Git开发基础

# Git环境搭建

## Ubuntu OS安装配置

### 安装git

安装命令：

***$ sudo apt-get install git***

基本设置：

***$ git config --global user.name “username”***

***$ git config --global user.email “username@baidu.com”***

### 安装repo

用户主目录下创建bin目录：

***$ mkdir bin***

安装repo：

***$ curl http://10.32.37.53:8088/public/repo > ~/bin/repo***

修改执行权限：

***$ chmod a+x ~/bin/repo***

### 安装python

查看Python版本：

***$ python***

若不为2.6以上（含2.6），执行以下安装命令：

***$ sudo apt-get install python***

### 设置.netrc文件

在用户主目录中编辑.netrc文件：

***$ vi ~/.netrc***

machine 172.22.1.83 #此为git服务器地址

login username #git账户，同svn账户相同

password 123 #git账户密码，同svn账户密码相同。

保存退出。

例如：

***machine 172.22.1.83***

***login lijinsuo***

***password 123***

## Windows安装配置

### 安装msysgit，默认安装即可

必须要先安装msysgit，后安装tortoisegit，下载地址：***http://code.google.com/p/msysgit/Git-1.7.4-preview20110204.exe***

### 安装Tortoisegit

默认安装即可，下载地址：

***http://code.google.com/p/tortoisegit/Tortoisegit-1.6.5.0-32bit.msi***

### 设置用户名和email地址

点击“开始”——“程序”——“TortoiseGit”——“Setting”，然后点击“git”——“config”，输入用户名和email地址，然后点击“应用”。

# Git代码获取

## 获取所有分支代码

### millennium-baidu

进入开发主目录：

***$ cd ~/dev***

创建millennium-baidu目录：

***$ mkdir millennium-baidu***

***$ cd millennium-baidu***

repo初始化：

***$ repo init --repo-url=http://172.22.1.83:8088/git/repo.git -u http://172.22.1.83:8088/git/manifest.git -b millennium-baidu***

代码同步：

***$ repo sync***

设置编译环境：

***$ . ccienv/da80\_env***

进行编译：

***$ make -j8 (or make –j4)***

### master

进入开发主目录：

***$ cd ~/dev***

创建master目录：

***$ mkdir master***

***$ cd master***

repo初始化：

***$ repo init --repo-url=http://172.22.1.83:8088/git/repo.git -u http://172.22.1.83:8088/git/manifest.git***

代码同步：

***$ repo sync***

设置编译环境：

***$ . build/envsetup.sh***

进行编译：

***$ lunch 3 (lunch 5 for NS)***

***$ make -j8 （or make –j4）***

### far2-baidu

进入开发主目录：

***$ cd ~/dev***

创建far2-baidu目录：

***$ mkdir far2-baidu***

***$ cd far2-baidu***

repo 初始化：

***$ repo init --repo-url=http://172.22.1.83:8088/git/repo.git -u*** [***http://172.22.1.83:8088/git/manifest.git***](http://172.22.1.83:8088/git/manifest.git) ***-b far2-baidu***

代码同步：

***$ repo sync***

设置编译环境：

***$ chmod a+x setmtkenv.sh***

***$ ./setmtkenv.sh***

进行编译：

***$ ./makeMtk –t hazel new***

更换产品重新编译：

***$ ./makeMtk –t hazel clean***

***$ ./makeMtk –t far2 clean***

#### 其他编译命令

编译OTA target files：

***$ ./makeMtk –t hazel otapackage***

编译recovery image：

***$ ./makeMtk –t hazel recoveryimage***

#### 常用编译命令

Full build：

***$ ./makeMtk –t hazel new***

clean build：

***$ ./makeMtk –t hazel clean***

增量build：

***$ ./makeMtk –t hazel remake***

单独build component：

***$ ./makeMtk –t hazel mm <模块目录>***

### n880e

进入开发主目录：

***$ cd ~/dev***

创建n880e目录：

***$ mkdir n880e***

***$ cd n880e***

repo初始化：

***$ repo init --repo-url=http://172.22.1.83:8088/git/repo.git -u http://172.22.1.83:8088/git/manifest.git -b n880e-baidu***

代码同步：

***$ repo sync***

设置编译环境：

***$ cd device/zte/atlas40 && ./copy-files.sh && cd – $ . ./build/envsetup.sh***

进行编译：

***$ lunch 6***

***$ make -j8 （or make –j4）***

*编译后可以按附件中说明把Image(boot.img, system.img.ext4, userdata.img.ext4) 下载至N880E中(N880E 中目标Image格式采用ext4, 没再用yaffs2)。*

#### 其他编译命令

编译OTA target files：

***$ make target-files-package***

### 所有分支统一切换

如果使用同一个开发目录进行所有分支项目的开发，可以使用repo命令快速切换所有分支项目代码到指定分支。

检出所有git库到某一分支：

***$ repo init –b <branch-name>***

同步所有git库：

***$ repo sync***

查看所有git库：

***$ repo brances***

切换所有git库到某一分支：

***$ repo start <branch-name> --all***

## 获取单个分支代码

获取millennium-baidu分支：

***$ git clone***[***http://172.22.1.83:8088/git/chunlei/baidu/packages/apps/BaiduUpdate.git***](http://172.22.1.83:8088/git/chunlei/baidu/packages/apps/BaiduBackupRestore.git)***-b millennium-baidu***

