关于Git的“技术”分享

1. 创建空目录:(1)mkdir hyx

(2)cd hyx

(3)pwd(用于显示当前目录)

2. git init把目录变成可以管理的仓库，如果没有看到.git的目录，就是默认隐藏，用ls-ah命令就可以看见。

3. 把文件放到git仓库：（1）git add(添加)

(2)git commit(提交，-m是本次的提交说明)

4. git status用来查看本地库的当前状态。

5. git diff用来查看文件的不同之处,修改内容。

6. 提交修改和提交新文件：(1)git add 文件名

（2）git commit(提交前后可先查看一下当前状态)

git log命令显示最近到最远的提交，输出信息太多，可改为git log --pretty=oneline。

在Git中，用HEAD表示当前版本，也就是最新的提交，上一个版本HEAD^，上上个版本HEAD^^，版本过多如100个可以用HEAD~100。

7. 退回上一个版本用git reset --hard版本，版本处可以用commit id。

8. Git提供了一个命令git reflog用来记录每一次命令，查看历史命令，以便要回到未来的哪个历史版本。

9. 工作区：在电脑里能看到的目录。

版本库：工作区的隐藏目录.git。

暂存区：stage

（

添加文件具体执行步骤：（1）用git add把文件添加进去，实际上就是把文件添加修改到暂存区。

（2）用git commit提交更改，实际上就是把暂存区的所有内容提交到当前分支。

（创建版本库时，Git自动创建了唯一一个master分支，所以git commit是往master分支

上提交更改。）

举例：使用两次git add把read.txt和LICENSE添加后暂存区状态：



提交后状态：



10. 用git diff HEAD – 文件名可以查看工作区和版本库里面最新版本的区别。

11. git checkout -- file可以让文件回到最近一次git add或git commit的状态。

（一定要注意-和空格问题）

场景1：当改乱了工作区文件的内容，直接丢弃，用git checkout – file命令。

场景2：不但改乱工作区文件的内容，还添加到了暂存区时，想丢弃修改，首先用git reset HEAD<file>,回到场景1，然后同上。

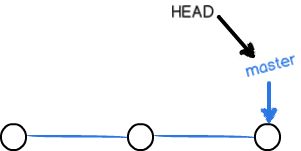
场景3：改乱并已经提交到版本库时并没有推送到远程库时，撤销提交，可以采用版本回退。

1. 用rm <file>命令删除文件。确实要删除，就用git rm <file>,并且git commit,文件就会从版本库中被删除。如果误删，用git checkout -- <file>恢复。

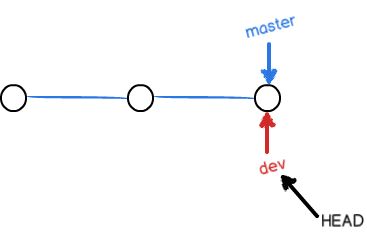
（从来没有被添加到版本库就被删除的文件不可恢复，提交的也只能恢复到最新版本）

1. 用cat <file>读取文件内容。
2. 目录里面有两个文件，是SSH Key的秘钥对，其中id\_rsa是私钥，id\_rsa.pub是公钥。
3. 在本地仓库下运行命令git remote add origin git@github.com:heyingxu/hyx.git可以关联远程仓库，用git push -u origin master可以把本地库上的所有内容推送到远程库上。用git push命令实际上是把当前分支master推送到远程。由于远程库是空的，第一次推送加上了-u参数，Git不但会把本地的master分支内容推送到远程新的master分支，还会将二者关联起来。以后的推送或拉取可以简化命令。推送成功后，可在GitHub页面中看到和版本库一模一样的内容，从现在起只要在本地提交后，用git push origin master推送到GitHub。
4. SSH警告：第一次使用Git的push命令连接GitHub会被警告，输入yes回车即可。
5. 从远程库克隆：在GitHub上创建名为gitskills的新仓库（勾选Intialize this repository with a README会看到README.md文件，然后用git clone命令克隆一个本地库:git clone git@github.com:heyingxu/gitskills.git然后cd gitskills Is就可以看到README.md文件。
6. 创建与合并分支：

* 开始时master指向最新提交，HEAD指向master

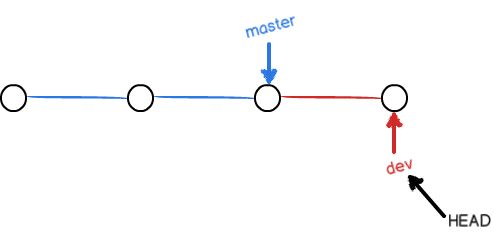


* 当我们创建分支时（如dev），Git新建了一个指针叫dev,指向master相同的提交，再把HEAD指向dev

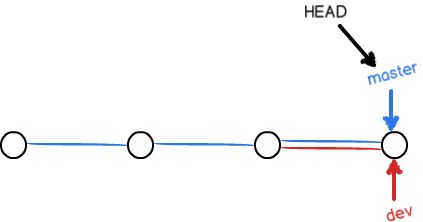


创建dev分支并切换：git checkout -b dev相当于给git branch dev和git checkout dev。

* 从此，对工作区的修改和提交针对dev，新一次提交后，dev指针往前移动一步，而master指针不变



* dev上的工作完成后，直接把master指向dev的当前提交，就可以完成合并



* 合并可后删除dev指针。

1. 创建分支：git branch <name>

查看分支：git branch(会列出所有分支，当前分支前面会标有\*号)

