版本控制工具

版本控制VCS（Version Control Systems）是一种记录一个或若干文件内容变化，以便将来查阅特定版本修订情况的系统。这个系统可以自动帮我们备份文件的每一次更改，并且可以非常方便的恢复到任意的备份（版本）状态。

解决的问题：

\* 代码合并

\* 协同开发

\* 提高效率

\* 回退版本

\* 分支管理-多任务处理

分类：

\* 集中式：SVN

  \*\* 在集中式版本控制工具中，都会有一个中央服务器。将所有源代码都放到中央服务器上。

    \*\*\*首先，从中央服务器上下载代码（或者更新代码），在本地开发。接下来，如果开发完毕，要通过网络将完成的代码，提交到中央服务器上。

  \*\* 必须联网、中央服务器、所有源代码集中放在中央服务器

  \*\* 没有网络，玩不转 --缺点

  \*\* 中央服务器坏掉了，就会导致所有源代码丢失。 --最大缺点

\* 分布式：Git

\*\* 不是必须要有中央服务器，在每个开发人员的电脑上，都会存有一个完整源代码，在本地创建版本库，每一次开发完成提交的代码，都在本地计算机上操作。

  \*\* 不需要联网，中央服务器也不是必须的，每个开发人员的电脑上都是完整的源代码

  \*\* 在实际开发时，通常会搭建一个中央服务器。

  \*\* 为了方便合并所有开发人员的代码块

  \*\* 为了方便开发人员之间的代码共享

综合上述，git的优势比svn要好，摒弃了svn的缺点。但是在大项目开发中也有的团队用SVN作为版本控制工具。在下一部分我会复习svn的操作方法和git的命令的使用。

SVN的使用

开头：

对于svn的详解，我不是很熟，只是用过svn的客户端，没使用过服务端，在这里我只是简单说一下在svn的客户端怎么拉取代码，提交代码和修改冲突等等。

svn的客户端我在Mac中用的smartSVN，在window系统中用的是TortoiseSVN。两者的使用方式差别不大。

这两个软件的下载地址，我放到网盘里，有需要自行下载。

链接:<https://pan.baidu.com/s/1dZcKGY> 密码:kuoe

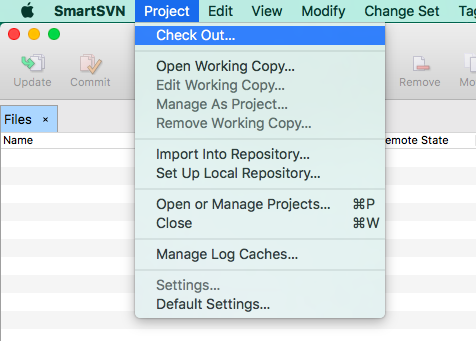
以我前段时间开发的感觉：一般安装之后，作为开发人员，会分配一个账号密码，还有服务器所在的地址，就是项目仓库的地址。

上一篇讲了版本控制工具的好处以及分类，[传送门](http://www.cnblogs.com/sqh17/p/8432270.html)  这篇讲的是是svn的使用

