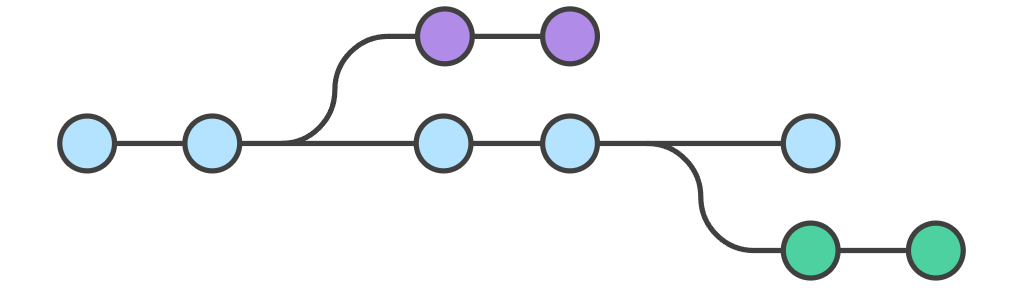
# **Git进阶使用**

在之前的操作中，所有的操作都是基于一条主线完成的。就好比，咱们学习的时候，记学习笔记，今天学点，那么就写一点，明天学点，再写一点，最后，完全学完了，这个笔记也就记全了。但实际上，有些文件可能再不同的场合需要同时使用不同的内容，而且还不能冲突，比如项目的配置文件，我需要本地进行测试，同时还要部署到服务器上进行测试。本地和服务器上的环境是不一样的，所以同一个配置文件就需要根据环境的不同，进行不同的修改。本地环境没问题了，修改配置文件，提交到服务器上进行测试，如果测试有问题，再修改为本地环境，重新测试，没问题了，再修改为服务器配置，然后提交到服务器上进行测试。依次类推，形成迭代式开发测试。

从上面的描述上看，就会显得非常繁琐，而且本质上并没有太重要的内容，仅仅是因为环境上的变化，就需要重新修改，所以如果将本地测试环境和服务器测试环境区分开，分别进行文件版本维护，是不是就会显得更合理一些。这个操作，在Git软件中，我们称之为branch，分支。

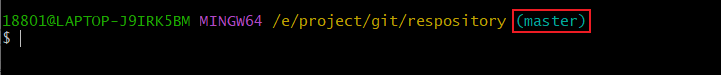
这里的分支感觉上就是树上的分叉一样，会按照不同的路线生长下去。有可能以后不再相交，当然，也可能以后会不断地纠缠下去，都是有可能的。



## 3.1 Git分支

### 3.1.1 主干分支

默认情况下，Git软件就存在分支的概念，而且就是一个分支，称之为master分支，也称之为主干分支。



这就意味着，所有文件的版本管理操作都是在master这一个分支路线上进行完成的。

不过奇怪的是，为什么之前的操作没有体现这个概念呢，那是因为，默认的所有操作本身就都是基于master分支完成的。而master主干分支在创建版本库时，也就是git init时默认就会创建。

### 3.1.2 其他分支

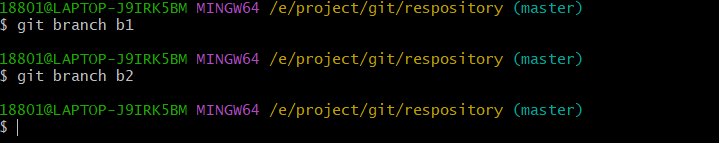
就像之前说的，如果仅仅是一个分支，在某些情况并不能满足实际的需求，那么就需要创建多个不同的分支。

