git -- 分布式版本控制软件，免费而超好用的git

gitHub 是使用git进行版本控制的代码管理网站

Linux系统不断发展，已经成为最大的服务器系统软件了。

CVS和SVN都是集中式的版本控制系统，而Git是分布式版本控制系统。

区别：集中式版本控制系统是，干活的时候用的都是自己的电脑所以要先从中央服务器取得最新的版本，然后开始干活，干完活了，再把自己的活推送给中央服务器。

中央服务器就好比是一个图书馆，你要改一本书，必须先从图书馆借出来，然后回到家自己改，改完了，再放回图书馆

集中式版本控制系统最大的毛病就是必须联网才能工作。

分布式版本控制系统根本没有‘中央服务器’，每个人的电脑上都是一个完整的版本库，这样就可以不用联网了，

但是如果你在自己电脑上修改了文件A，你的同事也修改了A，你们两个需要互相推送就可以看到对方的修改了。

分布式版本控制系统通常也有一台充当‘中央服务器’的电脑，这个服务器用来‘交换大家的修改’，交换修改更方便。

Git极其强大的分支管理

创建版本库：

版本库又名仓库，英文名repository，可以理解成一个目录，这个目录里所有文件都可以被Git管理，每个文件的修改、删除，Git都能跟踪和修改。

$ mkdir learngit

$ cd learngit

$ pwd

pwd命令用于显示当前目录，也就是该仓库位于/Users/michael/learngit

第二步，通过git init命令把这个目录变成Git可以管理的仓库

$ git init

瞬间Git就把仓库建好了，而且也是一个空的仓库，learngit文件下多了一个.git目录。这个目录是Git来跟踪管理版本库的，不要手动修改。

把文件添加到版本库

版本控制系统是没法跟中word文件的改动，如果要真正使用版本控制系统，就要以纯文本方式编写文件

千万不要使用windows自带的记事本编辑任何文本文件，会遇到很多不可思议的问题。建议使用Nodepad++去代替记事本。

言归正传

编写一个readme.txt文件 一定要放在learngit目录下，因为这是一个Git仓库

第一步

命令git add readme.txt告诉Git，把文件添加到仓库里面。没有任何消息，说明添加成功。

第二步

命令git commit把文件提交到仓库：

git commit -m "wrote a readme file"

解释：git commit命令，-m后面输入是本次提交说明，简介通俗易懂

提交完出现：

[master (root-commit) 9dfd485] wrote a readme file

1 file changed, 2 insertions(+)

create mode 100644 readme.txt

解释：1个文件被改动，插入两行内容

//commit一次可以提交很多文件，所以你可以多次add不同的文件

比如：

$ git add file1.txt

$ git add file2.txt file3.txt

$ git commit -m "add 3 files."

第三步

手动修改readme.txt文件

git status命令查看结果：git status命令时刻掌握仓库当前的状态，readme.txt被修改了，但还没有提交

git diff readme.txt命令查看不同，查看具体修改了哪些内容。

接下来知道修改了具体内容，便提交修改文件了

第四步

git add readme.txt

git status 运行该命令随时查看仓库状态

git commit -m "add content"

git status 提交后再次用该命令查看仓库的当前状态，当前没有需要提交的修改，工作目录是干净的

总结：git status 命令可以随时掌握工作区的状态，如果有文件被修改过，那么就用git diff查看修改内容

版本回退

再次对readme.txt进行修改，然后提交。

git log 获取所有提交修改的内容版本号

git log --pretty=oneline 该命令在一行显示提交的修改的内容版本号

如何回到readme.txt回退到上一个版本，也就是"add content"这个版本？

git中，HEAD表示当前版本。上一个版本 HEAD^, 上上一个版本 HEAD^^,上100个版本写成HEAD~100

现在将当前版本回退到上一个版本"add content"

git reset --hard HEAD^ 便可以得到当前版本的上一个版本

cat readme.txt 该命令可以获得readme.txt中的详细内容

git log 查看当前版本库的状态，刚才的版本不见了，21世纪回到了19世纪，想再回去21世纪如何？

只要命令行没有关掉就可以往上面找，找到刚才版本的commit id是多少

git reset --hard 33288ec89fa2 (只需要写前几位，没必要写全，git会自动去找) 该命令回到当前版本。

cat readme.txt 查看已经回到刚才版本

比如说现在你回退到前一个版本，关掉电脑。第二天想恢复到新版本该如何？找不到新版本的commit id？想要回退到最新的版本，就必须找到最新版本的commit id

