****一：Git是什么？****

       Git是目前世界上最先进的分布式版本控制系统。

****二：SVN与Git的最主要的区别？****

      SVN是集中式版本控制系统，版本库是集中放在中央服务器的，而干活的时候，用的都是自己的电脑，所以首先要从中央服务器哪里得到最新的版本，然后干活，干完后，需要把自己做完的活推送到中央服务器。集中式版本控制系统是必须联网才能工作，如果在局域网还可以，带宽够大，速度够快，如果在互联网下，如果网速慢的话，就纳闷了。

      Git是分布式版本控制系统，那么它就没有中央服务器的，每个人的电脑就是一个完整的版本库，这样，工作的时候就不需要联网了，因为版本都是在自己的电脑上。既然每个人的电脑都有一个完整的版本库，那多个人如何协作呢？比如说自己在电脑上改了文件A，其他人也在电脑上改了文件A，这时，你们两之间只需把各自的修改推送给对方，就可以互相看到对方的修改了。

****三：在windows上如何安装Git？****

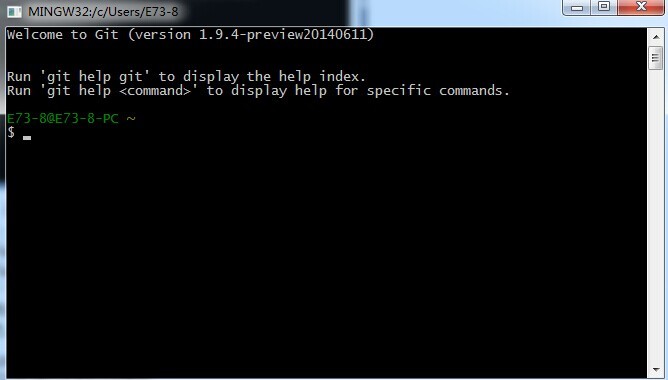
      msysgit是 windows版的Git,如下：

IMG_256

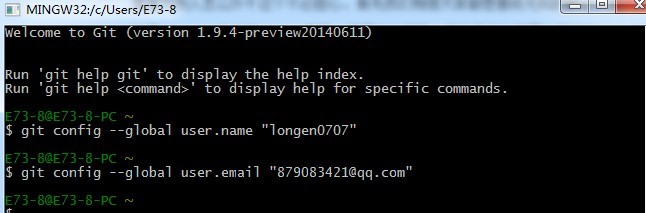
 需要从网上下载一个，然后进行默认安装即可。安装完成后，在开始菜单里面找到 “Git –> Git Bash”,如下：



会弹出一个类似的命令窗口的东西，就说明Git安装成功。如下：



安装完成后，还需要最后一步设置，在命令行输入如下：



  因为Git是分布式版本控制系统，所以需要填写用户名和邮箱作为一个标识。

****注意：****git config  –global 参数，有了这个参数，表示你这台机器上所有的Git仓库都会使用这个配置，当然你也可以对某个仓库指定的不同的用户名和邮箱。

****四：如何操作？****

     一：创建版本库。

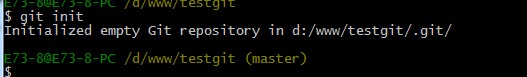
     什么是版本库？版本库又名仓库，英文名repository,你可以简单的理解一个目录，这个目录里面的所有文件都可以被Git管理起来，每个文件的修改，删除，Git都能跟踪，以便任何时刻都可以追踪历史，或者在将来某个时刻还可以将文件”还原”。

    所以创建一个版本库也非常简单，如下我是D盘 –> www下 目录下新建一个testgit版本库。

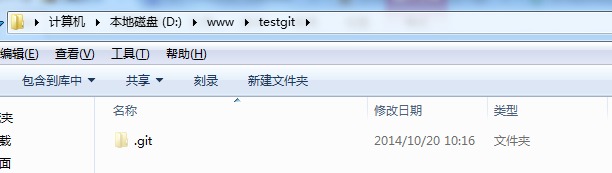


pwd 命令是用于显示当前的目录。

      1. 通过命令 git init 把这个目录变成git可以管理的仓库，如下：



     这时候你当前testgit目录下会多了一个.git的目录，这个目录是Git来跟踪管理版本的，没事千万不要手动乱改这个目录里面的文件，否则，会把git仓库给破坏了。如下：



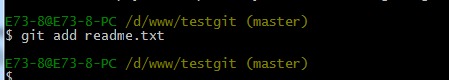
      2. 把文件添加到版本库中。

          首先要明确下，所有的版本控制系统，只能跟踪文本文件的改动，比如txt文件，网页，所有程序的代码等，Git也不列外，版本控制系统可以告诉你每次的改动，但是图片，视频这些二进制文件，虽能也能由版本控制系统管理，但没法跟踪文件的变化，只能把二进制文件每次改动串起来，也就是知道图片从1kb变成2kb，但是到底改了啥，版本控制也不知道。

****下面先看下demo如下演示：****

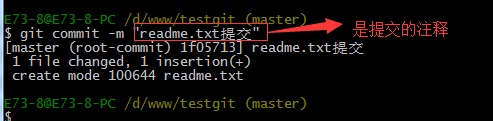
     我在版本库testgit目录下新建一个记事本文件 readme.txt 内容如下：11111111

     第一步：使用命令 git add readme.txt添加到暂存区里面去。如下：

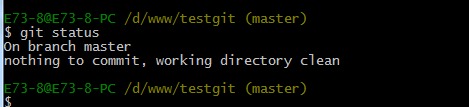


   如果和上面一样，没有任何提示，说明已经添加成功了。

   第二步：用命令 git commit告诉Git，把文件提交到仓库。



  现在我们已经提交了一个readme.txt文件了，我们下面可以通过命令git status来查看是否还有文件未提交，如下：



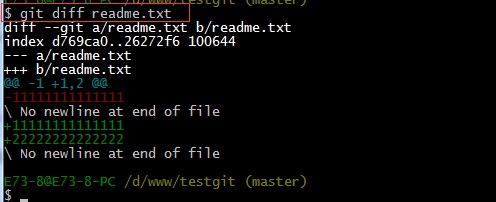
 说明没有任何文件未提交，但是我现在继续来改下readme.txt内容，比如我在下面添加一行2222222222内容，继续使用git status来查看下结果，如下：



上面的命令告诉我们 readme.txt文件已被修改，但是未被提交的修改。

接下来我想看下readme.txt文件到底改了什么内容，如何查看呢？可以使用如下命令：

git diff readme.txt 如下：



如上可以看到，readme.txt文件内容从一行11111111改成 二行 添加了一行22222222内容。

知道了对readme.txt文件做了什么修改后，我们可以放心的提交到仓库了，提交修改和提交文件是一样的2步(第一步是git add  第二步是：git commit)。

如下：



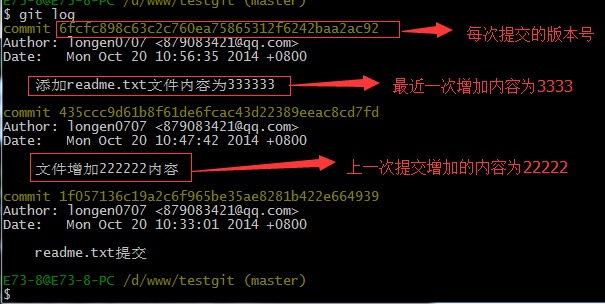
二：版本回退：

     如上，我们已经学会了修改文件，现在我继续对readme.txt文件进行修改，再增加一行

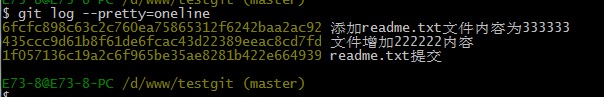
内容为33333333333333.继续执行命令如下：



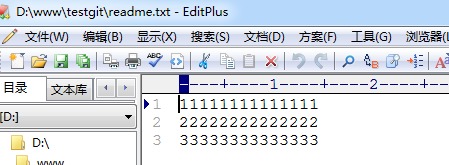
现在我已经对readme.txt文件做了三次修改了，那么我现在想查看下历史记录，如何查呢？我们现在可以使用命令 git log 演示如下所示：



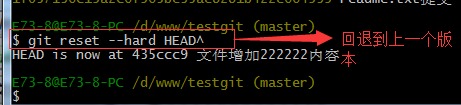
    git log命令显示从最近到最远的显示日志，我们可以看到最近三次提交，最近的一次是,增加内容为333333.上一次是添加内容222222，第一次默认是 111111.如果嫌上面显示的信息太多的话，我们可以使用命令 git log –pretty=oneline 演示如下：



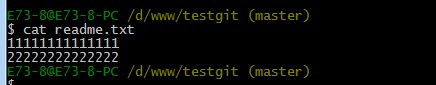
    现在我想使用版本回退操作，我想把当前的版本回退到上一个版本，要使用什么命令呢？可以使用如下2种命令，第一种是：git reset  –hard HEAD^ 那么如果要回退到上上个版本只需把HEAD^ 改成 HEAD^^ 以此类推。那如果要回退到前100个版本的话，使用上面的方法肯定不方便，我们可以使用下面的简便命令操作：git reset  –hard HEAD~100 即可。未回退之前的readme.txt内容如下：



如果想回退到上一个版本的命令如下操作：



再来查看下 readme.txt内容如下：通过命令cat readme.txt查看



可以看到，内容已经回退到上一个版本了。我们可以继续使用git log 来查看下历史记录信息，如下：



我们看到 增加333333 内容我们没有看到了，但是现在我想回退到最新的版本，如：有333333的内容要如何恢复呢？我们可以通过版本号回退，使用命令方法如下：

git reset  –hard 版本号 ，但是现在的问题假如我已经关掉过一次命令行或者333内容的版本号我并不知道呢？要如何知道增加3333内容的版本号呢？可以通过如下命令即可获取到版本号：git reflog  演示如下：



通过上面的显示我们可以知道，增加内容3333的版本号是 6fcfc89.我们现在可以命令

git reset  –hard 6fcfc89来恢复了。演示如下：



可以看到 目前已经是最新的版本了。

三：理解工作区与暂存区的区别？

****工作区：****就是你在电脑上看到的目录，比如目录下testgit里的文件(.git隐藏目录版本库除外)。或者以后需要再新建的目录文件等等都属于工作区范畴。

****版本库(Repository)：****工作区有一个隐藏目录.git,这个不属于工作区，这是版本库。其中版本库里面存了很多东西，其中最重要的就是stage(暂存区)，还有Git为我们自动创建了第一个分支master,以及指向master的一个指针HEAD。

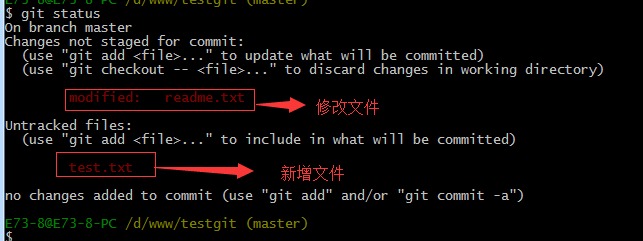
我们前面说过使用Git提交文件到版本库有两步：

  第一步：是使用 git add 把文件添加进去，实际上就是把文件添加到暂存区。

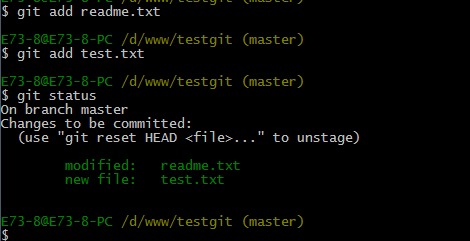
  第二步：使用git commit提交更改，实际上就是把暂存区的所有内容提交到当前分支上。

