Git Fundamental / Git基础

# 在本地机器上使用GIT

工作目录



## git init初始化库

执行命令：git init，把当前目前初始化为git库。（增加了暂存区和文档库区，但均是空的）。初始化后，当前目录下增加了一个隐藏的.git目录，用于git程序存放元数据，用户不要手工修改其中的内容，实际上暂存区和库区的内容，所部均保存于此，如果把此目录删除，则目录恢复为普通目录）



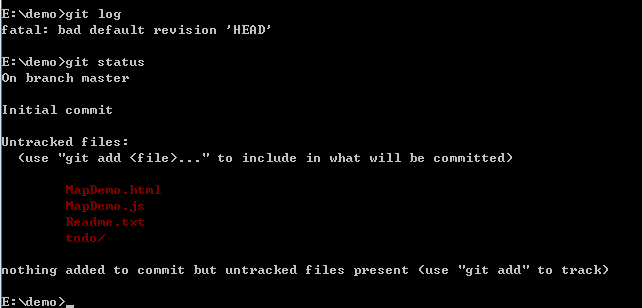


git init后的状态

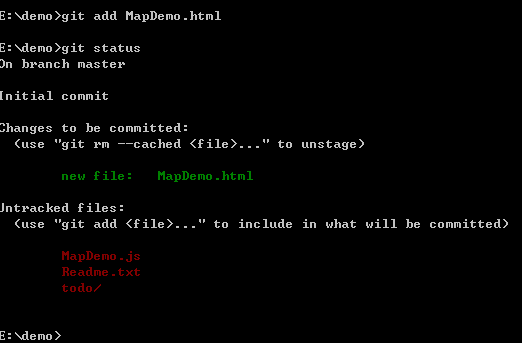
## git log查看commit历史/ status 查看库状态

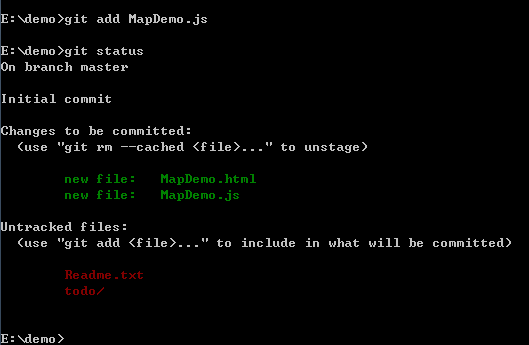
git log查看commit历史，因为无任何commit，故给出错误提示。

git status查看库的状态



## git add 添加新文件或更新文件内容到暂存区







git add 后的状态

几个高效率的命令：

git add -u ：针对所有之前已经被add的文件（即tracked file）的修改(modify)和删除(delete)操作，不处理新增的文件。

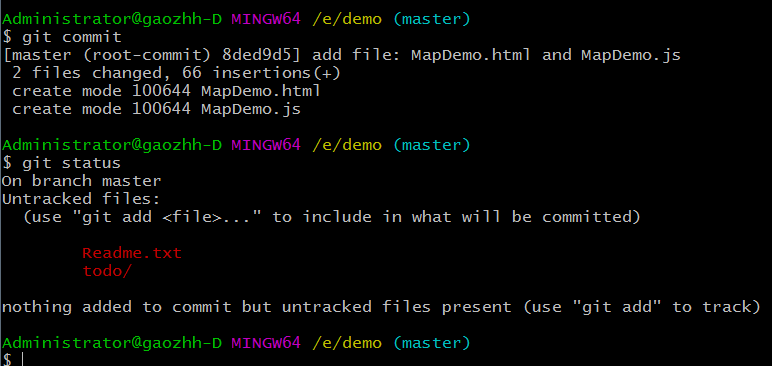
git add . ：针对所有已经被add的文件修改(modify)以及新的文件(new)，但不处理已被add文件的删除。

git add -A ：是上面两个功能的并集（git add --all的缩写）

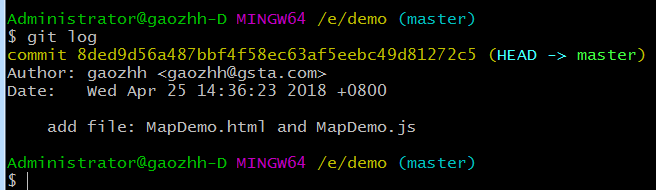
## git commit 提交暂存区的内容到库区

输入git commit命令，然后在弹出的编辑器中输入"add files MapDemo.html and MapDemo.js "

然后，再次输入 git status查看库状态。



最后，再次git log查看





首次git commit 后的状态

## git diff 找不同

修改文件:

