**Git入门学习笔记**

Git一种分布式版本控制系统

可以用touch 1.py来构建新文件



git commit -m "解释这个版本做了什么"

git add <file>

将文件从工作目录删除（git rm –cashed app.py）

**Untracked（未跟踪）**

文件在工作区中存在，但还没有被 Git 管理。

还未执行 git add。

**Unmodified（未修改）**

文件*已经被 Git 追踪*，并且内容与最近一次提交一致。

没有改动。

**Modified（已修改）**

文件内容被编辑过，但还没有被暂存（未执行 git add）。

Git 发现它与上次提交版本不同。

**Staged（已暂存）**

文件已经执行了 git add，进入暂存区（stage/index）。

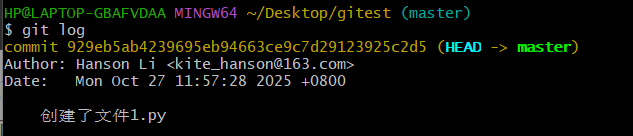
等待执行 git commit 提交到版本库。（*commit：提交操作*）

指令“git add .“可以将所有的未追踪文件还有修改都转移到暂存区；”git add <文件名>“可以追踪/将文件的修改到暂存区；”git add -u“可以将所有的修改（不包括未追踪的文件）转移到暂存区

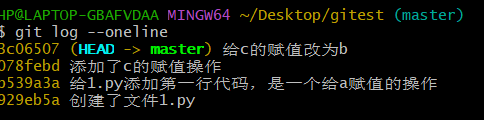
将所有的修改（不包括新文件）add到暂存区并且直接commit，可以使用“git commit -am”注释文字””

指令“git status“查看当前状态（工作区的文件情况）；使用“git status -s”可以给出简短的

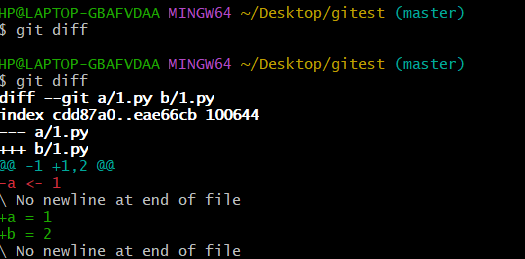
指令“git log“可以看更改日志（注解：这里的”master“是指主分支（有些版本是用”main“表示主分支））



指令“log --oneline“指令可以只看改动的文件以及注释

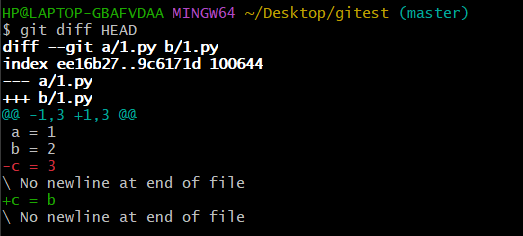


在修改了文件但是还没有add为staged（暂存）的状态时，可以使用”log diff”指令查看修改内容。



如果文件已经add了，可以用指令“git diff --cached“来查看修改内容

当然，也可以使用“git diff HEAD“命令，将当前工作区与暂存区的相对最近的一次（head）的提交的改动都显示出来



**⭐将文件回到之前的状态**

checkout方法（针对单个文件）:

指令“git checkout <对应状态的Hash值> -- <对应状态的文件名>“



这样做的特色是只是修改了现在·HEAD指针的指向，但是并没有删除之前的日志（由此生成了branch）

reset方法

*001 修改已经commit的版本内容*

情景：一个文件已经commit、生成版本了，现在想在不改变版本名的情况下修改这个版本的内容。

操作：将原本需要修缮的内容补add到暂存区，然后使用“git commit --amend “指令，这样的操作可以直接覆盖上一个保存的版本，而不是创建一个新的版本。



会出现一个这个界面，可以编辑提交信息（写注解）。如果只是覆盖版本不写内容，按“Esc“后输入”:wq“然后回车即可；如果要修改注解信息，按”i“然后开始写，写完，按“Esc“后输入”:wq“然后回车即可；如果要取消这次amend，按“Esc“后输入”:q!“后回车即可

“git commit --amend --no-edit“指令可以直接跳过这个编辑界面

*002 reset到add之前*

情景：如果一个文件，它被add到了暂存区，但是现在想将这个操作撤回，接着对文件做修改再add（目的是不产生两条日志），可以使用指令“git reset <文件名>“