#### 3.1.2.1 创建分支

**# git branch 分支名称**

git branch b1

git branch b2



现在我们创建了2个分支，不过这两个分支都是基于master主干分支为基础的。

#### 3.1.2.2 查看分支

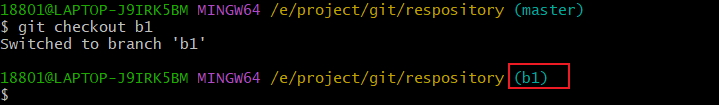


#### 3.1.2.3 切换分支

我们将工作线路切换到b1

**# git checkout 分支名称**

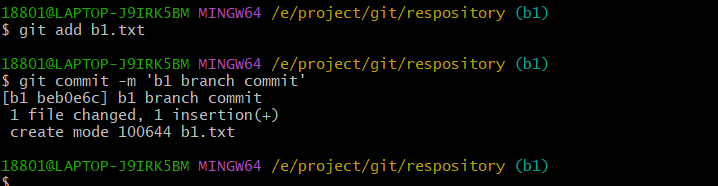
git checkout b1



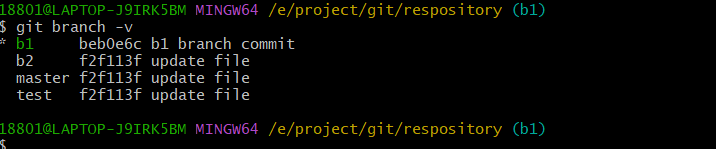
此时我们添加新的文件b1.txt



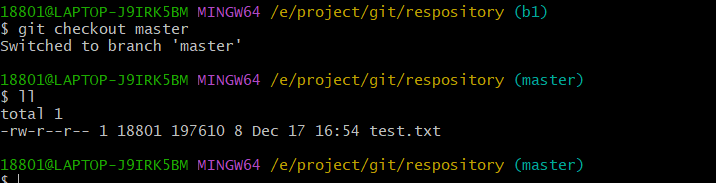
然后提交到版本库

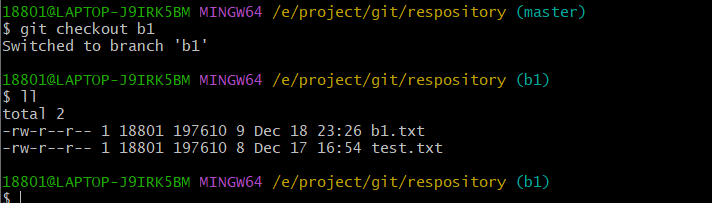


此时，查看分支信息，会发现不同分支的版本进度信息发生了改变



如果此时切换回到主干分支的话，那么b1.txt文件就不存在了，因为对应版本信息不一样。



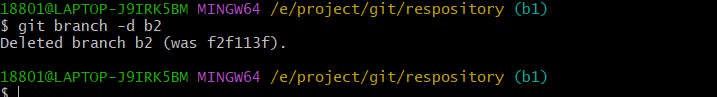


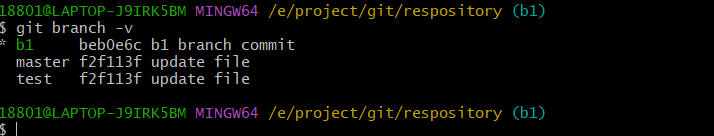
