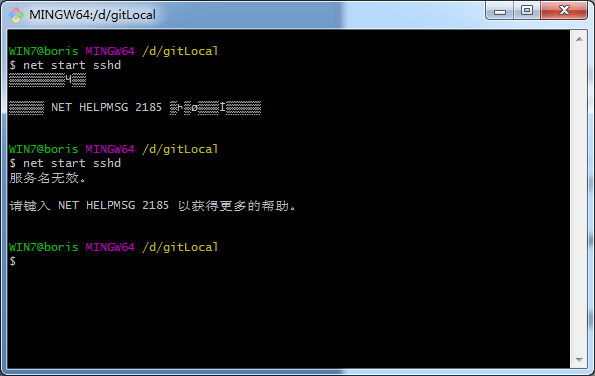
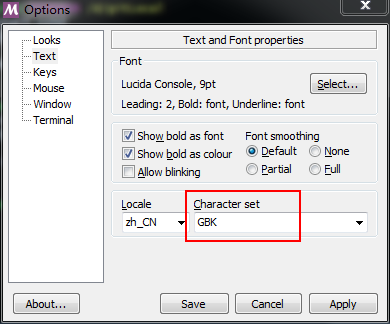
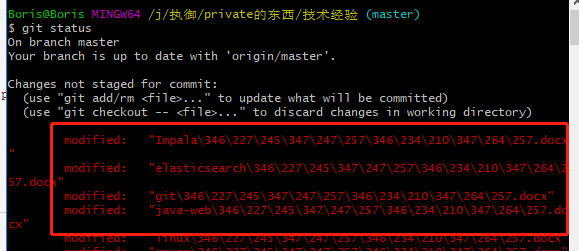
1、git bash中文乱码



解决办法：



Git status 命令显示列表，中文文件名不能显示。



解决办法：

git config --global core.quotepath false

在 git log 时中文依然不能显示，首先试试用 git --no-pager log 能不能显示中文，如果可以，则设置pager为more： git config --global core.pager more

参考：https://www.cnblogs.com/fallenmoon/p/8659678.html

2、拉去指定分支

git clone -b 分支仓库地址

例如：git clone -b develop http://git.jollycorp.com:8088/data/yuce\_platform.git

git clone http://git.jollycorp.com:8088/data/airflow.git -b yubo-dev

注：不需要加origin

3、新创建本地分支从远程分支上

git checkout -b 新分支名远程分支/本地分支

git checkout -b usercategory20180703 origin/master

从远程分支新建分支

git checkout -b test\_bran event\_bus\_origin/master

从本地分支新建分支

git checkout -b test\_bran2 master

4、从当前分支创建一个新本地分支

git checkout -b 新分支名

= git branch new\_branch + git checkout new\_branch

5、新建的本地分支是不能直接push的，因为没有远程分支对应

1>可以将本地的新建分支推到远程仓库

git push -u origin 远程分支名

例如：git push -u origin usercategory20180703

2>可以关联远程已有分支

#删除本地的某个分支

git branch -D hongchangfirst

#删除远程的分支

git branch -r -d origin/hongchangfirst

#注意这个只是删除本地的索引，而不是真正删除远程分支的内容，要想真正删除远程分支上的内容，可以这样：

git push origin :hongchangfirst

注意，冒号前面的空格不能少，相当于把一个空分支push到server上，等于删除该分支。

6、查看提交记录

git log--graph --pretty=oneline --abbrev-commit

7、推送到远程指定分支

git push origin 分支名

8、将本地代码推到远程仓库，新建的仓库

git init

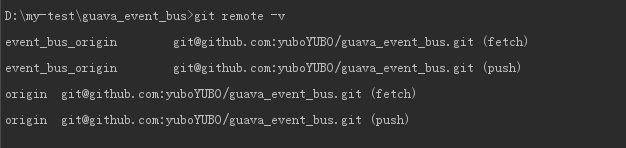
git add .

