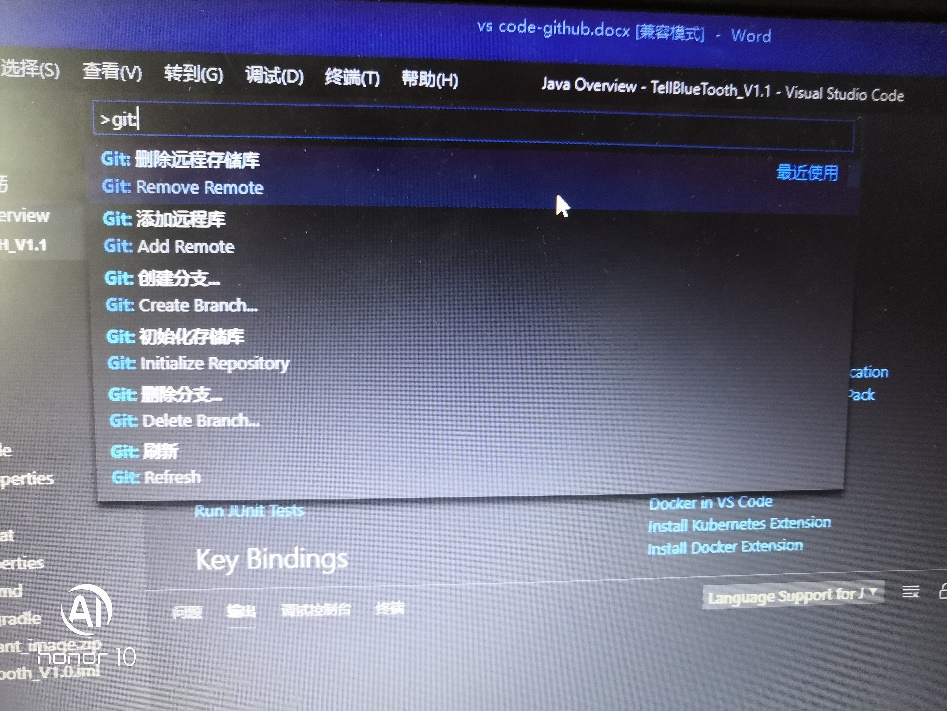


点击垃圾桶即可删除

1. VS Code上的添加与删除（即本地仓库分支）



点击该处，便会在搜索框中出现提示，点击创建分支即可，也可通过该方式更改当前的工程所在分支，两个分支之间不会产生影响，通过快捷键ctrl+shift+P打开搜索框，输入git:，即可搜索到删除本地分支的指令，点击之后选择分支删除即可



创建一个分支后再push工程，如果远程仓库没有该分支，会提示发布该分支，将其同步到远程仓库，然后再push,如果本地仓库想要获取远程仓库的建立的分支，可以通过sync同步功能进行获取

## 六、工作区、版本库和暂存区

<https://brianway.github.io/2016/08/07/git-basic-git-commands/>

* **工作区**：就是你在电脑里能看到的目录，比如我的learngit文件夹就是一个工作区。
* **版本库**：工作区有一个隐藏目录.git，这个不算工作区，而是Git的版本库。
* **暂存区**：Git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。

我们把文件往Git版本库里添加的时候，是分两步执行的：

1. 第一步是用git add把文件添加进去，实际上就是把文件修改添加到暂存区；
2. 第二步是用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。

因为我们创建Git版本库时，Git自动为我们创建了唯一一个master分支，所以,现在git commit就是往master分支上提交更改。

**简单理解**: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。

详细知识见[工作区和暂存区](http://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/0013745374151782eb658c5a5ca454eaa451661275886c6000)和[Git 基础 - 记录每次更新到仓库](https://git-scm.com/book/zh/v2/Git-%E5%9F%BA%E7%A1%80-%E8%AE%B0%E5%BD%95%E6%AF%8F%E6%AC%A1%E6%9B%B4%E6%96%B0%E5%88%B0%E4%BB%93%E5%BA%93)

## 本地库和远程库

### 新建仓库

* 建立远程库(为空，不要加README.md，不然后面会push不上去)
* 本地新建文件夹
* git init初始化仓库，可以发现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。**勿人为瞎改**
* 远程库的名字就是origin，这是Git默认的叫法
* git remote add origin git@github.com:michaelliao/learngit.git 这个命令是在本地的learngit仓库下执行的。这两个地方的仓库名不需要相同，因为会通过在本地的仓库目录下执行这条命令（命令中包含远程库的名字）已经将两者建立了联系
* git push -u origin master 把本地库的所有内容推送到远程库上。把本地库的内容推送到远程，用git push命令，实际上是把当前分支master推送到远程。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(推送和关联)**，Git不但会把本地的master分支内容推送到远程新的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 origin master每次本地提交后,推送最新修改到远程库

### 从远程库克隆

假设github上面已经有一个远程库，但是本地没有，需要克隆到本地，远程库的名字叫gitskills

* git clone git@github.com:michaelliao/gitskills.git 克隆一个本地库,则在当前文件夹下会多一个gitskills的文件夹。
* cd gitskills进入克隆下来的本地库，默认的名字是和github上的一样的
* git push origin master 推送分支，就是把该分支上的所有本地提交推送到远程库。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

