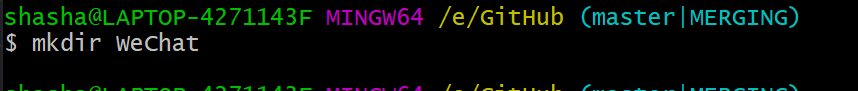
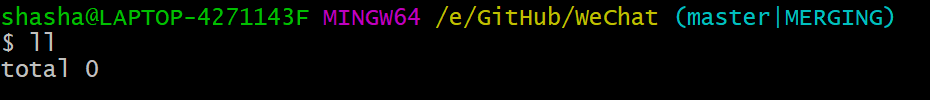
1. 对本地库进行初始化

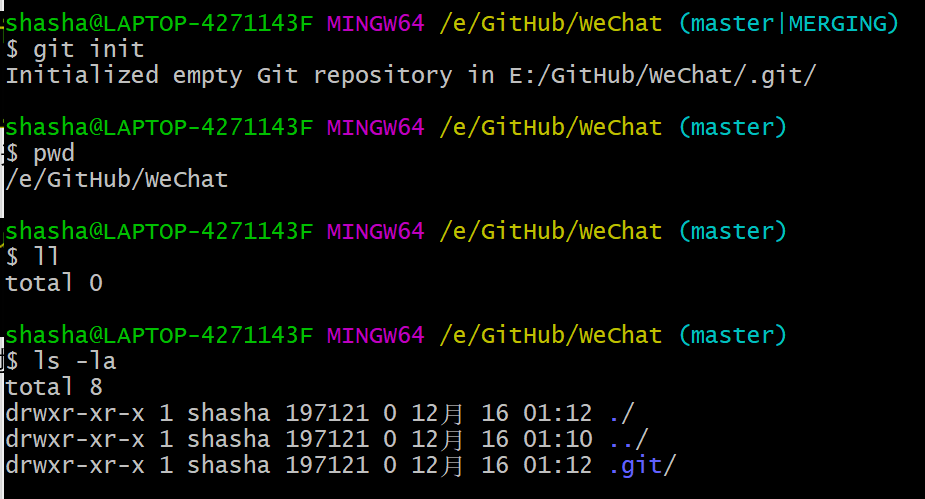
* Git base here打开命令行界面
* mkdir WeChat 创建一个名为“WeChat”的文件夹



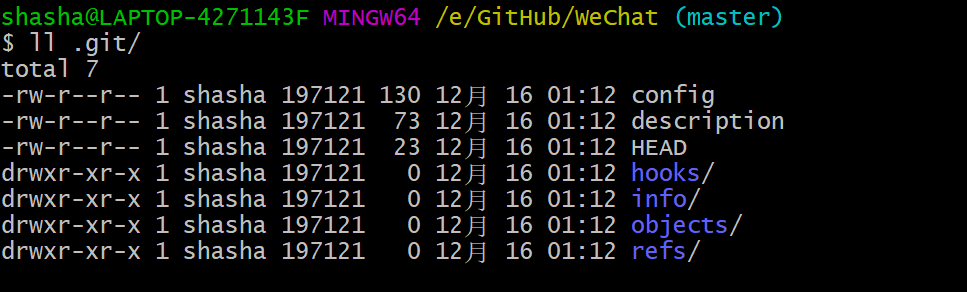
* cd WeChat/ 进入WeChat 文件夹，此时，该文件夹为空，可以通过ll来查看当前目录下文件：



git init初始化仓库，此时会生成.git隐藏文件，可通过ls -la查看当前目录下所有文件（包括隐藏文件）



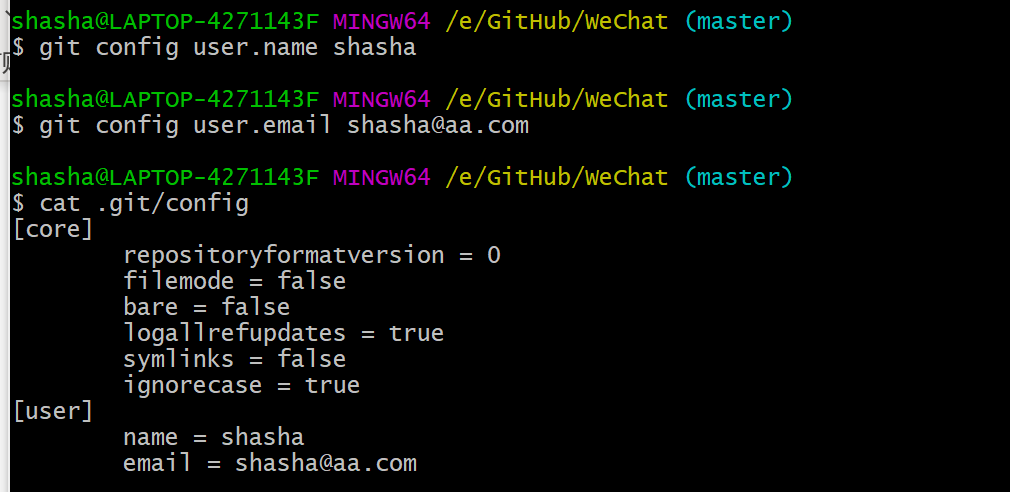
.git文件内部有如下文件



设置签名：git config 命令

项目级别:git config user.name xxx

Git config user.email [xxx@xxx.com](mailto:xxx@xxx.com)

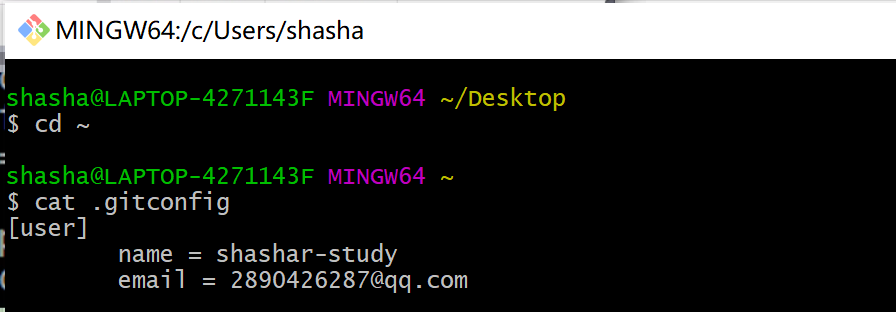


设置的信息存储在.git中的config文件中

系统用户级别：git config --global user.name xxx

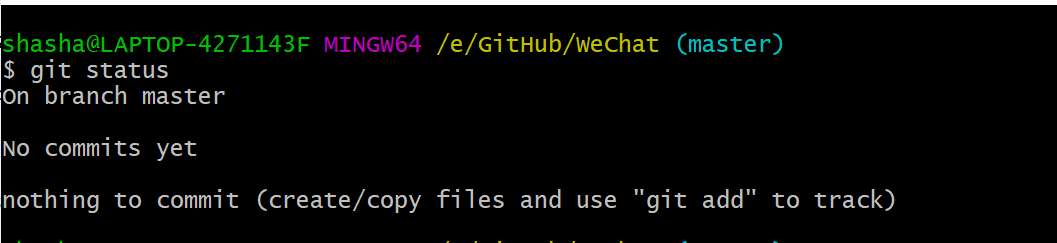
Git confid --global user.email [xxx@xxx.com](mailto:xxx@xxx.com)