我们继续使用demo来演示下：

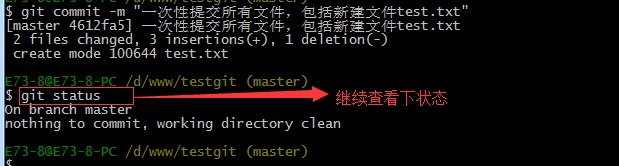
我们在readme.txt再添加一行内容为4444444，接着在目录下新建一个文件为test.txt 内容为test，我们先用命令 git status来查看下状态，如下：



现在我们先使用git add 命令把2个文件都添加到暂存区中，再使用git status来查看下状态，如下：



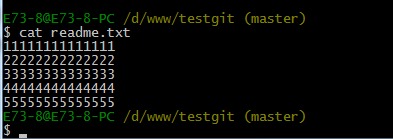
接着我们可以使用git commit一次性提交到分支上，如下：



四：Git撤销修改和删除文件操作。

****一：撤销修改：****

    比如我现在在readme.txt文件里面增加一行 内容为555555555555，我们先通过命令查看如下：

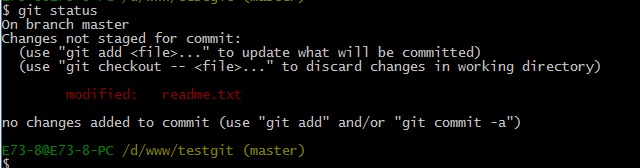


在我未提交之前，我发现添加5555555555555内容有误，所以我得马上恢复以前的版本，现在我可以有如下几种方法可以做修改：

第一：如果我知道要删掉那些内容的话，直接手动更改去掉那些需要的文件，然后add添加到暂存区，最后commit掉。

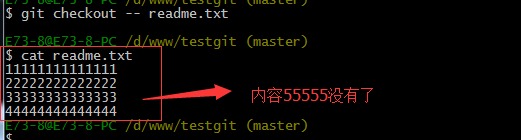
第二：我可以按以前的方法直接恢复到上一个版本。使用 git reset  –hard HEAD^

但是现在我不想使用上面的2种方法，我想直接想使用撤销命令该如何操作呢？首先在做撤销之前，我们可以先用 git status 查看下当前的状态。如下所示：



可以发现，Git会告诉你，git checkout  — file 可以丢弃工作区的修改，如下命令：

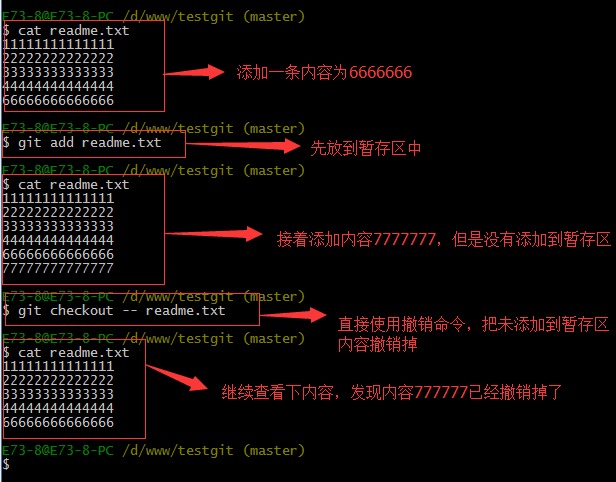
git checkout  —  readme.txt,如下所示：



命令 git checkout –readme.txt 意思就是，把readme.txt文件在工作区做的修改全部撤销，这里有2种情况，如下：

1. readme.txt自动修改后，还没有放到暂存区，使用 撤销修改就回到和版本库一模一样的状态。
2. 另外一种是readme.txt已经放入暂存区了，接着又作了修改，撤销修改就回到添加暂存区后的状态。

对于第二种情况，我想我们继续做demo来看下，假如现在我对readme.txt添加一行 内容为6666666666666，我git add 增加到暂存区后，接着添加内容7777777，我想通过撤销命令让其回到暂存区后的状态。如下所示：



****注意：****命令git checkout — readme.txt 中的 — 很重要，如果没有 — 的话，那么命令变成创建分支了。

****二：删除文件。****

     假如我现在版本库testgit目录添加一个文件b.txt,然后提交。如下：

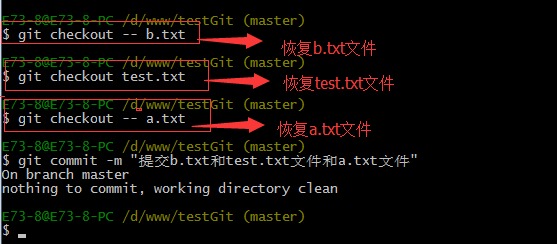


如上：一般情况下，可以直接在文件目录中把文件删了，或者使用如上rm命令：rm b.txt ，如果我想彻底从版本库中删掉了此文件的话，可以再执行commit命令 提交掉，现在目录是这样的，

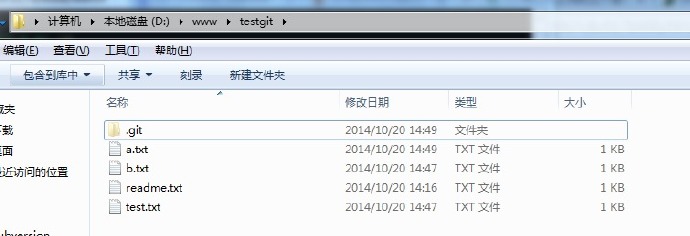


只要没有commit之前，如果我想在版本库中恢复此文件如何操作呢？

可以使用如下命令 git checkout  — b.txt，如下所示：



再来看看我们testgit目录，添加了3个文件了。如下所示：

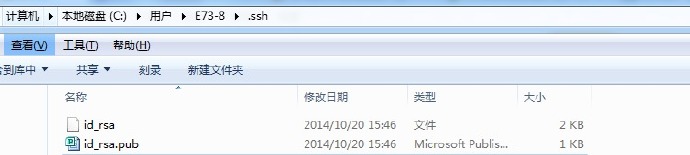


五：远程仓库。

     在了解之前，先注册github账号，由于你的本地Git仓库和github仓库之间的传输是通过SSH加密的，所以需要一点设置：

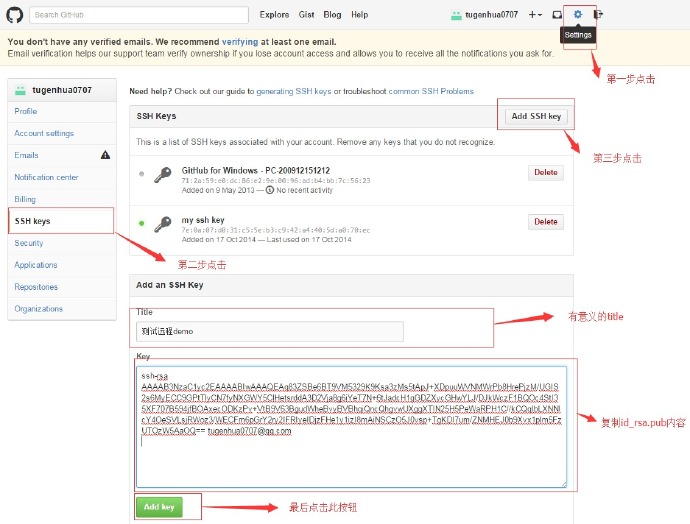
     第一步：创建SSH Key。在用户主目录下，看看有没有.ssh目录，如果有，再看看这个目录下有没有id\_rsa和id\_rsa.pub这两个文件，如果有的话，直接跳过此如下命令，如果没有的话，打开命令行，输入如下命令：

ssh-keygen  -t rsa –C “youremail@example.com”, 由于我本地此前运行过一次，所以本地有，如下所示：

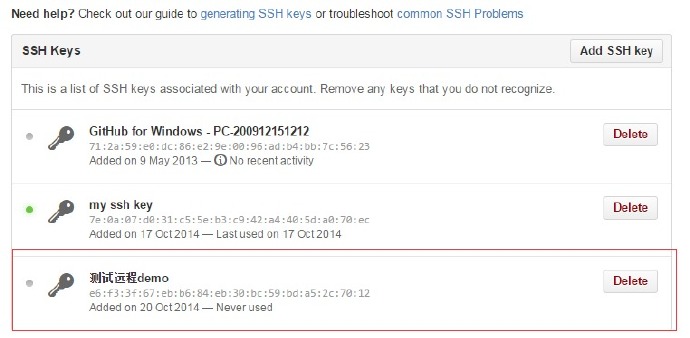


id\_rsa是私钥，不能泄露出去，id\_rsa.pub是公钥，可以放心地告诉任何人。

第二步：登录github,打开” settings”中的SSH Keys页面，然后点击“Add SSH Key”,填上任意title，在Key文本框里黏贴id\_rsa.pub文件的内容。



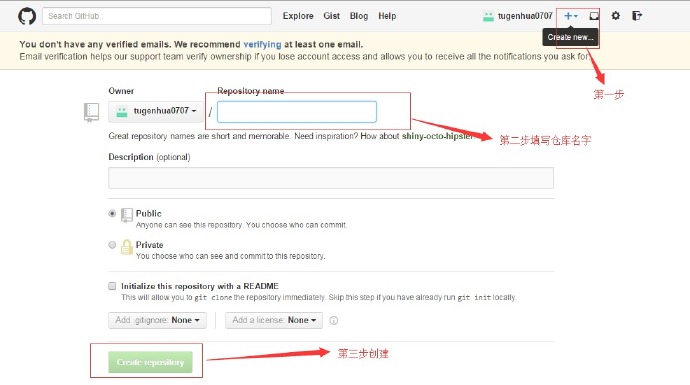
点击 Add Key，你就应该可以看到已经添加的key。



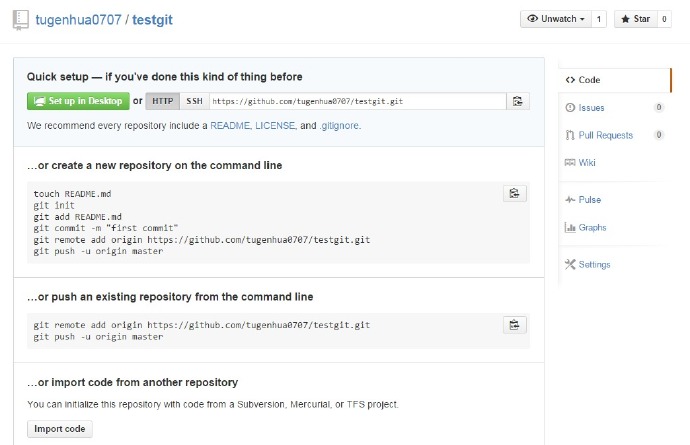
1. 如何添加远程库？

         现在的情景是：我们已经在本地创建了一个Git仓库后，又想在github创建一个Git仓库，并且希望这两个仓库进行远程同步，这样github的仓库可以作为备份，又可以其他人通过该仓库来协作。