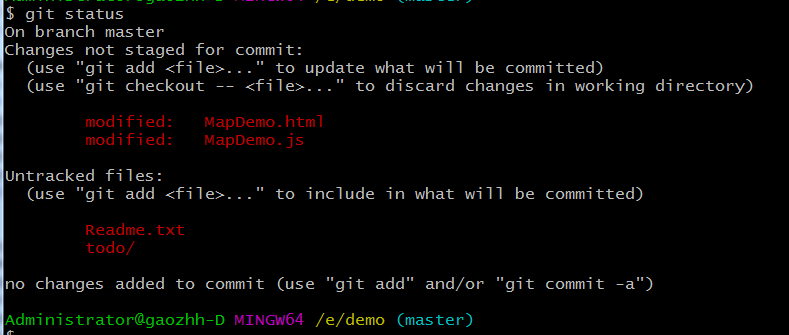
1)把MapDemo.js第30行的blue改为red



2)把MapDemo.html第15、16两行合并为一行



此时，执行git status，系统提示：有未被暂存的修订文件



执行git diff，可详细查看文件的修改内容。

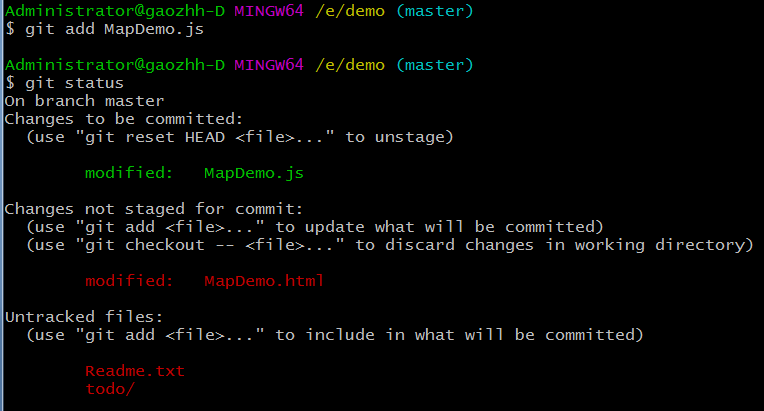
（图待补充）

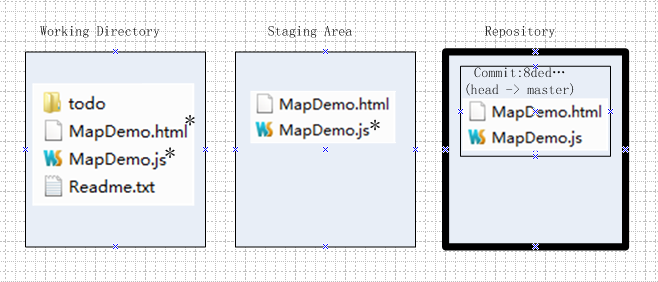


执行“git add MapDemo.js”，把文件添加到暂存区。

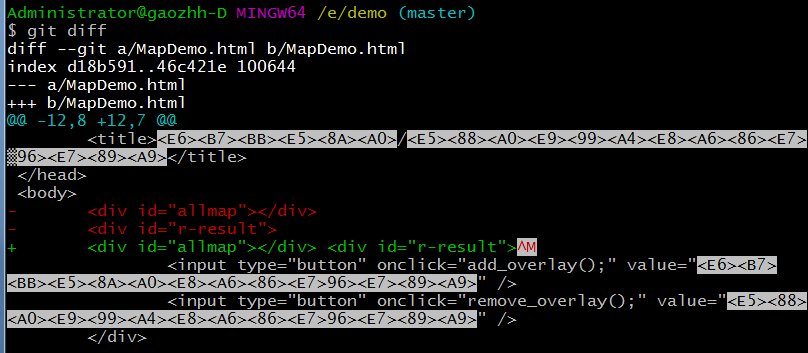
然后“git status”查看状态，系统提示：

1. MapDemo.js已暂存，待提交
2. MapDemo.html已修订，未暂存

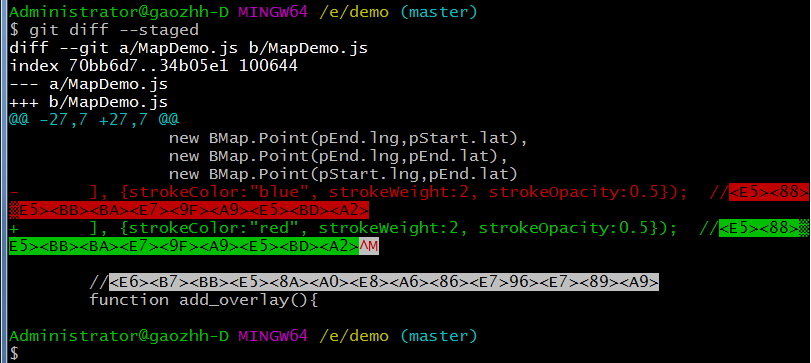


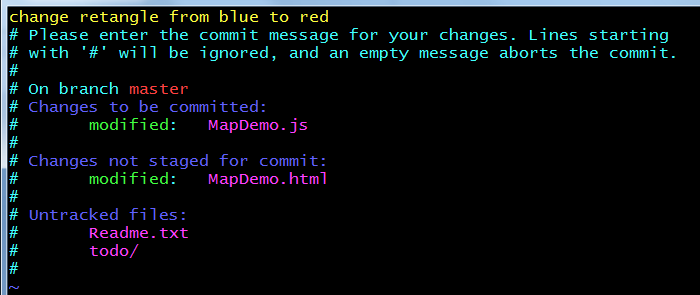


此时，再次输入“git diff”，显示当前工作目录内容与暂存区内容的不同。

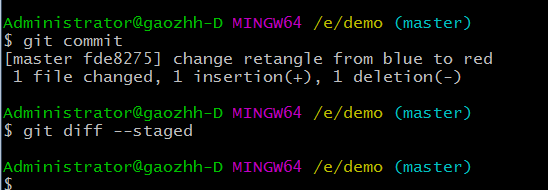


输入git commit命令，可以把暂存区的内容提交到库区。提交之前，我们用“git diff –staged”查看暂存区与库区head节点的不同。





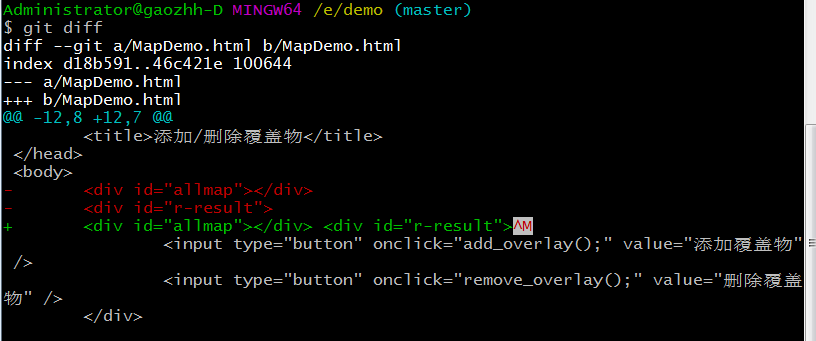
git commit后，再次用“git diff –staged”查看暂存区与库区head节点的不同。（无显示任何差异信息，表示两者是一致的）