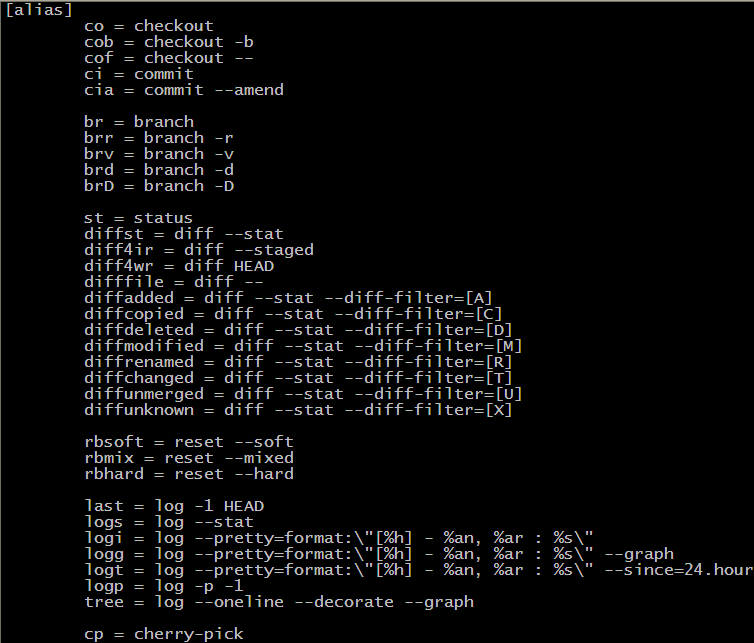
获取master分支：

***$ git clone***[***http://172.22.1.83:8088/git/chunlei/baidu/packages/apps/BaiduUpdate.git***](http://172.22.1.83:8088/git/chunlei/baidu/packages/apps/BaiduBackupRestore.git)***-b master***

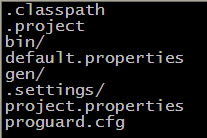
# Git环境配置

*先进行Git环境的个性化配置，因为后续命令示例都是据此简写。*

## .gitconfig参考



## git\_ignore\_global.txt参考



# Git代码修改

*注：以millennium-baidu分支BaiduUpdate为例，主目录为~/dev/millennium-baidu，以下命令默认初始当前目录为该主目录。*

## 场景一：一般代码修复

### 常规流程

进入git工程目录：

***$ cd ./baidu/packages/apps/BaiduUpdate***

查看当前git分支：

***$ git br***

******

检出远程主分支到本地主分支（建议本地主分支名称和远程分支名称相同，但本地临时分支可以根据需要自由命名，请参照“注意事项”）：

***$ git cob millennium-baidu origin/millennium-baidu***

确保本地主分支代码最新：

***$ git pull origin millennium-baidu***

修改本地主分支代码，然后本地检入：

***$ git ci –am “Commit message.”***

注意“-am”是“-a”和“-m”的缩写，其中“-a”表示将多有修改文件（不包括新增文件，有新增文件要用“git add <file-name>”命令添加后再执行以上命令）做本地提交。

本地编译确认（*第一次进入工程目录时需要运行前三条命令*）：

***$ cd ~/dev/millennium-baidu***

***$ . ./ccienv/da80\_userdebug\_env***

***$ cd -***

***$ mm***

代码远程提交（提交之前执行git st确定没有遗漏需要添加的文件）：

***$ repo upload .*** [--re=somebody@baidu.com](mailto:--re=somebody@baidu.com)

### 同步服务器

同步服务器上最新的代码修改：

***$ git fetch --all***

***$ git rebase origin/branch-name***

### 其他方法

如果已经存在和主分支名称相同的本地分支，并且做了本地提交，建议做如下的分支名称转换，以便让代码提交历史更加清晰：

将当前与主分支同名的分支检出到临时分支mill-baidu：

***$ git cob mill-baidu***

***$ git br***

******

强制删除与主分支同名的分支：

***$ git br –D millennium-baidu***

检出远程主分支到本地主分支：

***$ git cob millennium-baidu origin/millennium-baidu***

确定本地主分支代码最新：

***$ git pull origin millennium-baidu***

本地主分支从本地临时分支mill-baidu合并已有的代码修改：

***$ git merge --squash mill-baidu***

然后本地检入：

***$ git ci –am “Commit message.”***

注意“-am”是“-a”和“-m”的缩写，其中“-a”表示将多有修改文件（不包括新增文件，有新增文件要用“git add <file-name>”命令添加后再执行以上命令）做本地提交。