   首先，登录github上，然后在右上角找到“create a new repo”创建一个新的仓库。如下：



在Repository name填入testgit，其他保持默认设置，点击“Create repository”按钮，就成功地创建了一个新的Git仓库：

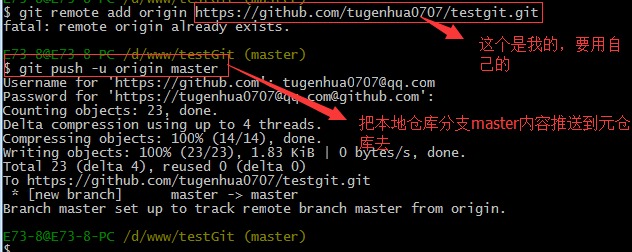


    目前，在GitHub上的这个testgit仓库还是空的，GitHub告诉我们，可以从这个仓库克隆出新的仓库，也可以把一个已有的本地仓库与之关联，然后，把本地仓库的内容推送到GitHub仓库。

现在，我们根据GitHub的提示，在本地的testgit仓库下运行命令：

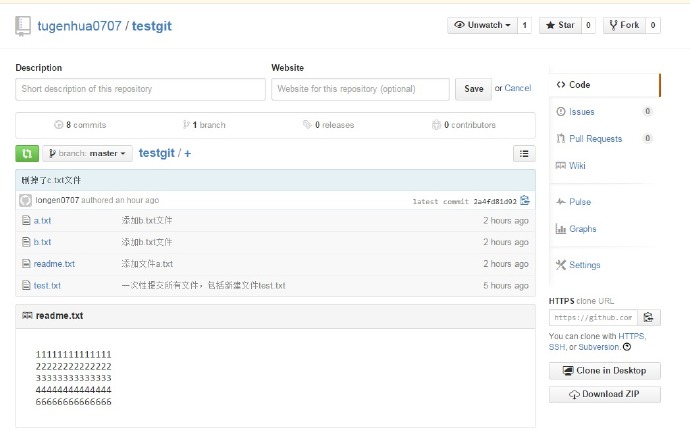
git remote add origin https://github.com/tugenhua0707/testgit.git

所有的如下：



把本地库的内容推送到远程，使用 git push命令，实际上是把当前分支master推送到远程。

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。推送成功后，可以立刻在github页面中看到远程库的内容已经和本地一模一样了，上面的要输入github的用户名和密码如下所示：



从现在起，只要本地作了提交，就可以通过如下命令：

git push origin master

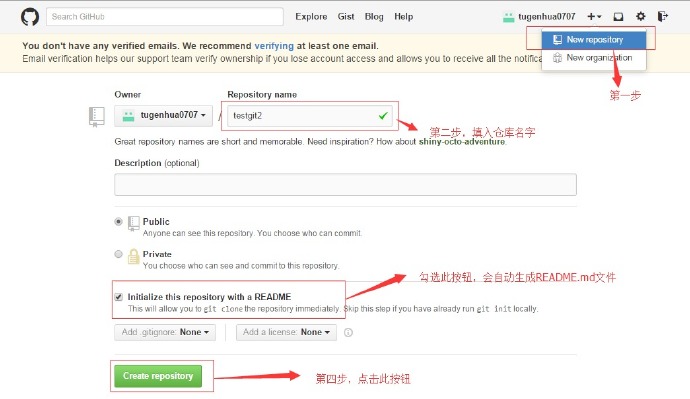
把本地master分支的最新修改推送到github上了，现在你就拥有了真正的分布式版本库了。

2. 如何从远程库克隆？

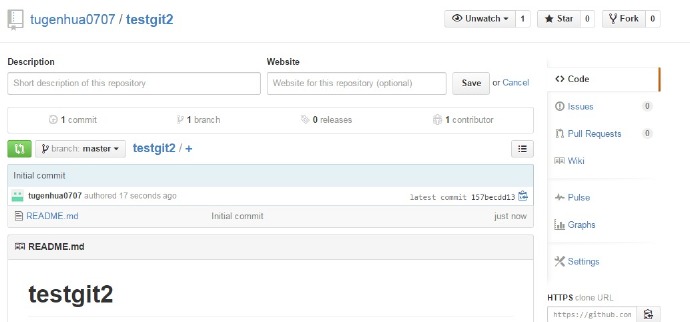
上面我们了解了先有本地库，后有远程库时候，如何关联远程库。

现在我们想，假如远程库有新的内容了，我想克隆到本地来 如何克隆呢？

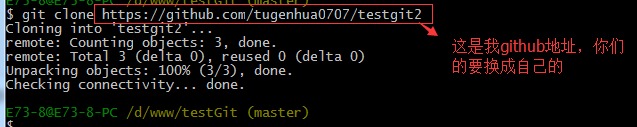
首先，登录github，创建一个新的仓库，名字叫testgit2.如下：



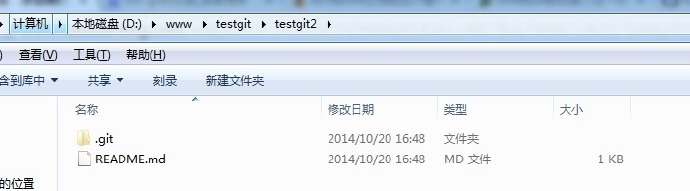
如下，我们看到：



现在，远程库已经准备好了，下一步是使用命令git clone克隆一个本地库了。如下所示：



接着在我本地目录下 生成testgit2目录了，如下所示：



六：创建与合并分支。

在  版本回填退里，你已经知道，每次提交，Git都把它们串成一条时间线，这条时间线就是一个分支。截止到目前，只有一条时间线，在Git里，这个分支叫主分支，即master分支。HEAD严格来说不是指向提交，而是指向master，master才是指向提交的，所以，HEAD指向的就是当前分支。

首先，我们来创建dev分支，然后切换到dev分支上。如下操作：



git checkout 命令加上 –b参数表示创建并切换，相当于如下2条命令

git branch dev

git checkout dev

git branch查看分支，会列出所有的分支，当前分支前面会添加一个星号。然后我们在dev分支上继续做demo，比如我们现在在readme.txt再增加一行 7777777777777

首先我们先来查看下readme.txt内容，接着添加内容77777777，如下：



现在dev分支工作已完成，现在我们切换到主分支master上，继续查看readme.txt内容如下：



现在我们可以把dev分支上的内容合并到分支master上了，可以在master分支上，使用如下命令 git merge dev 如下所示：



git merge命令用于合并指定分支到当前分支上，合并后，再查看readme.txt内容，可以看到，和dev分支最新提交的是完全一样的。

注意到上面的Fast-forward信息，Git告诉我们，这次合并是“快进模式”，也就是直接把master指向dev的当前提交，所以合并速度非常快。

合并完成后，我们可以接着删除dev分支了，操作如下：



总结创建与合并分支命令如下：

   查看分支：git branch

   创建分支：git branch name

   切换分支：git checkout name

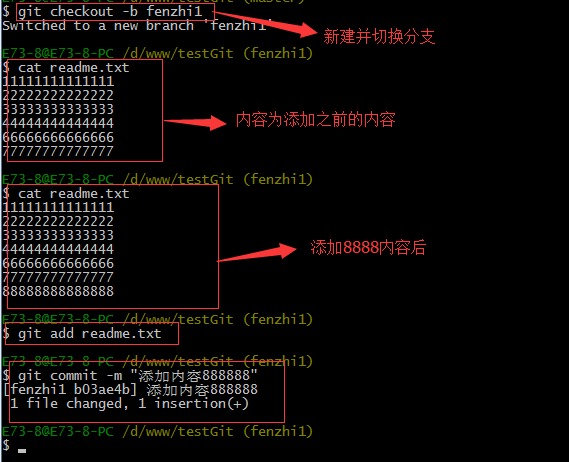
创建+切换分支：git checkout –b name

合并某分支到当前分支：git merge name

删除分支：git branch –d name

1. 如何解决冲突？

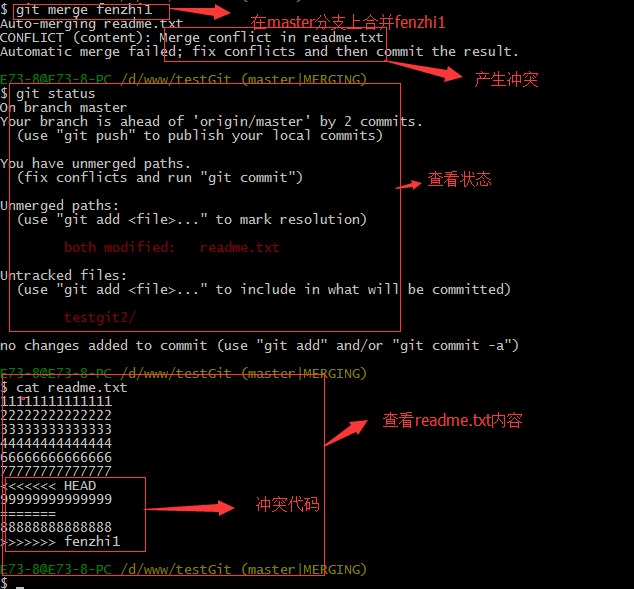
下面我们还是一步一步来，先新建一个新分支，比如名字叫fenzhi1，在readme.txt添加一行内容8888888，然后提交，如下所示：



同样，我们现在切换到master分支上来，也在最后一行添加内容，内容为99999999，如下所示：



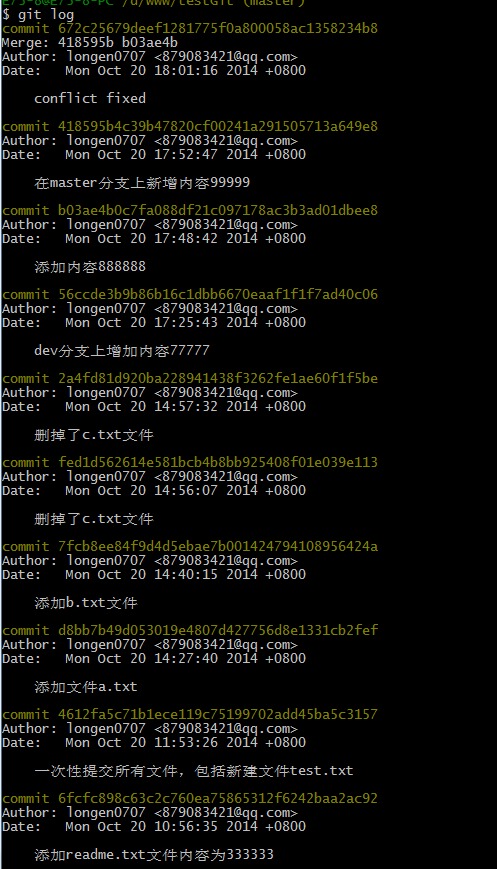
现在我们需要在master分支上来合并fenzhi1，如下操作：



Git用<<<<<<<，=======，>>>>>>>标记出不同分支的内容，其中<<<HEAD是指主分支修改的内容，>>>>>fenzhi1 是指fenzhi1上修改的内容，我们可以修改下如下后保存：



