GitHub

1.安装git for windows，下载地址是 <https://git-for-windows.github.io/>

2.打开GitBash，配置身份信息

git config --global user.name "用户名"

git config --global user.email "邮箱"

3.生成SSH密钥

查看是否已经有了SSH密钥:cd ~/.ssh

没有则生成密钥：ssh-keygen -t rsa –C "email"，按三个回车，设置密码为空，成功后在C:\Users\Administrator\.ssh中得到了两个文件:id\_rsa和id\_rsa.pub。

4.将SSH 密钥上传到GitHub上

进入<https://github.com/settings/keys>页面，添加新的SSH Key, Title名字可随便取,用来标识是用的哪个机器。因为你有可能在多台机器上访问GitHub，这样每台机器上都要创建SSH Key并上传到GitHub上。Key是将id\_rsa.pub中的全部内容拷贝到文本区中。

5.测试是否可以用SSH连接到GitHub上:ssh -T git@github.com

The authenticity of host ‘github.com (207.97.227.239)’ can’t be established.  
RSA key fingerprint is 16:27:ac:a5:76:28:2d:36:63:1b:56:4d:eb:df:a6:48.  
Are you sure you want to continue connecting (yes/no)? yes  
Warning: Permanently added ‘github.com,207.97.227.239′ (RSA) to the list of known hosts.  
ERROR: Hi tekkub! You’ve successfully authenticated, but GitHub does not provide shell access  
Connection to github.com closed.

出现以上信息则表示连接成功。

6.Clone GitHub上的仓库到本地

I.SSH的方式

在$下执行命令:git clone git@github.com:账号/仓库名.git

其中账号就是你自己在GitHub上注册的登录用户名，仓库名就是repository。因为上面是采用SSH的方式进行Clone，已经将SSH Key传递到GitHub上，这样相关的操作就不用输入用户名和密码了。

II.http的方式

也可以采用https的方式进行Clone:git clone https://github.com/账号/仓库名.git

执行上面命令，也能把GitHub上的仓库内容下载到本地，但如果后续要进行操作，当涉及到对服务器push变更时，就要输入用户名和密码。

7.忽略不需要提交的文件和文件夹

在仓库根目录下创建名称为“.gitignore”的文件，写入不需要的文件夹名或文件，每个元素占一行即可，如：

node\_modules

bin

\*.config

8.GitHub远程操作



I．git remote

git remote:列出所有远程主机(origin)

git remote –v:查看远程主机网址

git remote show <主机名>:查看该主机详细信息

git remote add <主机名> <网址>:添加远程主机

git remote rm <主机名>:删除远程主机

git remote rename <原主机名> <新主机名>:修改远程主机名称

II．git fetch

git fetch <远程主机名>:取回远程主机所有分支的更新

git fetch <远程主机名> <分支名>:取回特定分支的更新

如取回origin主机的master分支:git fetch origin masier

在远程分支的基础上创建一个新分支:

如:git checkout –b newbranch origon/master，表示在远程分支origon/master的基础上，创建一个新分支；

在本地分支上合并远程分支:

如:git merge origin/master 或 git rebase origin/master，表示在当前分支上，合并远程分支origin/master。

III．git pull

git pull <远程主机名> <远程分支名>:<本地分支名>:取回远程主机某个分支的更新，再与本地的指定分支合并;

如:git pull origin next:master，表示取回origin主机的next分支，与本地master分支合并；

如果远程分支是与当前分支合并，冒号后面内容可省略:git pull origin next，实质上，这个操作等同于先做了git fetch，再做git merge，先git fetch origin，然后git merge origin/next。

IV．git tracking

git会自动在本地分支与远程分支之间，建立一种追踪关系（tracking）。比如，在git clone的时候，所有本地分支默认与远程主机的同名分支，建立追踪关系，也就是说，本地的master分支自动"追踪"origin/master分支；

Git手动建立追踪关系: git branch --set-upstream master origin/next

上面命令指定mater分支追踪origin/next分支；

如果当前分支与远程分支存在追踪关系，就可以省略远程分支名:git pull origin，

上面命令表示，本地的当前分支自动与对应的origin主机"追踪分支"进行合并；

如果当前分支只有一个追踪分支，连远程主机名都可以省略:git pull，

上面命令表示，当前分支自动与唯一一个追踪分支进行合并；

如果合并需要采用rebase模式，可以使用--rebase选项，

git pull --rebase <远程主机名> <远程分支名>:<本地分支名>。

V．git push

git push <远程主机名> <本地分支名>:<远程分支名>:将本地分支的更新，推送到远程主机；

注意，分支推送顺序的写法是<来源地>:<目的地>，所以git pull是<远程分支>:<本地分支>，而git push是<本地分支>:<远程分支>；

如果省略远程分支名，则表示将本地分支推送与之存在"追踪关系"的远程分支（通常两者同名），如果该远程分支不存在，则会被新建；

如git push origin master，表示将本地的master分支推送到origin主机的master分支，如果后者不存在，则会被新建；

如果省略本地分支名，则表示删除指定的远程分支，因为这等同于推送一个空的本地分支到远程分支，git push origin :master等同于git push origin --delete master，表示删除origin主机的master分支；