#### 3.1.2.4 删除分支

如果觉得某一个分支建立的不太理想或已经没有必要在使用了，那么是可以将这个分支删除的。

**# git branch -d 分支名称**

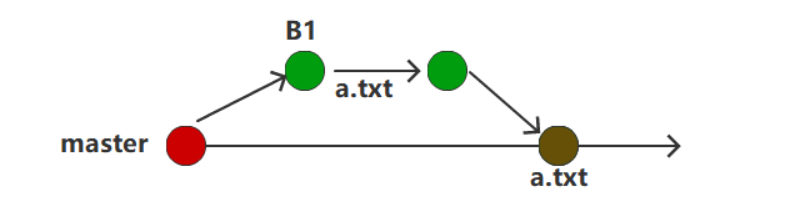
Git branch -d b2





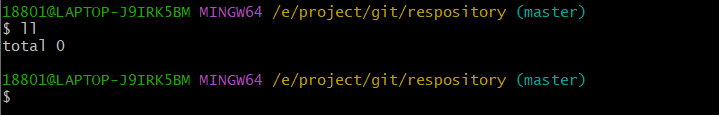
## 3.2 Git合并

无论我们创建多少个分支，都是因为我们需要在不同的工作环境中进行工作，但是，最后都应该将所有的分支合在一起。形成一个整体。作为项目的最终结果。

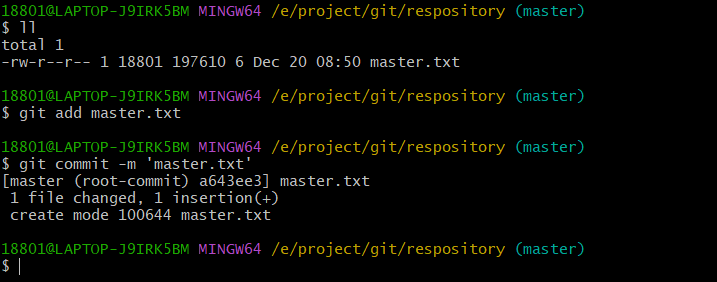


### 3.2.1 主干分支

首先我们先将主干分支的所有文件清空掉



在当前主干分支中创建一份文件master.txt，并提交

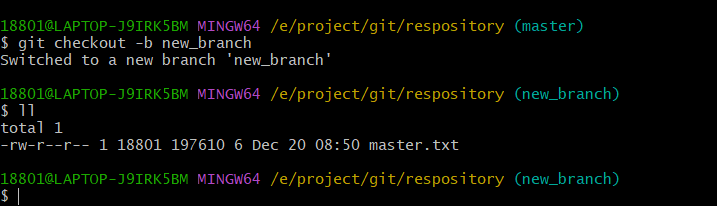


### 3.2.2 其他分支

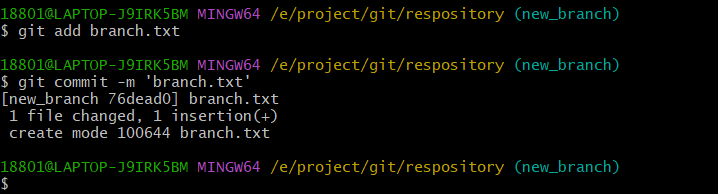
基于主干分支的内容，我们创建其他分支，并直接切换到新的分支

**# git checkout -b 分支名称**

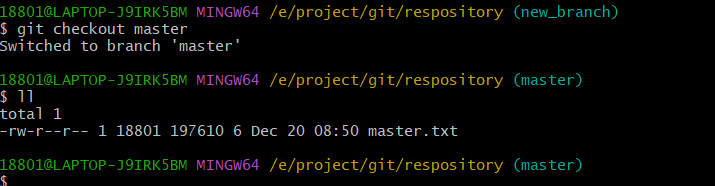
git checkout -b new\_branch