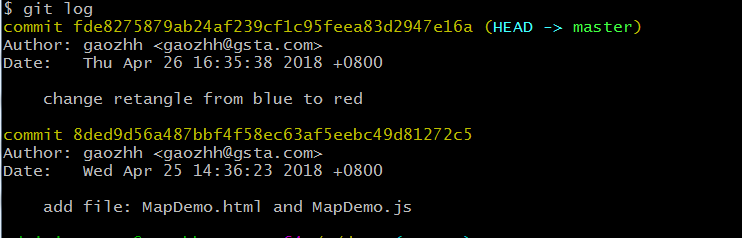
第二次git commit 后的状态

此时，再次运行git diff，查看工作目录和暂存区的区别。（依旧显示区别为MapDemo.html把15、16两行 合并为一行）



## git log再次查看commit历史

使用git log再次查看目前库区的信息。（显示每次commit）



使用git log –oneline可查看简化的信息。

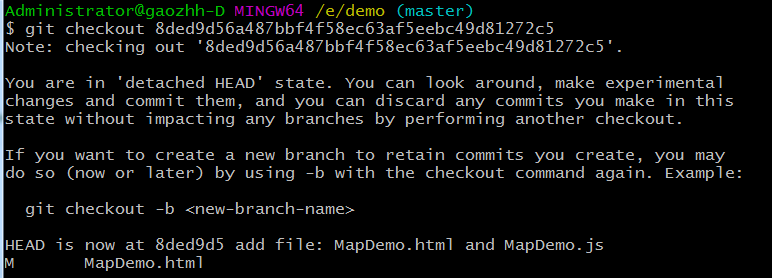


## git checkout切换到指定节点

git checkout 8ded9d56a487bbf4f58ec63af5eebc49d81272c5

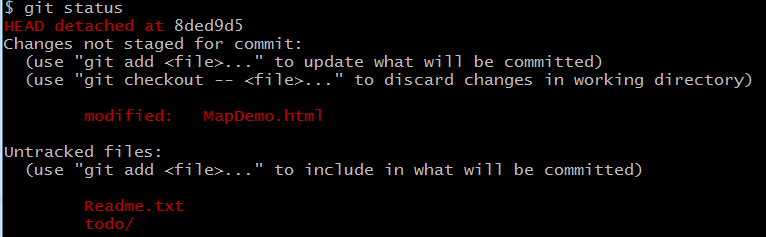
此命令把暂存区恢复为8ded…这个节点，把工作目录中的文件也恢复为8ded…这个节点，但需注意：

1. 工作目录中未被跟踪的文件，不做任何处理。
2. 工作目录中已被跟踪，但做了修改，而未被提交的文件，不做任何处理。（以免丢失修改内容）

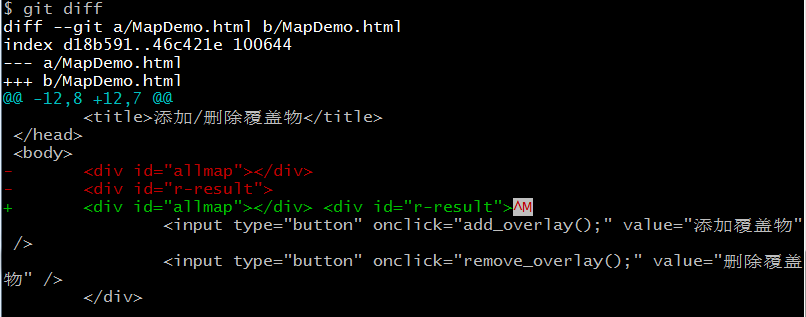


Git 提示“detache HEAD”，表示当前HEAD节点不归属于任何branch.

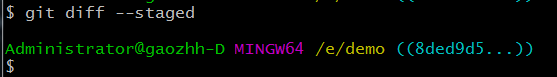
Git status 可看到，MapDemo.html文件已被修改，未暂存。



git diff可看到，MapDemo.html的修改内容



git diff --staged可看到， 暂存区与HEAD节点（即8ded…）是一致的



打开工作目录上的MapDemo.js文件，可以看到第30行中的red已被恢复为blue。

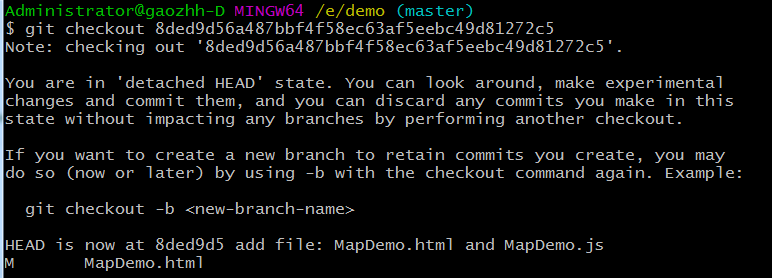


git checkout 8ded…引起的改动

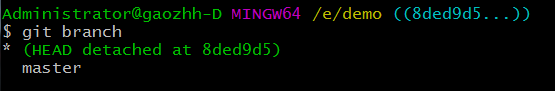
## branch分支

分支（BRANCH）是git为了提供多版本的管理而对每一次提交所标记的“标签”(label)。

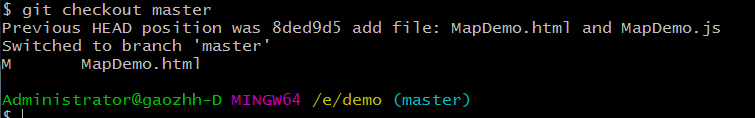
默认地，当库初始化时，即创建了一个称为master的分支，并处于（checkout）此分支上。每一次commit，git都把该次commit标记到已checkout的分支上。所以，此前两次commit，都是在master分支。而当我们checkout到某个节点时，此时由于没有checkout到任何分支，所以如下图所示，git会提示"detached HEAD"，表示当前并不处于任何分支，后续所做的commit不会归属到任何分支。（在git清理空间时，不属于任何分支的commit，会被删除。）



执行git branch，\*号显示当前处于某个节点，同时显示有另一个分支，名为master。

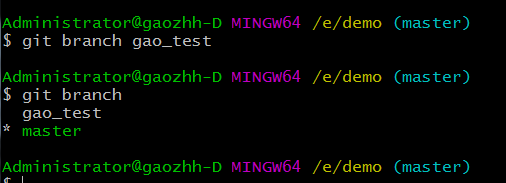


git checkout master 回到主分线，此时工作目录的内容回到master分支的head（即最新commit）。查看文件MapDemo.js的第30行，可以看到为“red”。注意，由于工作区的MapDemo.html被进行了修改，所以并没有恢复master分支中的状态。(以免丢失修改内容）

