**git**

前言：Git在乎的是文件的修改，并不在乎文件的本身。

1、创建版本库：

$ mkdir githome
$ cd githome
$ pwd

2、初始化

$ git init

3、添加文件

添加一个文件到版本库，直接新建，用notepad编写，注意格式，如新建一个 readme.txt

（注意：这一步代表你在工作区修改文件）

$ cat readme.txt

查看某文件的内容

$ ls

查看某目录内容

4、提交

$ git add readme.txt

（注意：这一步代表你将readme.txt这个对象的修改提交到了暂存区）

$ git **commit** -m "wrote a readme file"

（注意：这一步代表你将暂存区区所有的东西全部提交到仓库，-m后是你的修改说明）

（总结：修改部分得一个一个add到暂存区，但是只需commit一次就能将暂存区区所有文件提交到仓库）

5、版本回退

$ git log 查看提交日志

$ git log --pretty=oneline 简洁鲜明查看

1094adb7b9b3807259d8cb349e7df1d4d6477073 (HEAD -> master) append GPL
e475afc93c209a690c39c13a46716e8fa000c366 add distributed
eaadf4e385e865d25c48e7ca9c8395c3f7dfaef0 wrote a readme file

（注意：这三行代表你提交的三次，红色部分为修改说明，你可以通过修改说明对应到前面的大串数字）

$ git reset --hard HEAD 回退到head版本，head也就是当前指向的版本，也就是最新版。

（总结：可以根据这个格式改变HEAD，这样，我们就可以回退到任意版本了）

$ git reset --hard e475 回退到某个版本，写前几位数字就行

6、查看状态

$ git status

On branch master
Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git checkout -- <file>..." to discard changes in working directory)
modified: readme.txt
Untracked files:
(use "git add <file>..." to include in what will be committed)
LICENSE
no changes added to commit (use "git add" and/or "git commit -a")

（意思是：Git非常清楚地告诉我们，readme.txt【之前已经commit过】被修改了，但修改后还未add到暂存区。而LICENSE还从来没有被添加过，所以它的状态是Untracked。）

（总结：只要你的工作区有修改或新建的文件未提交到仓库，使用git status都会告诉你。如果工作区都提交了，

那么工作区就是干净的，如下）
On branch master
nothing to commit, working tree clean

7、撤销修改

前提：仓库中有个名为readme.txt的文件，你在文件里写了一句废话。

方法一：

$ git checkout -- readme.txt

命令git checkout -- readme.txt意思就是，把readme.txt文件在工作区的修改全部撤销，这里有两种情况：

一种是readme.txt自修改后还没有被放到暂存区，现在，撤销修改就回到和版本库一模一样的状态；

一种是readme.txt已经添加到暂存区后，又作了修改，现在，撤销修改就回到添加到暂存区后的状态。

总之，就是让这个文件回到最近一次git commit或git add时的状态。

方法二：

前提：仓库中有个名为readme.txt的文件，你在文件里写了一句废话，并且提交到了缓存区。

$ git reset HEAD readme.txt

（总结：git reset命令既可以回退版本，也可以把暂存区的修改回退到工作区。当我们用HEAD时，表示最新的版本。）

$ git checkout -- readme.txt 现在执行丢弃工作区所有修改

8、删除文件

$ rm test.txt 工作区test.txt文件被删除

$ git rm test.txt

$ git commit -m "remove test.txt"

后面两步删除版本库的文件

**git远程仓库**

1、$ ssh-keygen -t rsa -C "youremail@example.com" 创建账号的ssh。

在用户主目录里找到.ssh目录，里面有id\_rsa和id\_rsa.pub两个文件，这两个就是SSH Key的秘钥对，id\_rsa是私钥，不能泄露出去，id\_rsa.pub是公钥，可以放心地告诉任何人。

2、登陆GitHub，打开“Account settings”，“SSH Keys”页面：

然后，点“Add SSH Key”，填上任意Title，在Key文本框里粘贴id\_rsa.pub文件的内容：

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。

当然，GitHub允许你添加多个Key。假定你有若干电脑，你一会儿在公司提交，一会儿在家里提交，只要把每台电脑的Key都添加到GitHub，就可以在每台电脑上往GitHub推送了。

3、关联远程仓库

前提：github创建一个githome仓库，本地也是一个githome仓库。名字相同最好，当然也可以不同，因不同无法关联应自行百度。

$ git remote add origin git@github.com:lzwusrns5/githome.git

origin 为远程仓库的名字，可以自定义。以后可以绑定多个远程仓库，比如码云。

$ git push -u origin master 将本地仓库master分支的所有内容推送，关于master稍后讲。

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，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。

4、从远程仓库克隆

$ git clone git@github.com:lzwusrns5/githome.git

$ git clone https://github.com/michaelliao/gitskills.git 这种速度慢一点

5、分支

$ git checkout -b dev 创建dev分支，然后切换到dev分支

$ git branch dev 创建dev分支
$ git checkout dev 切换到dev分支

$ git branch 查看当前分支

前提：现在切换到dev分支。我们对readme.txt作了修改

$ git add readme.txt
$ git commit -m "branch test"