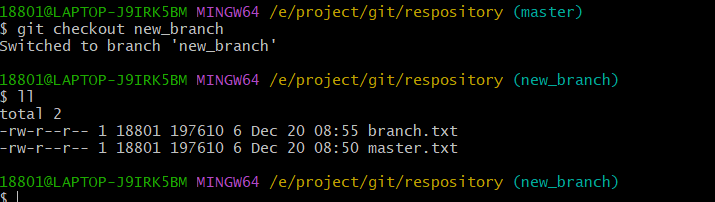
在新的分支中添加新文件branch.txt



此时切换回主干分支，只有master.txt文件。

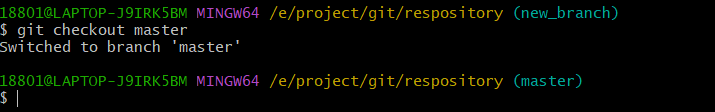


再切换回new\_branch分支，branch文件就又回来了。



### 3.2.3 合并分支

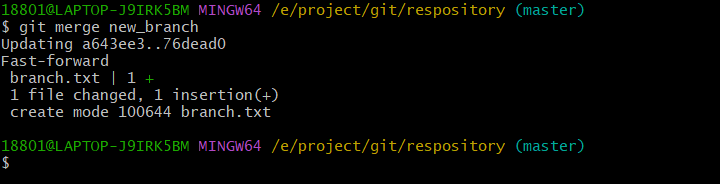
这里我们将new\_branch分支的文件内容合并到主干分支中。首先先切换回主干分支



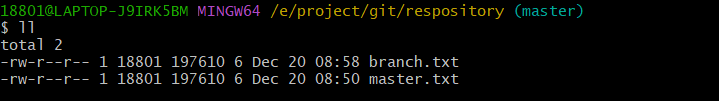
然后执行分支合并指令

**# git merge 分支名称**

git merge new\_branch

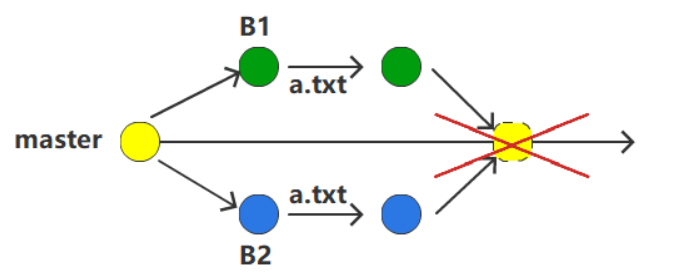


此时再次查看文件，就会发现branch.txt文件已经可以看到了。



## 3.3 Git冲突

在多分支并行处理时，每一个分支可能是基于不同版本的主干分支创建的。如果每隔分支都独立运行而不进行合并，就没有问题，但是如果在后续操作过程中进行合并的话，就有可能产生冲突。比如B1, B2的两个分支都是基于master分支创建出来的。B1分支如果和B2分支修改了同一份文件的话，那么在合并时，以哪一个文件为准呢，这就是所谓的冲突。