## 常用查看指令

* git status 查看仓库当前的状态
* git diff 文件名查看对文件做什么修改
* git diff 版本号1 版本号2 --stat查看两个版本的差异的文件列表，包括被修改行数和增删图。参数改为--name-status前面显示修改说明字母(A,M等)，无行数
* git log显示从最近到最远的提交日志
* git log --pretty=oneline 简化日志输出的显示信息，commit id很长,详细显示见[这里](http://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/0013744142037508cf42e51debf49668810645e02887691000" \t "_blank)
* git reflog 记录你的每一次命令，最先显示的是这个命令执行之后的版本的版本号的前七位，这样就算你清屏了或者重启了，也能找到某个版本的版本号，就可以轻松回退到那个版本
* git branch 查看当前所在的分支。git branch命令会列出所有分支，当前分支前面会标一个\*号
* git log --graph --pretty=oneline --abbrev-commit用带参数的git log可以看到分支的合并情况。用git log --graph命令可以看到分支合并图
* git remote 查看远程库的信息
* git remote -v 显示更为详细的信息

## 常用修改指令

* git add readme.txt添加，但是不提交
* git commit -m "提交描述"提交，**只有add后提交才有效**。“改文件->add文件->再改->提交”，则第二次修改无效,不会被提交，只会成功提交第一次的修改。

## 撤销修改和版本回退

* git checkout -- 文件名把没暂存(即没add)的干掉，或者说，丢弃工作区，回到到暂存状态
* git reset HEAD 文件名把暂存的状态取消，工作区内容不变，但状态变为“未暂存”。

简单来说，没有add过的修改，只需要git checkout -- 文件名即可撤销；add 过的修改，先git reset HEAD 文件名变成没add 过的修改，再git checkout -- 文件名撤销。操作示例可以看[这张图](http://7xph6d.com1.z0.glb.clouddn.com/git_%E6%92%A4%E9%94%80%E4%BF%AE%E6%94%B9.png)

* git reset --hard HEAD^ 会回退到上一个版本
* git reset --hard 某版本号前几位通过命令行上的历史信息（假如你没清屏的话），找到某版本 的版本号回到指定版本。不一定要全部的版本号，就像这个命令的例子，只要前面的约7、8位这样就可以。

## 分支管理

### 创建和合并分支

* git checkout -b dev创建一个新的分支：dev，并且会切换到dev分支。所以这条命令有两个作用。git checkout命令加上-b参数表示创建并切换，相当于以下两条命令：git branch dev和git checkout dev
* git branch dev，新建分支是新建指针,指向当前commit
* git checkout dev切换到dev分支
* git checkout masterdev分支的工作完成，我们就可以切换回master分支(**此时在dev分支的修改在master上是看不到的**)
* git merge dev 这是在master分支上执行的命令，作用是：把dev分支上的工作成果合并到master分支上
* git branch -d dev 删除已合并的分支。删除分支就是删除指针
* git branch -D devGit友情提醒，dev分支还没有被合并，如果删除，将丢失掉修改，如果要强行删除，需要使用git branch -D dev命令
* git rebase master变基。在当前分支(非master)下执行该命令，则相当于把当前分支和mater分支合并，和merge操作类似，但提交历史不同，rebase操作的log更干净。具体可参考[Git 分支 - 变基](https://git-scm.com/book/zh/v2/Git-%E5%88%86%E6%94%AF-%E5%8F%98%E5%9F%BA)

### 解决冲突

假设在master分支和feature1分支对同一文件做了修改

* git merge feature1 在master分支上执行该命令，与feature1分支合并。这种情况下，Git无法执行“快速合并”，只能试图把各自的修改合并起来，但这种合并就可能会有冲突，果然冲突了！Git告诉我们，readme.txt文件存在冲突，必须手动解决冲突后再提交。git status也可以告诉我们冲突的文件

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。如果要强制禁用Fast forward模式，Git就会在merge时生成一个新的commit，这样，从分支历史上就可以看出分支信息

* git merge --no-ff -m "merge with with no-ff" dev准备合并dev分支，注意--no-ff参数表示禁用Fast forward，因为本次合并要创建一个新的commit，所以加上-m参数，把commit描述写进去

### bug分支

Git还提供了一个stash功能，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保存工作现场
* git stash list 查看工作现场
* git stash apply恢复工作现场，但是恢复后，stash内容并不删除，有多个工作现场时可以git stash apply stash@{0}恢复特定的现场
* git stash drop删除stash的内容
* git stash pop恢复的同时也把stas内容删除了

### 远程分支

这部分只介绍常用的几个操作

* git fetch origin 这个命令查找 “origin” 是哪一个服务器，从中抓取本地没有的数据，并且更新本地数据库，移动 origin/master指针指向新的、更新后的位置
* git push (remote) (branch)推送本地的分支来更新远程仓库上的 同名分支。如前文提到的git push origin master就是将本地master分支推送到远程master分支；复杂一点的，git push origin serverfix:awesomebranch将本地的 serverfix分支推送到远程仓库上的awesomebranch分支
* git push origin --delete serverfix或者git push origin :remotebranch,删除远程的serverfix分支
* git pull在大多数情况下它的含义是一个git fetch紧接着一个git merge命令。具体可参考[Git远程操作详解](http://www.ruanyifeng.com/blog/2014/06/git_remote.html)和[Documentation git-pull](https://git-scm.com/docs/git-pull)