如果当前分支与远程分支之间存在追踪关系，则本地分支和远程分支都可以省略，git push origin，表示将当前分支推送到origin主机的对应分支；

如果当前分支只有一个追踪分支，那么主机名都可以省略，git push；

如果当前分支与多个主机存在追踪关系，则可以使用-u选项指定一个默认主机，这样后面就可以不加任何参数使用git push，git push -u origin master，表示将本地的master分支推送到origin主机，同时指定origin为默认主机，后面就可以不加任何参数使用git push了；

git push --all origin，表示将所有本地分支都推送到origin主机；如果远程主机的版本比本地版本新，推送时git会报错，要求先在本地做git pull合并差异，然后再推送到远程主机。这时，如果你一定要推送，可以使用--force选项，git push --force origin，使用--force选项，结果导致远程主机上更新的版本被覆盖，除非你很确定要这样做，否则应该尽量避免使用--force选项。

9.GitHub常用命令

git branch:查看本地分支

git branch -r:查看远程分支

git branch -a:查看所有分支

git branch <分支名>:创建本地分支，创建后不会自动切换为当前分支

git checkout <分支名>:切换分支

git checkout -b<分支名>:创建新分支并立即切换到新分支

git checkout <文件名>:撤销对工作区该文件的修改

git checkout:撤销对工作区所有文件的修改

git branch -d <分支名>:删除已经参与合并的分支，未合并的分支无法删除

git branch -D <分支名>:强制删除分支

git merge <分支名>:将该分支与当前分支合并

git push origin <分支名>:创建与该分支名称相同的远程分支

git push origin:<分支名>:删除远程分支

git push origin --delete<分支名>:删除远程分支

git status:查看当前状态

git add -A:提交所有变更

git add -u:提交被修改和被删除的文件，不包括新文件

git add .:提交新文件和被修改的文件，不包括被删除文件

git commit -m ‘注释’:只会提交添加到缓存区的文件(只提交添加的)

git commit -a -m ‘注释’:提交修改过，但是没有添加到缓存区的文件(修改过的就能提交)

git log:查看commit的日志信息

git log -stat: 显示commit历史，以及每次commit发生变更的文件

git log -n:显示过去n次的提交，n为正整数

git diff <文件名>:查看尚未暂存的更新

git rm <文件名>:移除文件(从暂存区和工作区中删除)

git rm -cached<文件名>:移除文件(只从暂存区中删除)

git rm -f <文件名>:强行移除修改后文件(从暂存区和工作区中删除)

git clean -n:告诉你哪些文件会被删除，不会真正的删除文件, 只是一个提醒

git clean -f: 删除当前目录下所有没有track过的文件，不会删除.gitignore文件里面指定的文件夹和文件，不管这些文件有没有被track过

git clean -f <path>:删除指定路径下的没有被track过的文件

git clean -df:删除当前目录下没有被track过的文件和文件夹

git tag:针对某一时间点的版本做标记，常用于版本发布

git tag:打印出当前仓库的所有Tag

git tag -l ‘v0.1.\*’ :搜索符合模式的Tag

git tag分为两种类型:轻量tag和附注tag，轻量tag是指向提交对象的引用，附注Tag则是仓库中的一个独立对象，建议使用附注Tag

git tag <Tag名>-light:创建轻量Tag

git tag -a <Tag名> -m <“Tag说明”>:创建附注Tag，参数a指定Tag类型，后附Tag名；参数m指定Tag的说明，信息会保存在Tag对象中。

git checkout <Tag名> :切换到指定Tag

git show <Tag名>:查看指定Tag信息

git tag -d <Tag名>:删除指定Tag

$ git tag -a <Tag名><commit 号>:给指定的commit打Tag

Tag推送到服务器

通常的git push不会将Tag对象提交到GitHub服务器，需要进行显式的操作：

git push origin <Tag名>:将指定的 Tag提交到GitHub服务器

git push origin –tags:将本地所有Tag一次性提交到GitHub服务器

GitHub回滚上一次commit:

I.没有push到远程仓库

git reset --hard <commit版本号>:把工作区和暂存区的内容都重置为指定的commit 版本

git reset -- soft<commit版本号>:保留现在工作区和暂存区的内容，HEAD指向制定的commit 版本

git reset -- mixed<commit版本号>:保留工作区的内容，将暂存区和HEAD内容恢复到指定的commit版本

II.已经push到远程仓库

1.先用git log找到要退回的commit号，然后执行git reset --soft <版本号>，将本地退回到相应版本，然后用git push origin <分支名> push到远程仓库，这时会提示本地版本落后于远程仓库，使用git push origin<分支名> --force强制覆盖即可，但这样可能会造成冲突；

2.git revert <commit版本号>:git revert用一个新提交来消除一个历史提交所做的任何修改，revert之后本地代码会回滚到指定的历史版本，再git push既可把线上的代码更新，这里不会像reset一样造成冲突的问题。

git revert HEAD  撤销前一次commit

git revert HEAD^    撤销前前一次commit

3.revert和reset的区别：

第一:如果你已经push到远程仓库, reset删除指定commit以后，git push可能导致一大堆冲突，但是revert并不会；

第二:如果以后现有分支和历史分支需要合并的时候，reset恢复部分的代码依然会出现在历史分支里，但是revert方向提交的commit并不会；

第三: reset是在正常的commit历史中删除了指定的commit，这时HEAD是向后移动了，而revert是在正常的commit历史中再commit一次，只不过是反向提交，他的HEAD是一直向前的。