本地编译确认（*第一次进入工程目录时需要运行前三条命令*）：

***$ cd ~/dev/millennium-baidu***

***$ . ./ccienv/da80\_userdebug\_env***

***$ cd -***

***$ mm***

代码远程提交（提交之前执行git st确定没有遗漏文件未添加）：

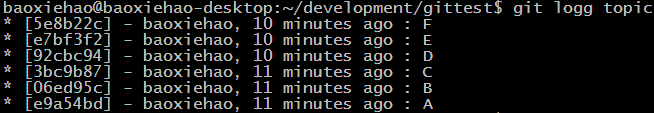
***$ repo upload .*** [--re=somebody@baidu.com](mailto:--re=somebody@baidu.com)

## 场景二：多分支代码管理

在git开发过程中经常会遇到某些修改涉及大量文件修改的特性开发，此时需要在本地特性分支进行代码修改和代码提交，等到特性分支开发完成再融合到主分支。因此，特别需要注意多分支代码的管理，避免代码融合的失败和代码历史的混乱。

例如，现在本地有master主分支和topic特性分支，其提交历史如下：





用图形示意图表示即如下：

**A-B (master)**

**\**

**C-D-E-F (topic)**

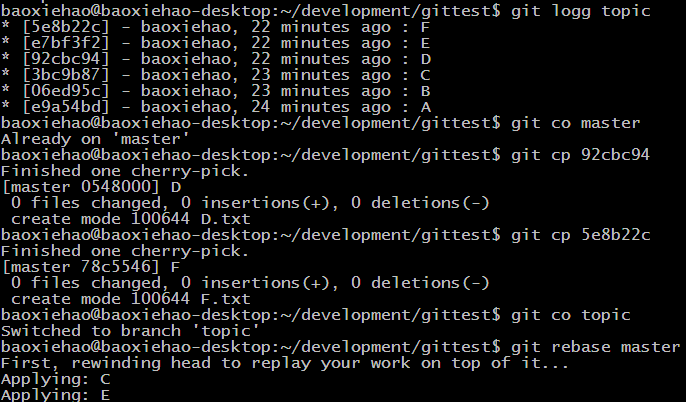
现在假设topic分支的D和F特性已经保证没有问题，需要融合到master主分支，而且又不能把提交历史搞得很乱，最好就变成如下：

**A-B-D-F (master)**

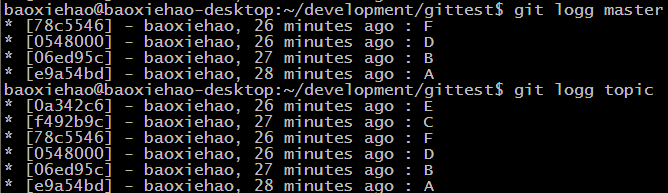
**\**

**C-E (topic)**

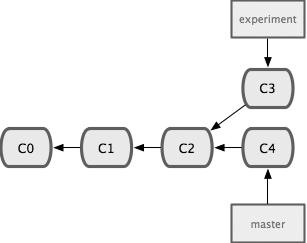
可以通过cherry-pick命令进行不同分支间指定提交的融合：



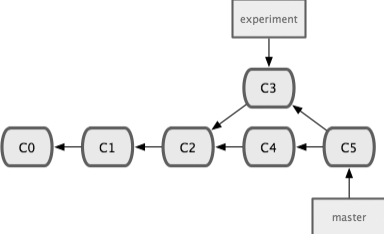
此时master分支和topic分支的提交历史如下：



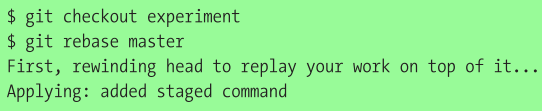
有人可能会问rebase和直接merge有什么区别，下面举例说明（选自《Pro Git》）。当前分支示意图：

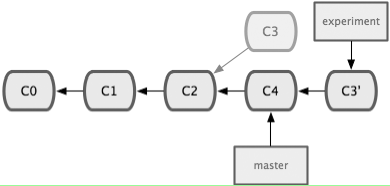


在master主分支执行“git merge experiment”命令，也称为3-way merge，得到的提交历史如下：

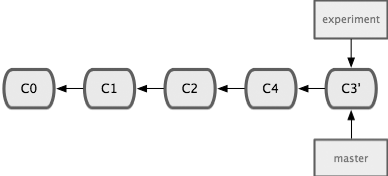


当然你也可以执行rebase命令，在master主分支上打上experiment分支各个修改对应的patch：





然后再执行fast-forword merge命令（即“git co master; git merge experiment”），得到更加清晰的提交历史：

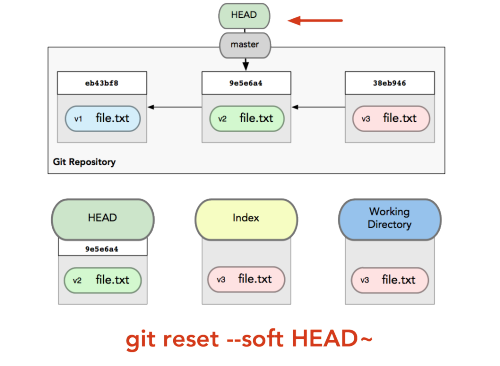


## 场景三：本地检入回滚

假设在本地分支已经检入当前修改，但是由于各种原因需要回滚本地仓库到指定版本，可以使用git reset命令来将修改回滚到指定的历史提交。回滚的时候有3种情况可选，一是回滚当前提交到指定历史提交的更改并暂存到索引文件，二是回滚当前提交到指定历史提交的更改并恢复到工作文件，三是回滚当前提交到指定历史提交的更改并丢弃（谨慎操作）。