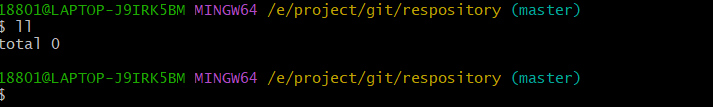


接下来，咱们就演示一下。

### 3.3.1 主干分支

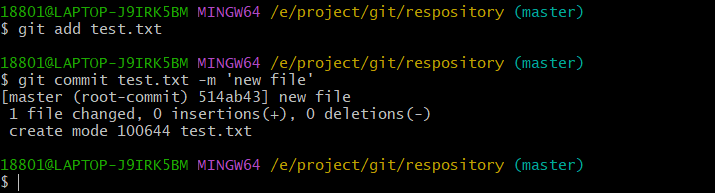


首先我们先将主干分支的所有文件清空掉



主干分支添加文件test.txt，文件内容为空

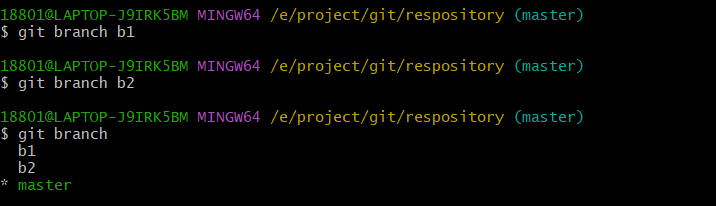




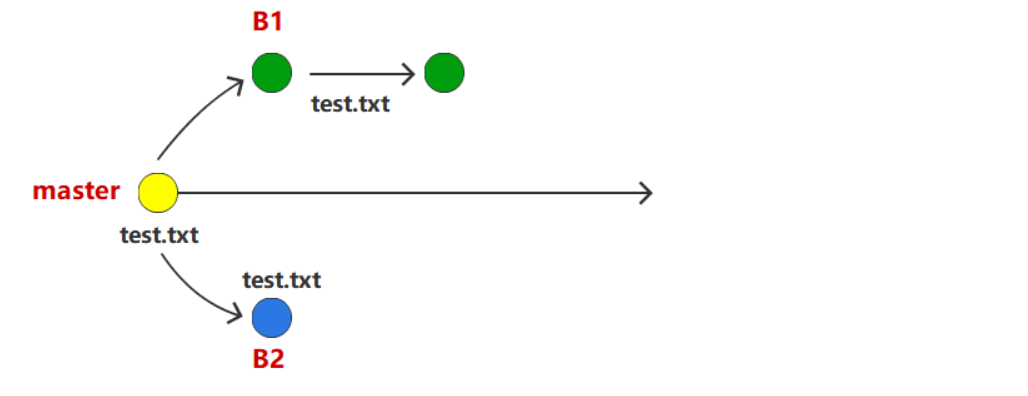
### 3.3.2 其他分支

基于主干分支，创建两个分支B1, B2

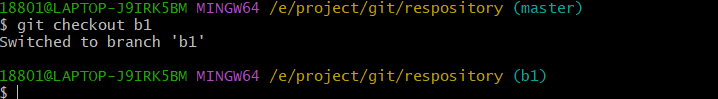




### 3.3.3 切换分支-B1

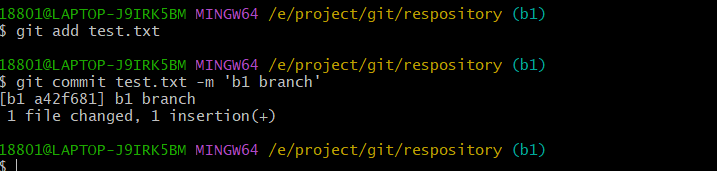


切换到B1分支，修改文件内容