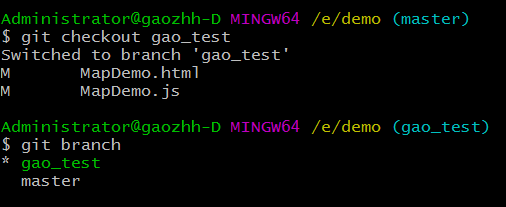


创建分支：

git branch gao\_test 创建gao\_test分支；git branch命令再次查看当前分支。

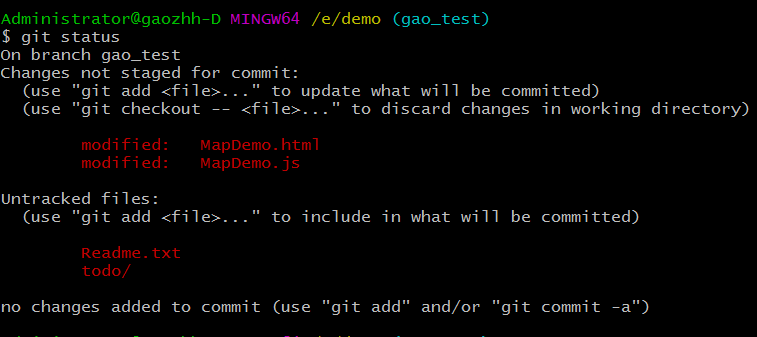


git checkout gao\_test切换到gao\_test分支；git branch命令再次查看当前分支。

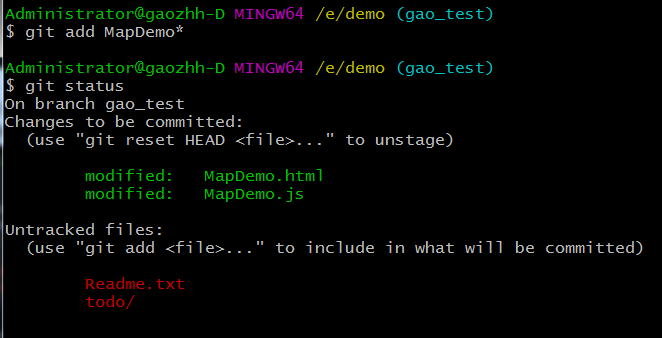


修改MapDemo.js的第11、13和21行，把blue改为green，第30行red改为green。

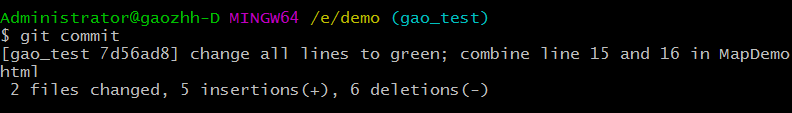
git status可看到有2个被修改的文件。



git add MapDemo\* 把两个文件同时加入暂存区。



git commit把修订提交到gao\_test分支



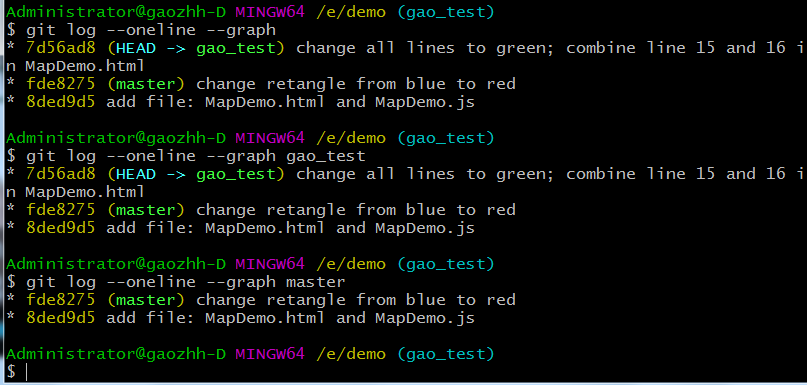
git log 再次查看当前的提交历史。



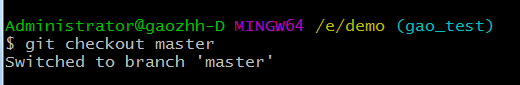
## git log 图示分支历史

git log --oneline --graph master查看master的commit历史

git log --oneline --graph gao\_test查看gao\_test的commit历史



git checkout master切换回master分支，可以看到MapDemo.js和MapDemo.html文件都恢复为master的最新状态。

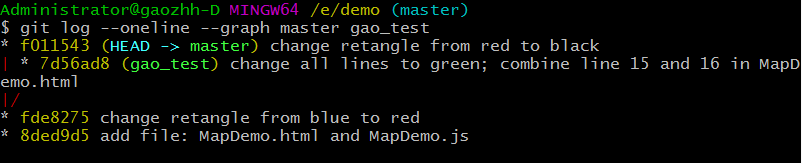


MapDemo.js第30行改为black，并提交。

git add MapDemo.js

git commit

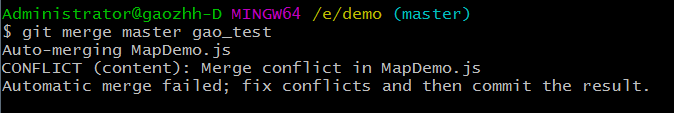
git log --oneline --graph master gao\_test 图示两个分支的提交历史



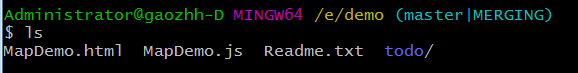
## merge合并及冲突解决

Checkout到master分支，并把gao\_test分支的修改内容合并到master。

git merge master gao\_test



git提示存在合并冲突。



为什么会冲突？

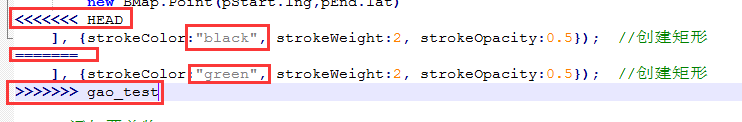
合并的目标：把两个版本所做的修订工作自动地叠加到一起。



打开MapDemo.js，可看到如下图所示的冲突内容。

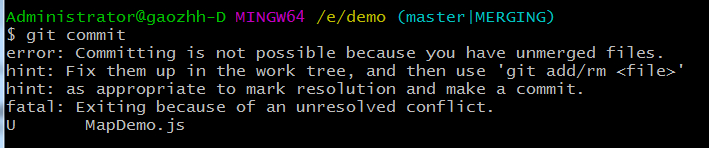
<<<<<<< HEAD 与 ======= 两行之间的内容，表示当前HEAD版本修中订的内容

=======与>>>>>>> gao\_test 两行之间的内容，表示gao\_test分支版本中修订的内容

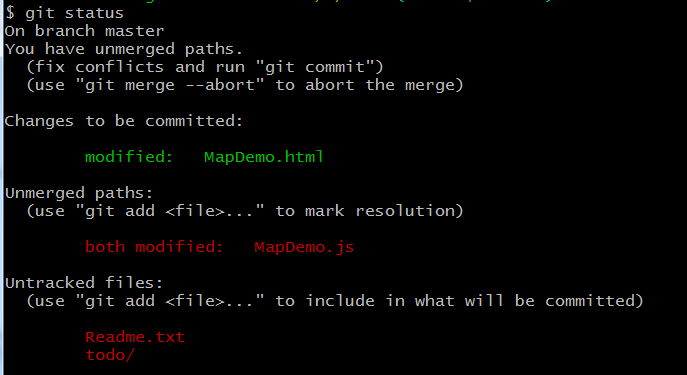


解决冲突：把冲突内容改为实际所需要的内容，并删除<<<<<<< HEAD、=======和>>>>>>> gao\_test三个标识行。

修改并保存，执行git commit命令，提示如下。表示暂存区尚有未合并的文件，无法提交。



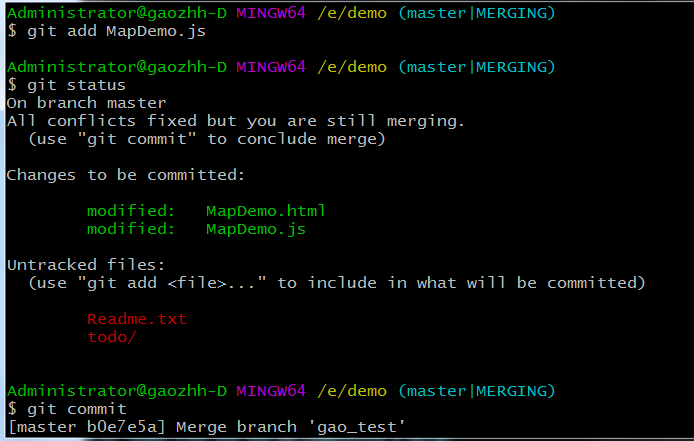
执行git status，可以看到Unmerged paths中提示MapDemo.js未能成功合并。



