1. git创建一个版本库，pwd显示所在位置

$ mkdir learngit

$ cd learngit

$ pwd

/Users/michael/learngit

$ git init

#变成git可管理的库

1. 添加文件

git add <file> #记得文件要放在库里面

1. 提交文件

git commit -m <message> #一次可以提交多个

1. 版本回顾

查看你的修改： Git log #记得每次修改完要add commit才会有记录！

想看起来短一点：git log --pretty=oneline

退回前一个版本：git reset --hard HEAD^ #文件会回到前一次储存的样子。

HEAD:当前版本，HEAD^:上个版本，HEAD^^:上上个版本

看一下TXT现在的样子： cat readme.txt

如果你回到过去之后还想回到未来，窗口未关闭可以去前面找版号用前四五位就行

$ git reset --hard 1094a

查看你的每一次命令：git reflog

1. 工作区和暂存区

工作区：电脑里能看到的目录：例 learngit文件夹

版本库：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。

（工作区add到暂存区，然后从暂存区commit到分支master）

查看目前文件的状态： git status

可以查看他们的添加和上传的情况

1. 管理修改

如果这么改：第一次修改 -> git add -> 第二次修改 -> git commit

那么第二次修改不会被添加！

git diff HEAD -- readme.txt

用于查看工作区和版本库里最新版本的区别

1. 撤销修改

git checkout -- file

丢弃工作区的修改

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：

1.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；

2.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。

注：一定要记得有--，不然就会变成切换分支的命令！

1. 删除文件

删除文件：$rm test.txt

当工作区和版本库不一致，git status就会记录到你删了文件。

然后，

如果你真的要删除：$git rm test.txt

$git commit –m “remove tset.txt”

如果你是误删的想要恢复：$ git checkout -- test.txt

1. 添加远程仓库

先在github上面建立库，然后根据提示用

$ git remote add origin git@github.com:\*\*\*\*\*/learngit.git

这样就可以建立一个叫做origin的远程库了

然后可以把本地库的内容推送push到远程

$ git push -u origin master

实际上是把当前分支推送到远程，-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。

之后，只要本地做了提交，用

$ git push origin master

就可以把master的最新修改推送到github

如果要删除远程库

先用 $ git remote –v 查看一下当前远程库

然后根据名字删除 $git remote rm origin

1. 克隆仓库

我们也可以先建立远程仓库，然后克隆一个本地仓库

$ git clone git@github.com:\*\*\*\*\*/gitskills.git

然后就可以看到本地有库了

$ cd gitskills

$ ls #可以列出文件

注：$ cd ..\ 回到上级目录

1. 创建和合并分支

主分支master，HEAD指向当前分支

当创建新的分支，相当于在时间线上多加一个指针，然后head指向新分支，然后之后的修改就会在新分支上进行。

在新分支上完成修改后可以和master合并，然后可以把新分支删了。

创建新分支并切换到dev分支

$ git checkout -b dev #-b表示创建并切换

Switched to a new branch 'dev'

等同于

$ git branch dev

$ git checkout dev

列出所有分支，当前分支前面会有\*

$ git branch

当在分支dev上add commit的时候，在查看master上的文件是没有被修改的。在dev修改完之后可以把修改结果合并到master上面：

$ Git checkout master

$ git merge dev

删除分支：

$ git branch –d dev

也可以用switch

$ git switch –c dev

$ git switch master

1. 解决冲突

当在分支1上提交，然后又在分支2做不同的提交，合并的时候就会发生冲突。

$ git status 可以告诉我们冲突的文件。

可以查看txt内容，然后git会用<<<<<<<<<, ===========,>>>>>>>>>显示不同分支的内容。然后我们去手动修改保存。然后再提交。

然后用 $ git log --graph --pretty=oneline --abbrev-commit 可以查看合并情况

查看分支合并图：

$ git log --graph

1. 分支管理策略

一般来说，master是最稳定的，一般只用来提交，然后一般在dev上工作，然后再合并到master上发布。

通常合并是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 no-ff" dev

因为生成新的commit所以要-m加注释。

1. bug分支

如果在dev分支上工作做到一般要去修复别的bug，可以缓存工作区

$ git stash

然后再去要修改的分支上建立新的分支然后修改合并，然后再回到dev干活,查看

$ git stash list

恢复工作的两种办法

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：

也可以多次stash，然后用git stash list查看，然后恢复想要的那个

$ git stash apply stash@{0}

因为dev是更早出来的分支，所以有时候dev里面的bug没改，如果想快捷修改：

$ git branch

\* dev

Master

$ git cherry-pick 4c805e2

[master 1d4b803] fix bug 101

1 file changed, 1 insertion(+), 1 deletion(-)

其中4c805e2是修改的那一次commit

1. feature分支

当你想要开发一个新功能，又不想打乱原本的分支，就开一个future分支然后开发，完成了再合并然后删除分支。

$ git switch -c feature-vulcan #然后去开发

$ git add vulcan.py

$ git status

$ git commit -m "add feature vulcan"

$ git switch dev

然后就可以合并了

如果这个时候项目不要了，要强行删除！

$ git branch -d feature-vulcan

会提示你无法删除，因为还未合并，这里就用大写D删除

$ git branch -D feature-vulcan

1. 多人协作

查看远程库信息，使用 $ git remote -v；

本地新建的分支如果不推送到远程，对其他人就是不可见的；

从本地推送分支，使用$ git push origin branch-name，如果推送失败，先用 $ git pull抓取远程的新提交；

在本地创建和远程分支对应的分支，使用 $ git checkout -b branch-name origin/branch-name，本地和远程分支的名称最好一致；

建立本地分支和远程分支的关联，使用$ git branch --set-upstream branch-name origin/branch-name；从远程抓取分支，使用 $ git pull，如果有冲突，要先处理冲突。

1. 打标签

首先切换到要打标签的分支上，然后

$ git tag <tag name>

想要给之前的提交加上tag：

$ git tag v0.9 f52c633

查看所有tag：

$ git tag

查看标签信息：

$ git show <tagname>

创建有说明的标签：

$ git tag -a v0.1 -m "version 0.1 released" 1094adb

删除标签：

$ git tag -d v0.1

推送标签到远程：

$ git push origin <tagname>

一次性推送全部未推送的tag到远程

$ git push origin --tags

如果已经推送了还想删除：先删除本地的，然后再用push去删除平台的

$ git tag -d v0.9

$ git push origin :refs/tags/v0.9