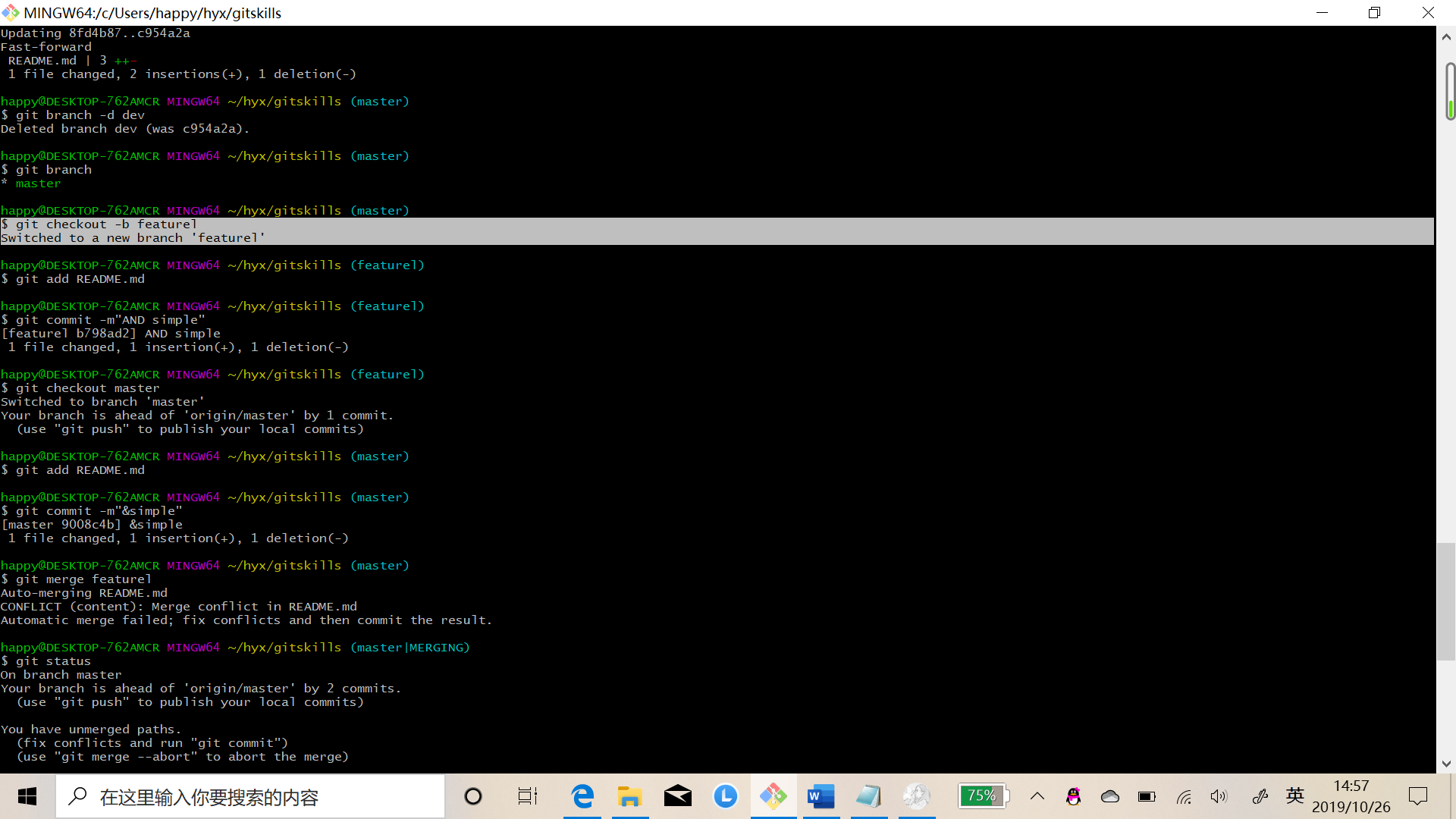
切换分支：git checkout <name>或者git switch<name>