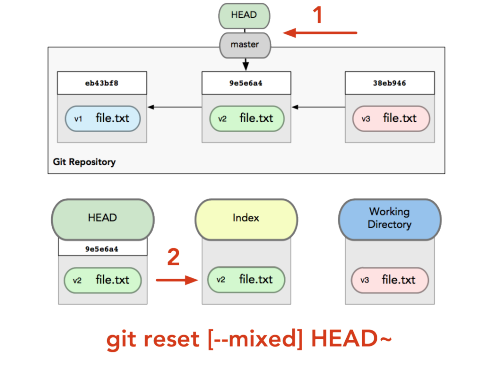
### 第一种情况（回滚到当前版本的前一个版本）

***$ git reset --soft HEAD~***



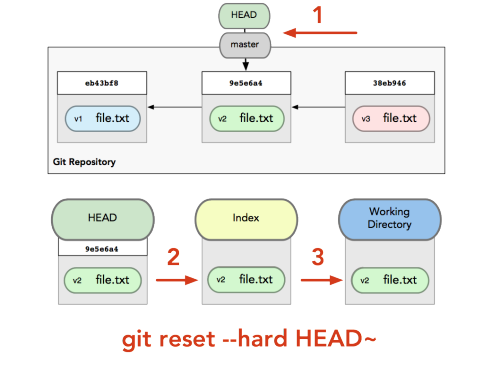
### 第二种情况（回滚到当前版本的前一个版本）

***$ git reset HEAD ~***



### 第三种情况（回滚到当前版本的前一个版本）

***$ git reset --hard HEAD~***



## 场景四：分支代码误删除

git reflog 可以查看所有分支的所有操作记录（包括commit和reset的操作），包括已经被删除的commit记录，git log则不能察看已经删除了的commit记录。举一个例子，假设有三个commit，git status：

*commit3: add test3.c*

*commit2: add test2.c*

*commit1: add test1.c*

如果执行 git reset --hard HEAD~1 则删除了commit3，如果发现删除错误了，需要恢复commit3，这个时候就要使用 git reflog：

**HEAD@{0}: HEAD~1: updating HEAD**

**63ee781 HEAD@{1}: commit: test3:q**

红色加粗的即是被删除了的 commit3，运行git log则没有这一行记录。可以使用如下命令恢复commit3：

***$ git reset --hard 63ee781***

现在git log后可以看到：

*commit3: add test3.c*

*commit2: add test2.c*

*commit1: add test1.c*

这里也可以使用另外一种方法来实现：

***$ git cp 63ee781***

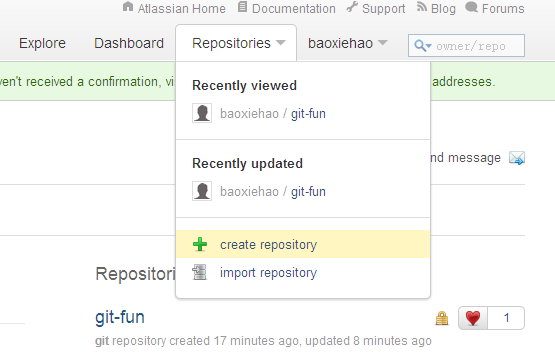
# Git实际操作

为了更好地熟悉和学习Git命令，可以在创建远程git仓库（有免费的git服务器，比如[github](https://github.com/)和[bitbucket](https://bitbucket.org/)），然后在本地进行代码拉取推送和分支管理等练习。下面以bitbucket进行示例。