同理的，系统用户变量存储在.gitconfig



基本操作：

git status查看工作区和缓存区状态



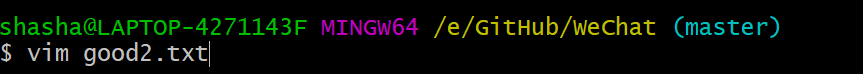
On branch master:在master分支上

No commits yet :没有任何提交

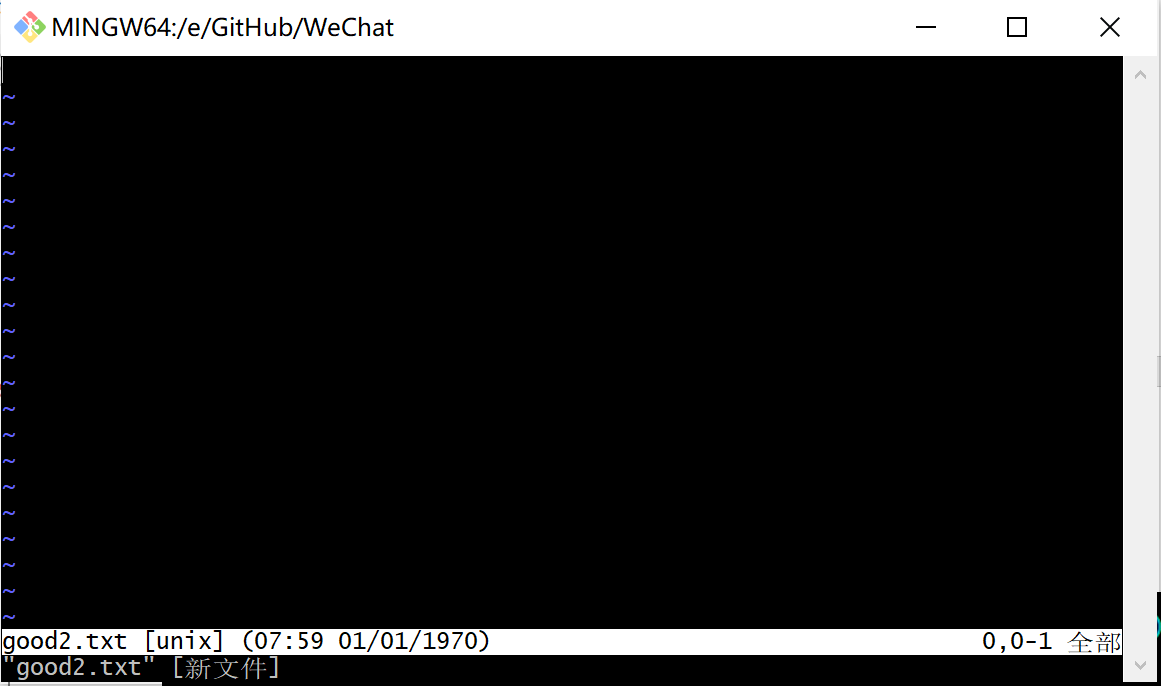
Nothing to commit :没有可提交的东西（暂存区里无东西）创建或者复制文件，使用git add命令追踪该文件