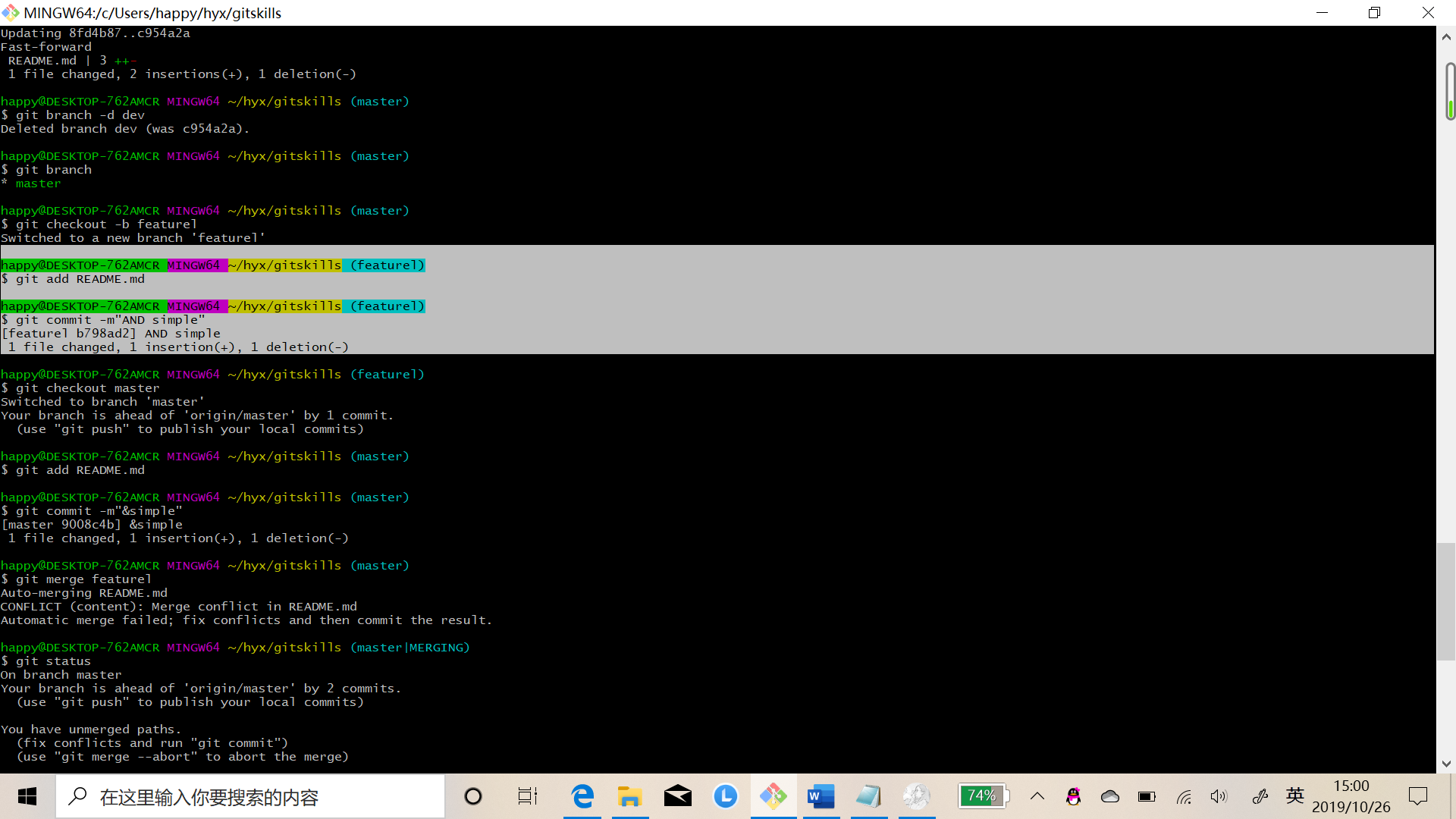
创建加切换分支：git checkout -b<name>或者git switch -c<name>