*003 reset到commit之前*

情景：现在想将现在所处的版本改到特定版本。

方法：

使用“git reset --hard HEAD^“可以跳到前一个版本

使用“git reset --hard <版本的哈希>“可以跳到特定版本

这样处理，原本修改过的版本将不再显示，这是因为在git中没改时候的版本没有指针去指了，但是其实之前的版本还在。

这个时候可以用指令“git reflog“看对于版本的所有改动（包括每个版本的对应哈希值），然后使用指令”git reset --hard <之前版本的哈希>“可以回到最初没改的情形。

**⭐分支管理**

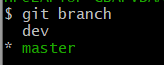
想法：一个项目经常会有不同的版本同时开发（如正式版和测试版）

001 看现在的分支结构（拓扑结构）

“git log --oneline --graph”方法可以看现在的结构

002 创建分支

“git branch <分支名>”可以创建分支；使用“git branch”可以看所有的分支名



在这个图中，※代表HEAD现在还在master分支上；可以使用”git checkout <分支名>“来实现HEAD的切换指向。

当然，可以使用“git checkout -b <分支名>“完成”分支创建和切换分支“两个动作

003 在其他分支上修改、更新版本

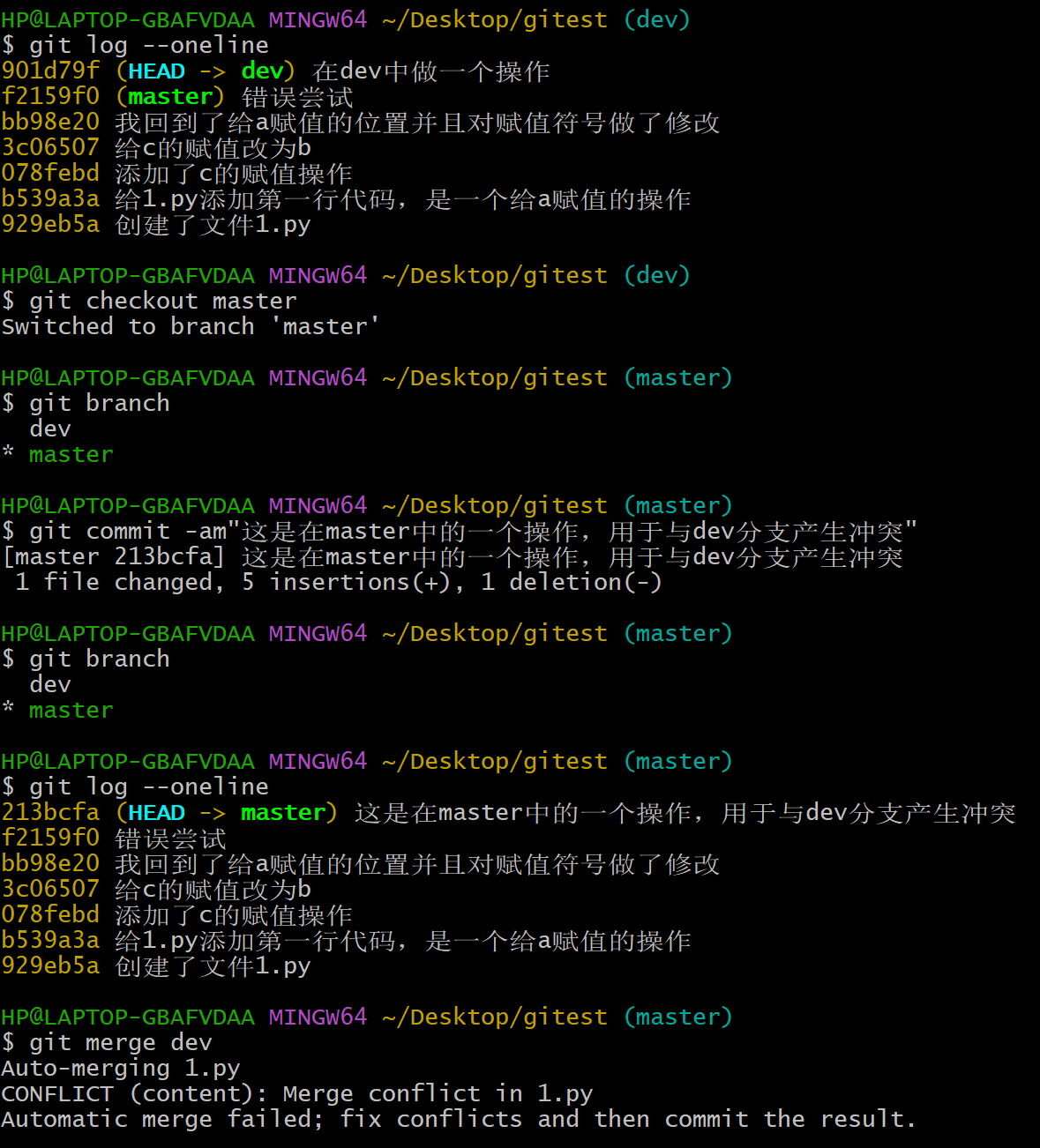
当HEAD移动到了其他版本，那么之后的版本更新都将记录到对应的分支上去，而不影响原先的分支。