git commit -m ""

git remote add origin 远程仓库地址

git push -u origin master

注：可以给远程仓库创建多个别名，如下图

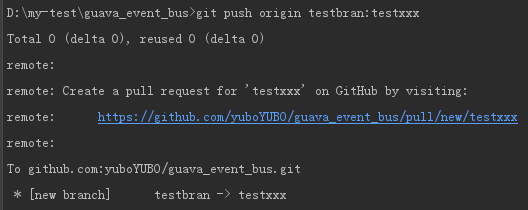


可以将本地分支推送到远程，分支不变

>git push origin testbran (本地分支)远程分支也是这名

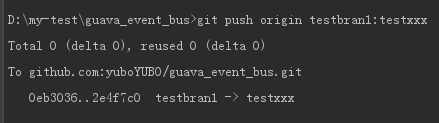


>git push origin testbran:testxxx 将本地分支推送到远程仓库，远程仓库的分支名为testxx

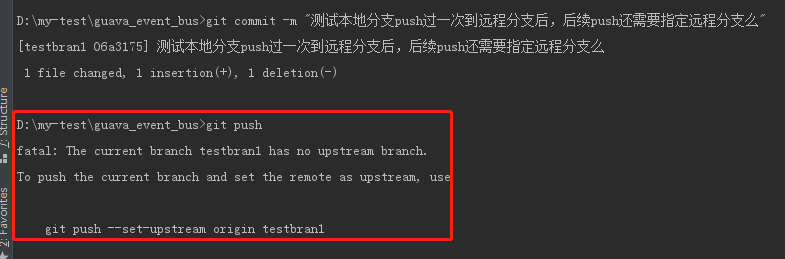


将新建的本地分支testbran1推送到远程已经存在的分支testxxx

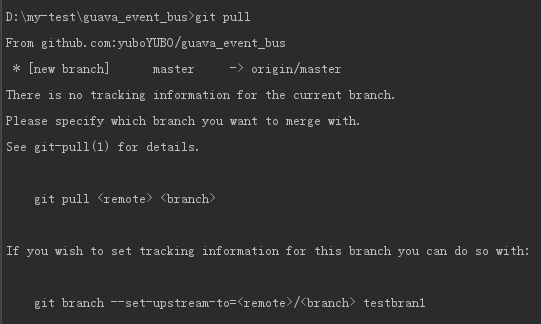
>git push origin testbran1:testxxx



下面的图证明已经push过一次到远程已经存在的分支。后续push还是不能简单的使用git push。证明本地分支和远程分支没有关联

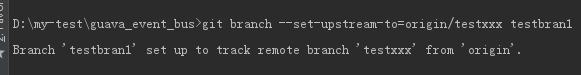


>新建的本地分支通过git push origin推送到远程仓库，或者同名或者不同名，git pull会有下面的信息提示，需要将本地分支和远程分支关联起来，才能git pull



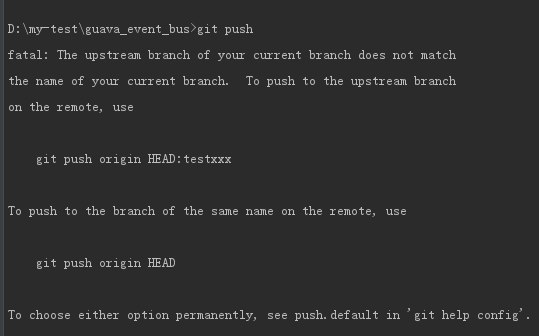
本地分支关联远程分支

>git branch –set-upstream-to=origin/testxxx testbran1



这下就可以从upstream关联的远程分支git pull了

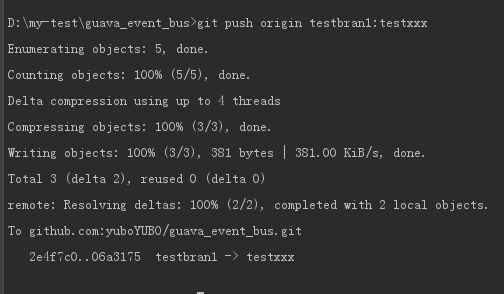
但是还是不能简单git push



提示有两种方案，1、修改配置；2、使用 git push origin HEAD:testxxx 或者git push origin HEAD

如果使用方案2显然很麻烦，如果修改配置不靠谱，会影响整个的git使用。

远程分支和本地分支同名，即本地分支push origin到远程的时候没有修改名字。是可以直接push的。



显然远程分支和本地分支如果名称不一样，操作很麻烦。所以尽量保证分支一致。

平时也很少交叉push到其他远程分支。

总结：

1、直接新建本地分支，而不是从远程仓库新建分支，然后拉取到本地。一般在不需要远程分支的情况才进行这种操作，比如个人任务，或者临时修改bug等等情况。

2、如果多人开发需要一个公共的远程分支。最好还是从远程仓库新建分支，然后拉取到本地。这样可以避免上面很多麻烦。

3、像github这种远程仓库，不能在远程直接新建，可以在本地新建分支，通过git push origin的方式推送到远程，然后将本地代码删掉，重新git clone。