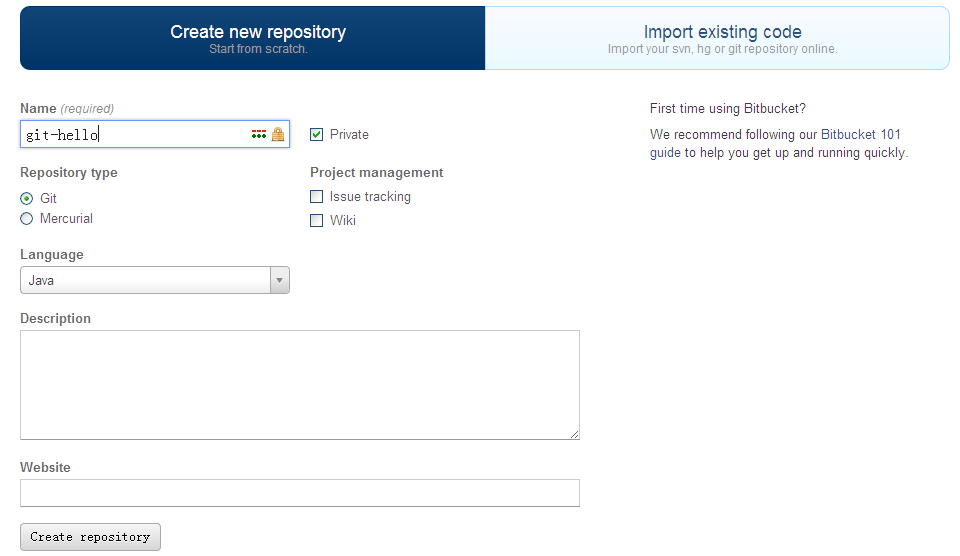
## 免费注册bitbucket账号

## 创建远程git仓库

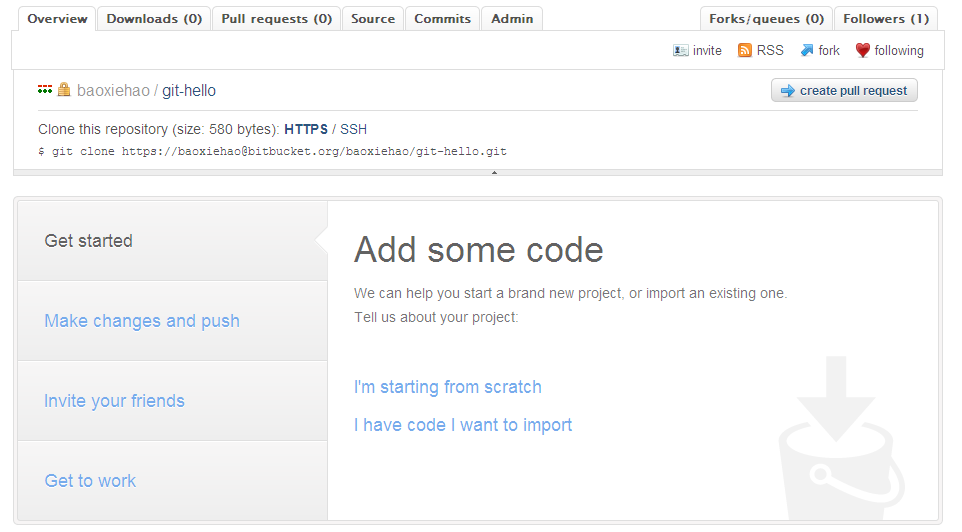
在个人账号页面选择“repositories”——“create repository”：