执行

git add MapDemo.js

git commit

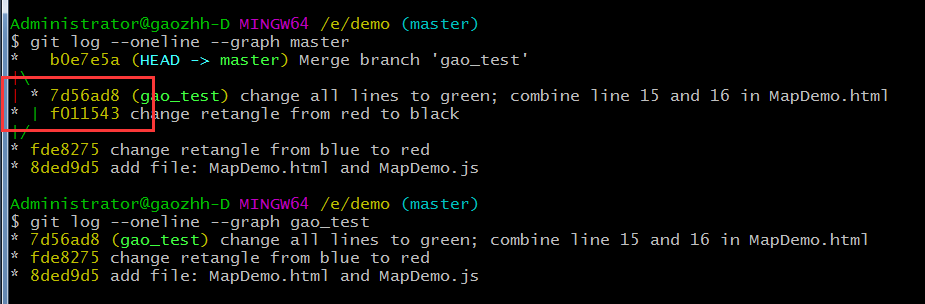




执行以下命令分别查看master和gao\_test两个分支的历史。

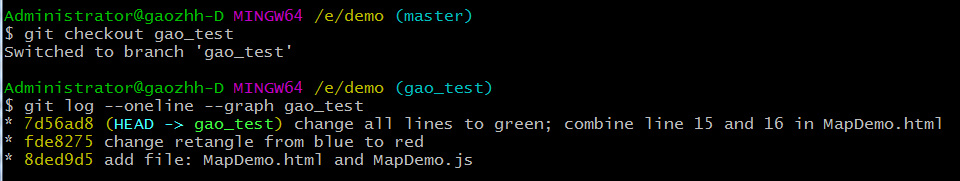
git log --oneline --graph master

git log --oneline --graph gao\_test



特别注意

1. 图中红框，表示节点f011543和7d56ad8分别源于fde8275,合并之后生成了节点b0e7e5a。
2. Merge只对当前check out的分支（master）产生影响，不影响另一个分支（gao\_test）,也就是说，合并后产生的新节点，只挂在当前分支上。（如下图所示，gao\_test的HEAD还在7d56ad8节点）