vim xxx.txt 使用vim命令创建一个xxx.txt 文件，随便写一点内容

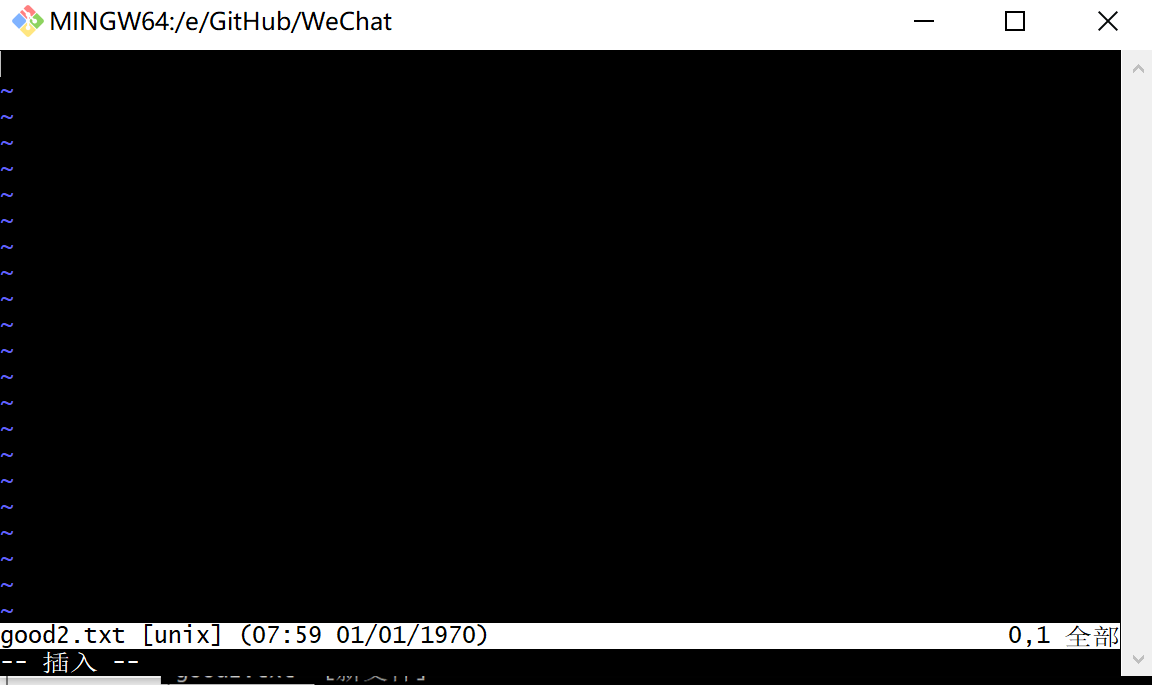
注意 :步骤1：vim good2.txt



回车，进入vi 的一般模式：

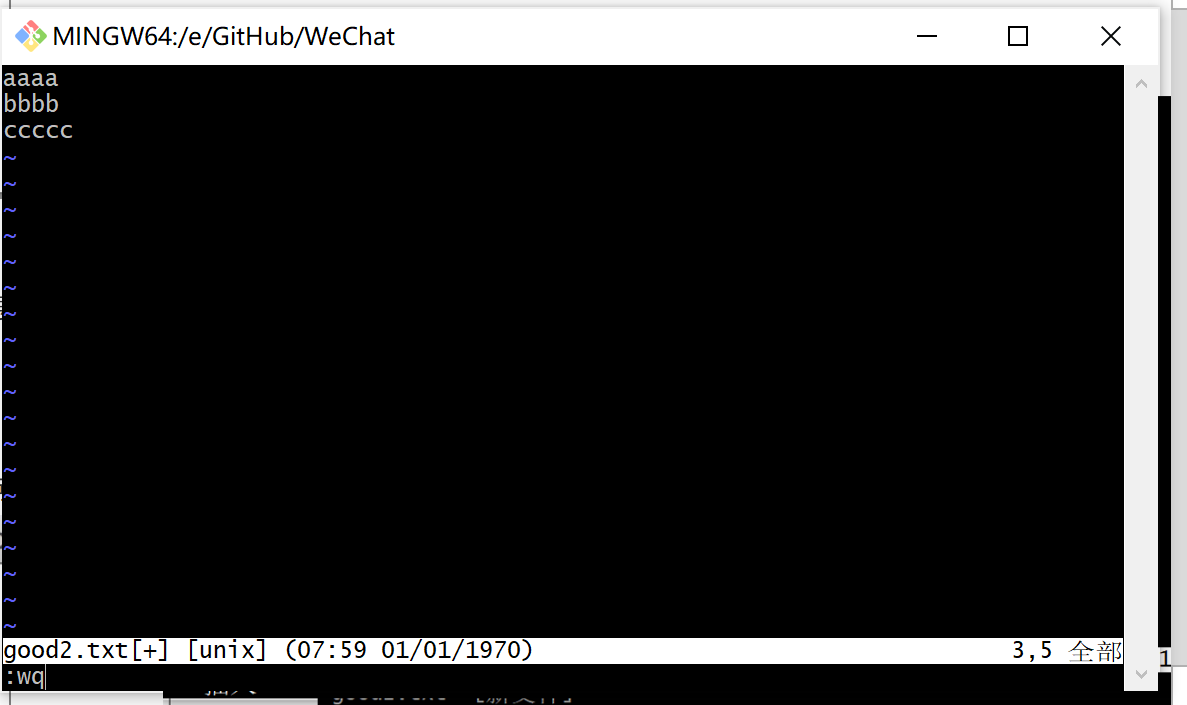


在这里~代表无内容，在一般模式下，按下I,o,a等字符可以进入编辑模式

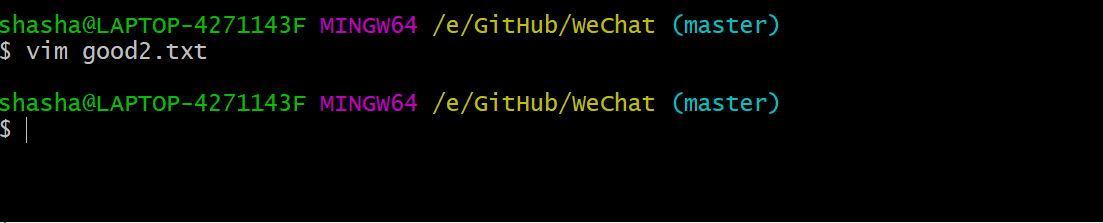


此时，可以进行写入部分内容，写入完成后，esc回到一般模式

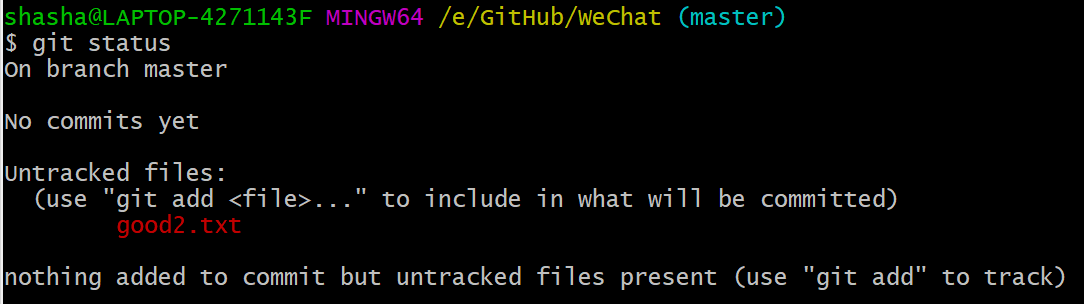
在一般模式下输入:wq储存后离开



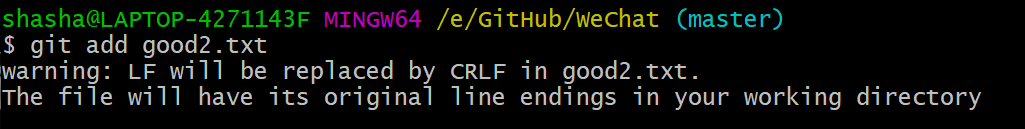
再次回车后回到上一界面：



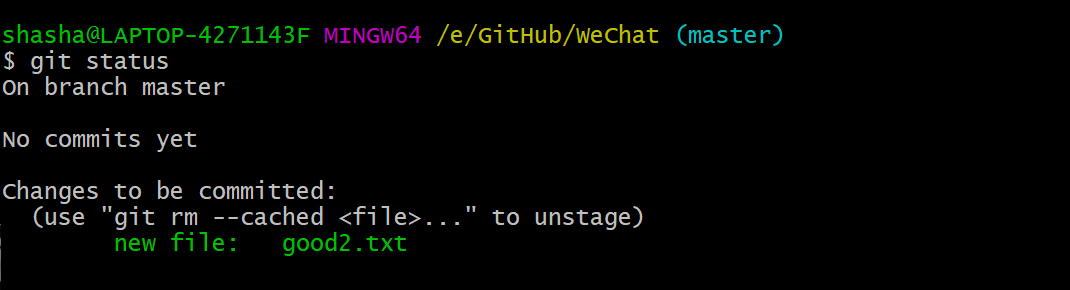
此时使用 git status 命令



发现未追踪文件，提示可以使用git add把该文件放入暂存区，根据提示使用命令