git reflog 该命令用来记录你的每一次提交便可以得到commit id

工作区和暂存区

工作区：

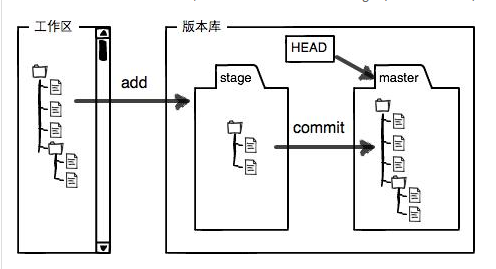
电脑里面可以看到的目录，比如当前studygit文件夹就是一个工作区。

版本库：

工作区有一个隐藏的目录.git 是git的版本库

git版本库存了很多东西，最重要的是stage（index）的暂存区

另外还有git自动创建的第一个分支master，以及指向master的一个指针叫HEAD



前面教程中把文件往git版本库提交的时候，是分两步执行：

第一步是用git add 把文件添加进去，实际上就是把文件修改添加到暂存区

第二步是用git commit提交修改，实际上就是把暂存区的所有内容提交到当前分支

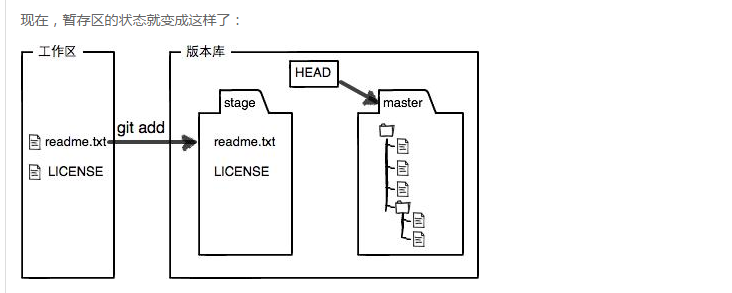
修改readme.txt,在工作区新增一个LICENSE文件

git status 查看状态：

readme.txt状态是：modified，而LICENSE没有被添加过，状态是Untracked

使用两次git add命令，将readme.txt和LICENSE添加到暂存区，

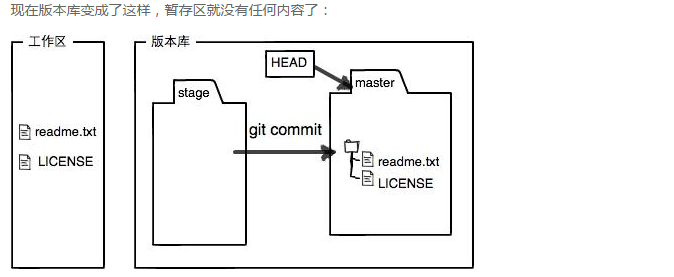
git status 查看状态



所以git add命令实际上把要提交的修改放到暂存区（stage）

git commit –m “understand how stage work”

然后执行git commit 就可以把暂存区的所有修改提交到分支



管理修改

git比其他版本控制系统设计的优秀在于Git跟踪并管理的是修改而不是文件

重要理解：第一次修改文本 🡪 git add 🡪 第二次修改文本 🡪 git commit

Git管理的是修改，当第一次使用git add命令后，在工作区的第一次修改被放入暂存区，准备提交。

但是在工作区的第二次修改并没有放入暂存区，所以git commit只负责把暂存区的修改提交了（也就是第一次的修改被提交），第二次的修改不会被提交。

当然如果想两次修改都提交到工作区

第一次修改文本🡪 git add 🡪 第二次修改文本🡪 git add 🡪 git commit



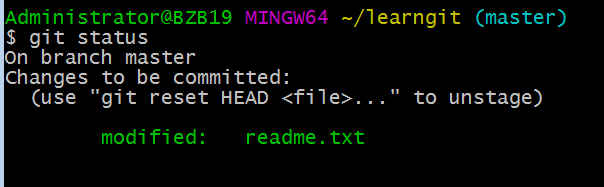
撤销修改

假如你一不小心在readme.txt的最后一行添加My boss is stupid

使用命令 git checkout – readme.txt 解释：把readme.txt文件在工作区的修改全部撤销。

1. readme.txt 从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，又作修改，撤销修改就回到添加暂存区后的状态

重点：你在最后一行添加了my stupid boss still prefers VSN.使用命令 git add 到暂存区了，但是没有git commit



Git同样提示，用命令git reset HEAD file可以把暂存区的修改撤销掉，重新放回工作区

1、git reset HEAD readme.txt 可以把暂存区的修改撤销掉，重新放回工作区

2、git checkout -- readme.txt 将工作区最后的修改丢弃

也就是工作区中已将my stupid boss still prefers VSN 这个修改丢弃

总结：

场景1：当你改乱了工作区某个文件的内容，想直接丢弃工作区的修改时，用命令git checkout –- file。

场景2：当你不但改乱了工作区某个文件的内容，还添加到了暂存区时，想丢弃修改，分两步，第一步用命令git reset HEAD file，就回到了场景1，第二步按场景1操作。

场景3：已经提交了不合适的修改到版本库时，想要撤销本次提交，参考[版本回退](http://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/0013744142037508cf42e51debf49668810645e02887691000)一节，不过前提是没有推送到远程库。

删除文件

在Git中，删除也是一个修改操作。

1. 确实要从版本库中删除该文件

新建test.txt->git add test.txt->git commit

执行删除操作：

git rm test.txt 删除文件

git commit –m ‘remove test.txt’ 提交文件从版本库中被删除

二、误删版本库中的文件

因为版本库里还有，所以可以很轻松把误删除的文件恢复到最新版本

git checkout -- test.txt

git checkout 其实是用版本库里的版本替换工作区的版本，无论工作区是修改还是删除都可以一键还原

远程仓库

介绍Git的杀手级功能

Git是分布式版本控制系统，同一个Git仓库，可以分不到不同的机器上，没有主次之分。

实际情况往往是，一台电脑充当服务器的角色，每天24小时开机，其他每个人都从这个’服务器’仓库克隆一份到自己的电脑上，并且各自把各自的提交推送到服务器仓库里，也从服务器仓库中拉取别人的提交。

