Git是一个分布式版本控制系统，目前世界上最好用的。

Windows下安装后还要再打两行命令

$ git config --global user.name "...."

$ git config --global user.email "...."

// 配置用户名与邮箱，不过这边我还是有点不懂的，这两个东西有啥用？

// 他奶奶的，居然不用分号,不用分号的都不是好语言。

////////////////////////////////////////////////////////////////////////////////////////////

* 初始化

/\*

\*\*划一块区域给Git，让他管理。巧妇难为无米之炊！

\*/

$ mkdir ....( make directory 的缩写，后面跟目录 )

// 这条指令可以不用，拿现成的目录就行

$ cd ....( 切换到该目录，使用前必切换 )

$ pwd ( print working directory 没太大用处 )

$ git init ( 找到目录后初始化，正儿八经的让git上！ )

//别高兴的太早，还没完呢。现在只是给git划定了一块区域让他管，到底该管什么呢？该文件夹下所用内容都管，喂喂，造反啊。

//所以，还得告诉它，那些东西是要管的。毕竟调教不是一天就成功的。

////////////////////////////////////////////////////////////////////////////////////////////

* 提交

$ git add .... ( 你要调教，哦不，提交的文件。。 )

有时候要提交的文件名有空格，很难受。

例：$ git add Apache Commons.docx

这样是会报错的，Apache无法匹配或找不到Commons.docx。

解决办法有两个：

1. Apache””Commons.docx // 引号中间为空格
2. “Apache Commons.docx” // 直接用引号括起来

$ git commit -m "..." ( 这条指令最重要，-m后跟的是提交说明，该指令前可跟多个提交指令。再多讲一点，这个git commit 的参数贼多，多到怀疑人生 -\_-||| 我目前只接触过-m和 -a)

$ git commit -a 【该指令会自动处理（原文为automatically stage，是在不知 道怎么翻）被修改或被删除的文件，没有提交到版本库的文件 不受影响】

////////////////////////////////////////////////////////////////////////////////////////////

* 不大会分类

$ git status (仓库当前状态，有没有文件被改变之类的)

$ git ls-files （查看版本库内有哪些内容）

////////////////////////////////////////////////////////////////////////////////////////////

* 比较不同

$ git diff ( difference的缩写，出现了哪些不同，英文可以正常显示，中文。。。要注意，该不同是相较于上一次提交而言的，因此要是已经重新提交了，他是不会显示有何不同的。 )

$git diff --cached （比较暂存区与版本库的不同，不加参数比较的是工作区与暂存区的不同）

// git是面向修改的，并不面向文件。因此，如果所做的修改没有使用 $git add 添加至暂存区，$git commit是不会将修改提交至master分支的。

$git diff HEAD -- readme.txt （这条指令的意思是工作区文件与版本库内最新版的区别，那个readme.txt是我瞎起的。也可以理解成是git diff 与 git diff --cached的合并内容。）

////////////////////////////////////////////////////////////////////////////////////////////

* 版本回退

上面扯了这么多，现在终于可以开始介绍最牛逼的功能了。

//哆啦A梦，我们坐时光机去玩吧！

$ git reset --hard .... ( 版本号，版本号写前7个就好，少了不能唯一标识 )

$ git reset --hard head^

$ git reset --hard head~1 ( head呢表示当前版本，head^表示上一个版本，head^^表示前两个版本，依次类推。不过版本要是多了，就不方便了，比如100。head~100表示前100个版本 )

要是回退过猛，想回到未来，

$ git reflog //展示曾用过的所用指令，ref log感觉像是两个单词拼凑在一起，又一种历史记录吧。

$ git log --pretty=oneline

( 设想一个超大的工程，有N次提交，人脑这么宝贵怎么可能记得住每次都提交了什么，备注了什么。该指令用于显示历史记录，超级详细的历史记录 ，包括：版本号，作者，日期，备注。后面那个参数可以不选，什么意思也很好理解，oneline。但你妈操蛋的是， --pretty = oneline 是错的，中间不能有空格。。。。。。)

介绍一点git回退原理，本质还是指针，将每一次变动分为一个小模块，head指针依次遍历，再刷新工作区就可以回退。

工作区里有个版本库，版本库里有个暂存区。

git add 指令就是将文件修改添加到暂存区，git commit 则是将暂存区内的所有内容提交到当前分支。

////////////////////////////////////////////////////////////////////////////////////////////

$ git checkout -- ....( 文件，注意“--”两边都是有空格的 )

// 我就不讲那么多东西了，感兴趣以后再研究吧。这条指令就是撤销修改，但在git里由于master分支与暂存区的缘故，撤销也分两种状态。一是返回到和版本库一模一样的状态，二是返回到暂存区一模一样的状态。总之就是返回到最近一次git add 或 git commit的状态。

$ git reset head ....