1、（先进入项目文件夹）通过命令 git init 把这个目录变成git可以管理的仓库

git init

2、把文件添加到版本库中，使用命令 git add .添加到暂存区里面去，不要忘记后面的小数点“.”，意为添加文件夹下的所有文件

git add .

3、用命令 git commit告诉Git，把文件提交到仓库。引号内为提交说明

git commit -m 'first commit'

4、关联到远程库

git remote add origin 你的远程库地址

如：git remote add origin <https://github.com/cade8800/ionic-demo.git>

要添加一个新的远程仓库,可以指定一个简单的名字,以便将来引用,运行 git remote add [shortname] [url]:

5、获取远程库与本地同步合并（如果远程库不为空必须做这一步，否则后面的提交会失败）

git pull --rebase origin master

6、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。执行此命令后会要求输入用户名、密码，验证通过后即开始上传。

git push -u origin master

将本地的master分支推送到origin主机，同时指定origin为默认主机，后面就可以不加任何参数使用git push了

= git push origin 将当前分支推送到origin主机上对应的分支

如果当前分支，只有一个追踪主机，主机名可以省略。

如果当前分支与多个主机存在追踪关系，那么这个时候-u选项会指定一个默认主机，这样后面就可以不加任何参数使用git push。

不带任何参数的git push，默认只推送当前分支，这叫做simple方式。此外，还有一种matching方式，会推送所有有对应的远程分支的本地分支。Git 2.0版本之前，默认采用matching方法，现在改为默认采用simple方式。

7、状态查询命令

git status

git remote 列出远程分支

列出远程分支(fork就是仓库的地址，不是branch)和地址 git remote -v | --verbose

Git global setup

git config --global user.name "boris@jollycorp.com"

git config --global user.email "boris@jollycorp.com"

Create a new repository

mkdir eshivedata

cd eshivedata

git init

touch README.md

git add README.md

git commit -m "first commit"

git remote add origin http://git.jollycorp.com:8088/boris/eshivedata.git

git push -u origin master

Push an existing Git repository

cd existing\_git\_repo

git remote add origin http://git.jollycorp.com:8088/boris/eshivedata.git

git push -u origin master

|  |
| --- |
| …or create a new repository on the command line  echo "# Jest-test" >> README.md  git init  git add README.md  git commit -m "first commit"  git remote add origin git@github.com:yuboYUBO/Jest-test.git  git push -u origin master  …or push an existing repository from the command line  git remote add origin git@github.com:yuboYUBO/Jest-test.git  git push -u origin master |

Git remote 列出已经存在的远程分支

Git remote -v | --verbose 列出详细信息，在每一个名字后面列出远程url

origin git@github.com:yuboYUBO/springboot-swagger2.git (fetch)

origin git@github.com:yuboYUBO/springboot-swagger2.git (push)

给本地仓库，添加一个远程仓库（关联一个远程仓库）。可以指定一个简单的名字，以便将来引用，git remote add [shortname] [远程url]

可以通过远程仓库的简单名字来访问远程仓库

git push 远程仓库名，比如：git push origin

如果本地仓库只是关联了一个远程仓库，可以省略远程仓库名或者地址，git push就好了。如果本地仓库关联了多个远程仓库，可以使用-u参数指定默认关联的远程仓库， git push -u 远程仓库名或者地址分支名，以后push变更时直接使用git bush就好了。

git push 这个将simple模式——只推送当前分支，

##git 回退到上一次提交的commit操作

git如何撤销上一次commit操作

1.第一种情况：还没有push，只是在本地commit

git reset --soft|--mixed|--hard <commit\_id>

git push develop develop --force (本地分支和远程分支都是 develop)