GitHub提供Git仓库托管服务的网站。

本地的Git仓库和GitHub仓库之间的传输是通过SSH加密的。

第一步：创建SSH Key 看看有没有.ssh目录，如果有，确定有没有id\_rsa 和id\_rsa.pub这两个文件，如果有请直接跳至下一步，没有请使用命令

Ssh-keygen –t rsa –C ‘759945821@qq.com’

id\_rsa 是秘钥，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，打开个人主页，SSH Keys页面

然后点击”Add SSH Key”,填写任意的Title,在Key文本框粘贴id\_rsa.pub文件的内容

点击Add Key.

为什么GitHub需要SSH key？

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只要知道你的公钥，就可以确认你自己才能推送。

当然GitHub允许你添加多个Key，可以在公司也可以在家，只要把每台电脑的key都添加到GitHub，就可以在每台电脑上推送了。

GitHub上免费托管的Git仓库，任何人都可以看到，但只有你才可以修改。

可以交钱变成私有的，也可以自己动手搭一个Git服务器，因为你自己的服务器，别人看不见。这样我们就可以开始学习远程库了。

添加远程库

在本地创建了一个Git仓库后，又想在GitHub创建一个Git仓库，并且让这两个仓库进行远程同步，这样GitHub上的仓库既可以备份，也可以让其他人通过该仓库来习作，一举多得。

在GitHub 中找到Create a new repository

输入Repository name：studygit

就成功创建了一个studygit远程仓库

在本地git输入命令，连接远程库

git remote add origin [git@github.com:mangoyi/studygit.git](mailto:git@github.com:mangoyi/studygit.git)

接下来就是把本地库的所有内容推送到远程库

git push –u origin master

git push命令，实际上是把当前分支master推送到远程库。

由于远程库是空的，第一次推送master分支时，加上-u参数，Git不但会把本地的master分支推送的远程新的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了。

从现在起，只要本地提交修改，就可以使用命令

git push origin master 现在也就是正式拥有了分布式版本库

总结：

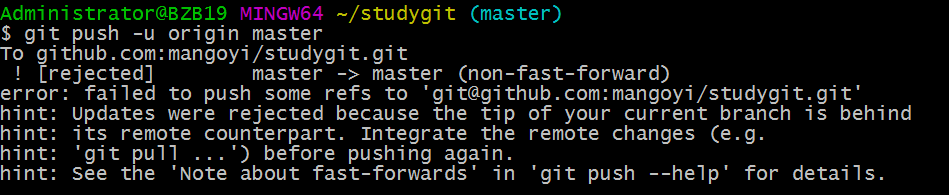
要关联一个远程库，使用命令git remote add origin [git@github.com:mangoyi/studygit.git](mailto:git@github.com:mangoyi/studygit.git)

关联后使用命令

git push –u origin master 第一次推送master分支的所有内容

在之后的提交中就可以直接使用git push origin master 推送最新修改

常见报错如下：



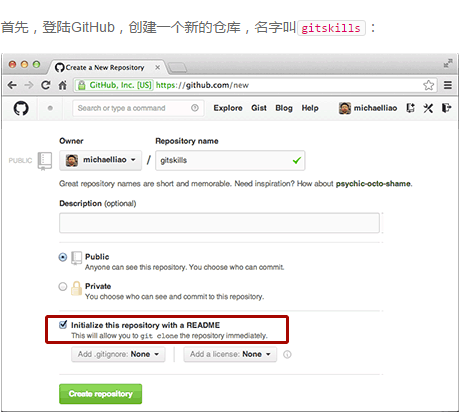
解决方法（强制推送修改，这不是最好的方式）：

1. git pull –f –all
2. git push –u origin master –f

删除远程库：git remote rm origin(当你关联错远程库的时候，可以先删除之前关联的远程库，重新关联需要相关联的远程库)

克隆远程库

Github上创建远程库，再从远程库克隆



勾选 Initialize this repository width a README

这样GitHub会在gitkills库中自动创建一个REANME.md文件（由于手误，以下都为gitkills仓库）

补充： 命令ssh [git@github.com](mailto:git@github.com) 可以来判断是否配置好权限了

克隆到本地库

克隆命令 git clone [git@github.com:mangoyi/gitkills.git](mailto:git@github.com:mangoyi/gitkills.git)

命令 cd gitkills

命令 ls

GitHub给出的地址不止一个还可以使用：

http://github.com/mangoyi/gitkills.git

实际上，Git支持多种协议，默认是git使用ssh，但也可以使用https等其他协议。https速度慢，还有个最大的麻烦每次推送都必须输入口令，但是在某些只开放http端口的公司内部就无法使用ssh协议而只能用https

