博客地址

<http://blog.csdn.net/Dawn_sf/article/details/78322163>

我们在上一篇博客[分支的基本操作](http://blog.csdn.net/dawn_sf/article/details/78240951" \t "http://blog.csdn.net/Dawn_sf/article/details/_blank)当中已经学会了，创建，查看，切换，合并，删除一个分支, 那么既然我们拥有了基础的知识之后，那我们需要如何

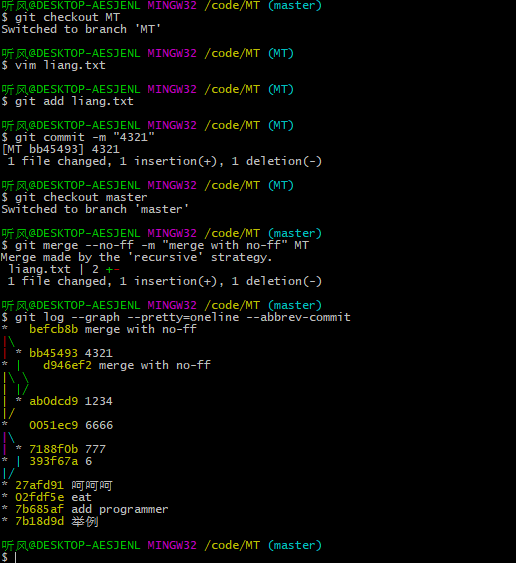
实际运用分支呢？ 那么在这个博客我们就能够找到答案. 通常，合并分支时，如果可能的话，GIT会用Fast forword模式，但是在这种模式下，删除分

支后，会丢掉分支信息.如果要强制禁用Fast forward模式，Git就会在merge时生成一个新的commit，这样，从分支历史上就可以看出分支的信息.

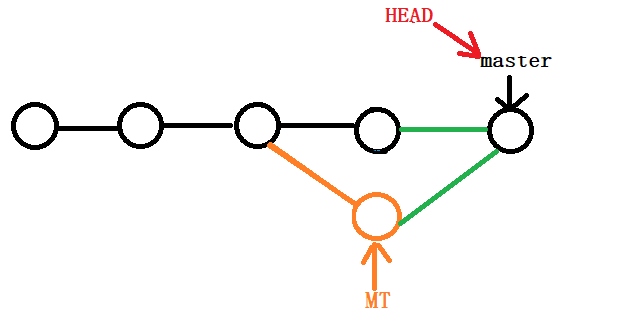
然后我们可以直接先使用文字，陈述一下我们的过程，首先我们创建一个MT分支: $ git checkout -b MT 然后在分支上面修改liang.txt文件，然后进

行git add和git commit操作，然后切回主分支master，准备合并dev分支，请注意--no-ff参数，表示禁用Fast forward 然后在程序运行结束后我们再

使用git log指令查看分支的历史情况,所以过程就是这样，我们来看看代码实现:



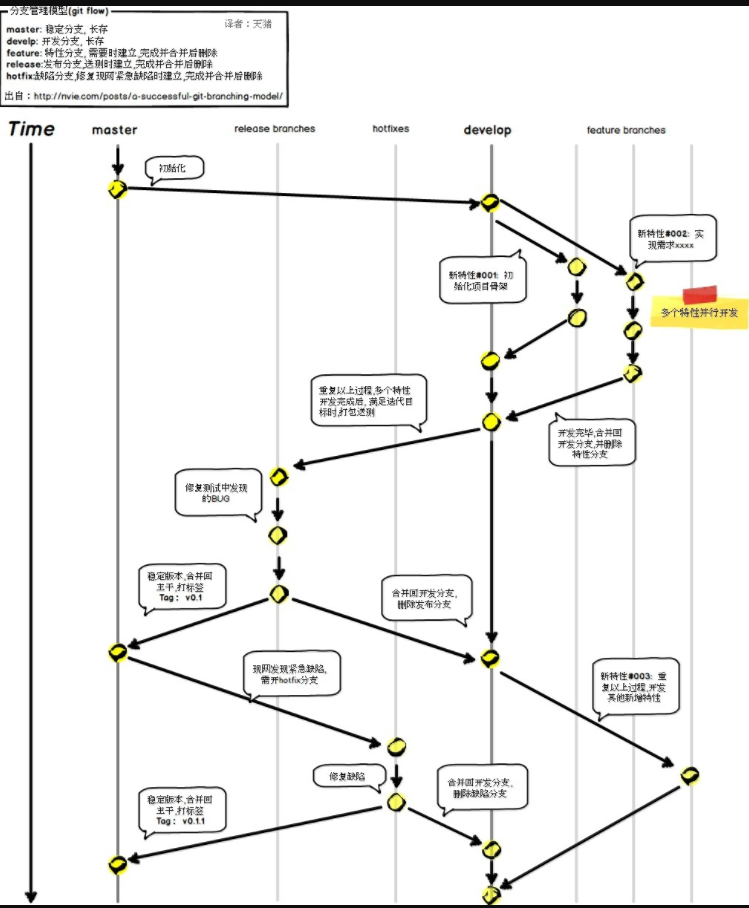
其实这里我们只需要注意合并前的最后一行即可. 不使用Fast forward模式的话，merge之后就是这样了:



我们在实际开发当中，首先master分支应该是非常稳定的，也就是仅仅用来发布新版本，平时不能在上面干活; 干活都在另外一个备用的稳定分支之上

先叫他develop分支，平时都是这样比如要发布3.0版本时，先把develop分支合并到master分支上面，然后在master分支之上发布3.0版本.