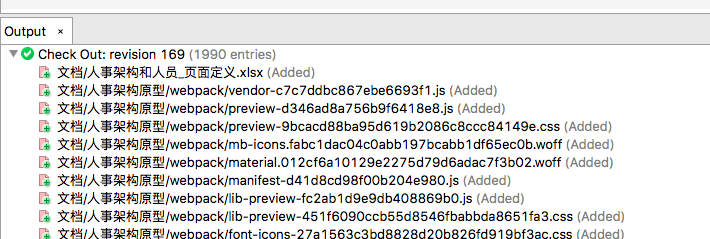
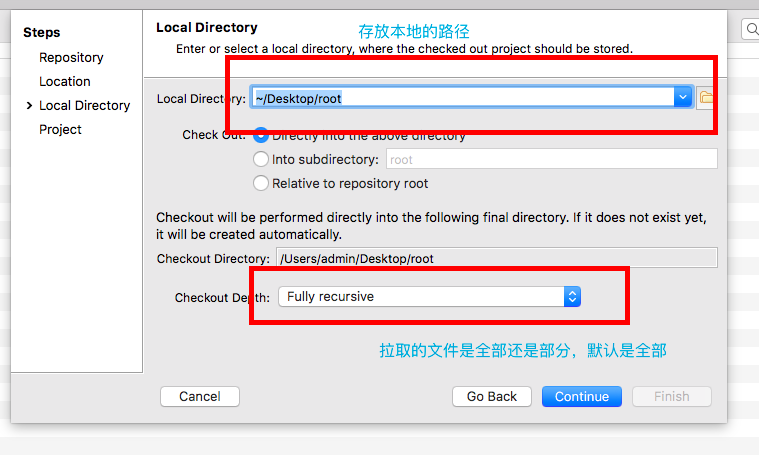
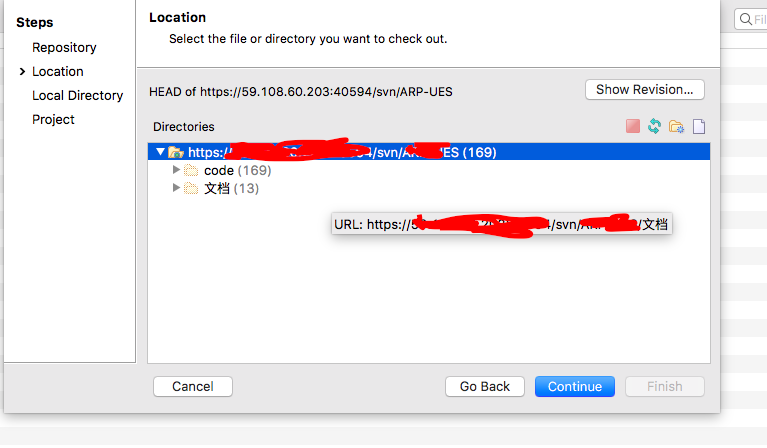
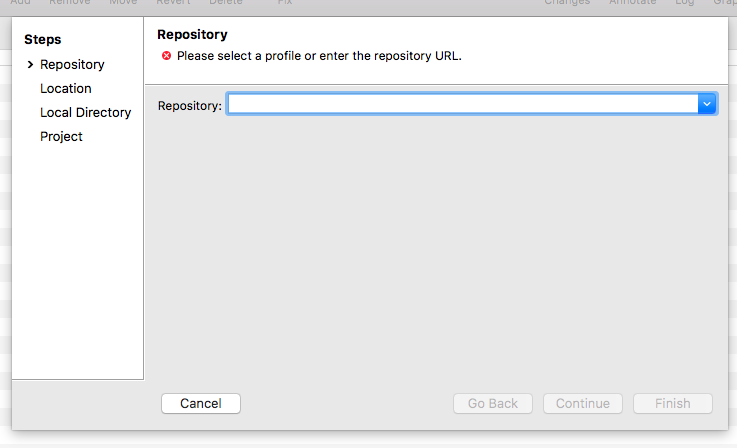
在**smartSVN**中：

拉取：

  1.先新建一个文件夹，打开服务 点击 open in smartSVN，一开始会出现这个页面，如果没有，可以点击菜单栏的 project -> check out，就出现一个弹窗。