创建并合并一个分支

1、 创建dev分支

git checkout –b dev 得到：Switched to a new branch ‘dev’

git checkout 命令加上-b参数表示创建并切换相当于两条命令：

git branch dev

git checkout dev

2、git branch命令查看当前分支

git branch \*dev (当前分支)

master

1. 当前在dev分支上，在文件夹中创建readme2.txt文件，此时文件是在dev分支上。
2. 提交

git add readme2.txt

git commit –m “branch test”

1. dev分支工作完成，切换到master分支

git checkout master

1. 切换到master分支时发现readme2.txt文件不见了，因为之前提交在dev分支上，而master分支此刻的提交点并没有改变。
2. 将dev分支上的工作成果合并到master分支上

git merge dev 命令用于合并指定分支到当前分支，此时readme2.txt出现了。

1. 合并完成后就可以放心地删除了dev分支了

git branch –d dev 删除dev分支

1. 删除后查看branch就只是剩下master分支了

git branch 当前仅仅只有master分支，\*master

10、创建合并删除非常快，git鼓励去使用分支完成某个任务。合并后在删掉分支，这样和直接在master分支上工作效果是一样的，但是过程会更安全。

查看分支： git branch

创建分支： git branch dev(名称)

切换分支： git checkout dev(切换到dev分支)

创建+切换分支： git checkout –b dev(分支名称)

合并某分支到当前分支： git merge dev (分支名称)

删除分支： git branch –d dev (分支名称)

解决冲突：在合并分支的时候会出现，比如两个用户修改了同一个文件区域，git会报告内容冲突 此时就必须解决冲突。

冲突产生的原因：比如在featrue1分支上，修改了readme.txt的最后一行文本内容，并且add和commit到分支上，然后到切换到master分支上，并在相同的位置上做出修改，然后add和commit。在合并git merge feature1:此时就会出现冲突。

详细步骤：

1、使用命令 cd studygit

2、准备新的分支 feature1分支 git checkout –b feature1

3、修改readme.txt最后一行，修改为Creating a new branch is quick and simple

补充：一：使用命令 vi readme.txt 便可以直接在命令行修改内容

修改完之后按ESC键，退出编辑模式，切换到命令模式

输入：wq 保存修改并且退出vi 编辑文档模式

二：当然也可以找到文件直接打开修改在保存。

4、git add readme.txt

git commit –m ‘quick & simple’

当前master分支和feature1分支各自都分别有新的提交

5、git在这种情况是无法执行快速合并， git merge feature1 会提示冲突，

必须手动解决，然后在提交

6、使用命令git status 查看冲突的文件

7、查看readme.txt的内容，

这里可以直接用命令vi readme.txt 来编辑内容，选择你需要的内容

8、修改完再次

git add readme.txt

git commit –m ‘conflict fixed’ 这样便解决冲突并且提交成功

9、git log –graph –pretty=online –abbrev-commit 可以查看分支合并的情况

10、最后删除分支

git branch –d feature1 工作完成

分支管理策略

通常合并分支时，git 会采用Fast forward模式，但这种模式下，删除分支后，会丢掉分支信息。

采用–-no-ff 方式的git merge

1、任然创建并且切换dev分支

git checkout –b dev

2、修改readme.txt文件，并且提交commit

git add readme.txt

git commit –m ‘add merge’

3、切换到master分支

git checkout master

4、合并dev分支，注意使用--no-ff参数，表示禁用Fast forward模式来合并

git merge -–no-ff –m ‘merge with no-ff’ dev

解释：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

5、合并后通过命令 git log 看看分支历史

总结：分支策略

后来话：在实际开发中，首先master分支时非常稳定的，也就是仅仅用来发布新版本，平时不能在上面干活。

干活都是在dev分支上面，也可以说dev分支不稳定。

团队合作每个人都有自己的dev分支，dev分支上面合并就可以了

Git分支十分强大，合并分支时，加--no-ff 参数就可以用普通模式合并，合并后的历史记录里面有分支的信息，而fast forward 合并就看不出来做过合并。

Bug分支

在软件开发中，bug就像家常便饭，git分支如此强大，所以每个bug都可以通过一个新的临时分支来修复，修复后，合并分支，然后将临时分支删除。

实际情况中：

当你接到一个101代号的bug任务时，很自然会想到创建一个issue-101来修复它，但是当前dev分支的工作还没有提交。实际情况是工作只进行到一半，还没法提交，预计一天才能提交，但bug要在2个小时内完成。肿么办？

幸好：Git 提供一个stash功能，可以把当前工作现场‘存储’起来，等之后恢复现场继续工作。

1. 在dev分支上，对文件readme.txt做出了一些修改工作没做完，但是bug在两个小时必须解决。

命令git stash 临时保存工作现场

提示：

Saved working directory and index state WIP on dev: 3826324 add merge

HEAD is now at 3826324 add merge

1. 使用git status查看当前的工作区状态，

首先确定在哪个分支上修复bug，假定是在master分支上修复，就从master分支创建临时分支

git checkout master 切换到master分支

1. 创建issue-101分支

git checkout –b issue-101

1. 修复bug，比如说将I am a loser 修改为 I am a achiever.