然后平时在公司，你和你的程序猿朋友们都拥有各自的分支，当你写好代码你就往develop上面添加就OK了.所以团队合作的分支就像是下面这个图片:



# bug分支

我们现在没写过多少程序也会发现，程序的bug简直就跟吃饭一样，闲了没事了就来一个BUG. 所以你这整个分支上肯定会不定时的出现各种各样的bug

这个时候，GIT岂能坐视不管，由于分支功能很强大，所以有bug我们完全可以通过一个新的临时的分支来修复，修复后，合并分支，然后将临时的分支

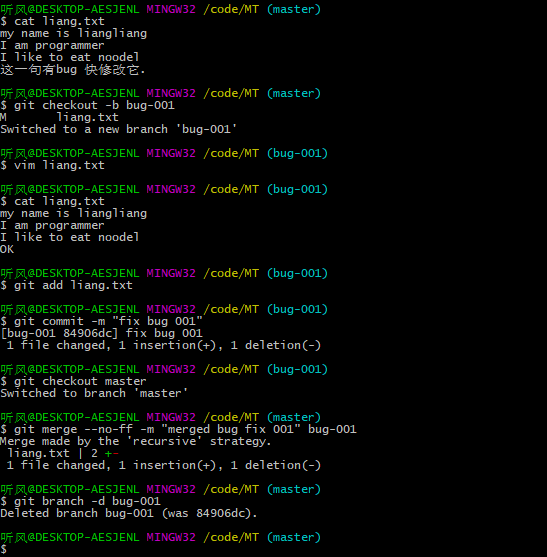
删除. 举个例子现在你在你自己的分支里面正在工作，然后你的老板突然打电话过来说master分支上面你负责的哪一个板块出现了bug，快赶紧过去抢

修，这个时候你就得马上放下手头的工作去修bug，但是我的工作这才写了一半，这里不用担心. 我们GIT有一个stash功能，可以把工作现场"储存"起来,等以后恢复现场之后继续工作:  $ git stash

好了，我们现在过来修改bug了，其实我们在上面那个图当中其实已经可以看到修改bug的过程了，具体就分为下面这几步:

假定我们在master分支上面修复，就从master创建临时分支bug-001. 然后修改bug，在文件中操作知道bug消除. 然后在该分支上面git add和git

commit，然后，切换回master分支，然后把bug-001分支合并到master分支上，这个时候master上面的bug解除了. 最后删除掉bug-001分支.



当然工程当中都是类似的修改bug方式，只是我的这个是在本地库修改. 然后当我们修改完bug.我们重新开始回到自己的分支上面干活，刚刚我们使用$git stash 保存了工作现场. 我们可以使用$git stash list查看stash存储的工作现场.  接下来我们使用指令进行工作现场恢复:

$git stash apply //恢复现场，但是stash存储工作现场内容不删除

$git stash deop  //删除git当中，stash指令存储的工作现场.

还有一种方式就是使用: $ git stash pop,//恢复的同时把stash内容也删了.

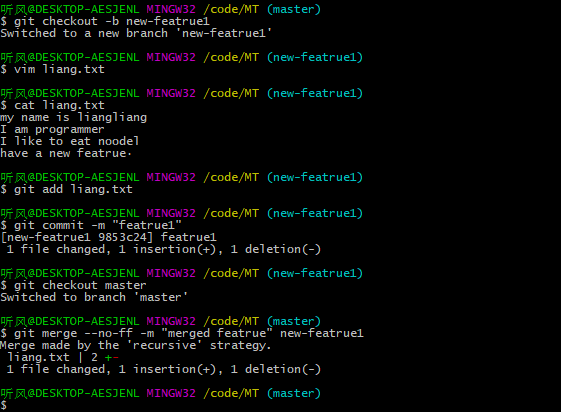
修复bug时，我们会通过创建新的bug分支进行修复，然后合并，最后删除. 但手头工作没有完成时，先把工作现场

git stash一下，然后修复bug，修复后再git stash pop，回到工作现场.

# Feature分支

软件开发当中，总会有新的功能需要添加进来这个时候. 添加一个新功能时，你肯定不希望因为一些实验性质代码，把主分支搞乱了.所以，每添加一个新功能，最好新建一个feature分支，在上面开发，完成后，合并，最后删除该feature分支.

从我上面那个图可以看出,创建一个feature分支的时候，其实跟bug分支没有什么区别,所以我们代码演示这个过程:



其实我们看到feature分支实现我们很容易理解的,我们只需要知道它的应用场景. 以及它的用法. 不过当你开发出新功能之后，你的组织上面突然有发

现了新功能的致命缺点，然后就让不要去往主分支上面合并了. 那你就只能删除这个分支了. 不过在GIT上面我们新创建的分支没有合并之前是没有办

法删除的. 所以我们需要一条大招指令: $git branch -D xxx   //强行删除

所以要丢弃一个没有被合并过的分支，可以通过git branch -D xxx 进行强行删除...