  ==》 

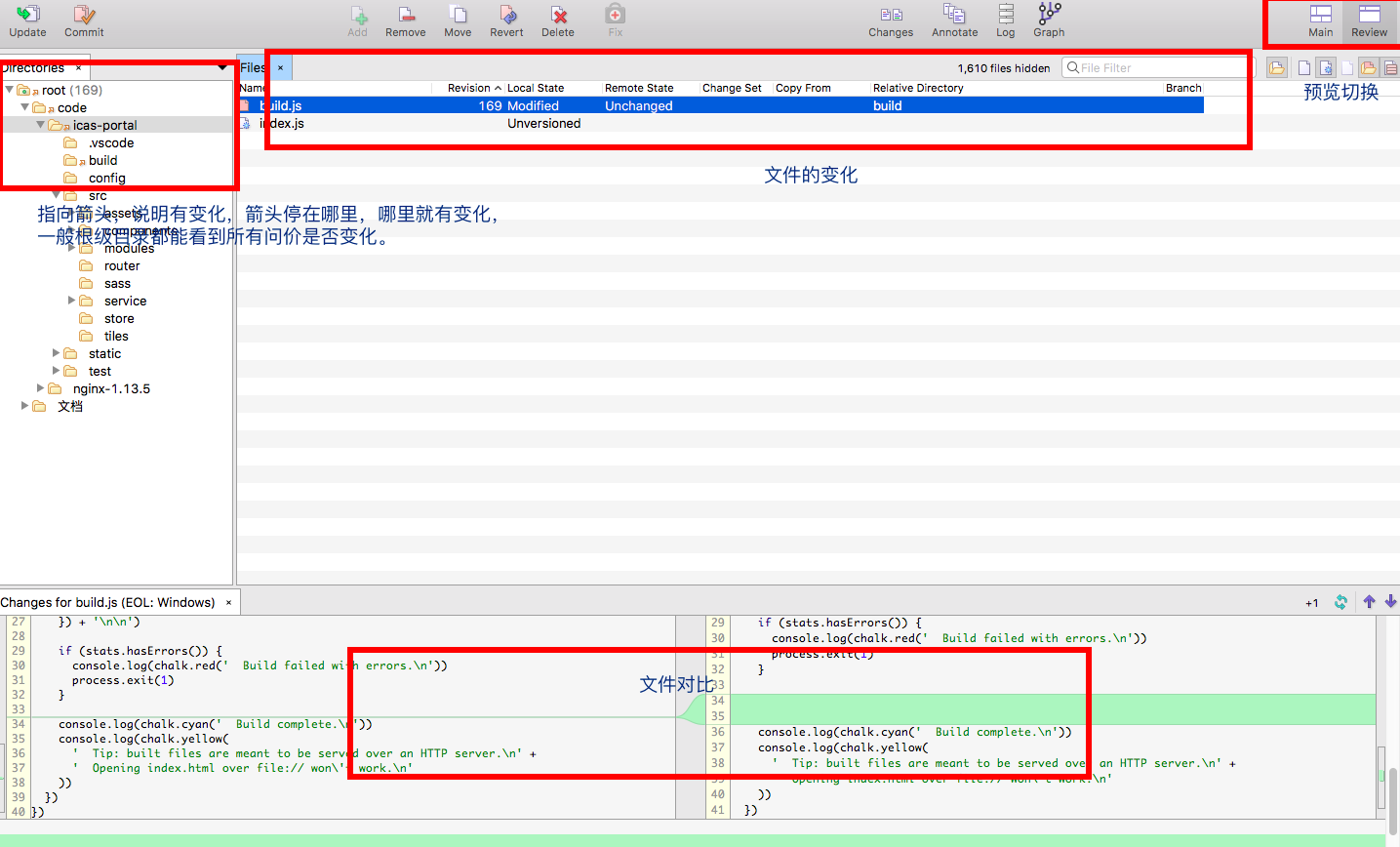
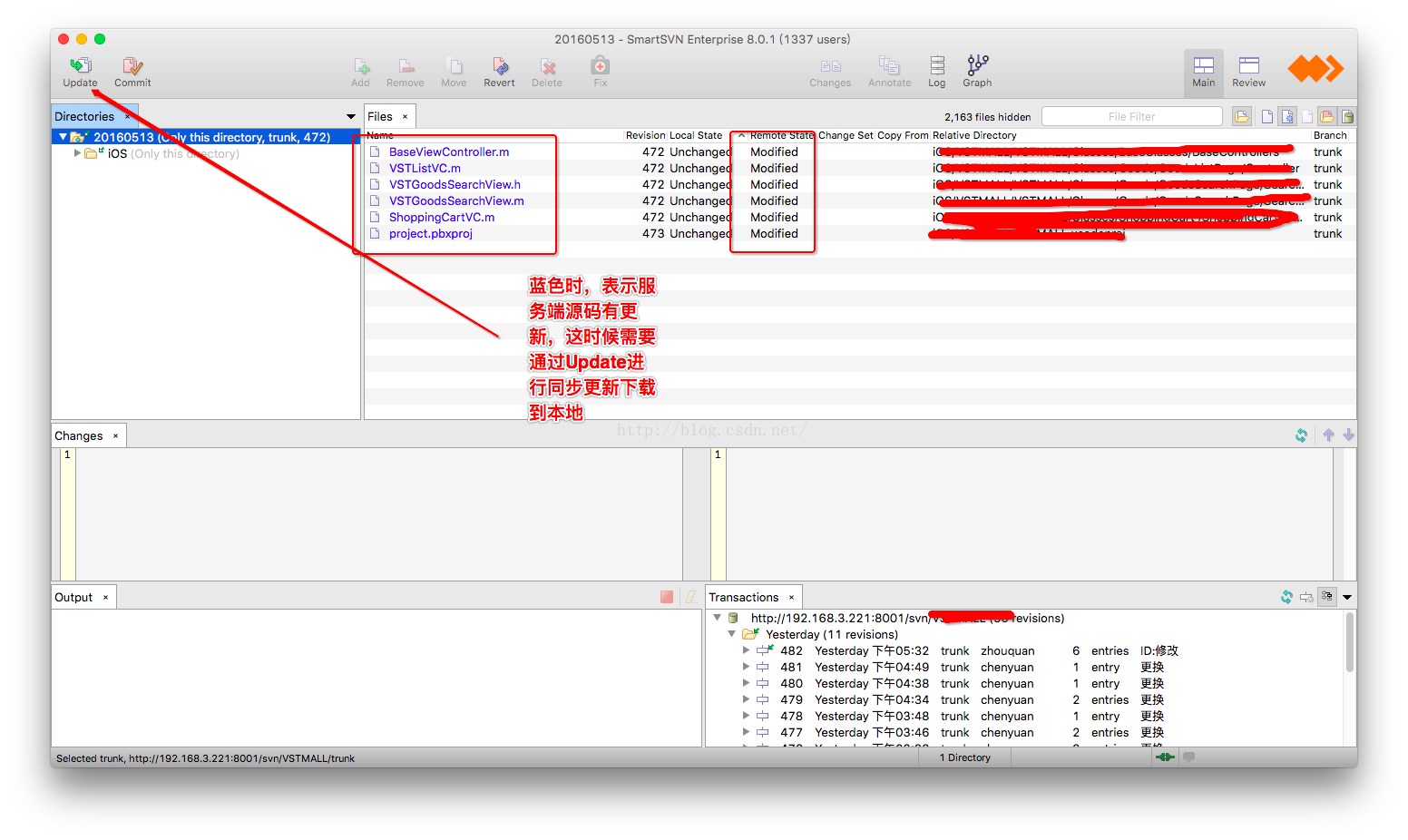
  2.接下来就是要输出项目的仓库地址。点击继续(continue)，再点继续。依次出来的是要存放项目的地址，以及获取的文件。 再一步默认，不用操作，直接点击完成（finish)。最后它会出现这种情况，说明检出成功，就是拉取代码成功。就可以开发了。