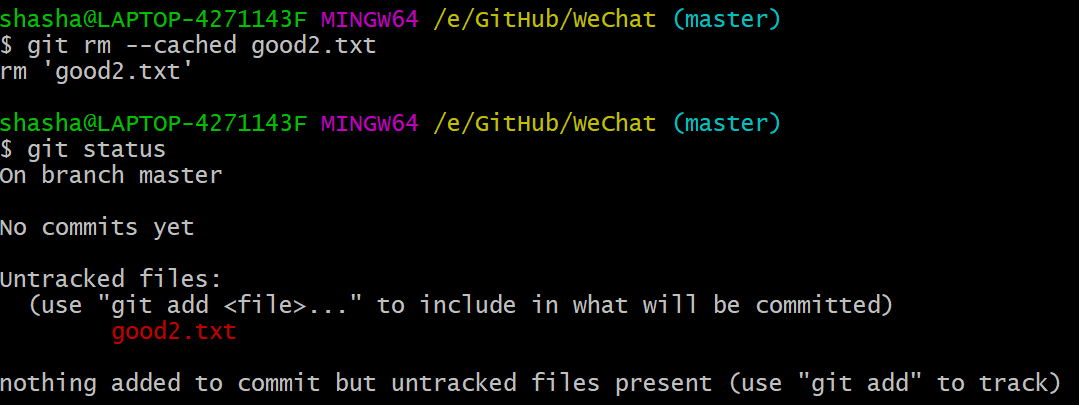
警告：LF会被替换为CRLF



此时，good2.txt状态发生变化，存入暂存区，颜色变为绿色

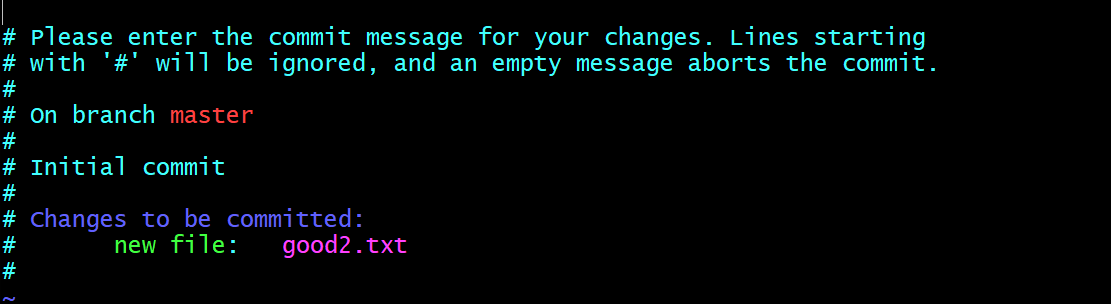
有改变可以被提交，出现了新的文件

根据提示，使用命令git rm --cached命令可以将文件重新转移出暂存区



可以看到 ，good2.txt回复之前的状态，在工作区内而不再暂存区中

接下来使用 git commit good2.txt命令进行提交，出现以下界面：

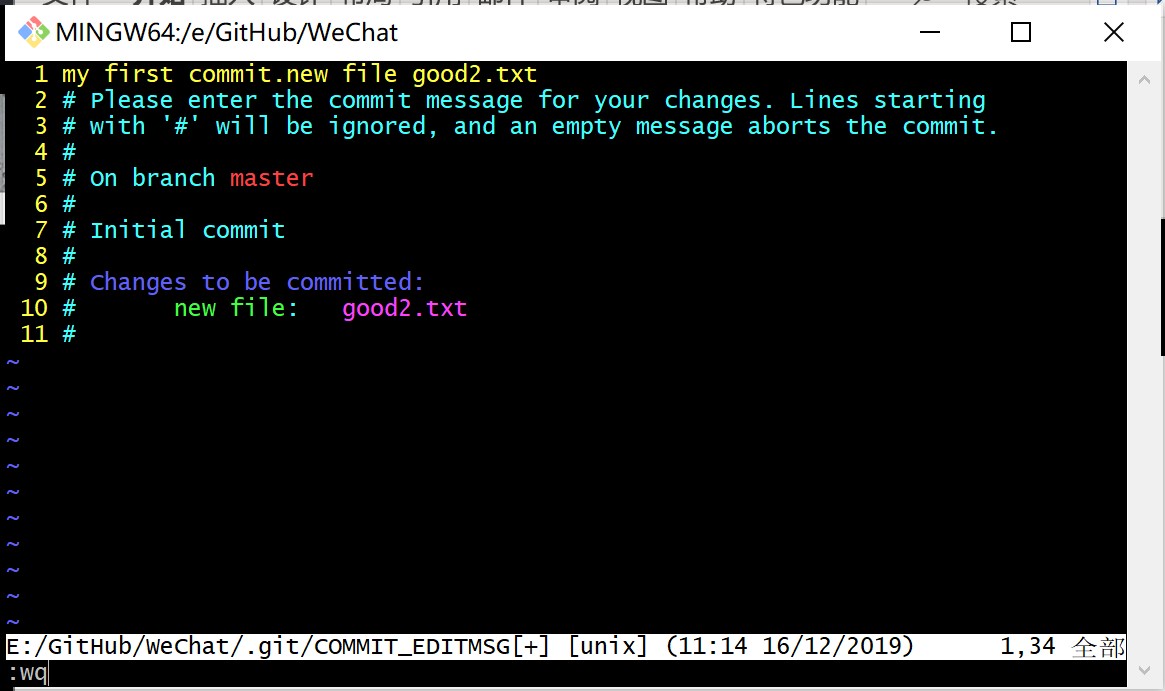


第一行提示输入提交信息，相当于注释，进入了vim编辑器中

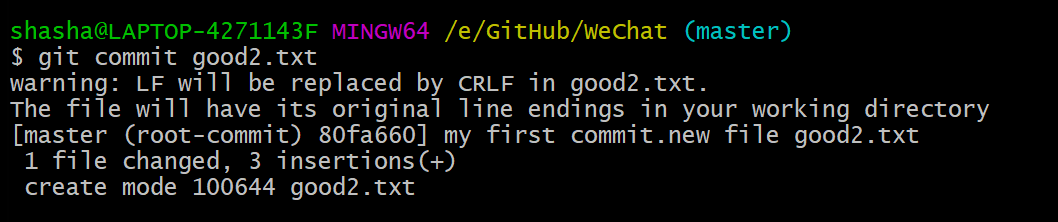
进入一般模式后，使用:set nu 可以显示行号



接下来，进入编辑模式，编辑信息后退回一般模式，使用:wq保存并退出



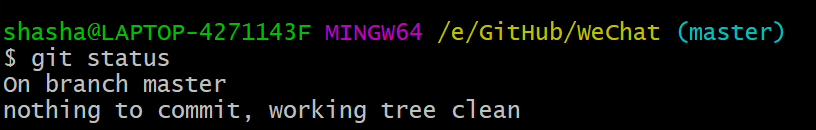
此时git commit 命令已经完成