输入“name”，选择“language”，点击“Create repository”：



以下是创建成功的页面：

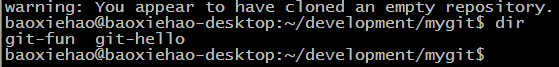


进入本地工作目录，执行以下git命令：

***$ git clone https://baoxiehao@bitbucket.org/baoxiehao/git-hello.git***



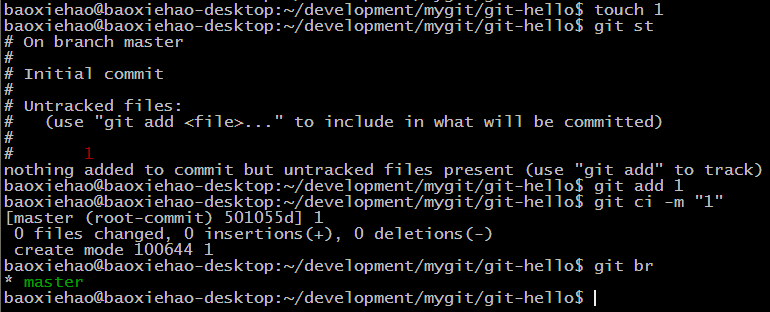
输入个人帐户密码后就从远端成功地复制代码分支到本地：



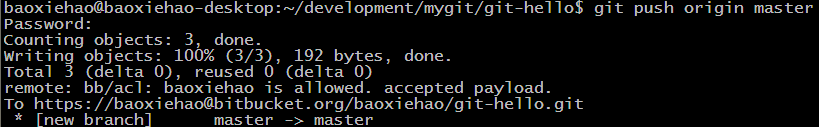
命令显示警告“复制空的仓库”，可以暂时忽略，因为创建的是空仓库。

## 代码推送到远程git仓库

现在可以进入git-hello目录进行正式地命令操作：

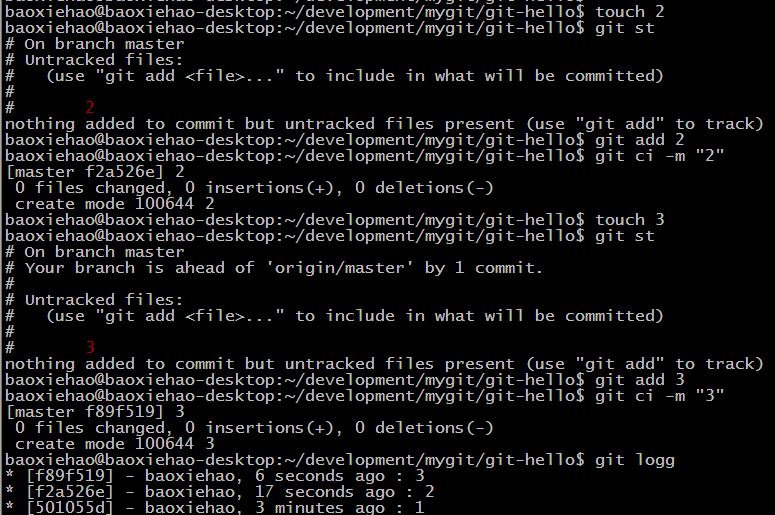


本地提交之后，推送到远端：

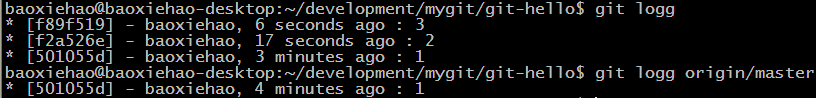


## 两种分支操作方式对比

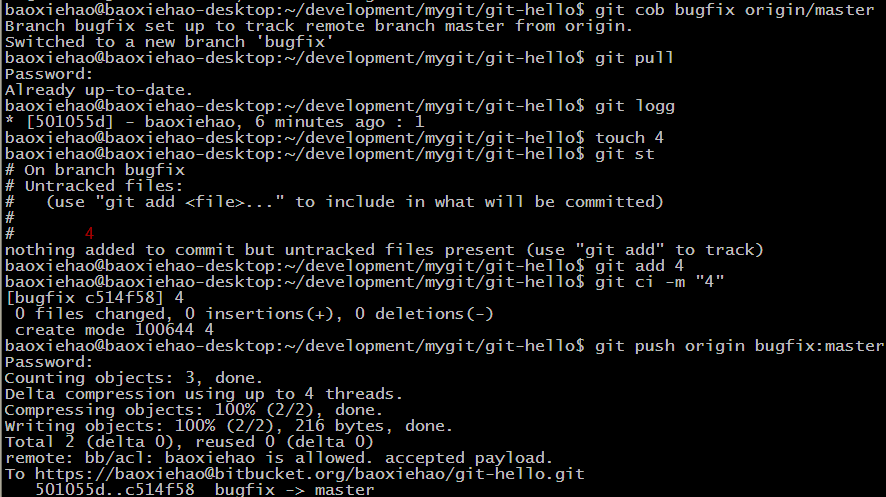
为了更好地说明分支管理操作，继续添加2和3两个文件：



此时本地分支有3个提交记录，而远程分支只有1个，因为还未push到远端：



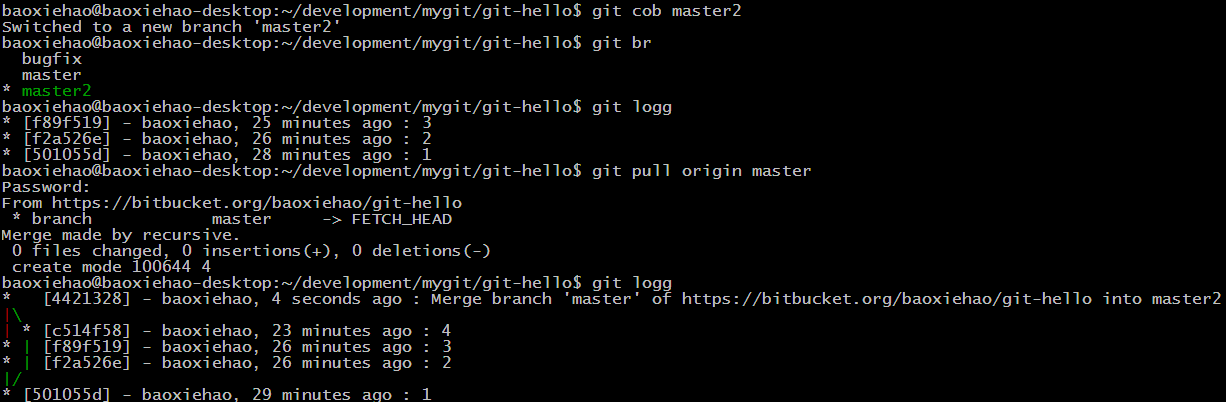
现在假设我们遇到一个需要紧急修复的bug，但是本地提交还未彻底验证，因此我们暂时切换到bugfix分支（源自远程master分支），进行代码修复，然后推送到远端：



紧急的bugfix完成之后，我们有足够的时间进行本地提交的测试了，假设测试完成没有问题，于是我们需要推送到远端，但是要注意的是，此时本地主分支和远程主分支已经分化开来了，本地主分支的提交历史是1—2—3，而远程主分支的提交历史是1—4。因此，现在需要解决分支的合并问题。通常有两种解决方法，下面分别介绍以示区别。