如果我想查看分支合并的情况的话，需要使用命令 git log.命令行演示如下：

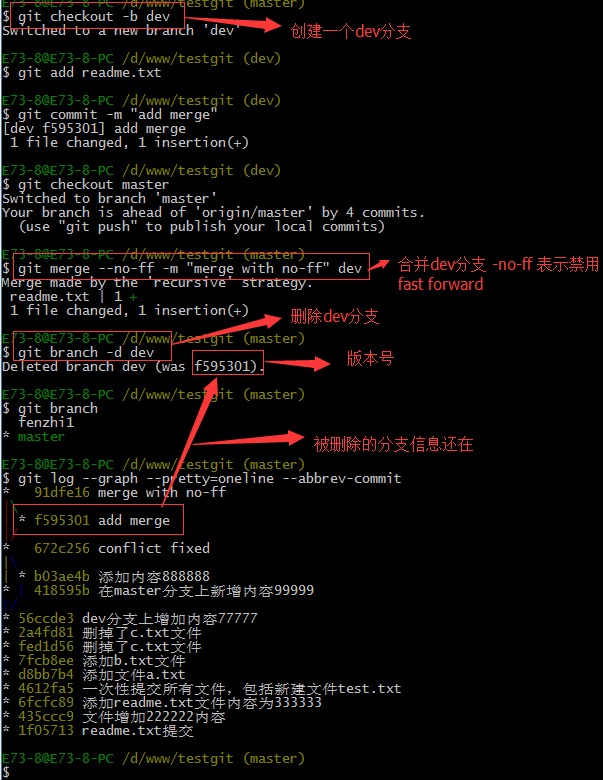


    3.分支管理策略。

      通常合并分支时，git一般使用”Fast forward”模式，在这种模式下，删除分支后，会丢掉分支信息，现在我们来使用带参数 –no-ff来禁用”Fast forward”模式。首先我们来做demo演示下：

1. 创建一个dev分支。
2. 修改readme.txt内容。
3. 添加到暂存区。
4. 切换回主分支(master)。
5. 合并dev分支，使用命令 git merge –no-ff  -m “注释” dev
6. 查看历史记录

截图如下：

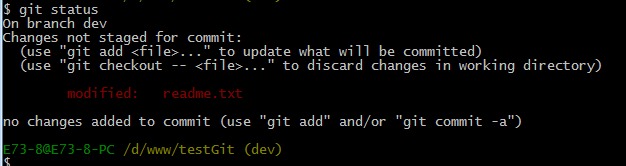


****分支策略：****首先master主分支应该是非常稳定的，也就是用来发布新版本，一般情况下不允许在上面干活，干活一般情况下在新建的dev分支上干活，干完后，比如上要发布，或者说dev分支代码稳定后可以合并到主分支master上来。

七：bug分支：

     在开发中，会经常碰到bug问题，那么有了bug就需要修复，在Git中，分支是很强大的，每个bug都可以通过一个临时分支来修复，修复完成后，合并分支，然后将临时的分支删除掉。

比如我在开发中接到一个404 bug时候，我们可以创建一个404分支来修复它，但是，当前的dev分支上的工作还没有提交。比如如下：

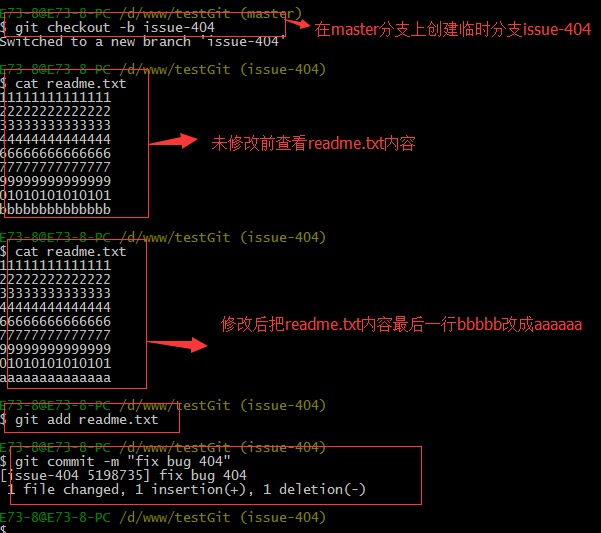


   并不是我不想提交，而是工作进行到一半时候，我们还无法提交，比如我这个分支bug要2天完成，但是我issue-404 bug需要5个小时内完成。怎么办呢？还好，Git还提供了一个stash功能，可以把当前工作现场 ”隐藏起来”，等以后恢复现场后继续工作。如下：

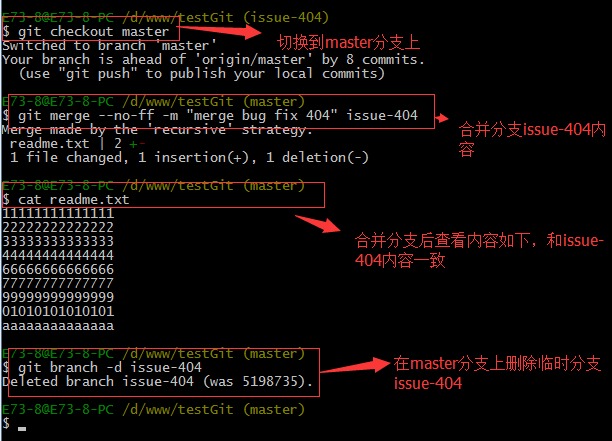


   所以现在我可以通过创建issue-404分支来修复bug了。

首先我们要确定在那个分支上修复bug，比如我现在是在主分支master上来修复的，现在我要在master分支上创建一个临时分支，演示如下：



修复完成后，切换到master分支上，并完成合并，最后删除issue-404分支。演示如下：



现在，我们回到dev分支上干活了。



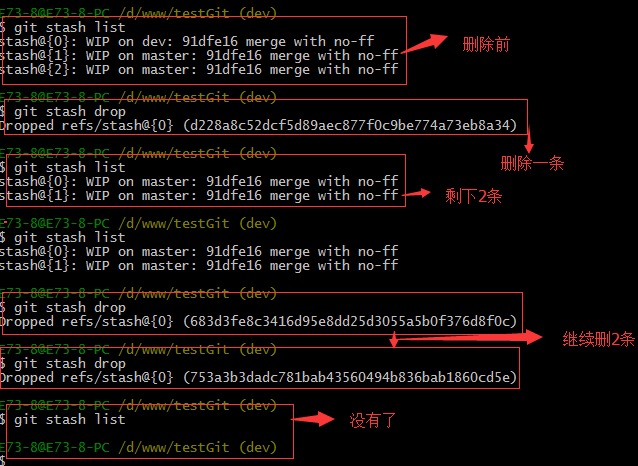
工作区是干净的，那么我们工作现场去哪里呢？我们可以使用命令 git stash list来查看下。如下：



工作现场还在，Git把stash内容存在某个地方了，但是需要恢复一下，可以使用如下2个方法：

1. git stash apply恢复，恢复后，stash内容并不删除，你需要使用命令git stash drop来删除。
2. 另一种方式是使用git stash pop,恢复的同时把stash内容也删除了。

         演示如下



八：多人协作。

当你从远程库克隆时候，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，并且远程库的默认名称是origin。

1. 要查看远程库的信息 使用 git remote
2. 要查看远程库的详细信息 使用 git remote –v

如下演示：



****一：推送分支：****

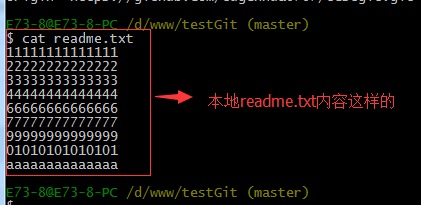
      推送分支就是把该分支上所有本地提交到远程库中，推送时，要指定本地分支，这样，Git就会把该分支推送到远程库对应的远程分支上：

      使用命令 git push origin master

比如我现在的github上的readme.txt代码如下：



本地的readme.txt代码如下：



现在我想把本地更新的readme.txt代码推送到远程库中，使用命令如下：



我们可以看到如上，推送成功，我们可以继续来截图github上的readme.txt内容 如下：



可以看到 推送成功了，如果我们现在要推送到其他分支，比如dev分支上，我们还是那个命令 git push origin dev

那么一般情况下，那些分支要推送呢？

1. master分支是主分支，因此要时刻与远程同步。
2. 一些修复bug分支不需要推送到远程去，可以先合并到主分支上，然后把主分支master推送到远程去。

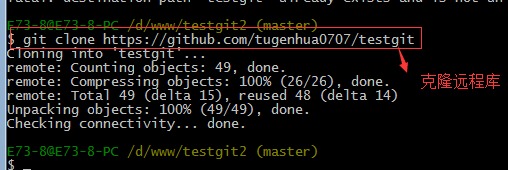
****二：抓取分支：****

多人协作时，大家都会往master分支上推送各自的修改。现在我们可以模拟另外一个同事，可以在另一台电脑上（注意要把SSH key添加到github上）或者同一台电脑上另外一个目录克隆，新建一个目录名字叫testgit2

但是我首先要把dev分支也要推送到远程去，如下



接着进入testgit2目录，进行克隆远程的库到本地来，如下：



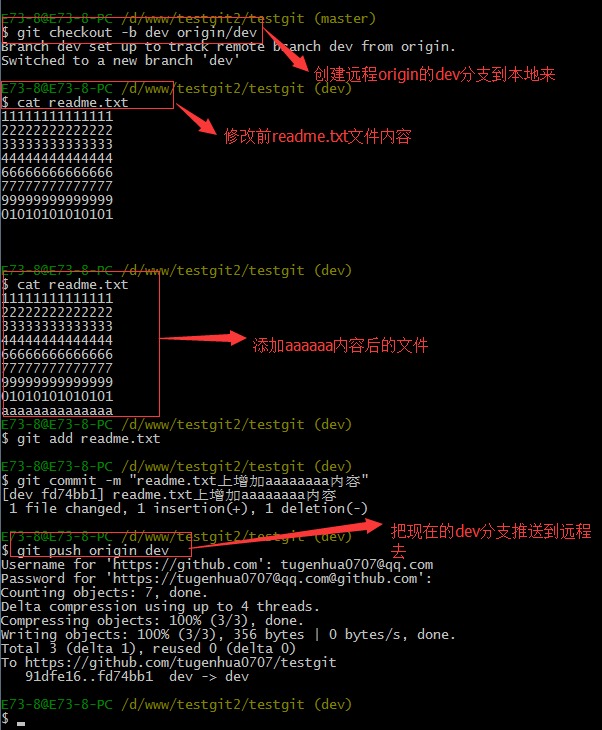
现在目录下生成有如下所示：



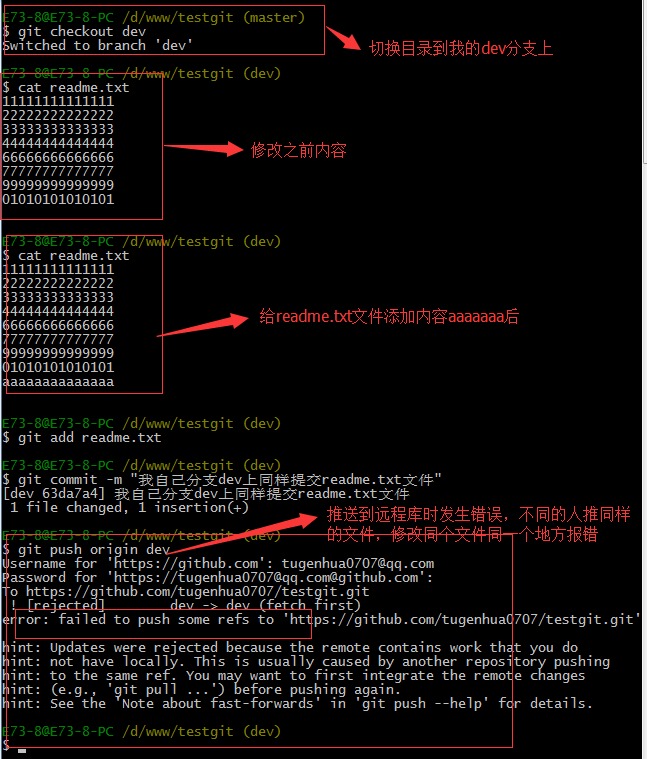
现在我们的小伙伴要在dev分支上做开发，就必须把远程的origin的dev分支到本地来，于是可以使用命令创建本地dev分支：git checkout  –b dev origin/dev