git add readme.txt

git commit - m ‘fix bug 101’

1. 切换到master分支，完成合并最后删除issue-101分支

git checkout master

git merge -–no-ff –m ‘merged bug fix 101’issue-101

解释：--no-ff参数保存合并的历史记录

git branch –d issue-101

6、101bug修复完成，现在要回到dev分支继续工作

git checkout dev

git status 解释：查看dev当前的工作区

1. 查看工作现场在哪里

git stash list

stash@{0}: WIP on dev: 3826324 add merge

7、恢复工作现场

一：git stash apply 恢复，但是恢复后stash内容并不删除，需要用git stash drop 来删除。

二：git stash pop恢复,恢复的同时把stash内容也删除；命令git stash list查看，这样就看不到stash内容了

Feature分支

软件开发中，总有无穷无尽的新功能添加进来，添加一个新功能你肯定不希望把主分支搞乱，所以每添加一个新功能，最好新建一个feature分支，在分支上开发，完成后，合并最后删除该feature分支

实战：接到一个新任务，开发代号为comment的新功能

1. 创建分支

git checkout –b feature-comment

1. 在文件夹中新建comment.html文件
2. git add comment.html

git status

git commit –m ‘add feature comment’

1. 切回master分支，合并

git checkout master

1. 此时上级命令取消新功能。

git branch –d feature-comment

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

1. Git提醒feature-comment分支还没有被合并，如果删除就会丢掉修改，如果想强制删除

git branch –D feature-comment 强制删除分支 完成任务

多人协作

---当你从远成仓库克隆的时候，实际上Git把本地的master分支和远程的master分支对应起来，并且远成仓库的默认名称是origin

查看远程库的信息 git remote

git remote 结果： origin

更加详细的是git remote –v

推送分支 ：

推送分支就是把该分支上的所有本地提交推送到远程库，推送时，要指定本地分支。Git就会把该分支推送到远程库对应的远程分支上。

git push origin master

推送dev分支：git push origin dev

master分支时主分支要时刻与远程同步

dev分支是开发分支，团队所有成员都需要在上面工作，也需要远程同步

bug分支只用于本地修复bug，就没必要推到远程了

feature分支是否推送到远程，取决于你是否和你的小伙伴合作在上面开发

抓取分支：

多人协作时，大家都会往master和dev分支上推送各自的修改。

比如在studygit库中新建一个文件test.txt

git add test.txt

git commit –m ‘commit test.txt’

git push origin master

这时就会将test.txt文件上传到远程库中，便可以在github中查看到

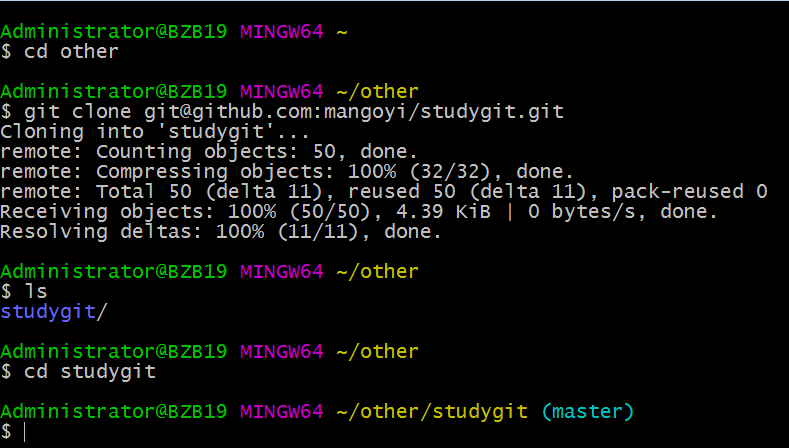
一个电脑要如何去模拟多人合作的实战，先创建另外一个文件目录other

1、cd other(该文件是和studygit在一个目录下)

2、git clone [git@github.com:mangoyi/studygit.git](mailto:git@github.com:mangoyi/studygit.git)

clone一份studygit库在other文件夹

4、cd studygit

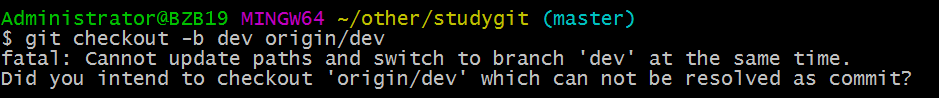


other文件里面的studygit也和远程库连接在了一起，就可以通过修改test.txt然后push到远程库中，但是注意现在的操作是在other文件夹里面的studygit库中，和other同一级目录里面还有一个studygit库。