提交后返回master分支

$ git checkout master

查看一下刚才readme.txt发现内容并未修改，现在就需要合并

$ git merge dev 切换到master分支并合并dev分支

$ git branch -d dev 删除dev分支

6、解决冲突

$ git checkout -b feature1

（在feature1分支中修改readme.txt文件，比如删个逗号）

$ git add readme.txt
$ git commit -m "随便怎么写"

$ git checkout master

(在feature1分支中修改readme.txt文件，比如加个名词)

$ git add readme.txt
$ git commit -m "任意"

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

$ git status 果然有冲突！
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
(use "git push" to publish your local commits)
You have unmerged paths.
(fix conflicts and run "git commit")
(use "git merge --abort" to abort the merge)
Unmerged paths:
(use "git add <file>..." to mark resolution)
both modified: readme.txt
no changes added to commit (use "git add" and/or "git commit -a")

直接查看readme.txt这个文件，Git用<<<<<<<，=======，>>>>>>>标记出不同分支的内容。我们修改后提交。

$ git add readme.txt
$ git commit -m "conflict fixed"

$ git log --graph --pretty=oneline --abbrev-commit 查看一下

$ git branch -d feature1 最后删除分支

7、分支管理策略

前提：建立一个dev分支，在此分支修改readme.txt

$ git checkout -b dev

$ git add readme.txt
$ git commit -m "add merge"

$ git checkout master

$ git merge --no-ff -m "merge with no-ff" dev 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。

$ git log --graph --pretty=oneline --abbrev-commit

8、bug分支

前提：当你接到一个修复一个代号101的bug的任务时，很自然地，你想创建一个分支issue-101来修复它，但是，等等，当前正在dev上进行的工作还没有提交：

$ git status
On branch dev
Changes to be committed:
(use "git reset HEAD <file>..." to unstage)
new file: hello.py
Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git checkout -- <file>..." to discard changes in working directory)
modified: readme.txt

$ git stash 把当前工作现场“储藏”起来

首先确定要在哪个分支上修复bug，假定需要在master分支上修复，就从master创建临时分支：

$ git checkout master

$ git checkout -b issue-101

修改后，提交，合并issue-101到master，即可。

现在继续到dev上进行工作：

$ git checkout dev

$ git status

$ git stash list

stash@{0}: WIP on dev: f52c633 add merge

一是用git stash apply恢复，但是恢复后，stash内容并不删除，你需要用git stash drop来删除；

另一种方式是用git stash pop，恢复的同时把stash内容也删了：

$ git stash pop

$ git stash apply stash@{0}

9、feature分支

前提：接到通知，开发代号为Vulcan的新功能，该功能计划用于下一代星际飞船。

$ git checkout -b feature-vulcan

$ git add vulcan.py
$ git status

$ git commit -m "add feature vulcan"

$ git checkout dev 切回dev准备合并，但是就在此时，接到上级命令，因经费不足，新功能必须取消！

$ git branch -d feature-vulcan

error: The branch 'feature-vulcan' is not fully merged.
If you are sure you want to delete it, run 'git branch -D feature-vulcan'.

$ git branch -D feature-vulcan

10、多人协作

当你从远程仓库克隆时，实际上Git自动把本地的master分支和远程的master分支对应起来了，并且，远程仓库的默认名称是origin。

要查看远程库的信息，用git remote

$ git remote

$ git push origin dev 推送dev分支

前提：现在，模拟一个你的小伙伴，可以在另一台电脑（注意要把SSH Key添加到GitHub）或者同一台电脑的另一个目录下克隆：

$ git clone git@github.com:michaelliao/learngit.git

$ git checkout -b dev origin/dev 你的小伙伴要在dev分支上开发，就必须创建远程origin的dev分支到本地，于是他用这个命令创建本地dev分支

$ git add env.txt
他修改文件readme.txt，提交
$ git commit -m "add env"

$ git push origin dev

但此时你修改了readme.txt，也提交到远程了，就会报错

$ git add env.txt
$ git commit -m "add new env"

$ git push origin dev

To github.com:michaelliao/learngit.git
! [rejected] dev -> dev (non-fast-forward)
error: failed to push some refs to 'git@github.com:michaelliao/learngit.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.

$ git branch --set-upstream-to=origin/dev dev 设置dev和origin/dev的链接

$ git pull 先用git pull把最新的提交从origin/dev抓下来，然后，在本地合并，解决冲突，再推送

合并有冲突，解决冲突后

$ git commit -m "fix env conflict"

$ git push origin dev

11、rebase

略

12、标签管理

$ git checkout master

$ git tag v1.0 默认标签是打在最新提交的commit上的

$ git log --pretty=oneline --abbrev-commit
12a631b (HEAD -> master, tag: v1.0, origin/master) merged bug fix 101
4c805e2 fix bug 101
e1e9c68 merge with no-ff
f52c633 add merge
cf810e4 conflict fixed
5dc6824 & simple
14096d0 AND simple
b17d20e branch test
d46f35e remove test.txt
b84166e add test.txt
519219b git tracks changes
e43a48b understand how stage works
1094adb append GPL
e475afc add distributed
eaadf4e wrote a readme file

git tag v0.9 f52c633 给原来某个commit打标签

$ git push origin v1.0 向远程推某标签

$ git push origin --tags 推所有标签

$ git tag -d v0.1删除某标签