现在小伙伴们就可以在dev分支上做开发了，开发完成后把dev分支推送到远程库时。

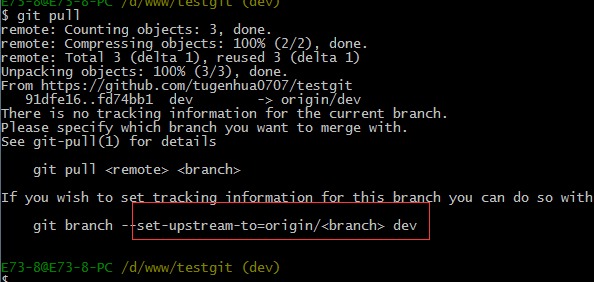
如下：



小伙伴们已经向origin/dev分支上推送了提交，而我在我的目录文件下也对同样的文件同个地方作了修改，也试图推送到远程库时，如下：



由上面可知：推送失败，因为我的小伙伴最新提交的和我试图推送的有冲突，解决的办法也很简单，上面已经提示我们，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。



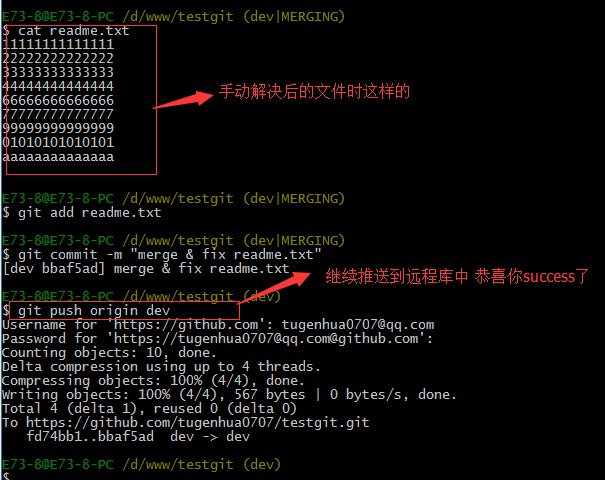
git pull也失败了，原因是没有指定本地dev分支与远程origin/dev分支的链接，根据提示，设置dev和origin/dev的链接：如下：



这回git pull成功，但是合并有冲突，需要手动解决，解决的方法和分支管理中的 解决冲突完全一样。解决后，提交，再push：

我们可以先来看看readme.txt内容了。



现在手动已经解决完了，我接在需要再提交，再push到远程库里面去。如下所示：  


因此：多人协作工作模式一般是这样的：

1. 首先，可以试图用git push origin branch-name推送自己的修改.
2. 如果推送失败，则因为远程分支比你的本地更新早，需要先用git pull试图合并。
3. 如果合并有冲突，则需要解决冲突，并在本地提交。再用git push origin branch-name推送。

****Git基本常用命令如下：****

   mkdir：         XX (创建一个空目录 XX指目录名)

   pwd：          显示当前目录的路径。

   git init          把当前的目录变成可以管理的git仓库，生成隐藏.git文件。

   git add XX       把xx文件添加到暂存区去。

   git commit –m “XX”  提交文件 –m 后面的是注释。

   git status        查看仓库状态

   git diff  XX      查看XX文件修改了那些内容

   git log          查看历史记录

   git reset  –hard HEAD^ 或者 git reset  –hard HEAD~ 回退到上一个版本

                        (如果想回退到100个版本，使用git reset –hard HEAD~100 )

   cat XX         查看XX文件内容

   git reflog       查看历史记录的版本号id

   git checkout — XX  把XX文件在工作区的修改全部撤销。

   git rm XX          删除XX文件

   git remote add origin <https://github.com/tugenhua0707/testgit> 关联一个远程库

   git push –u(第一次要用-u 以后不需要) origin master 把当前master分支推送到远程库

   git clone <https://github.com/tugenhua0707/testgit>  从远程库中克隆

   git checkout –b dev  创建dev分支 并切换到dev分支上

   git branch  查看当前所有的分支

   git checkout master 切换回master分支

   git merge dev    在当前的分支上合并dev分支

   git branch –d dev 删除dev分支

   git branch name  创建分支

   git stash 把当前的工作隐藏起来 等以后恢复现场后继续工作

   git stash list 查看所有被隐藏的文件列表

   git stash apply 恢复被隐藏的文件，但是内容不删除

   git stash drop 删除文件

   git stash pop 恢复文件的同时 也删除文件

   git remote 查看远程库的信息

   git remote –v 查看远程库的详细信息

   git push origin master  Git会把master分支推送到远程库对应的远程分支上

[git基本原理详解](http://blog.csdn.net/xiaoputao0903/article/details/23912561)

**1.什么是git**

        这个去google一下可以搜出一大堆，git就是一个软件管理器，不同一般的是它是分布式的，不仅有一个中心的服务器控制最新版本代码，而且每个开发者自己还有个本地仓库，所以在开发过程中都是先将代码提交到本地仓库再推送到中心服务器上的，这样的好处就是每个人都依赖于中心服务器来实现交互，但又不会被中心服务器限制，就算中心服务器挂了，也能很容易的找到最新版本的代码，而且我自己的工作依然可以顺利进行，提交到本地仓库，当中心服务器修复之后，再将自己仓库的东西推送到中心服务器。当然它还有很多的不同点，以后再来比较git和svn。

**2.自己的git仓库**

        我们要进行开发，提交代码和中心服务器进行交互，首先我们要有自己的一个开发基地，也就是我们自己的git仓库。拥有自己的git仓库的方式有两种，一就是在自己已有的目录里初始化自己的git仓库然后和中心服务器建立连接，更新最新代码到自己的git仓库。二就是将一个已经存在的项目克隆到自己的目录成为自己的git仓库。

        执行git init命令，就会创建并初始化git仓库，这个时候在该目录下会产生一个.git的隐藏文件夹，而该目录就是你的工作目录，你的一切行为都是在这个目录里，而这个.git文件夹就是你的本地仓库，当你进行了一些文件操作之后，认为可以提交了那么首先你就是提交到本地仓库也就是这个.git中，然后再推送到中心服务器。或者直接克隆一个仓库到本地作为git仓库，也是一样的。当你进入到这个.git目录中，会发现里面还有很多的子目录和文件，有的是很重要的，这里说几个，

config文件，这是你项目的配置文件，里面有中心服务器的信息和分支信息，

HEAD文件指向当前的分支，

index文件是暂存区的相关信息，

logs目录中都是相关操作产生的日志，这个很重要，因为日志是我们操作的唯一证据，我们本地的版本控制也靠它，

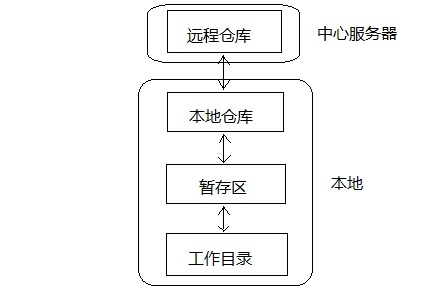
objects目录里面存储的就是所有 的数据，也就是快照，

refs目录里是存储指向数据提交对象的指针。

        乍看之下不知道说的什么，主要是里面的很多名字不知道是什么东西，下面再来看看一些名字概念，分支和提交对象以后再讲这个涉及版本的控制，我们先看最基本的。

**3.git的分层结构**

        刚开始学习git的时候，总是一头雾水，又是什么工作目录，又是什么暂存区，又是什么本地仓库，又是什么远程仓库，还尼玛快照，看的我头晕眼花的。但是只要把这几个概念弄清楚，那么最基本的开发就不是问题了。先来看看我所理解的git的分层结构：



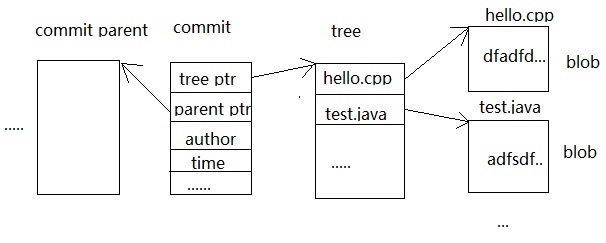
        git的工作总共分四层，其中三层是在自己本地也就是前面说的git仓库，包括了工作目录，暂存区和本地仓库，工作目录就是我们执行命令git init时所在的地方，也就是我们执行一切文件操作的地方，暂存区和本地仓库都是在.git目录，因为它们只是用来存数据的。远程仓库在中心服务器，也就是我们做好工作之后推送到远程仓库，或者从远程仓库更新下来最新代码到我们的git仓库。git所存储的都是一系列的文件快照，然后git来跟踪这些文件快照，发现哪个文件快照有变化它就会提示你需要添加到暂存区或是提交到本地仓库来保证你的工作目录是干净的。

        这个怎么理解呢，git中的文件有两种状态，一种是被跟踪的，也就是提交到本地仓库的文件，因为本地仓库要保管它们当然得跟踪他们，对它们负责，还有一种就是未被跟踪的。那么当我们添加新的文件时，它不是被跟踪的，因为本地仓库里面没有这个文件，它是外来的，本地仓库目前还不需要对他们负责。但是如果是对仓库已经存在的文件进行修改，那么这些文件就是被跟踪的文件，就可以通过git status查看他们的状态来进行相应的操作。当然我们也可以生成一个.gitignore文件，里面指定要忽略的文件类型，然后这些文件就不会被跟踪，不管怎么改变它们，git status都不会提示你需要做什么操作的。

        所以当我们在工作目录中进行文件操作后，要先添加到暂存区，然后再将暂存区中刚添加的文件快照提交到本地仓库，然后再将本地仓库的最新版本文件快照推送到远程仓库。这个文件快照其实就是各个文件的在被添加到暂存区时的状态，就和照相一样的，留下每个不同时刻的快照，方便以后查询，而git存储的就是这些一系列的快照。说到这个快照就要说说git的对象了。

**4.git的对象**

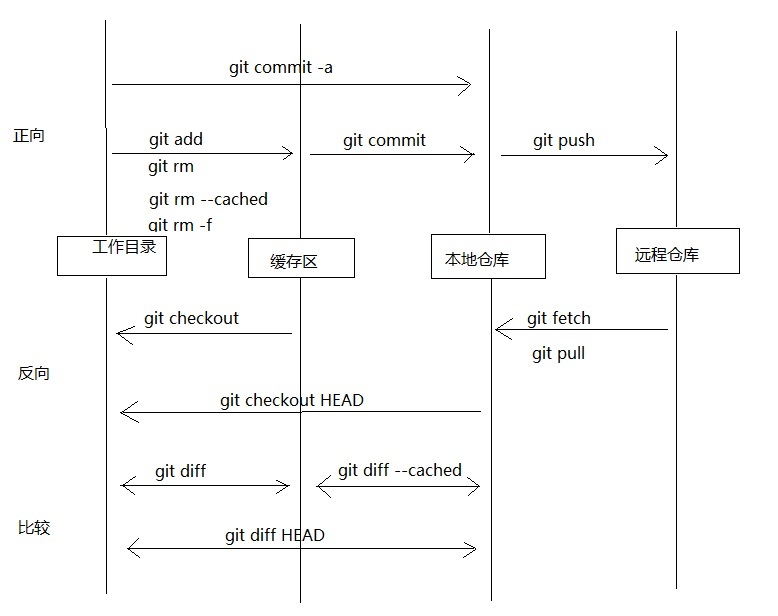
        从根本上讲，git是一套内容寻址的文件系统，它存储的也是key-value键值对，然后根据key值来查找value的，说到寻址就会想到指针吧，不错，git也是根据指针来寻址的，这些指针就存储在git的对象中。git一共有3种对象，commit对象，tree对象和blob对象。下面便是这3个对象：