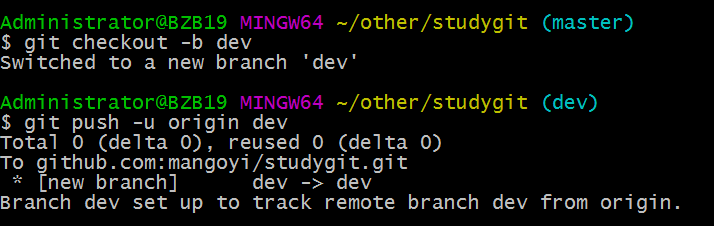


当前在master分支，而模拟的是想在dev分支上开发，所以就必须远程创建origin的dev分支到本地。

一、git checkout –b dev origin/dev



解决方法：<http://blog.csdn.net/weichuang_1/article/details/48437911>



git checkout –b dev //创建本地dev分支

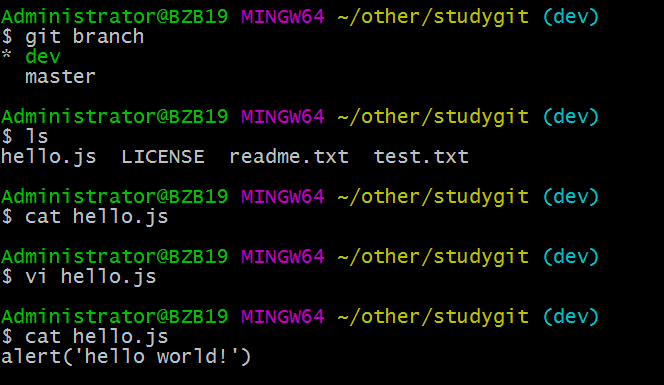
git push –u origin dev //将dev分支推送到远程库

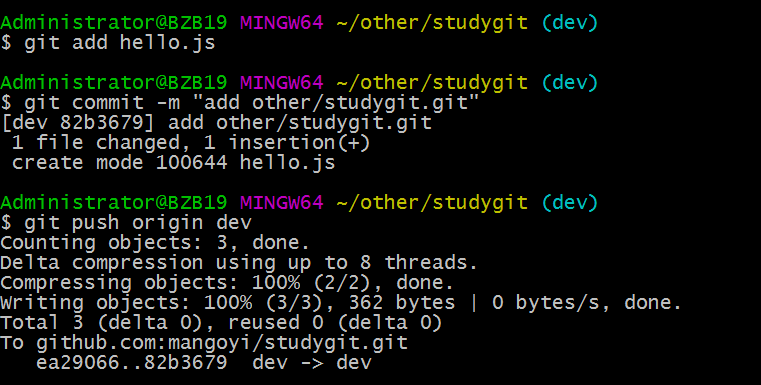
/\*也就是理解成如果原创仓库中没有origin/dev这个分支的话，

你只需要创建一个本地分支dev，然后将它推送到远程仓库\*/

二、在dev分支上创建一个文件hello.js

在hello.js中添加一行内容：alert(“hello world!”)





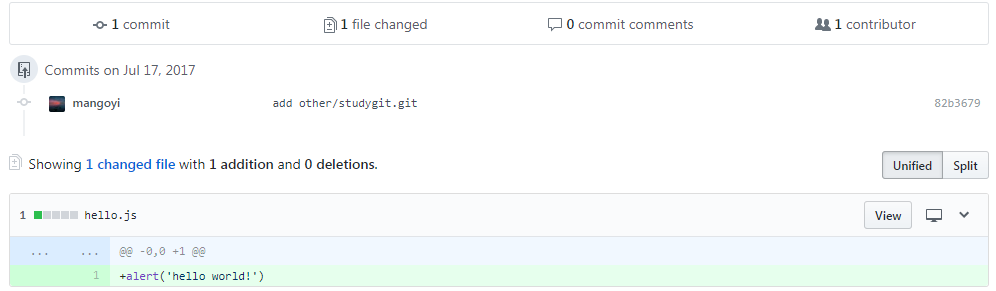
git add hello.py

git commit –m ‘add other/studygit/hello.js’

git push origin dev //推送到远程库的dev分支

此时（相当于你的同事）other目录下的分支已经推送了他的提交。

推送完可以在github上面可以看到dev分支的推送内容

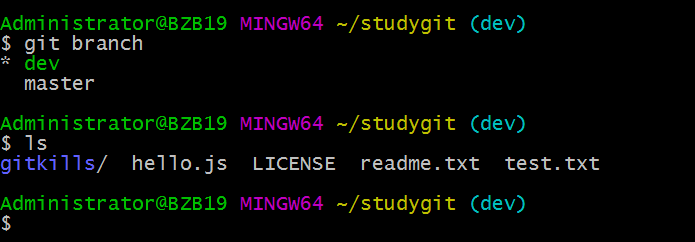


返回上一级目录 cd .. (cd和..中间存在一个空格)