第一次提交作为root-commit,后面的80fa660目前可以认为是版本号

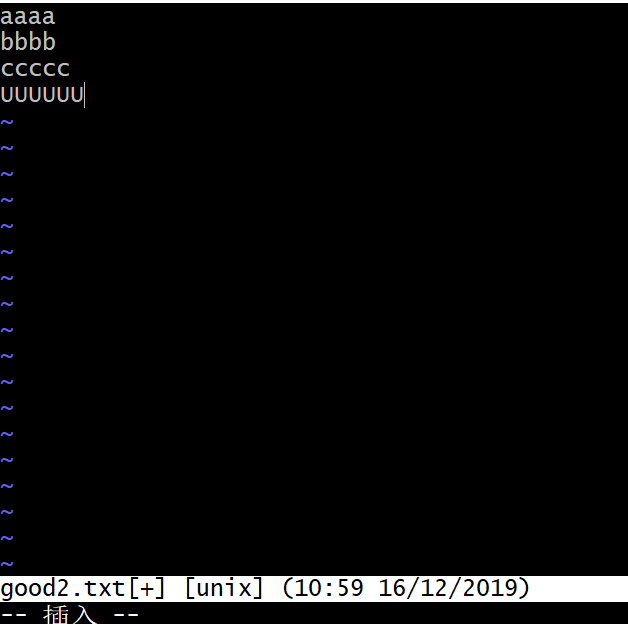
一个文件改变了，新增了3行，因为之前在good2.txt中写入了三行文字们可以对应的上

此时，再看一下状态

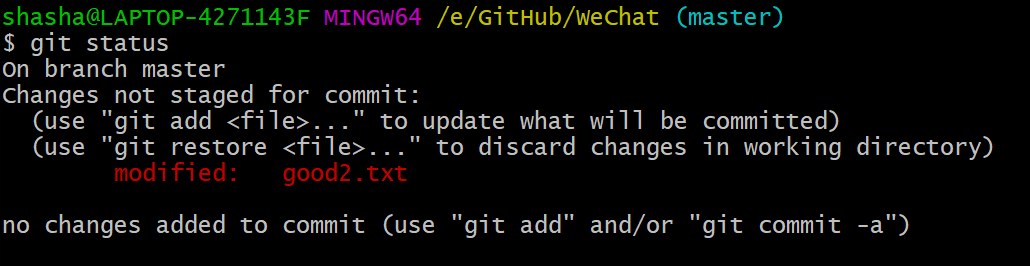


目前暂存区里没有要提交文件，工作区无修改

接下来，我们对于good2.txt做一下修改，使用vim 命令，写入保存退出，如下：



再使用git status查看



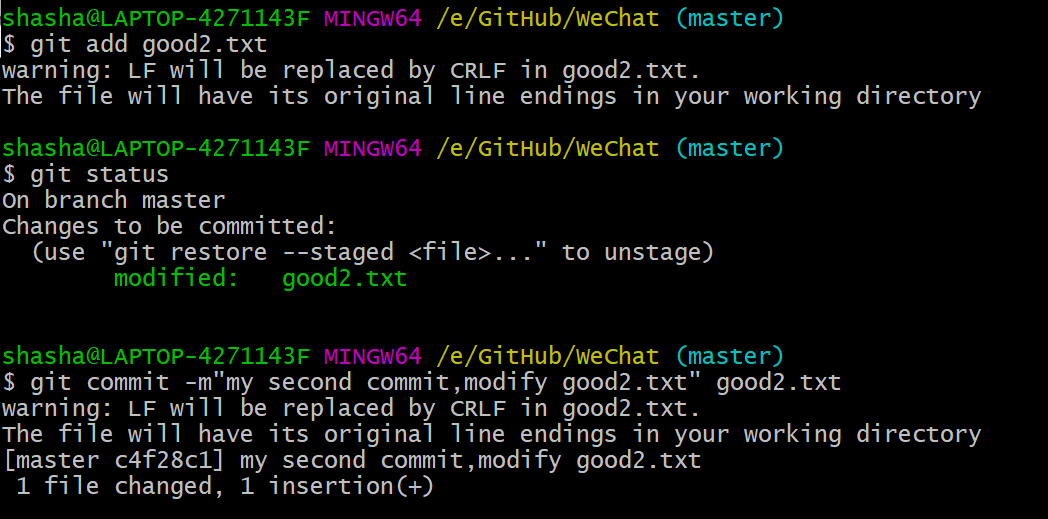
红色字体提示modified,文件修改，但是没有放进暂存区

提示可以使用git add命令进行update;或者使用 git checkout 取消修改

可以使用git add或者直接使用 git commit -a提交

接下来使用git add，修改放入暂存区

提交时，推荐使用git commit -m”XXX” good2.txt命令，可以不进入VIM编辑器，直接在XXX部分写入注释，结果如下，可以看到，版本号已经更新