### 本地主分支merge远程主分支

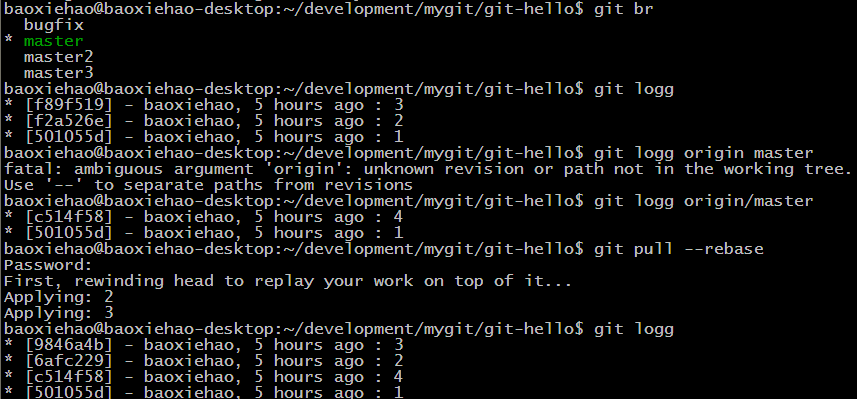
此时我们可以直接merge远端代码——git pull origin master（先检出本地master分支到master2分支）：



可以看到，正如前面所说的，merge是三方合并，导致提交历史不甚清晰，特别是当本地主分支和远程主分支有较大差异的时候（示例中是有1个提交先于本地的2个提交）。

### 本地主分支rebase远程主分支

现在试试另一种rebase远端代码的方法——git pull --rebase [origin master]（加上rebase的意思是：1、把本地 repo.从上次 pull 之后的变更暂存起来；2、恢复到上次pull时的状态；3、套用远端的变更；4、套用刚刚暂存下来的本地变更。详细说明可以参考 [pull with rebase](http://www.gitready.com/advanced/2009/02/11/pull-with-rebase.html)）：



可以看到，rebase和merge之后的提交历史显然是前者更加清晰，因此强烈推荐使用rebase命令进行本地主分支和远程主分支的融合。

注意：有可能pull远程分支到本地之后会产生conflicts，此时只要解决冲突文件，然后git add <conflicting files>；git rebase --continue即可。

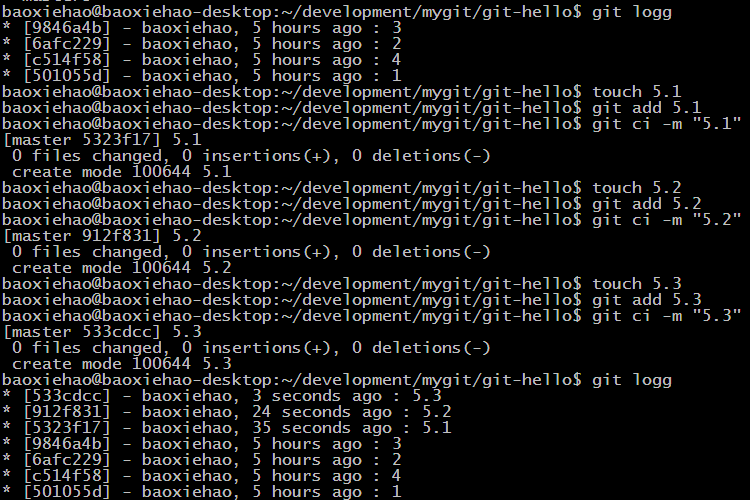
### merge和rebase的比较

到底何时该用merge，何时该用rebase？rebase跟merge类似，出现conflict会暂停rebase，需要手动修复后才可以继续。这也是rebase比merge复杂一点的地方：merge如果发生conflict，只需要解决冲突一次，然后commit即可；而rebase的conflict可能会发生在每次套用变更的时候，所以可能需要解决conflict好几次。因此，如果修改比较多，预期会有较多的conflict，建议使用merge（不过多次大范围的修改是不是一开始就该多开一个branch来做呢？）；如果修改比较少，不大预期有conflict，则建议使用rebase。

## 本地检入多变一

Git的核心思想是少量修改多次检入，而对于同一个特性的大量修改，本地可能对应多个检入，因此当最后测试通过需要远程检入的时候，我们就需要把本地的多个检入合并成一个检入，然后再进行远程检入。