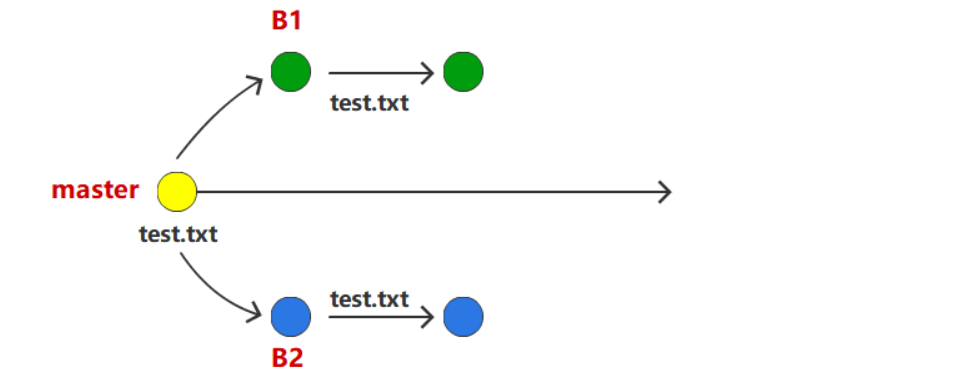




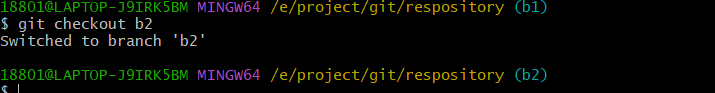
提交修改后的文件



### 3.3.4 切换分支-B2

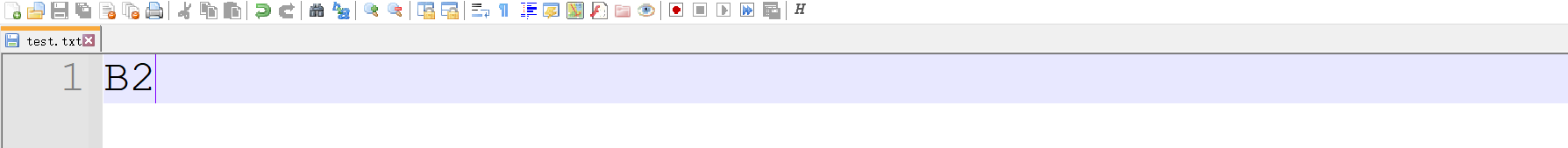


切换到B2分支，查看文件内容

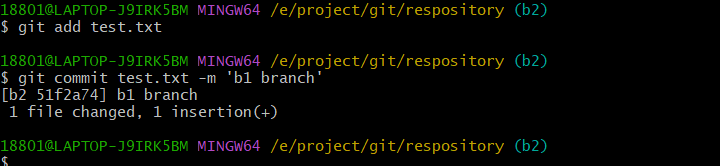




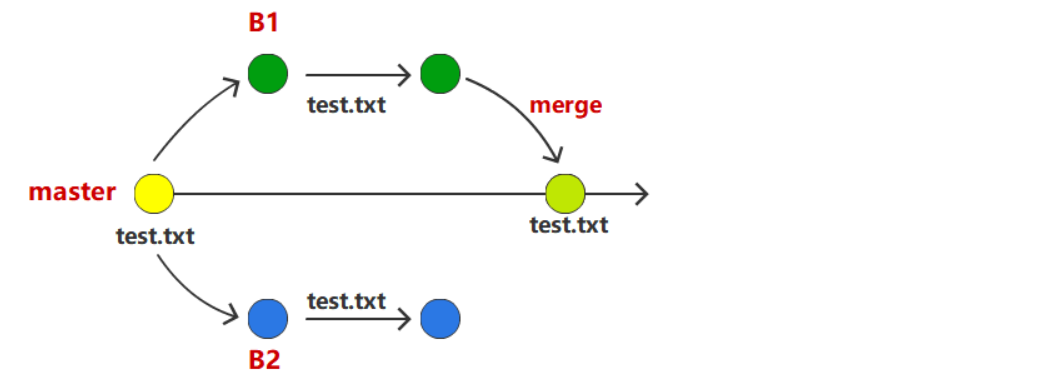
修改文件内容：



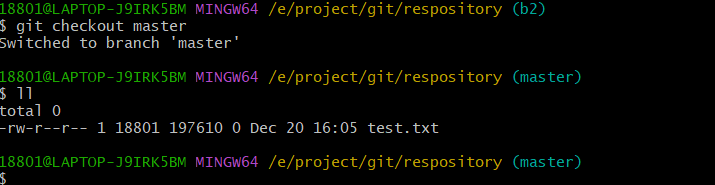
提交文件



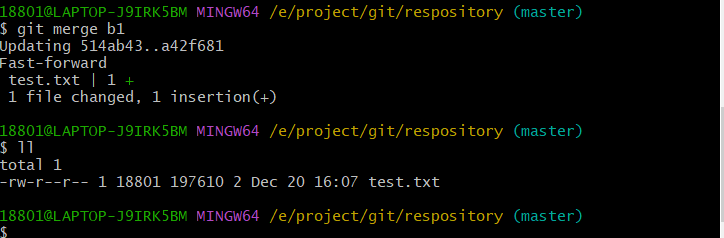
### 3.3.5 合并分支-B1



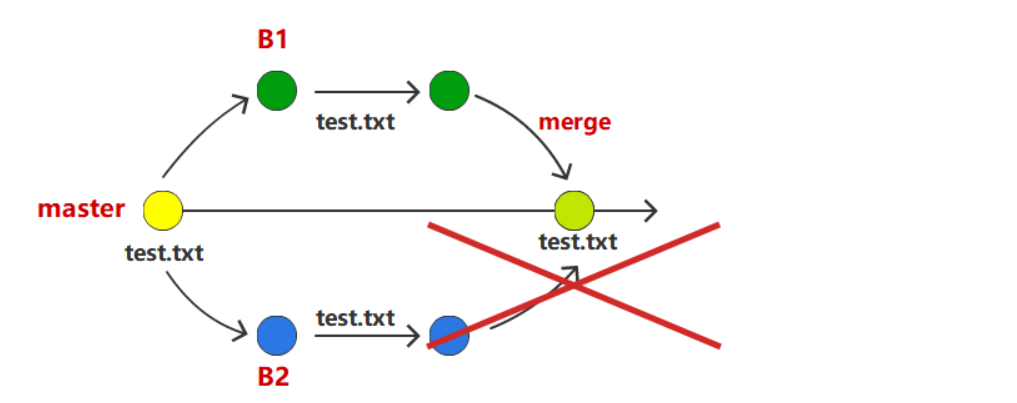