（以上的操作是在模拟同事提交也就是在other目录下操作）

二：回到studygit库

1、在库里创建一个dev分支，并新建hello.js文件

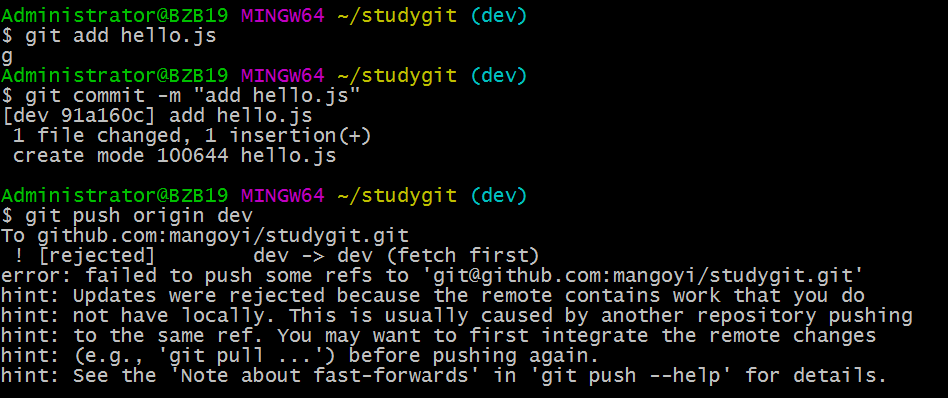


2、git add hello.js

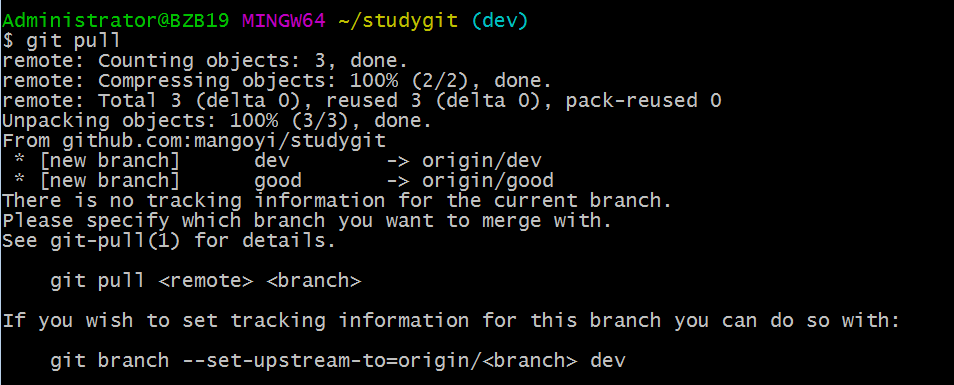
git commit –m ‘add utf-8’

git push origin dev //将本地仓库dev分支推到远程仓库的dev

推送失败：因为你的模拟推送的提交和你试图推送的提交有冲突，git提示我们，先用pit pull 把最新的提交从origin/dev抓下来，解决冲突在推送



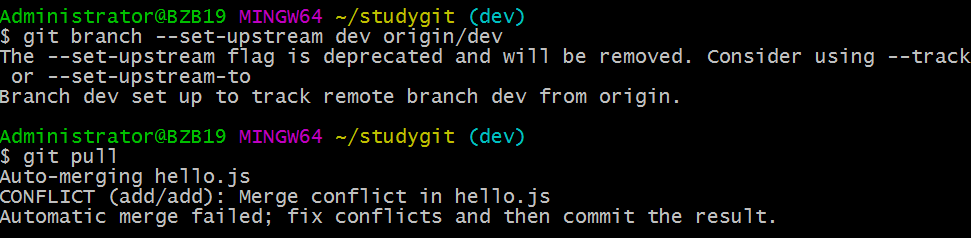
3、使用命令git pull



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链接

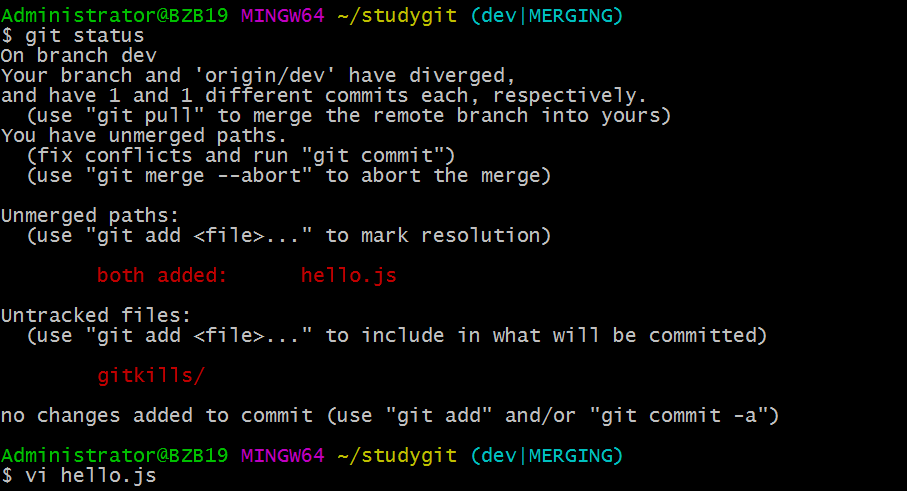
4、使用git branch –-set-upstream dev origin/dev 命令设置链接。