这里的<commit\_id>就是每次commit的SHA-1，可以在log里查看到

--mixed    会保留源码,只是将git commit和index 信息回退到了某个版本.  
--soft   保留源码,只回退到commit信息到某个版本.不涉及index的回退,如果还需要提交,直接commit即可.  
--hard    源码也会回退到某个版本,commit和index 都会回退到某个版本.(注意,这种方式是改变本地代码仓库源码)

 当然–soft 和–hard --mixed的区别可以如下理解：

* soft  
  只操作了HEAD，暂存区和work都没有被影响
* hard  
  操作了HEAD、暂存区和work，都被影响了
* mixed  
  操作了HEAD、暂存区，work没有被影响

当然有人在push代码以后,也使用 reset --hard <commit...>回退代码到某个版本之前,但是这样会有一个问题,你线上的代码没有变,线上commit,index都没有变,当你把本地代码修改完提交的时候你会发现全是冲突.....这时换下一种

2.commit push 代码已经更新到远程仓库

对于已经把代码push到线上仓库,你回退本地代码其实也想同时回退线上代码,回滚到某个指定的版本,线上,线下代码保持一致.你要用到下面的命令

git revert <commit\_id>

revert 之后你的本地代码会回滚到指定的历史版本,这时你再 git push 既可以把线上的代码更新。

注意：git revert是用一次新的commit来回滚之前的commit，git reset是直接删除指定的commit，看似达到的效果是一样的,其实完全不同。

第一:上面我们说的如果你已经push到线上代码库, reset 删除指定commit以后,你git push可能导致一大堆冲突.但是revert 并不会.  
第二:如果在日后现有分支和历史分支需要合并的时候,reset 恢复部分的代码依然会出现在历史分支里.但是revert 方向提交的commit 并不会出现在历史分支里.  
第三:reset 是在正常的commit历史中,删除了指定的commit,这时 HEAD 是向后移动了,而 revert 是在正常的commit历史中再commit一次,只不过是反向提交,他的 HEAD 是一直向前的.

git rebase 和commit的差异

master分支，节点链表指向为：c1<--c3<--c4  
dev分支，节点链表指向为：c1<--c2<--c5  
master分支和dev分支祖先为c1，假定在master分支上做git merge dev合并，得到的提交历史为：  
c1<--c2<--c3<--c4<--c5<--c6（c1、c4、c5做了一次三方合并发现冲突，手工处理完毕后git add/commit增加了提交节点c6）  
采用git merge dev处理提交log是按照时间戳先后顺序的。

假定采用的是git rebase处理过程为：

git checkout dev

git rebase master # 将dev上的c2、c5在master分支上做一次衍合处理

# git提示出现了代码冲突，此处为之前埋下的冲突点，处理完毕后

git add readme # 添加冲突处理后的文件

git rebase --continue # 加上--continue参数让rebase继续处理

此处处理后的节点为：

c1 c3 c4 c2 c5 # 此处不是按照时间顺序处理的综合表现，git rebase可以得到一个更加简洁的提交历史，无需多了c6。处理完毕后，git checkout master加上git merge dev，git会智能采用f-f（fast-forward）处理。

总结为：  
git rebase过程相比较git merge合并整合得到的结果没有任何区别，但是通过git rebase衍合能产生一个更为整洁的提交历史。  
如果观察一个衍合过的分支的历史提交记录，看起来会更清楚：仿佛所有修改都是在一根线上先后完成的，尽管实际上它们原来是同时并行发生的。

一般我们使用衍合的目的，是想要得到一个能在远程分支上干净应用的补丁，比如某个项目你不是维护者，但是想帮点忙，最好使用衍合处理。  
先在自己的一个分支进行开发，当准备向主项目提交补丁的时候，根据最新的orgin/master进行一次衍合操作然后再提交，这样维护者就不需要任何整合工作。

实际为：把解决分支补丁同最新主干代码之间的冲突的责任，划转给由提交补丁的人来解决。  
作为维护项目的人只需要根据你提供的仓库地址做一次快进合并，或者直接采纳你提交的补丁。

衍合的风险，请务必遵循如下准则：  
一旦分支中的提交对象发布到公共仓库，就千万不要对该分支进行衍合操作。一句话rebase慎用。

什么时候我应该用git merge?