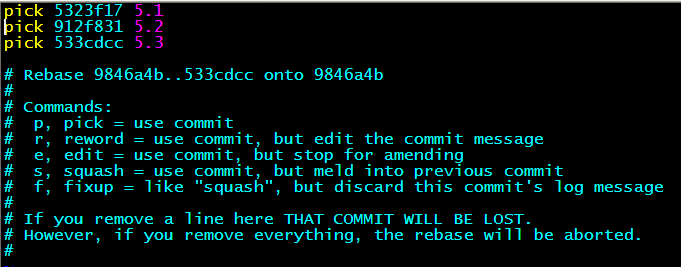
首先，我们在本地进行3次检入：



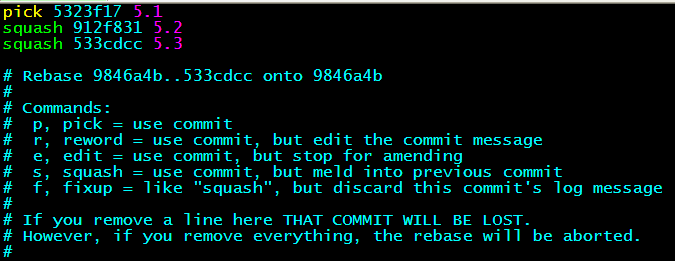
然后，我们执行git rebase命令进行检入合并：



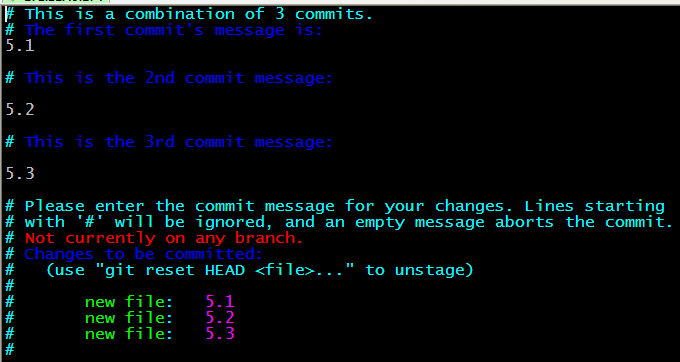
其中，HEAD~3表示进行前3次检入的合并，命令输入之后显示如下：



参照说明将pick改成squash进行检入合并（还有其他选项可选，不一一介绍，参见如下英文文档）：



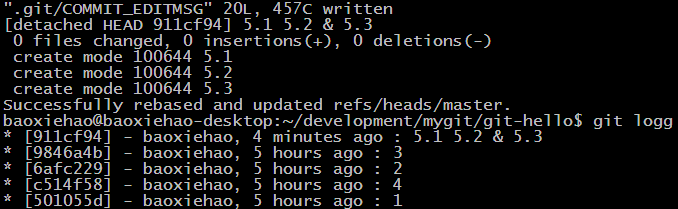
保存后显示编辑检入描述：



根据需要修改检入描述：



保存后显示rebase成功，此时查看log就只有1次本地检入了：

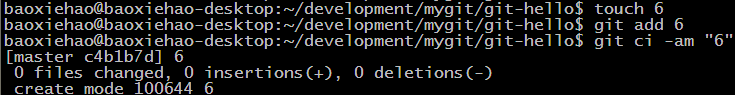


千万注意：不要将已经push到远程仓库的检入进行rebase操作！

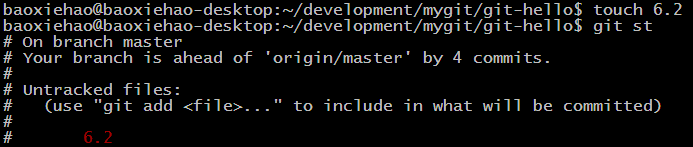
## 本地检入修改

其实如上的多次检入合并可以在第二次检入的时候执行“git ci --amend”命令避免，也就是说每次本地检入都将合并到前一次本地检入，因此多次本地检入最后仅得到一个检入。其优点是操作简单，但缺点是本地检入历史不清晰，另外修改检入历史局限性更大。

首先进行本地检入：



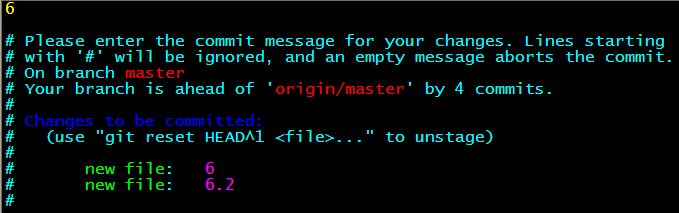
再次进行本地编辑，新增文件：



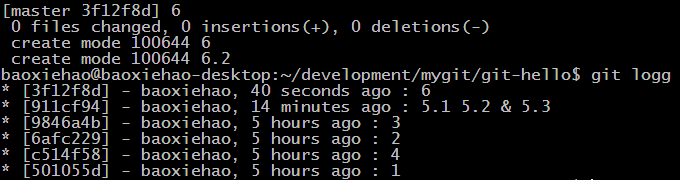
然后执行“git ci --amend”命令进行检入修改，即合并当前检入到前一检入（新增文件需要执行“git add”方可）：



修改检入描述：



保存后，检入修改成功：



# 其他命令

## 版本回滚

***$ git reset target-commit --merge***

***$ git reset ORIG\_HEAD***

***$ git commit -a -m***

## 跟踪文件提交历史

***$ git log -p <file-name>***

***$ git blame -L line\_from,line\_to <file-name>***