// 将暂存区的文件放回工作区。

$ git rm ....

// 删除指令，只要文件已被提交到版本库，就不用担心误删，放心用。

--cached 参数只会将文件从版本库内移除，不修改工作区。

$ ssh-keygen -t rsa -C ["726692245@qq.com"](mailto:\"726692245@qq.com\")

//该指令用于生成秘钥，不过，每一次生成其实都不一样。

////////////////////////////////////////////////////////////////////////////////////////////

* 远程操作

本地关联远程有两种方式。

* 第一，本地推送远程：

$ git remote add <主机名> <主机地址>

// 这条指令是添加远程仓库的意思，主机名可以自己取，但也别取的太魔幻。添加有两种方式，ssh与https。（https推送每次都要输用户名和密码，很烦的，但不知道为什么我以前用https传输时没这回事，今天倒是有。以前真怪。）

该指令其实是git remote 的延伸之一。

$ git remote 的作用是列出与本仓库关联的所有远程主机。

$ git remote -v 加个 -v参数可以列出具体的地址

除此之外还有

$ git remote show <主机名> 列出指定主机的详细信息

$ git remote rm <主机名> 删除某主机

$ git remote rename <原主机名> <新主机名> 滴，更名卡

此处再补一条，ssh连接与http连接时可以互相更换的，这个还是挺有用的，总之要多查文档。

$ git remote set-url origin <远程库的地址>

* 第二，本地克隆远程：

$ git clone .....（远程库的地址，可以是ssh协议，也可以是https协议，各有千秋吧。）

这样直接克隆的话，远程主机会被自动命名为origin。

$ git clone -o <自定义主机名> <远程库地址> // -o参数

OK了，接下来的操作大家都一模一样。

* $ git fetch <远程主机名>

// 这个操作会将远程库的更新统统取回本地，但注意，只是取回，本地或远程的内容均不做修改。所以，一般用来看看其他人的工作进度。

$ git fetch <远程主机名> <分支>

// 这种写法只取回特定的分支

取回分之后可以有选择的将其与本地分支进行合并。

$ git fetch origin master

$ git merge master/master（这个不知道对不对，因为两个分支都一样）

* $ git pull <远程主机名> <远程分支名>:<本地分支名>

// 这条指令算比较长的了。

git pull 是取回远程仓库某分支的更新，并选择性的与本地某分支进行合并

举个例子：

$ git pull origin next:master

这个的意思是，取回远程origin仓库next分支的更新并与本地master分支合并，但如果是和当前分之合并的话，冒号后可省略。

$ git pull --rebase origin master 这是我平时用的比较多的，至于rebase模式是什么，惭愧，还不清楚。

* $ git push <远程主机名> <本地分支>:<目标分支>

将本地分支的更新有选择性的推送到远程仓库中的某分支。

//本地内容推送到远程，-u参数是将本地分支与远程分支也关联起来

$ git push -u origin master

//此后只要本地做了提交，使用以下指令就可以将本地的修改同步推送至GitHub

$ git push origin master

//要保证github里新建的项目是空的！也就是说不能有readme.md。这边我想到一个超级好的比方，全双工通信与半双工通信。本地仓库与github之间的push、pull就属于半双工通信，即在每一次操作前，相较于上一个版本，只能有一方有改动，若两方都有修改便会失败。

$ git pull --rebase origin master

$ git push -u origin master

////////////////////////////////////////////////////////////////////////////////////////////

* .gitignore

// 用来忽略那些不想提交的东西，毕竟有很多玩意都是不需要版本控制的。

有意思的是，它不仅可以忽略，还能指定将某些文件添加到版本库中，只不过我还没见过这么做的。

! 1.txt

**ecplise** ########################################################

确切的说是EGit，ctrl + enter键的组合是往本地版本库提交代码。