合并某分支到当前分支：git merge <name>

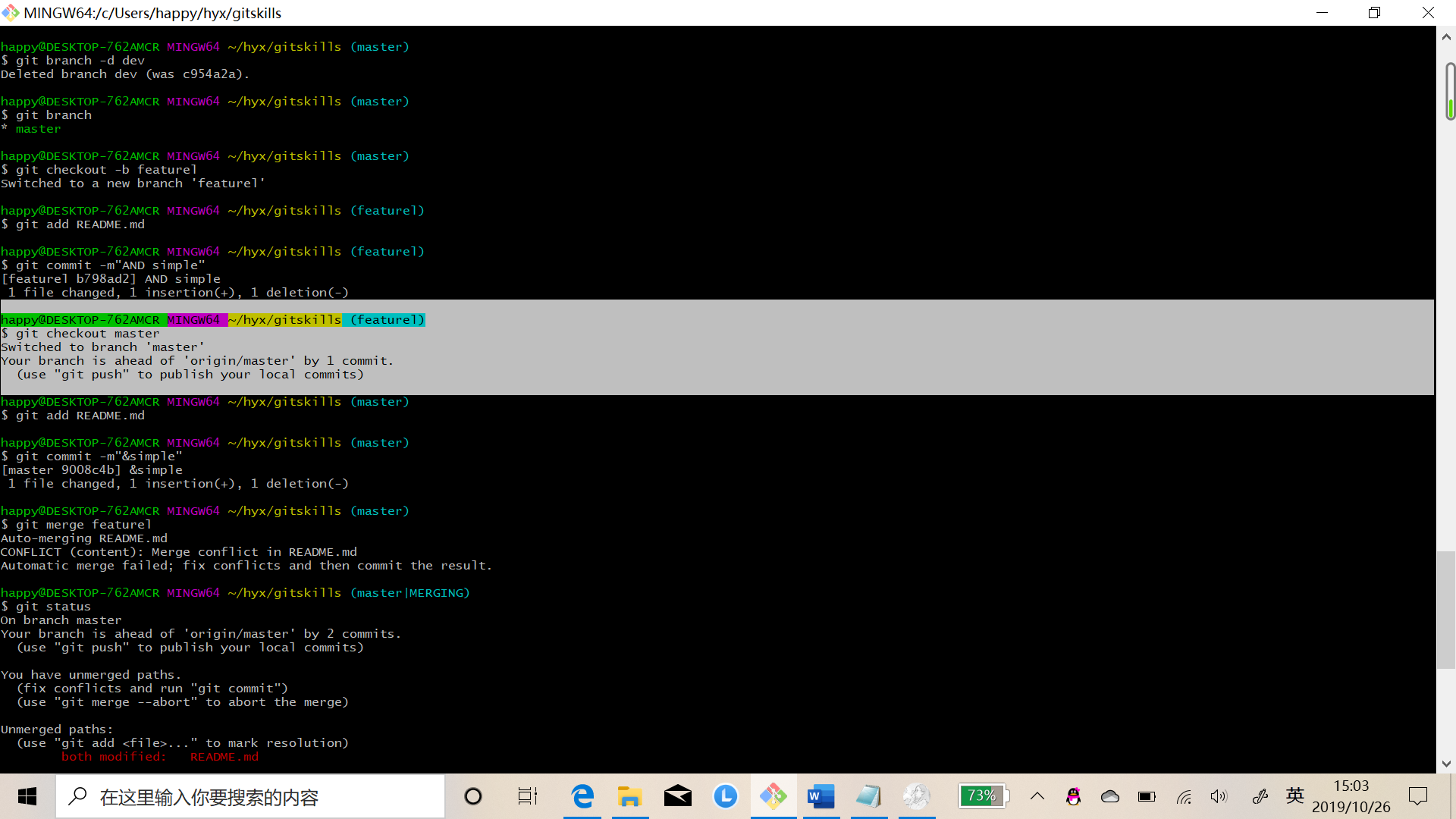
删除分支：git branch -d<name>

1. 解决冲突：备新的feature1分支，继续新分支开发：

  
修改README.md最后一行，改为：Creating a new branch is quick AND simple.  