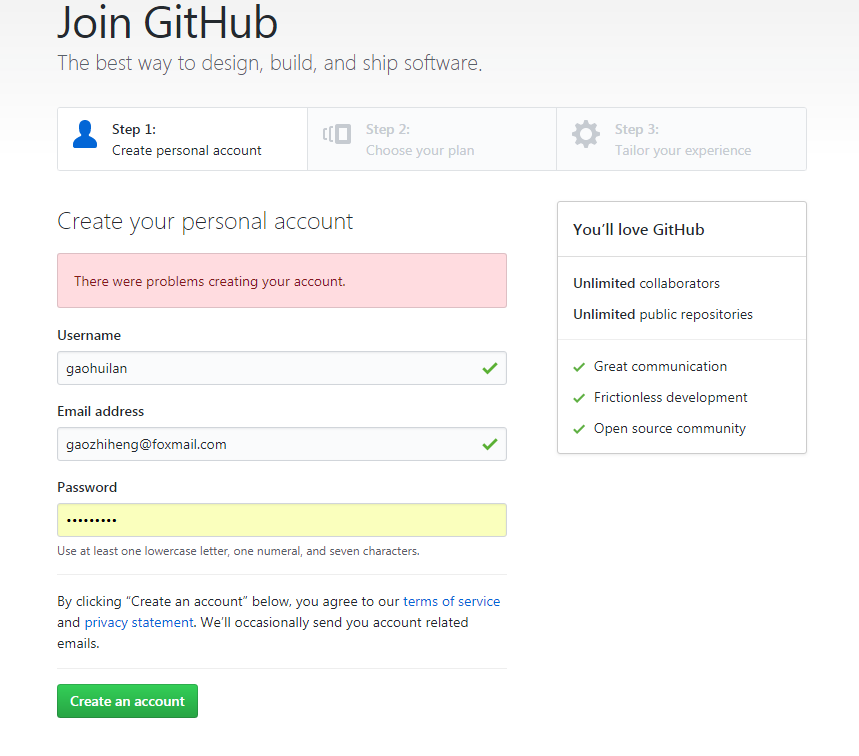


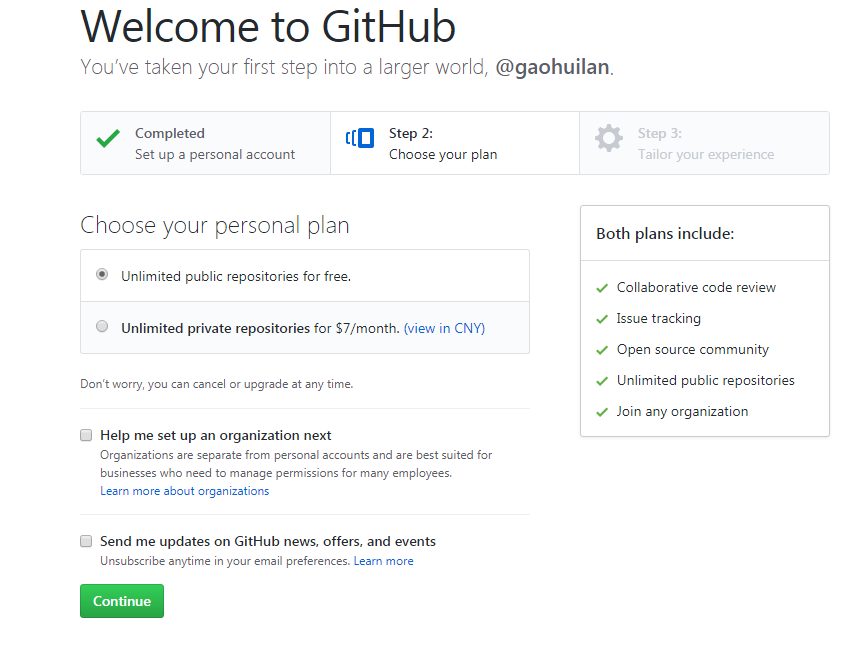
# 使用GITHUB进行多人协作

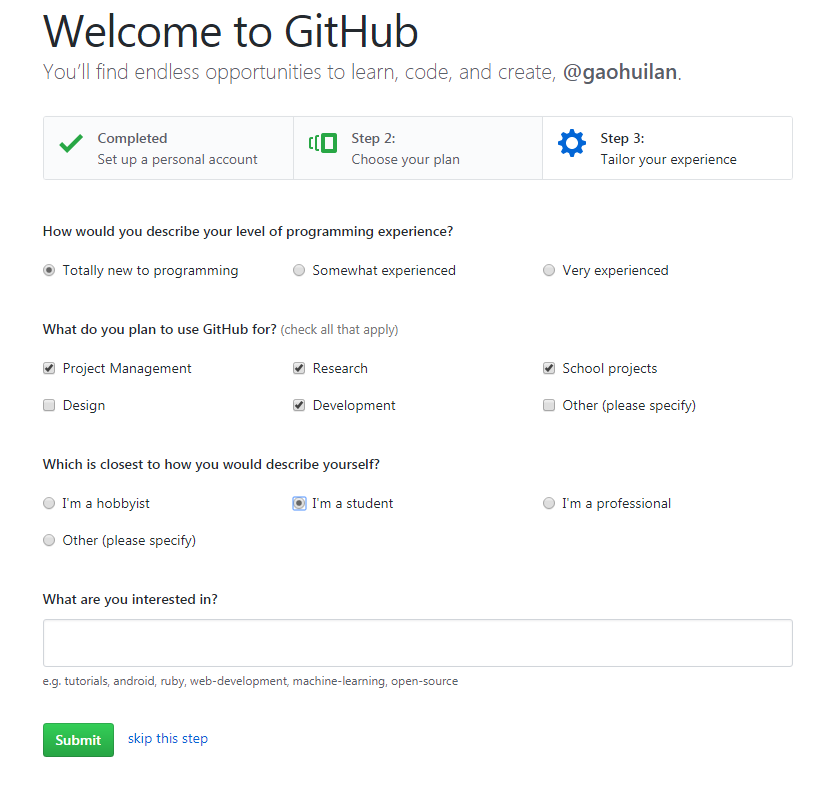
## 创建github帐号

登录：<https://github.com/join>

输入用户名、邮箱和密码，创建帐号





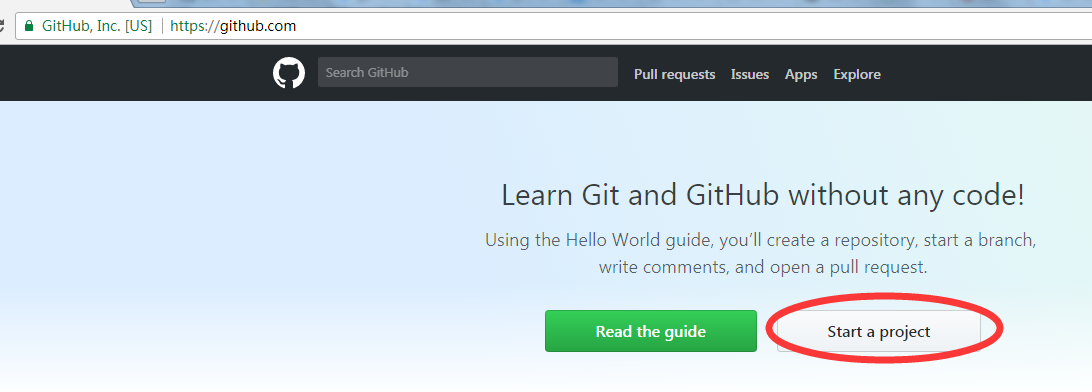


设置帐号密码自动提交，参见：

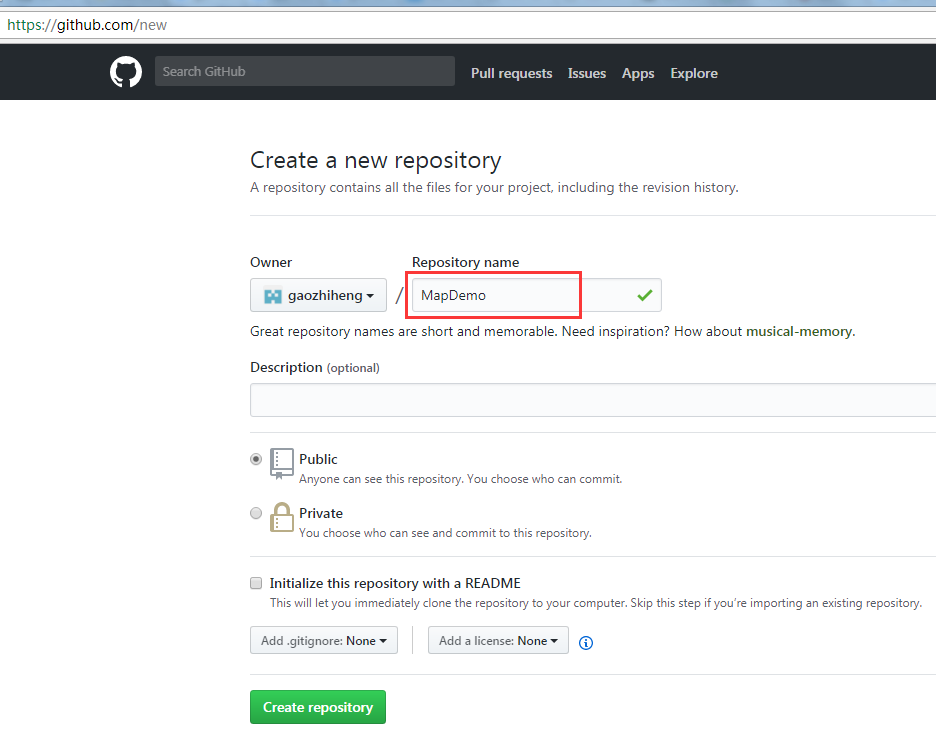
<https://help.github.com/articles/caching-your-github-password-in-git/>