merge执行一个合并，合并其他分支的工作，从而放弃其他的分支。比如开发分支完成了，要提测，需要就可以在测试分支上面merge开发分支

1、仅仅是一个local的，临时性的分支，创建它的目的仅仅是为了在开发它的同时不影响其他分支。

2、如果我们从master新建开发分之后，master分支有新的提交。我们想保持master分支清晰，我们可以将新建开发分支的提交点后移，可以在开发分支上面使用rebase命令。

3、git rebase的应用场景。当几个同事在开发分支上开发一个功能，你开发了一些功能，需要提交push到远程分支。这时其他同事已经push了变更，当你提交的时候，不被允许，因为你的代码太陈旧了。

这时需要git rebase 一下（用git pull不好（git fecth + git merge）,会产生一个空提交，没必要）

4、在开发分支工作完成，需要和到master分支，这时master分支已经有很大的变化了。可以先git rebase master 后，然后再将开发分支merge到master

5、git rebase -i master(梳理历史信息，比如合并成一个commit)+git merge

git merge 默认是 fast forward模式

$ git merge testff

|  |
| --- |
| Updating 30a974d..5a036ee  Fast-forward  readme | 1 +  1 file changed, 1 insertion(+)  create mode 100644 readme |

git merge --no-ff -m "merge with no-ff" dev

禁用Fast forward模式，Git就会在merge时生成一个新的commit

在git merge的时候，如果有冲突产生，需要解决冲突，也会产生一个commit

问题：fatal: refusing to merge unrelated histories

出现问题的场景：在远端建立了一个仓库，并有提交（比如初始化了readme.md文件）

现在要将本地的仓库的代码关联到远端仓库

>git init

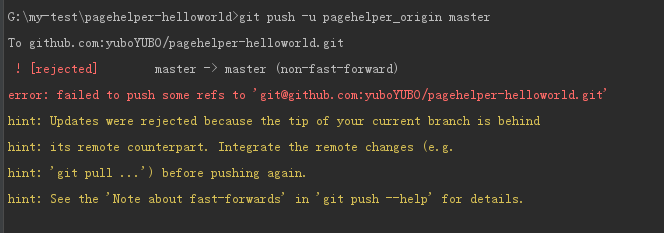
>git add .

>git commit –m “first-commit”

>git remote add pagehelper\_origin github.com:yuboYUBO/pagehelper-helloworld.git

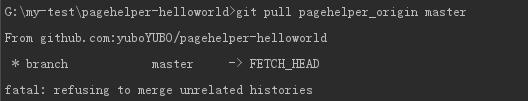
>git push –u pagehelper\_origin master

这一步会提示：



就是说不能push，需要先pull。

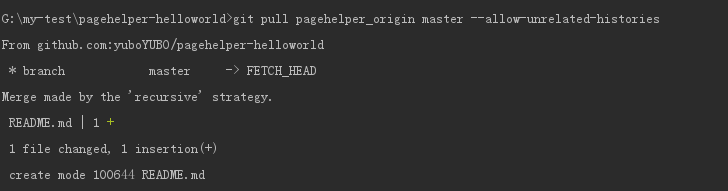
>git pull pagehelper\_origin master



拒绝合并未关联的历史提交。现在就卡这儿了。不能讲本地代码推到远程仓库去。

解决办法git pull的时候添加参数--allow-unrelated-histories

>git pull pagehelper\_origin master --allow-unrelated-histories



然后

>git push –u pagehelper\_origin master

完成。

删除远程分支feature-qe-life-circle-initial

> git push origin --delete feature-qe-life-circle-initial

删除本地分支master

> git branch -d master

git push -u <远程仓库名> --all //这里是推送所有分支

git push -u origin master //这是我们平时使用的推送命令，把master分支推送到origin仓库中

git push -u <远程仓库名> --tags //这里是推送所有tag,如果你平时有用到tag的话

比较两个分支的不一样

git diff testff newtestff

也可以在add之前，比较文件有哪些变化

git diff file

没有提交的commit，要取消掉

git reset HEAD~1 --hard

git reset 适合取消本地未push的 commit，不适合已经push到远程仓库的commit。会产生很多冲突，可能一万个。。。

git revert 向前提交一个commit（以前的commit）