更新：

在代码开发中，要养成一个习惯，就是先更新再开发或者提交。

  1.新增文件和修改文件，都能在smartSVN上发现。只要有指向箭头，就说明文件有变化。smartSVN都能看到整个变化。



  2.左上角有update和commit，一开始都是灰的，一般先开发前就要update，点一下根级目录，左上角就变亮了。就可以update了。

  3.点击中间的一个文件正上角有add(新增) remove(删除) move(移动) revert(回退) delete(删除) fix

     \* add 代表是文件新增，可以提交上去。

     \* remove和delete的区别应该在于新增的文件和已存在的文件的删除不同。

     \* move 代表移动文件，平常来说，这基本上不用，因为开发中就已经弄好路径了。

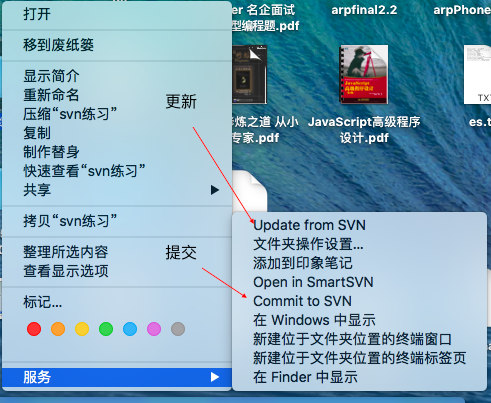
     \* revert 版本回退，就是代码写的不对，可以退到上一个版本的状态。

 提交：

  提交前也要update。确定提交了，直接点击左上角的commit就可以。一般开发时提交需要备注，写明开发了哪些。

 最简单的方式

  就是这个弄法了，点击文件的服务，就可以update和commit，但是不能准确知道哪些需要提交和修改的文件，建议还是用smartSVN来update和commit为好。



冲突问题：

所谓冲突问题，就是开发人员1也修改了这文件并提交了，开发人员2也修改了这文件并提交，但提交时发生问题，这问题就是冲突问题。(这就说明先更新再开发的重要性)