## 创建Github上的Repository

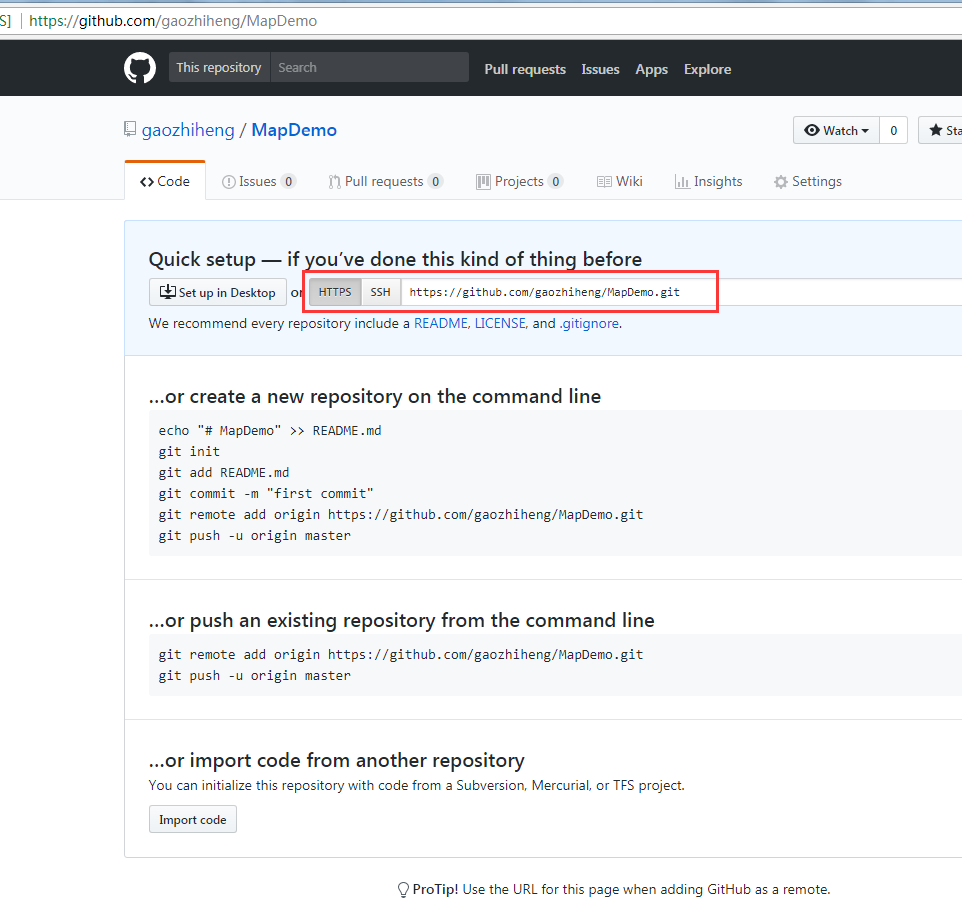
点击首页上的“start a project”按钮



输入Repository的名称，然后点击“Create repository”按钮，完成一个空库的创建。

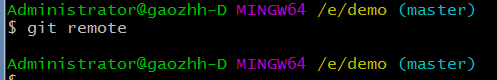


下图显示，该库的URL为： <https://github.com/gaozhiheng/MapDemo.git>



## 添加远程库到本地

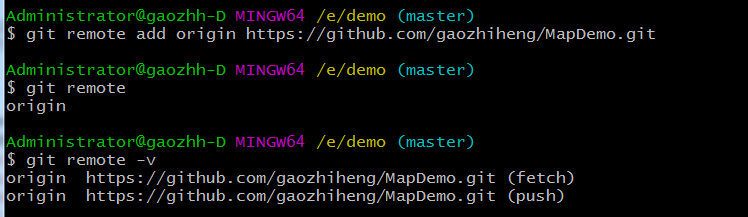
执行git remote命令，可以后到，目前本地库尚未添加任何远程库。



执行命令git remote add origin <https://github.com/gaozhiheng/MapDemo.git>，把远程库添加到本地。（注意：远程库可添加多个，origin是远程库的名称，可自定义，在只使用一个远程库的情况下，业界的通用规范是使用名称origin。）

执行git remote，可看到目前有一个名称为origin的远程库。

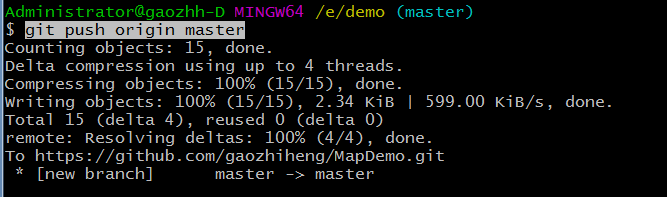
执行git remote -v 可显示该远程库的详细信息。



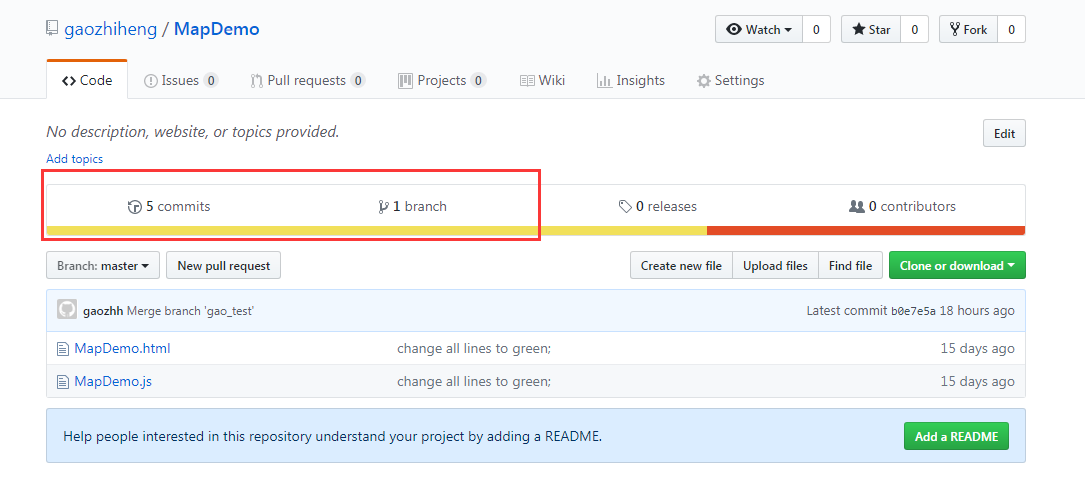
## 把本地分支push(推送)到远程库

执行以下命令，把本地库中的master分支推送到远程库origin。

git push origin master

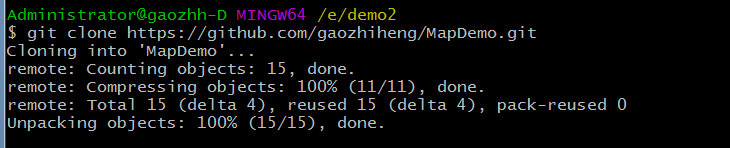


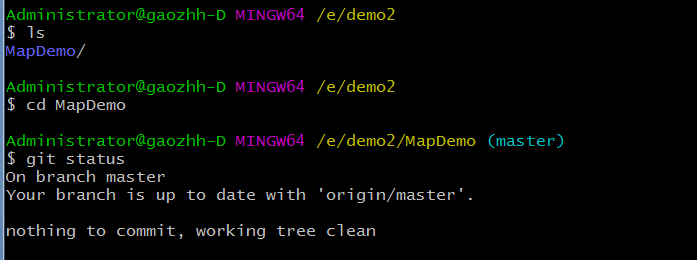
完成后，在该库的主页https://github.com/gaozhiheng/MapDemo/， 可看到已把本地的master分支的所有5个commit历史信息上传到远程库。