        这个blob对象对应的就是文件快照中那些发生变化的文件内容，而tree对象则记录了文件快照中各个目录和文件的结构关系，它指向了被跟踪的快照，commit对象则记录了每次提交到本地仓库的文件快照，从上图看出其中有两个指针，一个指向tree对象，一个则指向上一个commit对象。这个怎么理解呢，怎么还有上一个commit对象，在开发过程中，我们会提交很多次文件快照，那么第一次提交的内容会用一个commit来记录，这个commit没有指针指向上一个commit对象，因为没有上一个commit，它是第一个，当第二次提交时，又会有另外一个commit对象来记录，那么这次commit对象中就会有一个指针指向上一次提交后的commit对象，经过很多次提交后就会有很多的commit对象，它们组成了一个链表，当我们要恢复哪个版本的时候，只要找到这个commit对象就能恢复那个版本的文件不是吗。而我们所谓的HEAD对象其实就指向最近一个提交的commit对象，也就是最后一个commit对象。

**5.git的基本操作**

        上面说了这么多东西，也该看看基本操作了，基本上的操作无非是文件的增删改和版本的提交更新回溯。当把上面的内容弄清楚之后，这些基本操作根本就是小草一碟：



       上面列出了基本上能用到的所有命令，当然还没涉及到分支，这个概念后面再说，基本的用法google一下一大堆，这里不用多说，还有就是要经常用到git status这个命令，它可以指引你该干嘛干嘛，确定你的工作目录是否干净。干净的意思就是和暂存区还有本地仓库保持一致。还有一个命令也是经常用到的，那就是git log，这个命令会列出你的操作产生的日志，有很多的信息，还有提交后的commit对象的id，这个checkout版本的时候用处很大。

**6.学习感受**

        以前在用svn的时候，基本两条命令就可以搞定，svn up，svn ci，svn di更新，提交和比较，就知道哦svn up是将服务器的代码更新到本地，哦svn ci是将本地代码提交到服务器，有冲突哦svn di解决冲突了再提交。根本不知道其结构式怎样，这些会导致什么结果，用起来总是小心翼翼，生怕把服务器搞挂了。学习就是要知其所以然，用起来才会顺手，基本原理弄懂之后，操作这些就是相关命令多用用自然搞定。我也是个git初学者。推荐一篇文章，这篇文章很深入的分析了git内部原理，以至于我都没看懂多少，惭愧：[http://www.open-open.com/lib/view/open1328070620202.html](http://www.open-open.com/lib/view/open1328070620202.html" \t "http://blog.csdn.net/xiaoputao0903/article/details/_blank)

        git分支后面再讲。话说这个图画的确实是小学生的水平啊，不过能能看懂就好，这个是用windows自带画图板画的，哈哈，不喜勿喷，有啥好用的画图软件求推荐。

[git分支branch详解](http://blog.csdn.net/xiaoputao0903/article/details/23933589)

上一篇git的基本原理[http://blog.csdn.net/xiaoputao0903/article/details/23912561](http://blog.csdn.net/xiaoputao0903/article/details/23912561" \t "http://blog.csdn.net/xiaoputao0903/article/details/_blank)，介绍了git最基本的原理和相关操作，这篇就来讲讲git的分支。

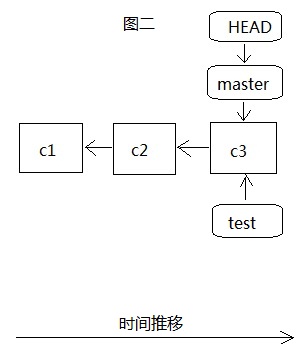
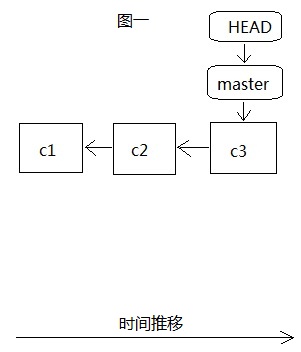
**1.git的分支是什么**

        顾名思义，分支就是从主线上分离出来进行另外的操作，而又不影响主线，主线又可以继续干它的事，是不是有点像线程，最后分支做完事后合并到主线上而分支的任务完成可以删掉了。这样是不是很方便，主线继续做它的事，分支用来解决临时需求，二者互不相干。

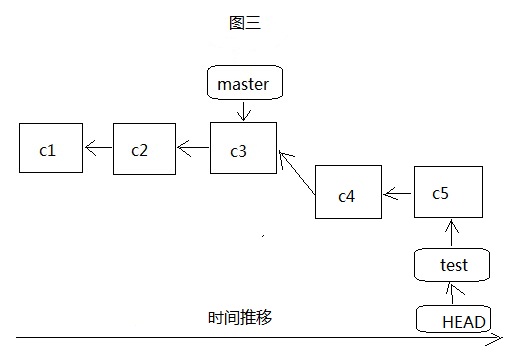
        git的分支功能特别的强大，它不需要将所有数据进行复制，只要重新创建一个分支的指针指向你需要从哪里开始创建分支的提交对象(commit)，然后进行修改再提交，那么新分支的指针就会指向你最新提交的这个commit对象，而原来分支的指针则指向你原来开发的位置，当你在哪个分支开发，HEAD就指向那个分支的最新提交对象commt。没弄清楚没关系，先有这么一个概念，后面慢慢就会弄清的。

**2.分支的新建与合并**

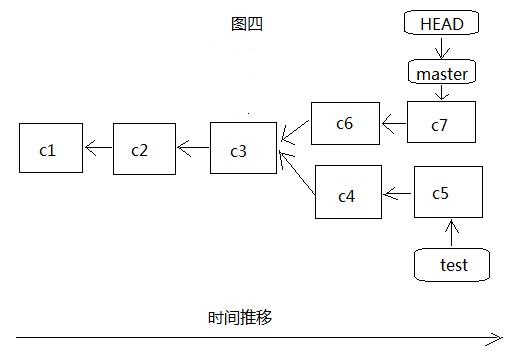
        我们可以用命令git branch来查看我们的git仓库有几个分支，而我们目前工作处于那个分支，前面有个\*号的就为我们目前所处的分支。我们可以通过命令git branch name来创建分支，而这个分支的指针就指向最新的commit对象，也就和HEAD指向同一对象。我们可以通过命令git checkout name来切换到目的分支，我们默认的主分支为master。在分支的创建和切换，其实只是简单的创建指针找指针而已，而根据找到的指针找到所指向的commit对象，然后将工作空间恢复成该commit对象所指的文件快照让我们来工作。当提交一次，指针就重新指向这个最新提交的对象，特别的简单。



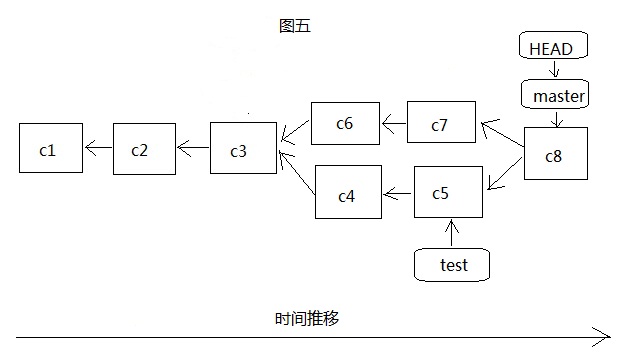
        当我们建立分支teset之前，只有master一个主分支，如图一，我们所有的开发都是在这个分支上，而且HEAD是指向最近一次提交的commit对象c3，c3以前还有两次提交c1和c2，这时我们通过git branch test创建test分支，如图二，这时HEAD还是指向master分支最近一次提交的c3，当git checkout test切换到test分支后，HEAD就指向test分支的最近一次提交c3，这个时候其实在.git里面都是指向同样一份数据c3。



        这个时候，当我们在test分支上进行了几次开发提交了c4和c5两个版本后，那么test和HEAD都指向test分支的最近一次提交c5，如图三，而master此时还没有变化，任然指向的是c3，如果这个时候将test分支合并到master分支，那么git根本不用做什么，只要将master移动，指向c5就可以了，这个过程称之为Fast-forward快进。如果此时test的任务完成，我们就可以通过git branch -d test将它删除掉，继续在主分支master上进行开发。如果是这样的话，那么test分支就白建了。



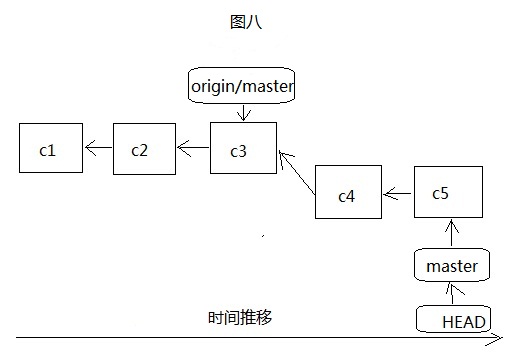
       那么如果此时master分支上又进行另外的开发，提交了两个版本c6和c7，那么此时的master和HEAD指针都指向的是c7，如图四，可以看出在哪个分支上开发，那么HEAD就指向的是哪个分支上的commit，这个时候合并两个分支的话，就如下。



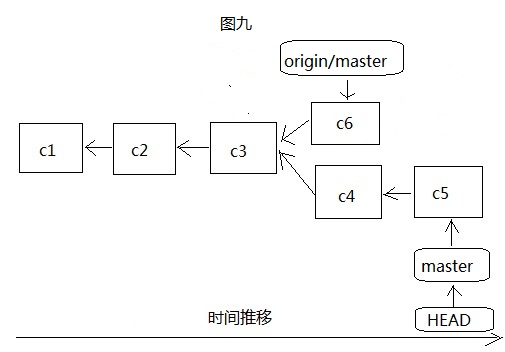
        如图五，我们先切换到master分支，然后通过git merge test将test分支合并到master分支，这个时候，git就不是简单的移动指针了，因为两边都有开发，所以git就要对于两个分支的最新提交c5和c7还有两个分支共同的祖先commit对象c3来进行一次简单的三方合并，产生新的文件快照并用新的commit对象c8记录，这个合并的过程不需要太在意，如果产生了冲突，也就是两个分支对同一个文件进行了修改，那么git就会停下合并操作，让你处理好冲突后，再提交(c8)，然后再进行合并。这时master和HEAD都指向c8，但是test是没有移动的，此时还可以在test上继续开发，再合并到master，如果test已经没有利用价值了就可以删掉了。