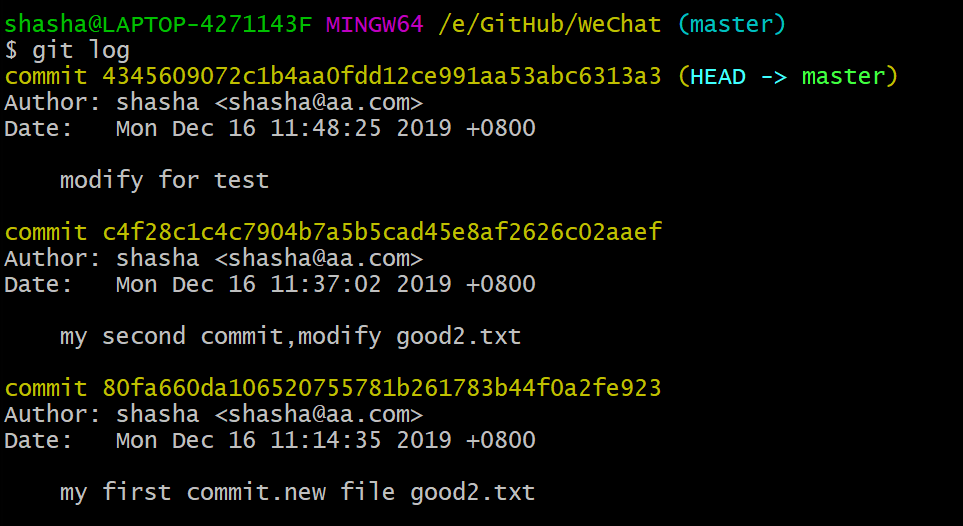


使用 git log,查看历史版本记录，可以看到，有两个版本的历史记录



HEAD是一个指针，指向当前版本

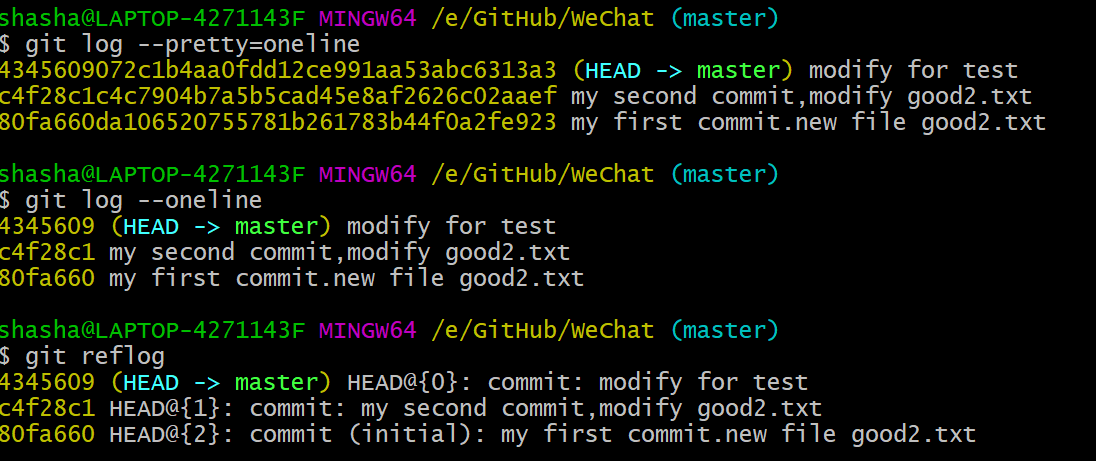
为了更直观的感受，我们再进行一次版本更替



若日志过多时，多屏显示控制方式：  
空格向下翻页  
b 向上翻页  
q 退出

可以使用 git log --pretty=oneline

或者 git log --oneline或者 git reflog

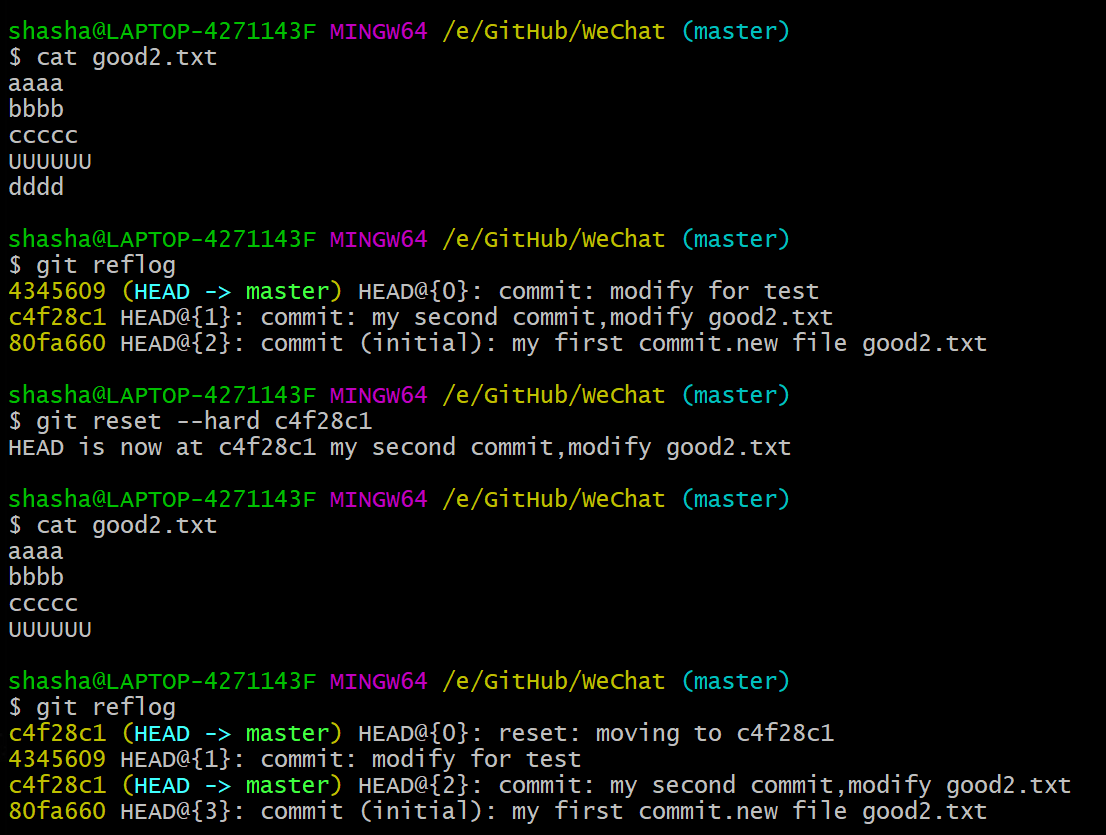


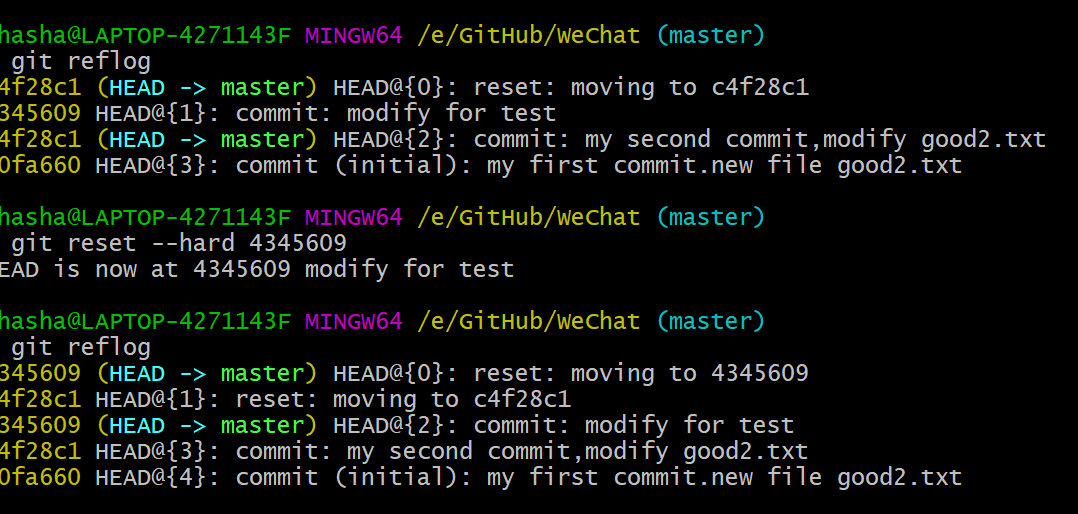
HEAD@{移动到当前版本需要多少步}

进行版本控制时，需要先将git reflog 命令调出局部索引值

进行版本控制的三种方法：

1. 基于索引值操作[推荐] 可以进行前进或者后退  
 git reset --hard [局部索引值]