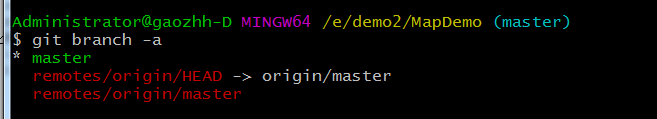
## 把远程库clone(克隆)到本地

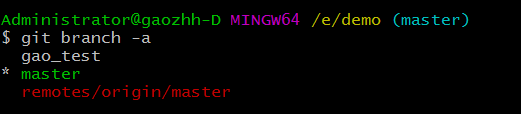
git clone https://github.com/gaozhiheng/MapDemo.git





执行git branch -a命令，分别查看两个库下的所有分支情况。可以看出，demo2下的本地库只有master一个分支，没有gao\_test分支，因为该分支并未上传到github。





## 在demo2本地库上新增一个commit并push到github

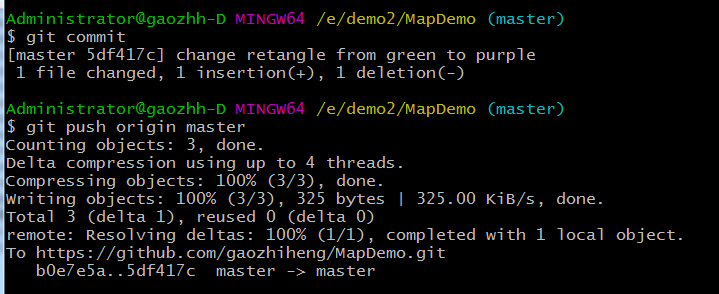
打开demo2本地库上的MapDemo.js文件， 把30行上的green改为purple，并保存文件。

执行git add MapDemo.js加入暂存区



执行git commit提交到库区。

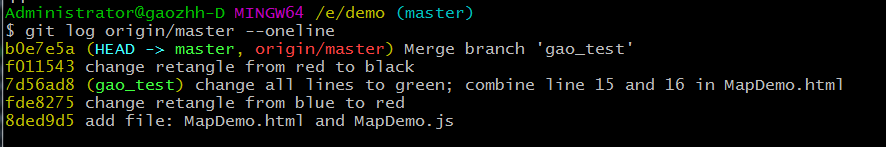
执行git push origin master推送到github。



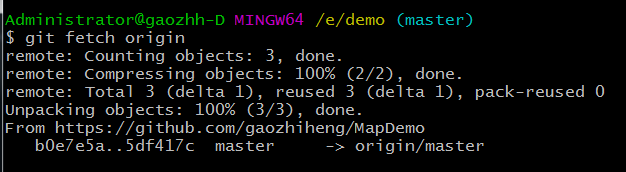
## git fetch:把demo2提交到github上的更新内容同步到demo库

Demo库下执行以下命令，查看远程github库在该库的本地副本的commit历史。可以看到，上一步中从demo2库推送到github上的commit节点(5df417c)尚未同步到本地

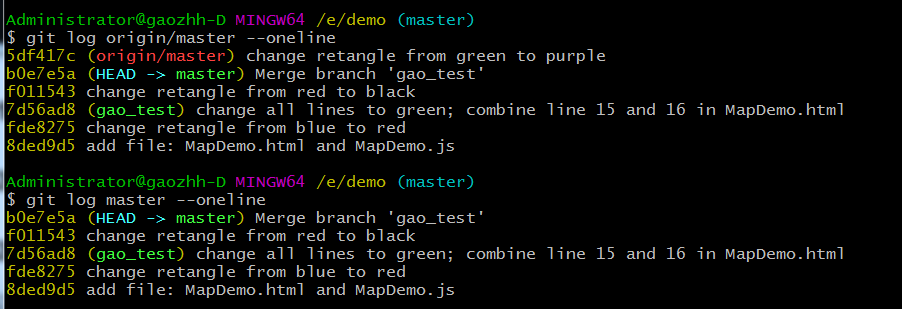
git log origin/master --oneline



执行git fetch origin，把远程库的更新内容同步到本地。



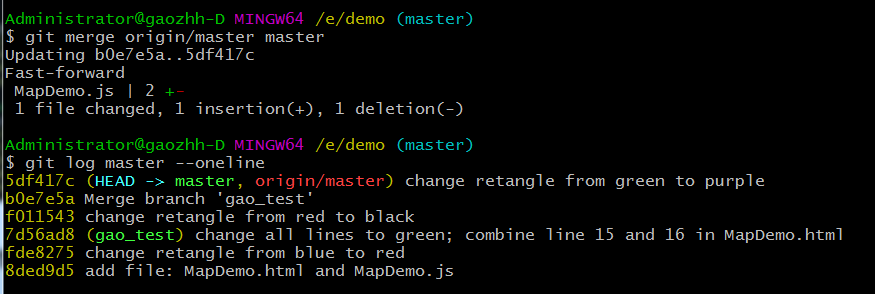
此时，分别执行git log origin/master -- oneline 和git log master --oneline，可以看到，分支 origin/master（这是远程github分支master对应于本地的副本）上已出现了commit节点(5df417c)，而master分支则尚未有该节点。



## 合并origin/master的内容到master

执行git merge origin/master master，合并origin/master的内容到master。

然后执行git log master --oneline，可看到commit节点(5df417c)已出现在master分支了。



## 使用git pull实现fetch和merge

git fetch origin

git merge origin/master master

以上两个命令，可以使用下面一个命令完成，提高操作效率

git pull origin master

注意 ：如在存在冲突，则相当于git fetch完成了，但git merge失败，需要解决冲突后，再次提交git merge命令。

github特有的操作

1. fork(复刻)

把github上别人的库复制到你自己在github上的库（类似于git clone，但git clone是复制到本地）

1. pull request

多人协作时，申请把自己提交的内容合并到master。

优点：a)合并前由另一个人进行多一次确认

b)github上提供了便利的消息传递，通知相关人员。

第一次commit的git