1. 再使用git pull 成功。（但是合并有冲突，需要解决冲突在提交）

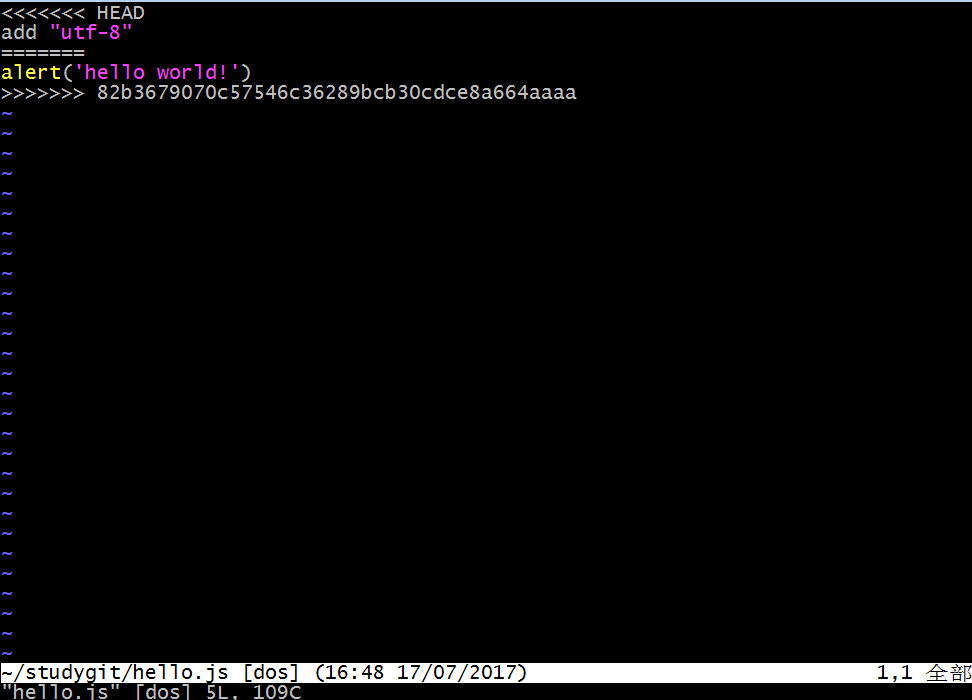


6、git status查看状态

使用命令vi hello.js进行修改，解决冲突



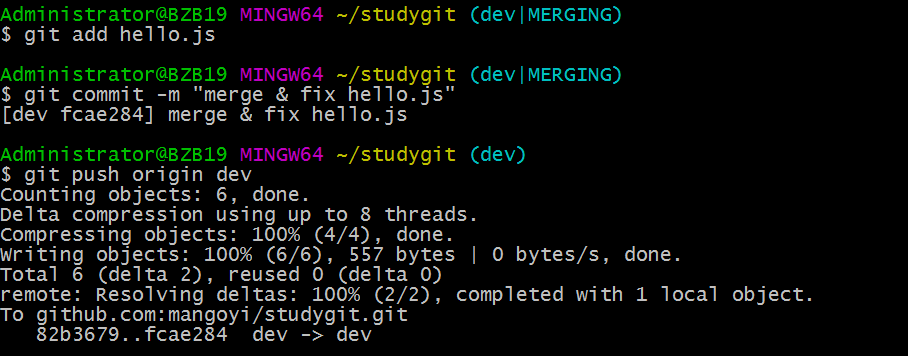
使用vi hello.js 命令之后出现下图



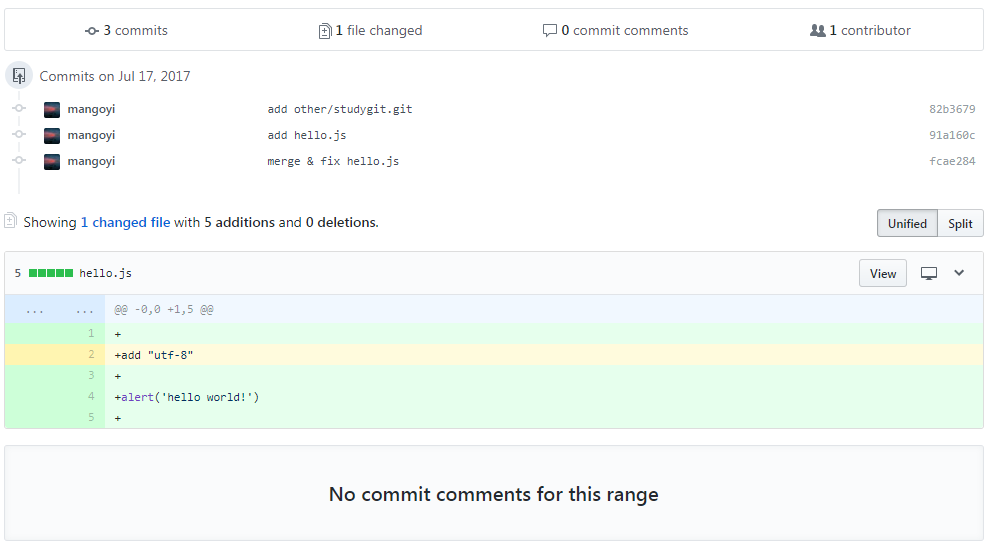
7、git add hello.py

git commit –m ‘merge and fix hello.js’

git push origin dev //推送分支到远程库



推送成功，再次去刷新自己github页面，这也就完成了你和同事的协作



内容来自学习廖雪峰老师的教程的个人总结

<http://www.liaoxuefeng.com/> （廖雪峰老师的官方网站）