解决办法就是，找到当前的文件，先更新一下再提交，如果成了提交，没啥问题，如果还是失败，那说明都改了同一内容，需要打开相应的文件，会发现多了几个文件和压缩包。并且改的文件也有以下符号。

<<<<<<.mine ==== 当前用户修改的内容；

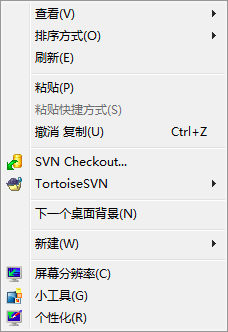
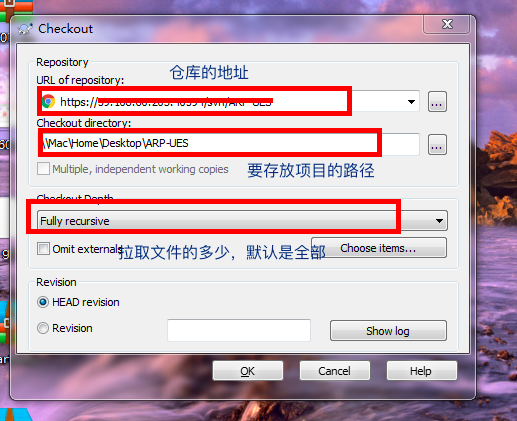
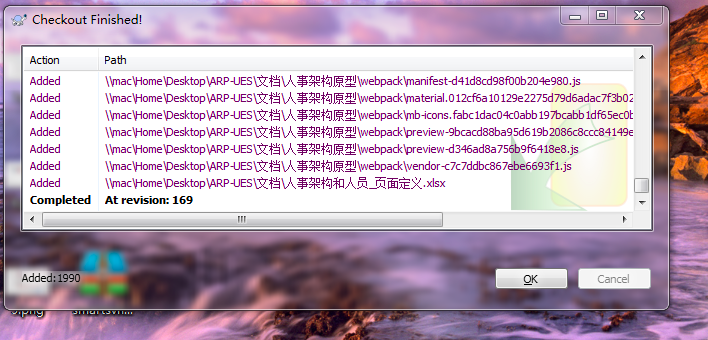
　　====到>>>>>.r10 版本库中的内容：

根据情况进行对比删除。并把以上的符号删了，再提交。

在**TortoiseSVN**中：

 拉取：

  在某个文件夹点击右键 选中check out 会出现此弹窗，把项目仓库地址填进去，点击ok，会出现账号和密码，就是分配的账号密码，输入后，ok就check out界面。出现complete时代表拉取成功。 打开拉取的文件夹会发现都是带有绿色对勾的，说明文件是最新版本的

更新**:**

 还是原来的那句话，先更新再开发。

提交：

 点击某个文件就可以提交。

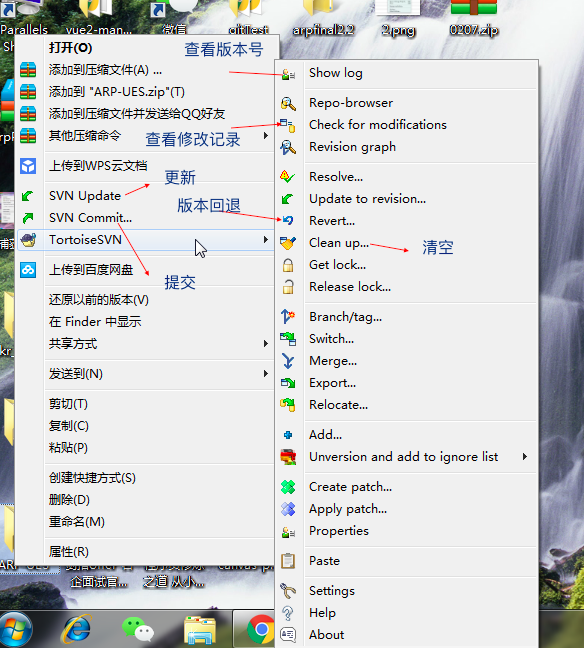
 一般开发的时候可以简单通过文件的左角标来判断对文件的操作。

   \* 带问号的角标： 说明版本库里不存在此文件，属于新增的文件。

   \* 带对勾的角标： 说明是最新的文件，和版本库一致。

   \* 带感叹号的角标： 说明此文件已经修改。

我在项目开发时，一般都是这几个流程。



 冲突问题：

目前不能用svn进行版本控制了，再加上不会svn的服务端的配置。所以没办法制造冲突问题，(上一个的smartSVN的版本控制是我在回忆中想到的，不知道对不对，感觉都是svn，差别不大)有一个博主的博文写的不错，如果有冲突问题没法解决的可以去看看。