在feature1分支上提交：

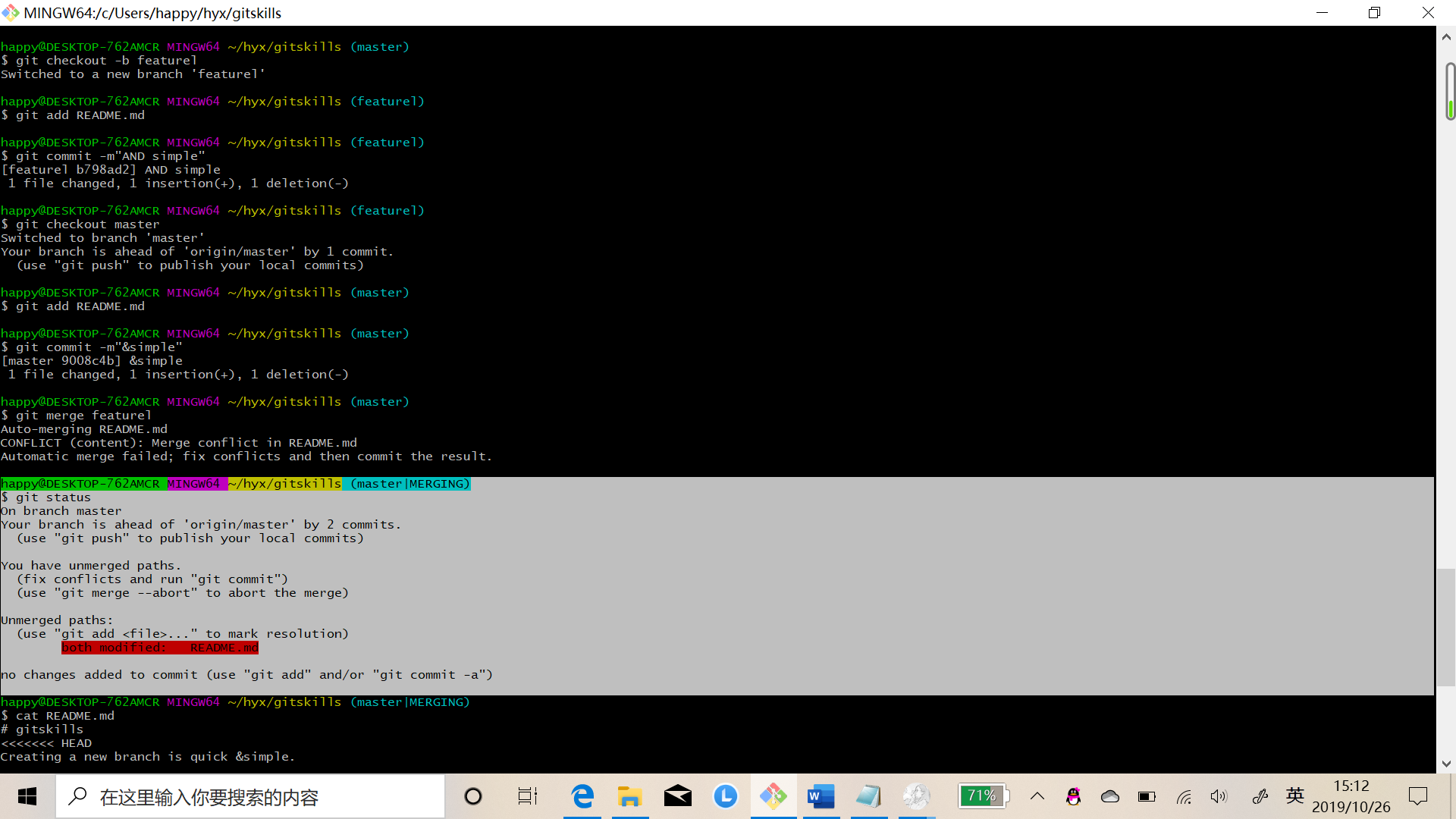


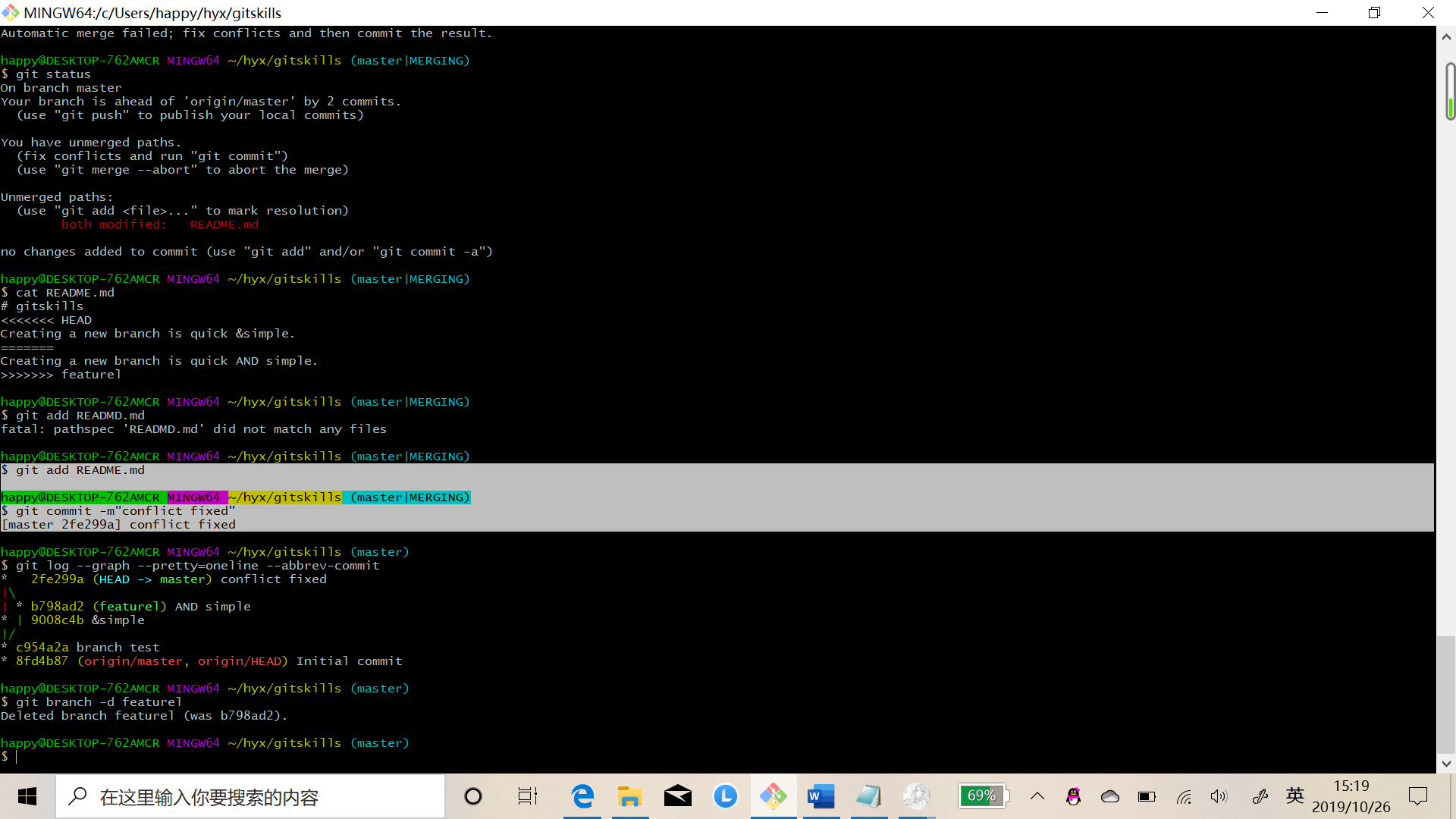
切换到master分支：



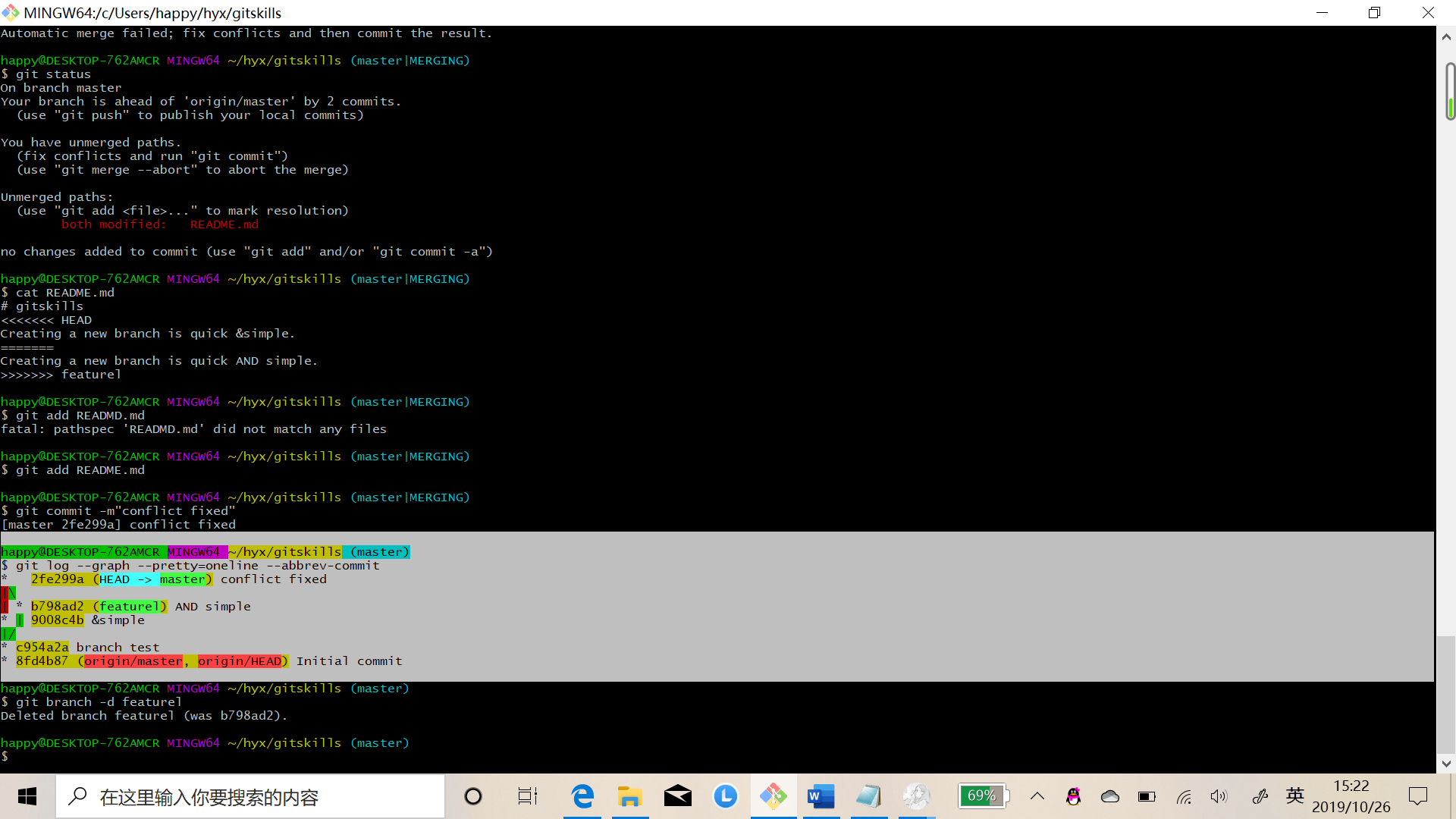
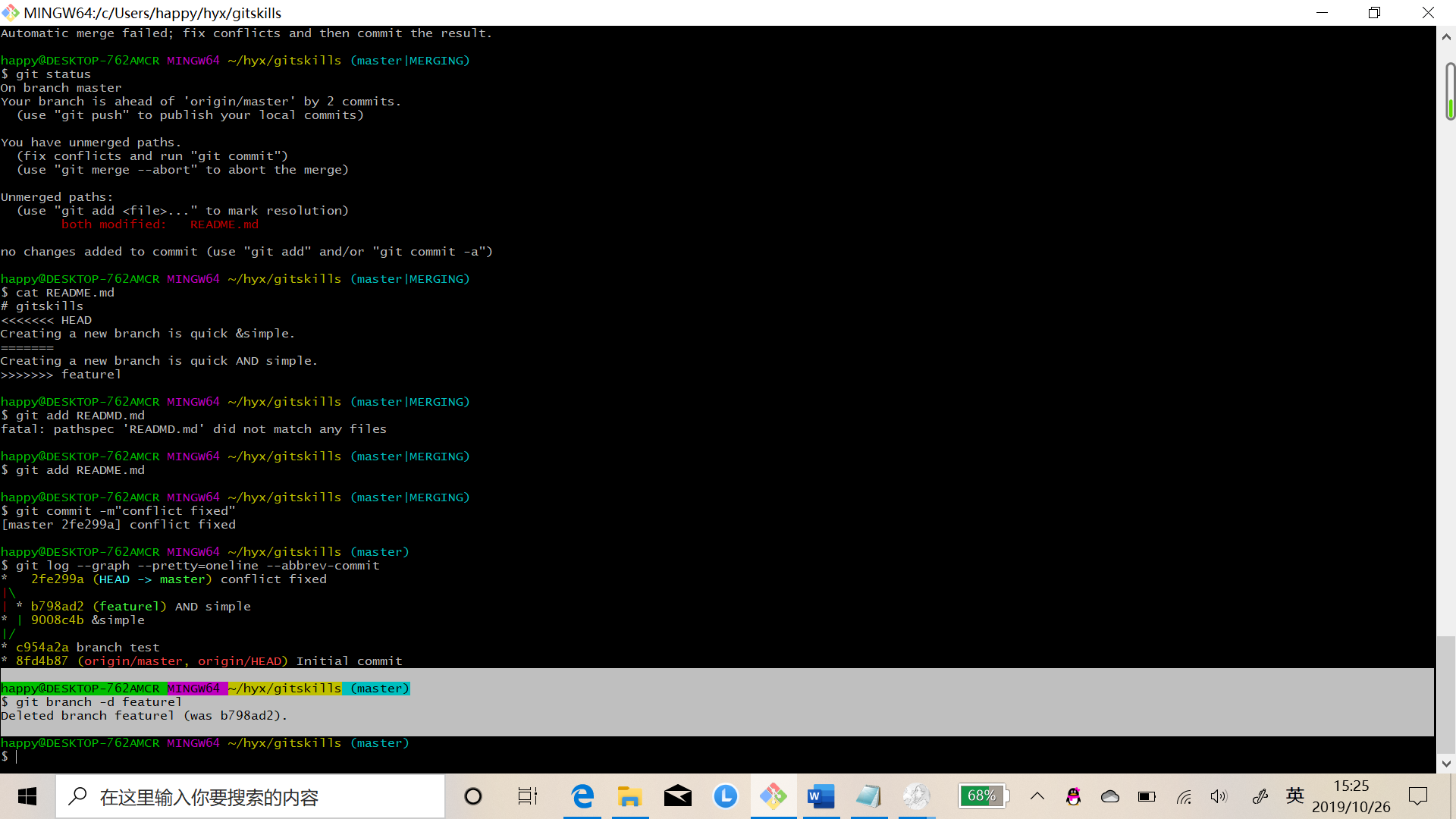
Git还会自动提示我们当前master分支比远程的master分支要超前1个提交。  
在master分支上把readme.txt文件的最后一行改为：Creating a new branch is quick & simple后提交。现在，master分支和feature1分支各自都分别有新的提交，变成了这样：

  
这种情况下，Git无法执行“快速合并”，只能试图把各自的修改合并起来，但这种合并就会冲突，必须手动解决冲突后再提交。git status也可以告诉我们冲突的文件：

  
查看README.md可以发现Git用<<<<<<<，=======，>>>>>>>标记出不同分支的内容，我们修改如下后保存：Creating a new branch is quick and simple.  
再提交：