**3.本地分支，追踪分支和远程分支**

        这里有三个概念，本地分支就是我们可以通过git branch查看到的分支，也就是我们自己git仓库所拥有的分支，我们都可以利用。远程分支是对远程仓库的分支的索引，它其实也是本地分支，只是我们无法移动它，必须要在和中心服务器交互根据服务器更新到本地来的代码移动的，远程分支的作用就是我们上次和中心服务器交互更新得到的最新版本，它也是个指针。追踪分支比较难理解，它也是一个本地分支，只是它对应了一个远程分支，如果我们本地的某个分支对应了一个特定的远程分支，那么它就是追踪分支，比如我们最初的master分支就是一个追踪分支，它对应远程分支origin/master，这里origin是远程仓库名，当我们在master分支里执行更新(fetch，pull)或是推送(push)，在不指定分支的情况下，默认就是从origin/master分支更新来或者提交到origin/mster分支。



        从图七和图八很容易看出来，和我们本地创建分支很相似，只是origin/master远程分支只有在连接服务器并更新服务器代码到本地后才会移动，如下图九：



更新远程代码到本地有两个命令，fetch和pull，fetch是将远程代码更新到本地，但是不会执行合并操作，需要自己查看，解决冲突什么的，然后自己再执行merge将更新来的代码合并到我们自己制定的分支，但是pull就将这两个操作合成了一步，直接更新服务器代码更新并合并到到本地指定分支，当然遇到冲突也必须要自己解决。所以我们一般都使用fetch来实现更新，虽然麻烦了点，但是不容易出问题。

        将本地代码推送到远程仓库，也就是中心服务器，一般我们推送数据都是git push origin master:master，这里指定远程仓库名，本地分支名和远端分支，也就是将我们本地master分支的数据推送到远程仓库origin的master分支。如果本地的master分支是追踪分支，那么在不指定的情况下，它会自己找到远程仓库中对应的分支来推送数据。或者我们直接进行git push origin操作，只指定远程仓库名，那么git会根据我们目前所在分支和它所对应的远程仓库的分支来实现数据推送，前提是我们目前所在分支必须是追踪分支。当然如果是git push origin :master，这里本地分支名是空的，这个操作就是将空分支推送到远程仓库的master分支，结果就是将master分支删除。

        既然追踪分支这么好用，那么我们怎么建立追踪分支呢，有两种方式，第一种方式是根绝远程分支创建追踪分支，如果不指定该追踪分支的名字，默认和远程仓库的分支名字一样：git checkout --track origin/test，这样我们就建立了一个名为test的追踪分支，如果重新指定追踪分支的名字：git checkout -b name origin/test，这样我们就创建了一个名为name的追踪分支，它对应远程仓库的test分支。第二种方式是已经存在某个本地分支，要让它来对应某个远程分支来成为追踪分支，也有两个命令可以用，git branch --set-upstream test origin/test 或者git branch -f --track test origin/test 这里我们就让我们本地已经存在的test分支来追踪远程的test分支。

**4.git分支管理**

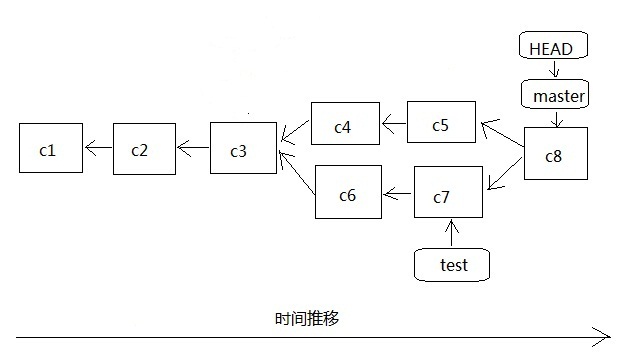
        git创建分支于合并分支是如此简单快捷，那么在我们的开发过程中可以疯狂的使用分支，而且git的核心玩法之一就是分支，非常提倡使用分支，但是是不是我们可以肆无忌惮的使用分支呢，创建这么多的分支我们要如何来管理呢，分支不在多而在恰到好处，如果分支创建多了，管理起来就麻烦了，所以推荐一种分支的管理策略，git-flow，同时推荐一篇文章来了解这种策略：[http://nvie.com/posts/a-successful-git-branching-model/](http://nvie.com/posts/a-successful-git-branching-model/" \t "http://blog.csdn.net/xiaoputao0903/article/details/_blank)，让你的git使用更加顺手。

[git merge，rebase和\*(no branch)](http://blog.csdn.net/xiaoputao0903/article/details/23995055)

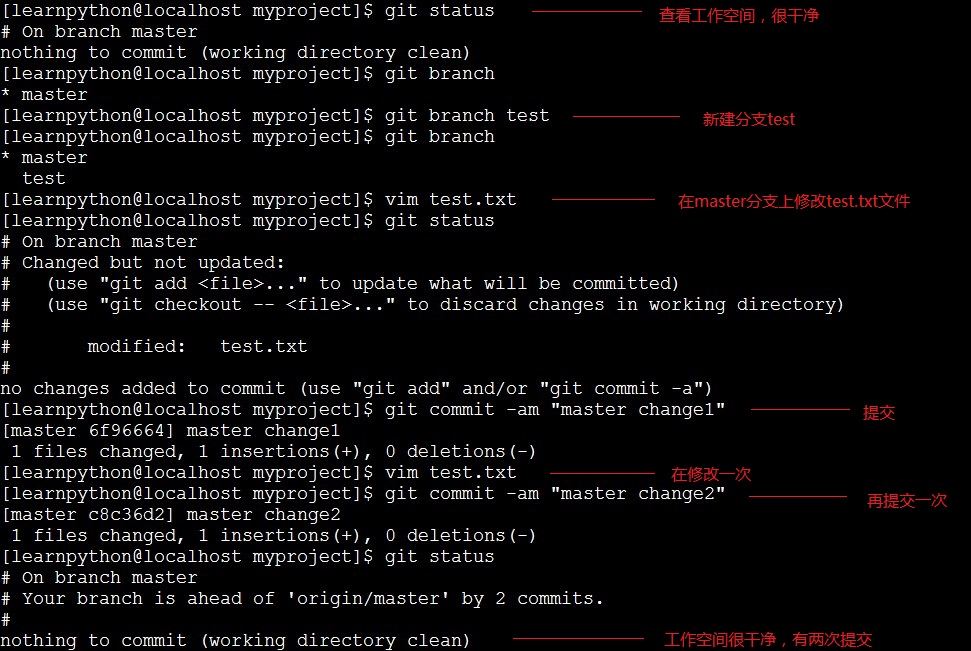
上一篇：[http://blog.csdn.net/xiaoputao0903/article/details/23933589](http://blog.csdn.net/xiaoputao0903/article/details/23933589" \t "http://blog.csdn.net/xiaoputao0903/article/details/_blank)，说了git的分支，相关的用法没说到但是只要google就能搜出一大片，这里还有几个细节要注意一下，就是merge合并和rebase合并的区别，以及\*(no branch)的处理。

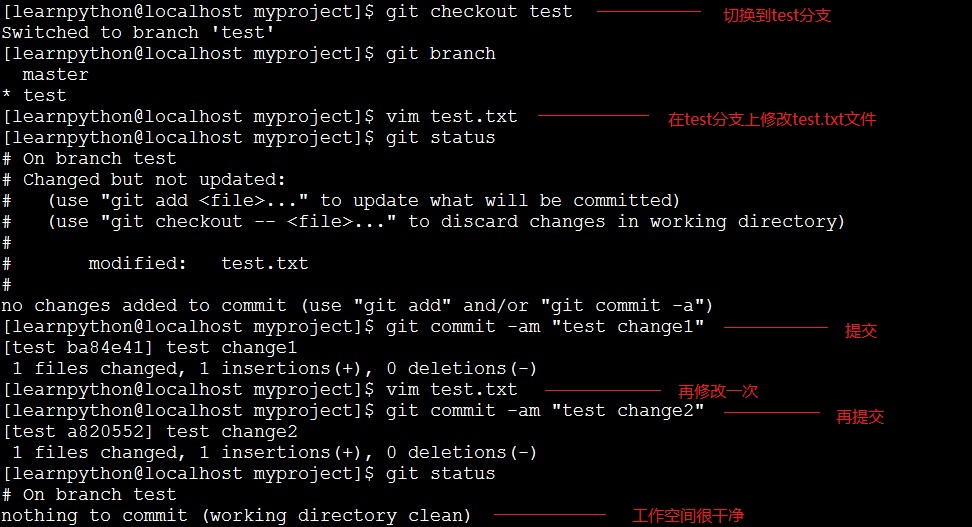
**1.merge**

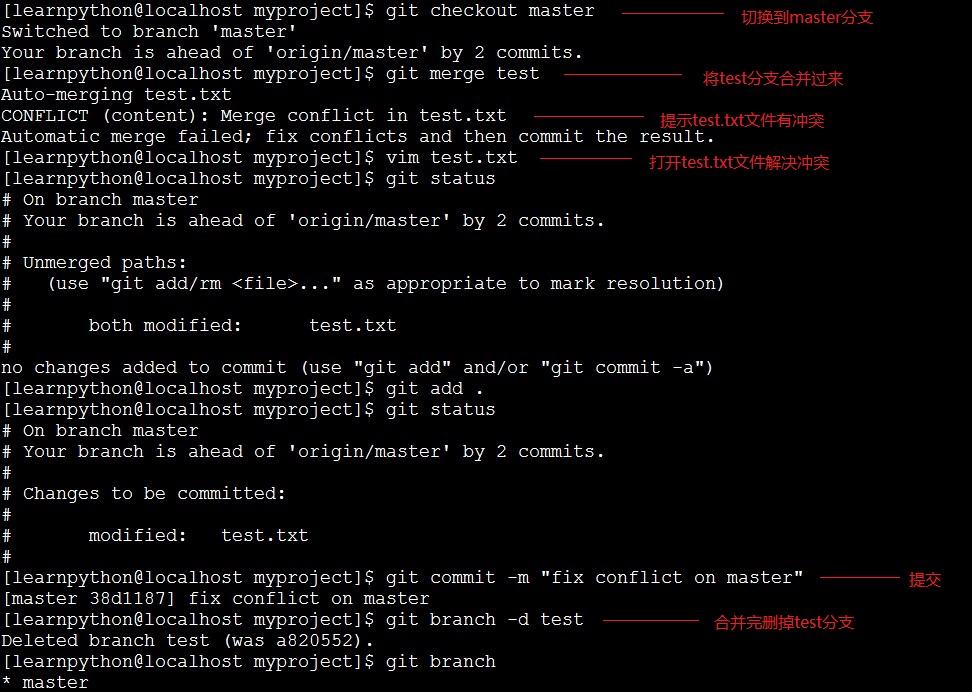
        在上篇介绍分支的时候有简单的说了一下分支的创建和合并，当时合并就是写的merge，这是根据两个不同分支的最后一次提交的commit对象c5，c7和两个分支的交叉点的commit对象c3进行一次简单的三方合并，最终得到一个新的commit来作为最终的提交commit对象c8，指针指向c8，而且c4，c5，c6，c7是存在于本地仓库的历史版本，我们可以通过日志查看找到这两个commit，同样也可以恢复到这两个版本。也就是下面这个图：



        上图是将test分支合并到master分支，然后我们来实现一次，假如现在已经到了c3，我新建一个分支test，然后先用master分支修改我的test.txt文件并提交重复两次，得到c4和c5，然后再切换到test分支同样对test.txt修改并提交两次，得到c6和c7，然后切换到master分支执行合并操作，这时会提示有冲突，最后我们解决冲突了再提交：