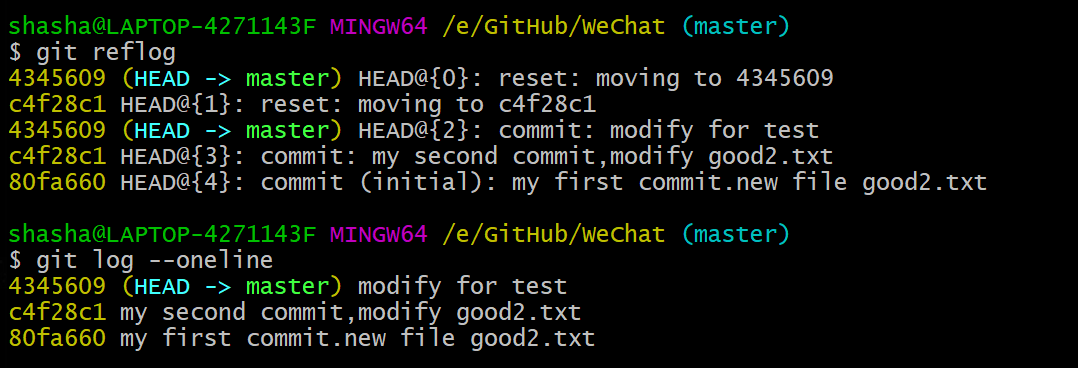


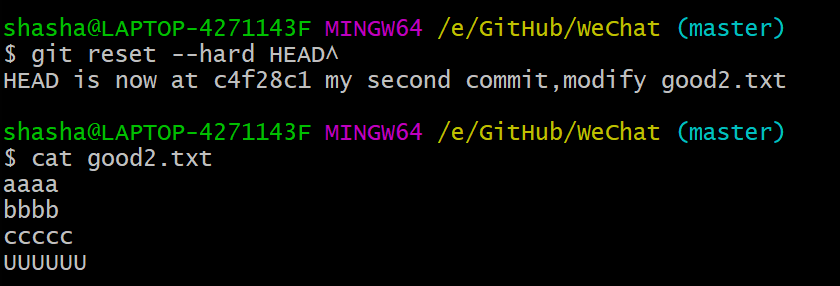


1. 使用^符号, 只能后退  
    git reset --hard HEAD^  
   注： 一个^表示后退一步， n 个表示后退 n 步

这个更适合用git log --oneline来显示版本

我们可以和git reflog 进行比较一下





1. 使用~符号： 只能后退  
    git reset --hard HEAD~n  
   注： 表示后退 n 步

git reset [<模式>] [<提交>]

该表格会将当前分支头重置为，<commit>并可能根据来更新索引（将其重置为的树<commit>）和工作树<mode>。如果<mode>省略，则默认为--mixed。的<mode>必须是下列之一：

-- soft

仅仅在本地库移动HEAD指针，而不会更改暂存区和工作区。完全不触摸索引文件或工作树（而是将头重置为<commit>，就像所有模式一样）。就像git status 这样，这会将所有更改的文件保留为“要提交的更改” 。

使用git reset --soft 命令之后，git reflog命令变化与--hard一致，但是good2.txt文件内容没有变化，使用git status命令可以看到待提交的修改（绿色modify）

--mixed

在本地库移动HEAD指针且会重置暂存区，但不会更改工作区。重置索引，但不重置工作树（即，已更改的文件将保留，但未标记为提交），并报告尚未更新的内容。这是默认操作。

使用git reset --soft 命令之后，git reflog命令变化与--hard一致，但是good2.txt文件内容没有变化，使用git status命令可以看到待上传至暂存区的文件（红色字体，提示git add）