004 将其他分支上的修改merge（合并）到特定分支

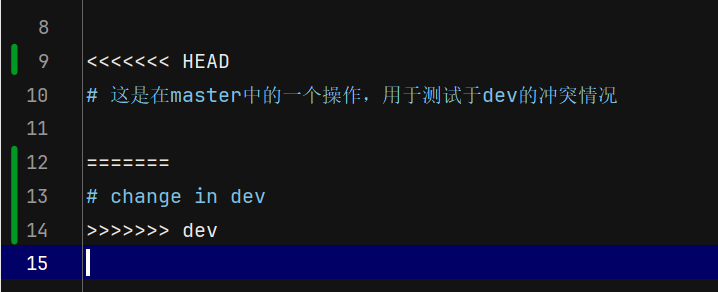
首先先找到特定位置“git checkout <分支名>”，然后直接将某分支的版本合并过来“git merge dev”

当分支A完全落后于分支B，且分支A近期没有过更新，那么这里的merge会直接采用“Fast-Forward（快进）”的方式，即直接将版本拼接到分支A的版本后面，从拓朴结构上看不出有merge的痕迹（即merge操作在这种情况下没有commit信息）。这个时候可以使用“git merge --no-ff <分支名>”的方式，使用非“Fast-Forward”的方式做merge，也可以用“git merge --no-ff -m”commit信息” <分支名>”来给这次merge添加commit信息。