<https://www.cnblogs.com/xing901022/p/4402686.html>

结尾：

这几天闲了学习了git，打算整一个笔记给自己复习用，又想到前段时间用的是svn进行版本控制，于是先整理了svn，但是美中不足的是，项目已开发完，没法进行操作，所以在截图和解释上略有空洞，

svn是不难，蛮好用的，用的多了，就能一眼看出来问题，所以还是希望多练习练习。下一部分复习git的使用

git

安装

git官网就有git各个系统的安装包，可以根据自己系统安装相应的安装包。

window的git安装包 依据国内网速原因，将安装放到网盘里，链接: <https://pan.baidu.com/s/1kWDLHDP> 密码: hkmx

默认安装就欧了，最后的结果就是在桌面点击右键，出现 git bash Here和git GUI Here 代表安装成功



操作：

在一个文件夹里点击服务进入终端(window直接右键进入git命令面板)

首先第一次使用git时，需要进行账号和邮箱绑定，因为git是分布式管理系统，每个人的电脑上都有自己的git，因此就有了每一个账号和邮箱

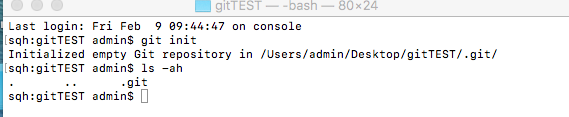
$ git config --global [user.name](http://user.name/) "Your Name"。 ----自己的名字

$ git config --global user.email ["email@example.com](mailto:%22email@example.com)" ----邮箱，我建议用gitHub的对应的邮箱就好

(--global 代表 全局 ，这台电脑都能使用这git的配置)

git在mac的终端和window的git面板的输入的命令都是相似的，对于其他的系统我没有试过。复习时我用mac的终端进行操作

初始化仓库： 新建一个空的仓库，

新建一个文件夹，启动这个文件夹的终端： git init 之后会有一个隐藏的.git的存在，在mac看不到，可以输入ls -ah查看，这就代表创建了git空仓库。结果如下：

我们开发了一个文件叫 text.html，放到该文件夹中，下面就要放到git仓库里了,按顺序依次输入：

git add text.html

git commit -m ‘第一次提交‘

git add 代表把选中的文件放到暂存区中。

（单个文件放到暂存区中 git add 文件名<有后缀名>

多个文件放到暂存区中 git add .）

git commit 是将暂存区的文件全部提交到仓库里去。 (-m代表的是提交的说明，就是引号里面的内容)

对于add和commit的注意事项：

1.只有add之后才能commit

2.add之后又在工作区修改了内容，再commit，只提交了add在暂存区的文件，工作区的依然没提交。

3.可以add多次，最后时 一次commit就可以。

4.可以参考状态的三个区别。

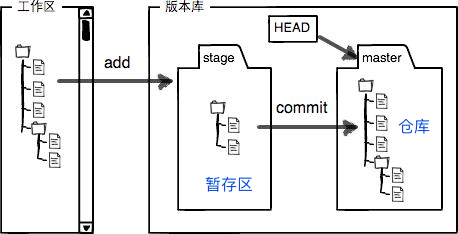
状态：

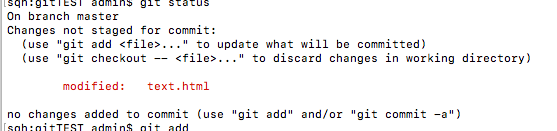
工作区和暂存区以及仓库的区别：

工作区：就是在本地看到的文件和目录

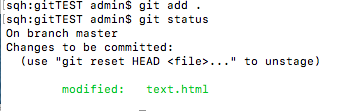
暂存区：add之后文件所在的地方

仓库：commit之后所在的地方



一般开发都会这样的操作。先add再commit。可以通过git status查看状态。如果当前的文件夹已经修改过但未add会出现红色的文件名,这说明这个文件已操作。例如：