切换到master主干分支，此时test.txt文件内容为空



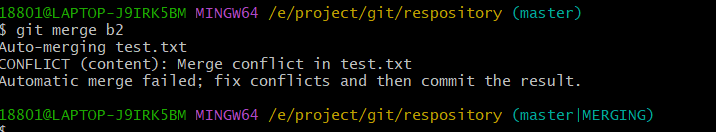
将B1分支合并到主干分支中



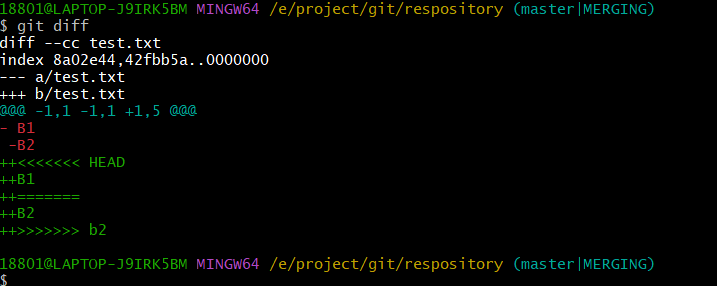
### 3.3.6 合并分支-B2



因为B2分支也对文件进行了修改，所以如果此时合并B2分支,就会提示冲突

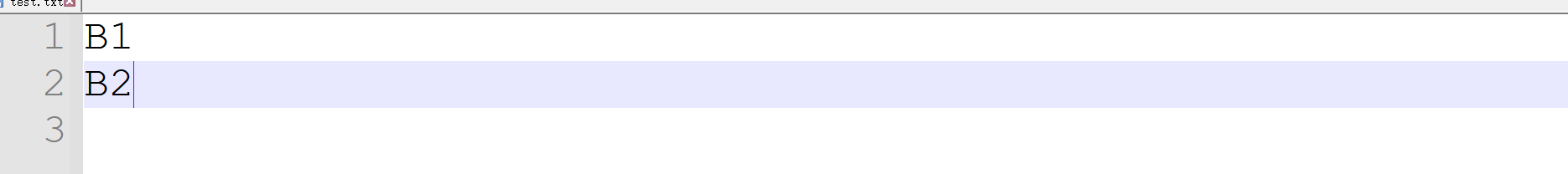


查看文件内容差异





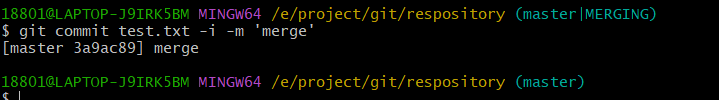
这里的冲突，软件是无法判断该如何出来处理的，所以需要人工进行判断，将冲突的文件内容进行修正。



重新提交到master主干分支中。



**# git commit 文件名称 -i -m 注释**



再查看一下Git软件的操作日志

**# git log --graph**