但是如果在分支A中有做不同于分支B的改动，那么直接merge会出现冲突（报：CONFLICT）并且告知冲突的文件名：

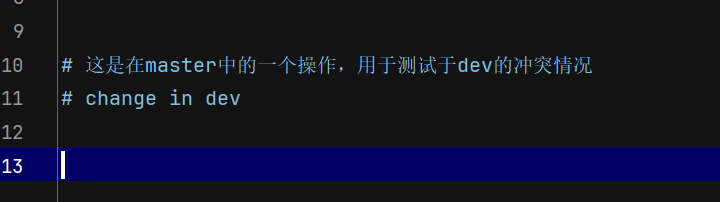


这个时候，在分支A的对应文件（这个地方显示的是1.py）会出现如下的一段内容：

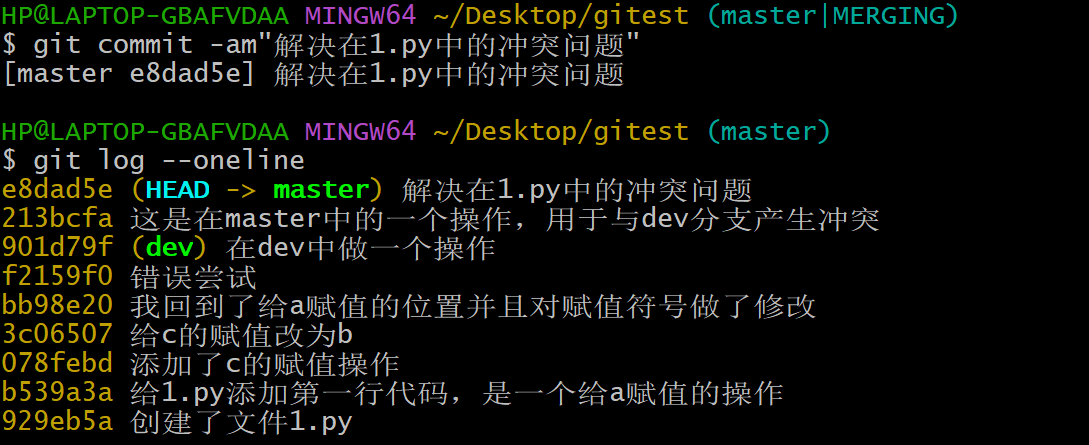


“<<<<<<< HEAD”下面的内容是在分支A中出现conflict位置的内容，而“=======”下面到“>>>>>>> dev”上面的内容则是在分支B中对应文件出现conflict的内容。

现在需要的操作是：直接在文件这个地方操作，保留希望留存的内容（比如下面）



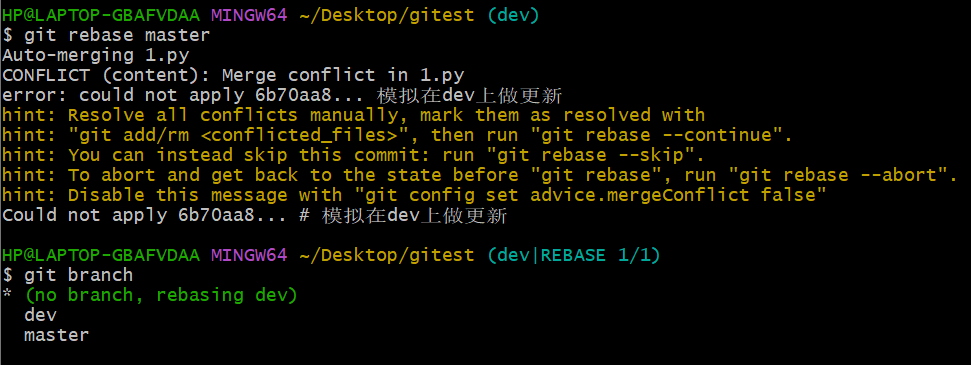
然后再add并commit现在的文件即可：



005 rebase修改

情景：假定现在在某个节点上有了分支，其中主分支master有做一些更改，而子分支dev正在基于之前的主分支未修改的版本进行开发；现在我们希望在分支dev上的版本能够在更新后的master分支内容上进行开发（而不是就的master分支上的内容）。

用法：假定在dev分支上用新的master内容，指令“git rebase <分支名>”（这里是master）即可。如果没有冲突情况则直接成功；如果有冲突：



会进入到一个类似于之前的冲突回复，回复内容包括冲突文件，并且会给标好冲突的地方，需要手动调整；同时这个时候，HEAD指针会在一个称为rebase的区域



在这之后，add更改的文件，并且使用命令“git rebase --continue”，即可完成rebase的操作。

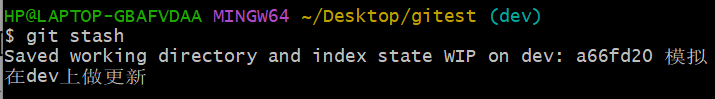
006 stash临时修改

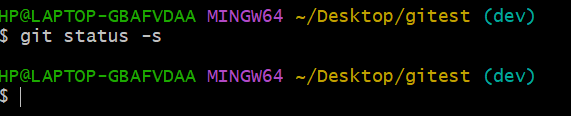
场景：假设现在正在dev分支中开发，并且内容有add到暂存区，但由于这个版本还没搞完，所以没有commit；但这个时候出现了一个任务，要求摆脱掉现在的开发立刻去做——这个时候需要我们将还没有commit的开发版本临时保存在另一个地方，然后着手于其他的优先级更高的工作，等工作做完之后，再从临时地点将还没开发完的版本取回来继续开发。

用法：

*储存操作：*

“git stash”就直接保存当前的暂存区中的内容了；也可以添加补充描述“git stash save”描述文字””；使用“git stash list”可以看当前所有临时储存的内容





*拿出操作：*

“git stash apply”是应用上一次的储存内容，但是不将其从储存list中删除；“git stash pop”是应用并将其从list中删除

其实这里对stash的操作就是对一个列表的操作，可以使用“git stash apply @stash{0}”这种方法访问元素然后拿出来



**⭐Github在线管理**