如果已经add了但commit，git status的状态里也会出现绿色的文件名，是说明未提交，如：



可以直接通过git diff 来查看修改的内容

git diff 是工作区和暂存区的比较

git diff --cached 是暂存区和仓库的比较

如果status状态里没有任何提醒，只有这个，说明已经add并commit了。



版本回退和前进：

版本控制就是为了方便修改文件，那么提交之后就会有很多个版本，如果想要回退到某个版本重新开发，那么就可以用git reset了。

git log 查看版本历史，都有对应的commit的id，作者，提交的时间以及提交的内容

注意：如果回退到某个版本，那么git reset 之后这个版本以后的修改内容就看不到了。

git reflog 查看命令历史，通过id进行回退和前进



如果将要进行版本回退或者回退之后将要再恢复刚才的文件就进行以下操作

git reflog 通过git reflog查看版本历史，确定要回退到某个版本的id。

git reset --hard id 通过id进行回退，

根据版本回退有两种写法：

1. git reset --hard HEAD^ 这代表回退到上一个版本，若回退到上上一个版本，就 HEAD^^，依次类推，如果是100次，那就HEAD～100

2. git reset --hard id 这一步骤需要知道上一个版本的id

如果只回退上一个版本，次数不多的话那就用第一种写法，如果次数太多就用第二种写法。

撤销修改：

有的文件发现修改错误，可以用git checkout进行补救，分三种情况：

1. 文件已修改，但未add到暂存区

git checkout -- 文件名 （--代表是撤销，如果不加--代表切换分支）

2. 文件已add到暂存区，但未commit

git reset HEAD 文件名

git checkout -- 文件名

3. 文件已commit到仓库

git reset --herd HEAD^

注意：如果文件已经提交到远程库，此办法多此一举。

删除文件：

git rm 文件名 这是删除版本库的文件。

rm 文件名 删除工作区的文件

如果是工作区的文件删除，就可以add再commit即可

注意：rm后就要commit，最近修改的内容也就没了。

远程仓库

github是git所用的远程仓库，免费的也有私用的。

github的账号注册，----省略。

创建ssh key：

$ ssh-keygen -t rsa -c 'github的账号'

查看各个系统的ssh key的路径

mac 命令行输入： open ～／.ssh

window 在地址栏输入 C:\Users\Administor 即可看到

git remote add origin [git@github.com.sqh](mailto:git@github.com.sqh)17/one.git 将本地仓库和github的空仓库绑定

⚠注意

若出现 fatal：remote origin already exists。说明远程仓库已存在，已经和其他的绑定过，输入以下命令：

git remote rm origin

下一步，将本地仓库的内容推送到远程库中

git push -u origin master 将当前的本地库（已commit）推送到远程仓库 -u代表第一次推送时加上，若不是则不用。

以后本地有任何提交的，想推送，就

git push origin master

克隆仓库

github是开源社区，代码可以下载下来。用两种方式， 一个是下载压缩包的形式，一个就是克隆：

git clone [git@github.com](mailto:git@github.com):sqh17/gitTest1.git 后面的是远程仓库的地址。

分支管理

git的最大的特点就是分支管理操作。直接从远程拷贝一份到本地就可以开发了，但是为了避免在主分支master开发影响其他开发人员的进度，都会有分支，分支开发完成，在合并到主分支。

创建分支：

git checkout -b dev 创建分支dev并且切换到dev分支，

这个相当于两个操作：

git branch dev 创建分支dev

git checkout dev 切换到dev分支

⚠注意

git branch 查看分支 星号代表所在的分支

master也算分支，主分支。

合并分支：

在分支dev上开发完成时（先add后commit），需要合并到主分支，再pull到远程，其他开发人员就可以看到自己最新的代码了

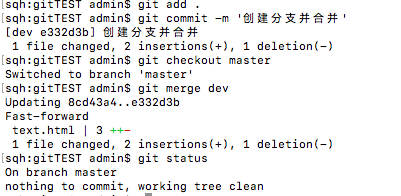
git add ...

git commit -m '...'