如果-N指定，则将删除的路径标记为要添加的意图（请参见 git-add（1））。

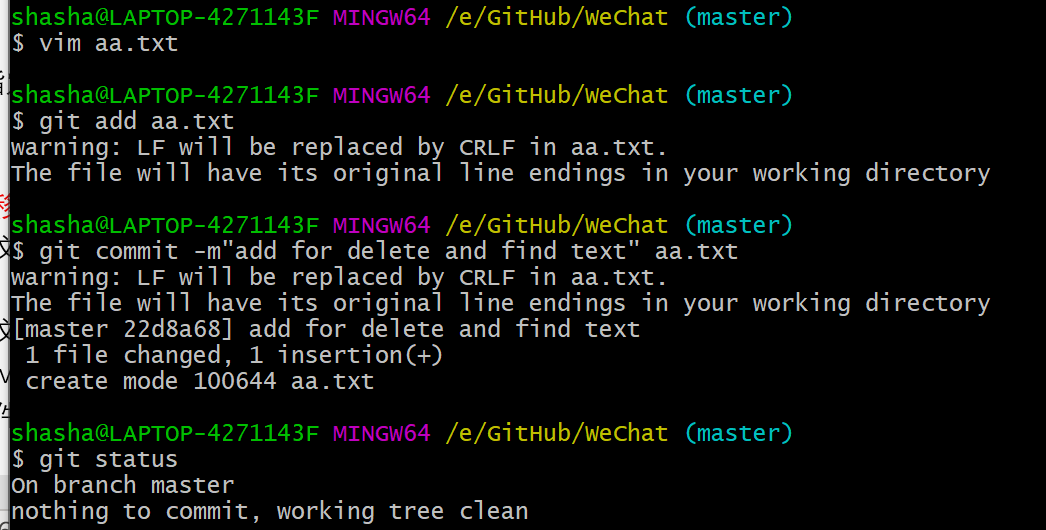
- -hard

在本地库移动HEAD指针，且会重置暂存区和工作区。重置索引和工作树。由于对工作树中跟踪文件的任何更改，<commit>都将被丢弃。

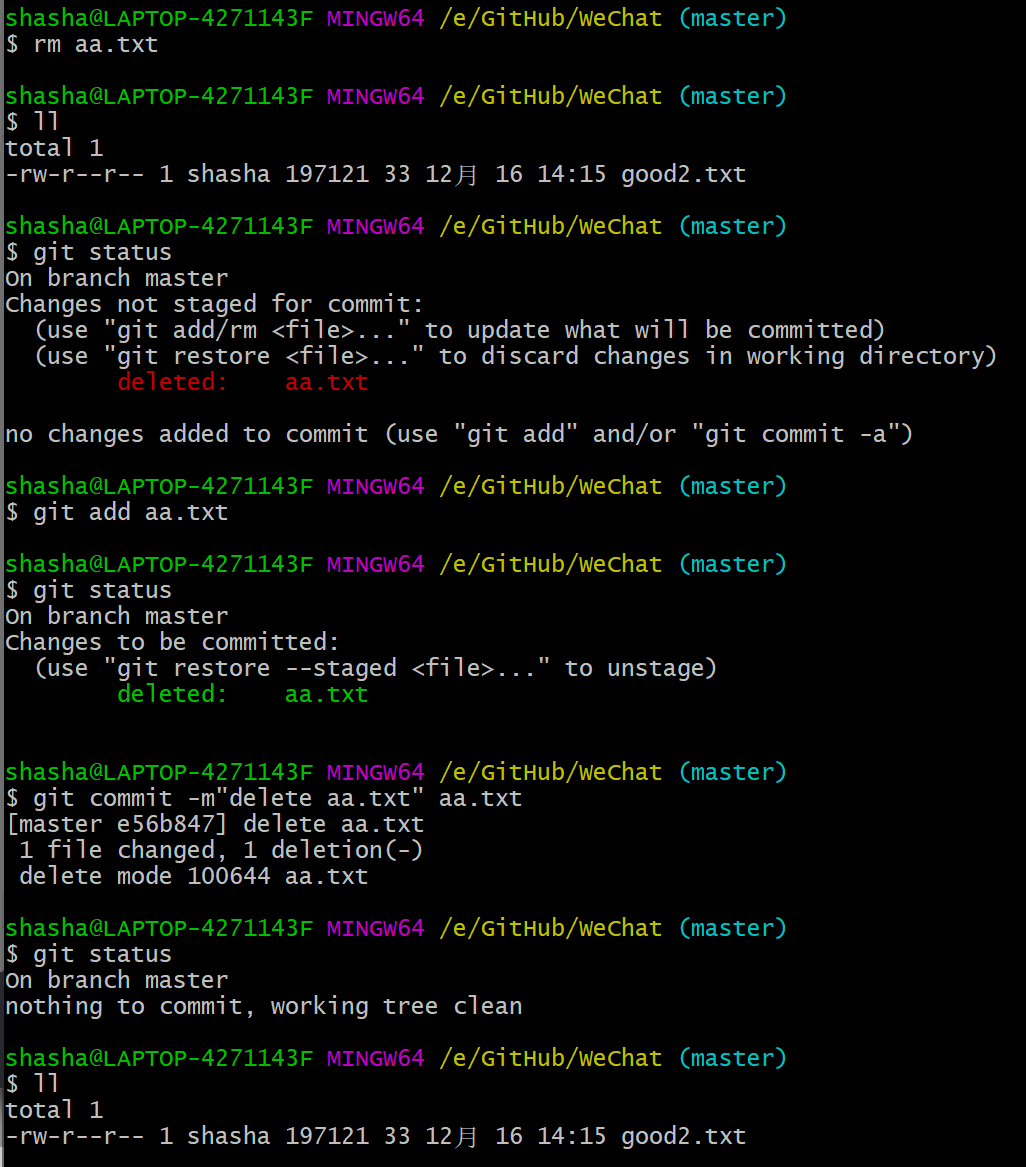
永久删除文件后找回

1. 删除文件——提交删除至本地库（产生了版本更替，可通过版本控制方式找回）

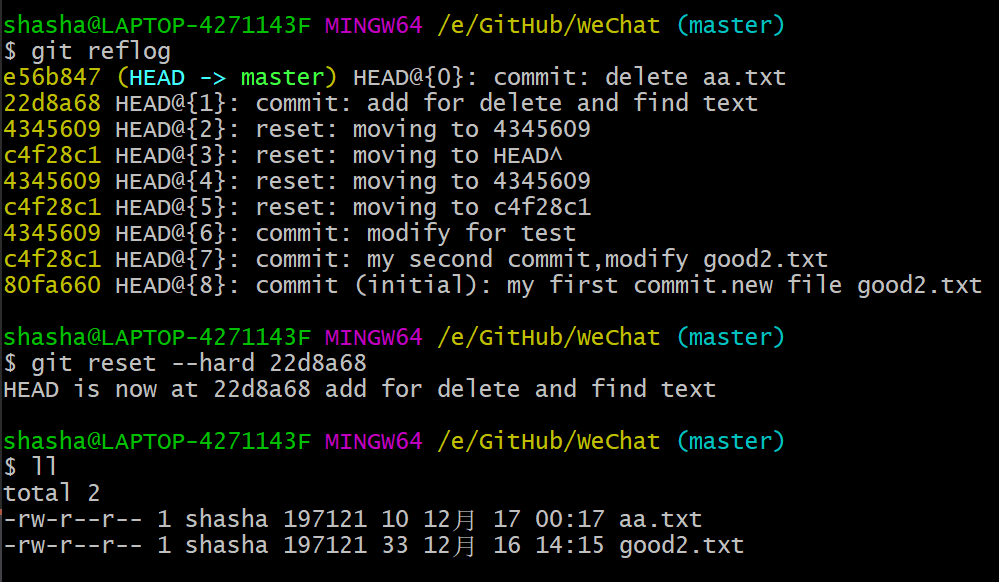
首先使用vim 命令新建一个文件，例如aa.txt,删除该文件通常指删除已经提交至本地库之后的文件



之后，使用rm 文件名 进行删除操作，删除之后，通过ll命令，可以发现本地库已经没有了这个文件，将改动提交



然后，可以通过版本控制方式，可以找会已经删除的文件

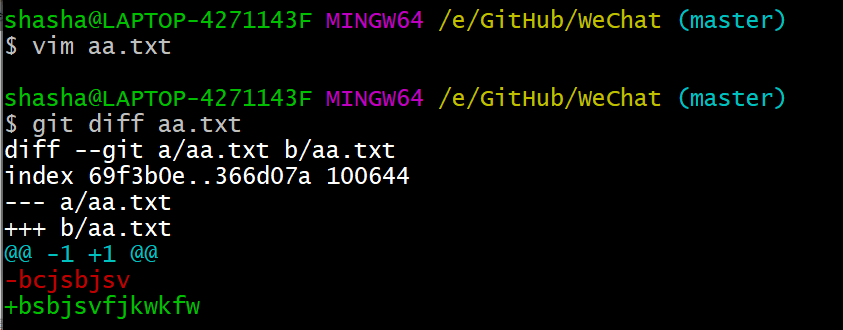


1. 找回文件——删除操作没有提交至本地库,仍然可以通过版本控制方式找回

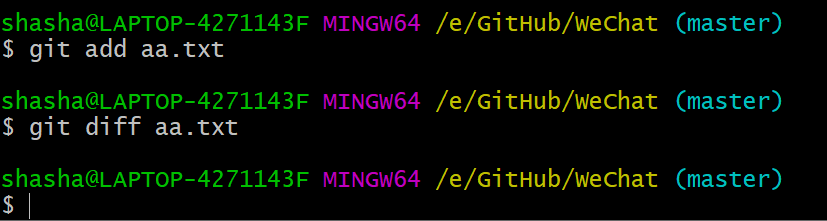
比较文件差异

修改aa.txt

git diff查找不同，红色是减少，绿色是增加；，因为git是以行来进行版本控制，所以相当于删掉原有行，又新增了一行



如果将改动提交至暂存区，再使用git diff命令，会发现，没有区别；说明，git diff不附加参数的话，是将工作区文件与暂存区中文件进行比较，不带文件名 是比较多个文件



若是增加HEAD参数，表明比较文件是本地库中的文件，此时，可以看到，又出现了差异

还可以和本地库中某一历史版本进行比较，如git diff HEAD^ aa.txt