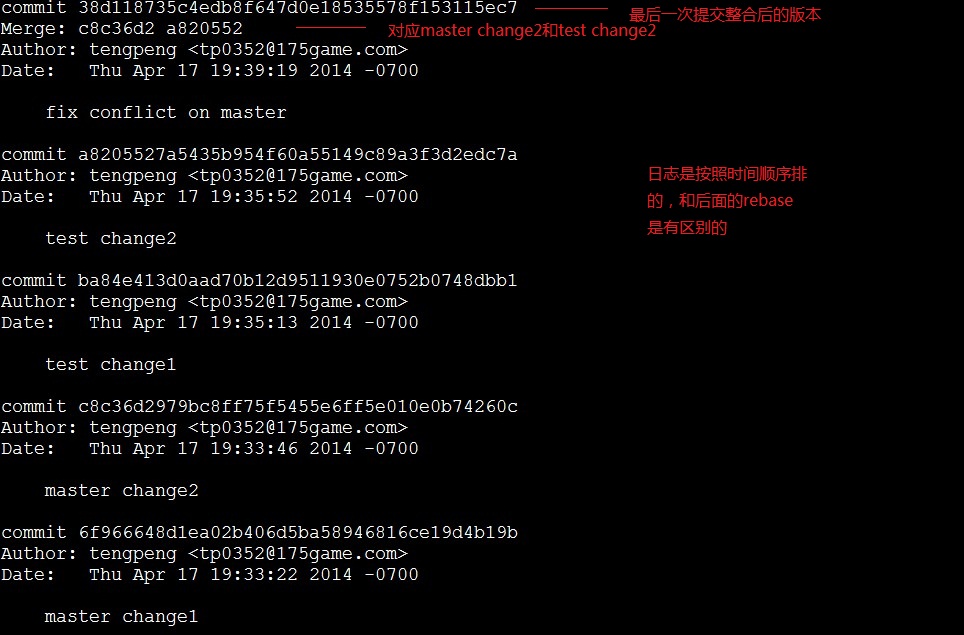




冲突：



日志：

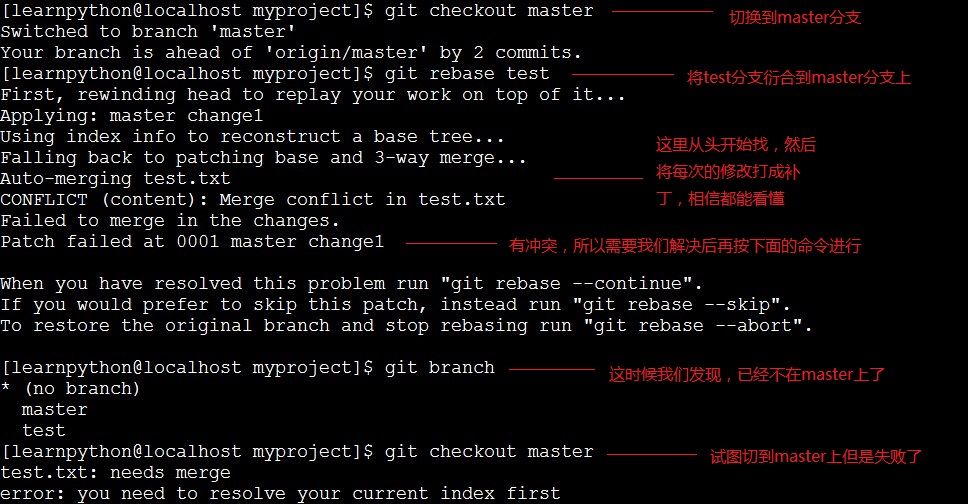


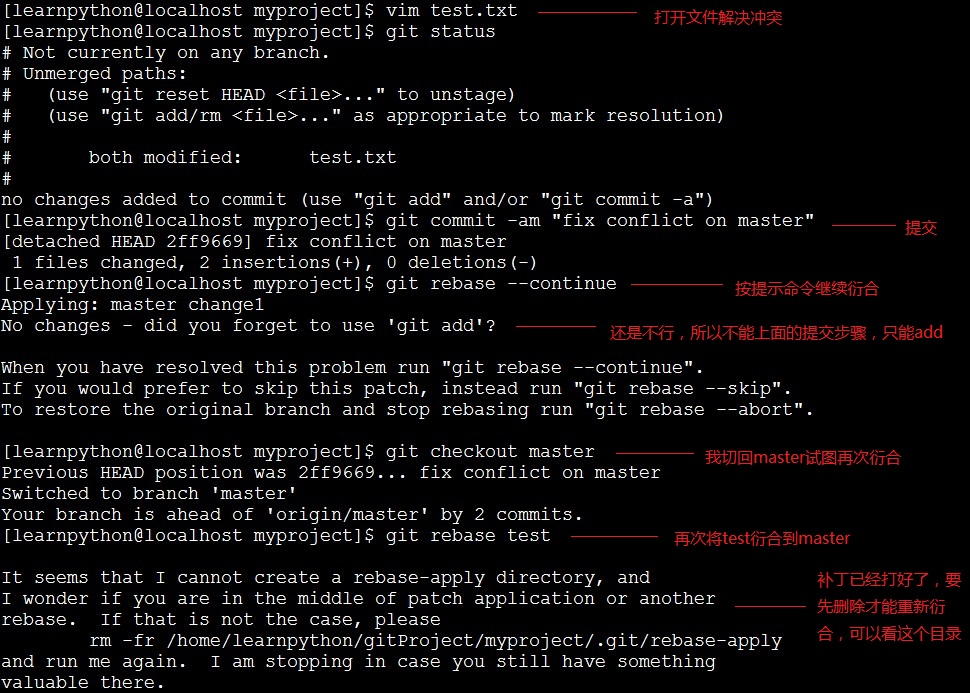
**2.rebase**

        rebase称为衍合，这是个什么概念呢，如下图，当我们把test衍合到master的时候，是将c5,和c6中发生的变化打成补丁然后再c8的基础上做修改的，如果这个时候遇到冲突，就必须要我们自己决绝好冲突之后，添加到暂存区然后再继续合并，c5的变化补丁在c8上发生变化之后得到c5'这个commit对象，而c6的变化补丁再在c5'上修改得到c6',然后指针指向c6'，这个过程我们称之为衍合，而且这个过程都是在一个\*(no branch)分支上做的，这个后面会说到。而且有一个要注意的地方，如果git执行git gc或者我们手动执行git gc，那么c5和c6就不再存在于我们的本地仓库，我们就再也找不回这两个commit了，这个过程就是一个线性的过程，在test执行完之后，再在test的基础上执行master的操作。以上就是rebase和merge的区别所在。



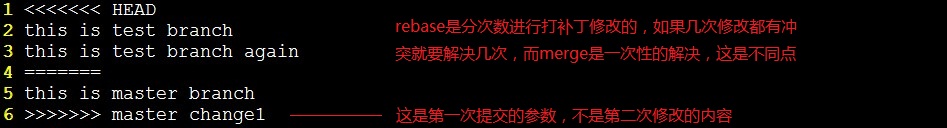
        和merge一样，我们实际来做一次，假设现在我们已经在c4了，这时候创建test分支，所有修改操作和merge一样，然后我们用rebase这种方式来把test衍合到master上：



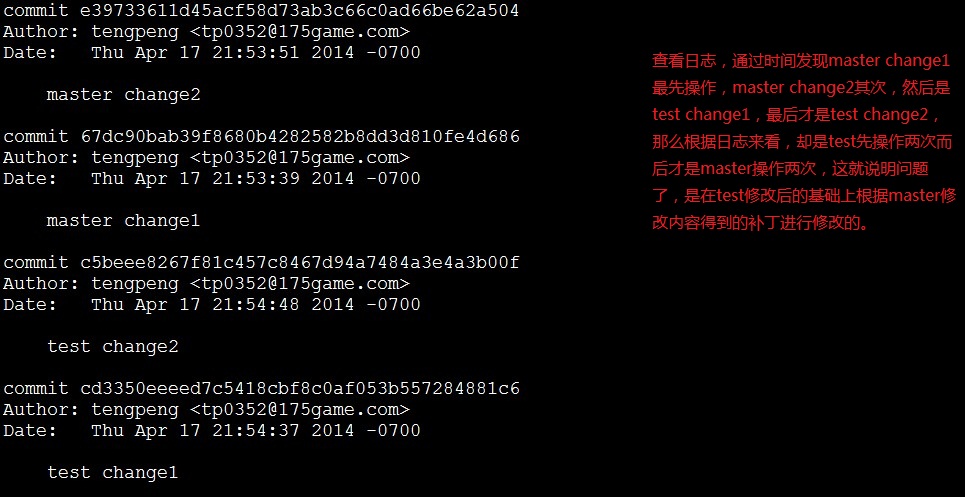




冲突：



日志：



**3.\*(no branch)**

        在执行命令git branch查看分支的时候，如果出现\*(no branch)，则表示不在任何分支上进行工作。出现这种情况我也是在几次不经意之间，用git checkou回溯版本的时候，用git pull或者merge和rebase的时候会出现\*(no branch)。目前我在rebase的时候都是在\*(no branch)上进行的，当衍合完成后自动切到master上，我觉得这是个正常现象，但是其他几种方式就不正常了，具体原因我也不是很清楚。

        由于\*(no branch)表示不在任何分支上进行，而有时我们不知道自己是在\*(no branch)上进行操作的，而且可能我们已经进行很久的开发工作了，已经提交好几个版本的代码了，突然执行git branch发现在\*(no branch)上，是不是一件很恐怖的事啊。

        当然经过提交的版本数据都会以快照的方式被记录在commit对象存在.git目录的objects子目录里，那么当我们发现是在\*(no branch)时应该怎么解决呢。有两种情况。

        第一种情况是我们还没离开\*(no branch)，这个时候，我们可以执行git checkout -b mybranch命令，这个时候会创建新分支mybranch，并将\*(no branch)里面的数据都checkout到mybranch分支上，然后我们再在mybranch上开发，最终合并到master上。

        第二种情况就不乐观了，我们已经离开\*(no branch)了，然后发现用git log都找不到之前的提交了，当然了，在\*(no branch)上提交的，在别的分支上怎么找的到在它上面提交的数据呢。不过也许还有救，如果git还没有执行git gc，那么我们可以通过执行git reflog找到在\*(no branch)上提交的数据，然后根据找到的commit的id来恢复该数据，这也是最后唯一的希望了，如果git已经执行了git gc或者你手贱自己执行了git gc，那么就真的不能在一起愉快的玩耍了。

        所以在执行过git checkout恢复过以前的数据或者是做过合并分支的操作，那么不要吝啬你们的git branch，敲这个命令又不要钱，却能让你之后的提交高枕无忧。

**4.总结**

        merge的合并是三方合并，而且历史版本都在本地方库中，但是却比较繁琐，而且开发的过程是个网状结构，如果创建的分支比较多，进行的merge也比较多，那么就算我们在纸上画它的提交历史都会画的手疼，但是rebase就不一样，它是根据另外一个分支的修改内容进行打补丁然后在前一个分支的最后提交上进行修改，而且将另外一个分支的提交历史删除，这样就是一个线性的过程，很清晰。所以在推送到远程仓库之前尽量多用rebase来衍合分支，但是如果将一个commit推送到远程仓库之后，就不要再对它进行衍合操作了，因为这样的话，它很可能在你的某次衍合过程中被删除，那么再推送到远程仓库就会造成很大的损失。切记，rebase虽好可不要贪杯。