git checkout master 切换到主分支

git merge --no-ff -m '合并分支说明' dev 将dev分支和master分支合并起来 （--no-ff -m ''这句话是防止快速合并，以防合并冲突时发生了什么，也可以不用加）

git branch -d dev 删除dev分支（不是必要的）



解决冲突

多人开发中都会出现代码合并导致冲突问题，这是难免的，只能手动去解决冲突。

何为冲突：

a开发的分支中有某段代码修改了，并且提交了，b之前没有pull到a最新代码，他也同样改了这段代码，并且提交，git发现这两段代码和远程仓库的代码不一致。这就发生了冲突

这最根本最根本的解决办法就是及时pull最新代码，但是有时候还是避免不了代码冲突。

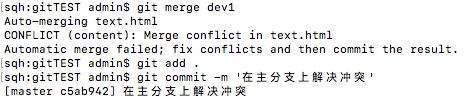
出现冲突的前兆：

当git merge时会出现以下命令：

Auto-merging ...

CONFLICT (content): Merge conflict in ...

Automatic merge failed; fix conflicts and then commit the result.



解决办法：

1.git status 查看冲突的文件

2.打开冲突的文件，会发现以下代码：

<<<<<<< HEAD

-----这是版本库里的内容-----

=======

-----这是自己分支的内容-----

>>>>>>> dev



3. 根据情况选择对的代码，删除HEAD一行和dev一行以及中间的=====一行，留下有用的。

4. git add

5. git commit -m

6. git branch -d dev

⚠注意

可以根据git log --graph查看分支图

以下是一些示例 -

$ git pull <远程主机名> <远程分支名>:<本地分支名>

Shell

比如，要取回origin主机的next分支，与本地的master分支合并，需要写成下面这样 -

$ git pull origin next:master

Shell

如果远程分支(next)要与当前分支合并，则冒号后面的部分可以省略。上面命令可以简写为：

$ git pull origin next

Shell

上面命令表示，取回origin/next分支，再与当前分支合并。实质上，这等同于先做git fetch，再执行git merge。

$ git fetch origin

$ git merge origin/next

Shell

在某些场合，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也允许手动建立追踪关系。

$ git branch --set-upstream master origin/next

Shell

上面命令指定master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，git pull就可以省略远程分支名。

$ git pull origin

Shell

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”(remote-tracking branch)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

Shell

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>

Shell

**git fetch和git pull的区别**

1. *git fetch*：相当于是从远程获取最新版本到本地，不会自动合并。

$ git fetch origin master

$ git log -p master..origin/master

$ git merge origin/master

Shell

以上命令的含义：

* 首先从远程的origin的master主分支下载最新的版本到origin/master分支上
* 然后比较本地的master分支和origin/master分支的差别
* 最后进行合并

上述过程其实可以用以下更清晰的方式来进行：

$ git fetch origin master:tmp

$ git diff tmp

$ git merge tmp

Shell

2. *git pull*：相当于是从远程获取最新版本并merge到本地

git pull origin master

Shell

上述命令其实相当于git fetch 和 git merge  
在实际使用中，git fetch更安全一些，因为在merge前，我们可以查看更新情况，然后再决定是否合并。

临时分支

在项目开发时，都会有bug出现，我们不得不放下手中的项目，去解决bug，但是当前的项目未完成，不能提交，这时候临时分支就用上了。

1 git stash 把当前工作现场储藏起来，等恢复现场后再继续工作。

2 下一步就按照创建分支，合并分支来解决bug即可。

（通过使用git stash list命令来查看已存在更改的列表。）

3 恢复现场有两种方式：stash内容是临时分支的内容？

a. git stash apply 恢复但stash内容并不删除，要结合git stash drop来删除

（恢复指定的stash git stash apply stash@{0}

b. git stash pop 恢复的同时把stash的内容删除。（只需执行git stash pop命令即可从堆栈中删除更改并将其放置在当前工作目录中。）

多人协作

查看远程仓库的信息

git remote -v （-v代表显示更详细的信息）

如果团队用git开发，那么每个开发人员都会建立自己的分支，在自己的分支上开发，要及时从远程仓库git pull

步骤：

1 先git push origin branch-name

2 如果失败了，是因为远程分支比自己的先更新，则先git pull

3 成功之后

4 如果有冲突问题，按照冲突解决办法解决，再提交

5 最后git push origin branch-name

⚠

git pull出现no tracking information

则说明本地分支和远程分支的链接关系没有创建，用命令

git branch --set-upstream branch-name origin/branch-name。