  
现在，master分支和feature1分支变成了下图所示：

  
用带参数的git log也可以看到分支的合并情况：

  
最后，删除feature1分支：  
  
当Git无法自动合并分支时，就必须首先解决冲突。解决冲突后，再提交，合并完成。解决冲突就是把Git合并失败的文件手动编辑为我们希望的内容，再提交。  
用git log --graph命令可以看到分支合并图。(git log –-graph –pretty=oneline –abovrev-commit)。

1. 合并分支时的 –-no-ff参数，表示禁用Fast forward。不使用Fast forward模式，merge后：



1. 团队合作的分支可以形象地表示为下图：



1. 合并分支时，加上 --no-ff参数就可以使用普通模式合并，合并后的历史有分支，能看得出来曾经做过合并，而Fast forward合并就看不出来曾经做过合并。

git stash可以把当前工作现场储藏起来，等以后恢复现场继续工作。可以用git stash list命令查看工作现场。

1. 工作现场还在，Git把stash存在某个地方，恢复方法有两种：

* 用git stash apply恢复后，stash内容并不删除，需要用git stash drop删除。
* 用git stash pop,恢复的同时把stash的内容删除。

1. 可以用git stash apply stash@{}恢复指定的stash。
2. Git专门提供一个cherry-pick命令复制一个特定的提交到当前分支。
3. 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。当手头工作没有完成时，先把工作现场git stash一下，然后去修复bug，修复后，再git stash pop，回到工作现场；在master分支上修复的bug，想要合并到当前dev分支，可以用git cherry-pick <commit>命令，把bug提交的修改“复制”到当前分支，避免重复劳动。
4. 如果要丢弃一个没有被合并过的分支，可以通过git branch -D<name>强行删除。
5. 远程库的默认名称是origin，用git remote查看远程库的信息，或者用git remote -v显示更详细的信息。
6. 推送分支用git push origin 分支名。
7. master分支是主分支，因此要时刻与远程同步。

dev分支是开发分支，团队所有成员都需要在上面工作，所以也需要与远程同步；  
bug分支只用于在本地修复bug，就没必要推到远程了，除非老板要看看你每周到底修复了几个bug。  
feature分支是否推到远程，取决于你是否和你的小伙伴合作在上面开发。

1. 多人协作时，当他人从远程库clone时，默认情况，他人只能看到本地的master分支。
2. 当他人和本人的提交有冲突时，将远程分支和本地分支连接起来，再用git pull把最新的提交抓下来，然后在本地合并，解决冲突，再推送。
3. 多人协作的工作模式通常是这样：
   1. 首先，可以试图用git push origin <branch-name>推送自己的修改；
   2. 如果推送失败，则因为远程分支比你的本地更新，需要先用git pull试图合并；
   3. 如果合并有冲突，则解决冲突，并在本地提交；
   4. 没有冲突或者解决掉冲突后，再用git push origin <branch-name>推送就能成功！
   5. 如果git pull提示no tracking information，则说明本地分支和远程分支的链接关系没有创建，用命令git branch --set-upstream-to <branch-name> origin/<branch-name>
4. 在本地创建和远程分支对应的分支，使用git checkout -b branch-name origin/branch-name，本地和远程分支的名称最好一致；
5. git rebase可以把本地未push分差的提交历史整理成一条直线，使得看上去更为直观，缺点时本地的分叉提交已经被修改。
6. 在Git中打标签，首先，切换到需要打标签的分支上：

$ git branch  
\* dev

master

$ git checkout master

Switched to branch &apos;master&apos;

然后，敲命令git tag <name>就可以打一个新标签：

$ git tag v1.0

可以用命令git tag查看所有标签：

$ git tag

v1.0

1. 默认标签是打在最新提交的commit上的。有时候，如果忘了打标签，方法是找到历史提交的commit id，然后打上。
2. 标签不是按照时间顺序列出，而是按字母排序。可以用git show<tagname>查看标签信息。
3. 可以创建带有说明的标签，用-a指定标签名，-m指定说明文字。
4. 用git tag命令可以查看所有标签。
5. 命令git push origin <tagname>可以推送一个本地标签；

命令git push origin --tags可以推送全部未推送过的本地标签；

命令git tag -d <tagname>可以删除一个本地标签；

命令git push origin :refs/tags/<tagname>可以删除一个远程标签。

1. 用git config –global color.ui ture让Git适当显示不同颜色。
2. 在Git工作区的根目录下创建一个特殊的.gitgnore文件，然后把要忽略的文件名填进去，Git就会自动忽略这些文件。
3. 想添加某个文件到Git，添加不了时，用git add -f <file>强制添加。
4. 用git check-ignore -v<file>命令检查.gitignore的错误之处。
5. 配置别名：st(statua)

co(checkout)

ci(commit)

br(branch)

git lg (git config –-global alias.ig”log –-color –-graph –-

1. 配置Git的时候，加上--global是针对当前用户起作用的，如果不加，那只针对当前的仓库起作用。
2. 每个仓库的Git配置文件都放在.git/config文件中。