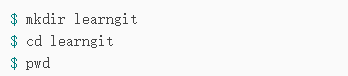
->创建文件夹名为’learngit’

->进入learngit文件夹

->查看当前文件夹

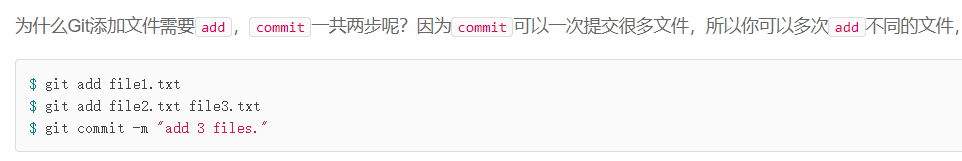


在该文件夹下用git init就可以将此文件夹作为管理的仓库了

用Notepadd++创建一个记事本txt，拖到这个仓库中，然后告诉git







1. 创建一个readme.txt文件，里面写一段话
2. 在git命令行中输入’git add readme.txt’, 回车，再输入’git commit -m created a readme.txt’, 回车
3. 这时候回到readme.txt，加上一句话，保存。这时候该文本已经编辑了
4. 回到Git,再输入一次‘git add readme.txt’准备提交修改, ‘’git commit -m modified my readme.txt’提交修改
5. 这时候commit提交了
6. Git会记录这些修改记录
7. 输入git log查看以往的修改记录以及commit id
8. Commit id可以帮助我们回到过去或者未来的某一个版本，只要我们在git commit -m <message>中将message写得详细且在之后留有印象，便可以坐时光机去到任何时间点
9. 当然若只是返回最近的几个版本没必要去查找这个commit id。

‘HEAD’代表当前的node，可以通过输入命令**’git reset --hard HEAD^’**返回上一次的修改节点

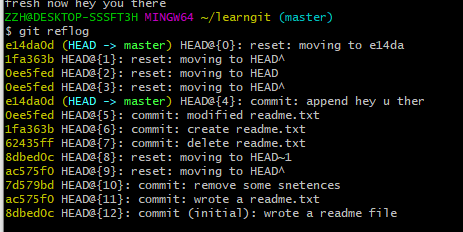
git reset --hard HEAD^^返回到上上次的修改节点

git reset --hard HEAD^^^再上一次，若要100次则git reset --hard HEAD~100

同时，这个HEAD指针随着这个修改指令不断移动，不能回到命令前

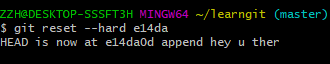
这时候就需要commit id了

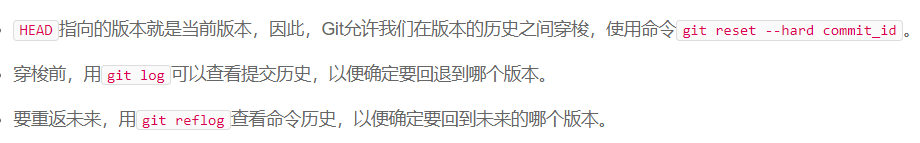
通过git reflog可以获得更全的日志信息

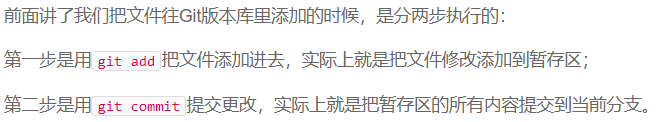


可以看到每次的更新都有记录，最左侧黄色黄色额的字符串便是commit id的前几位,我们也只需要前几位

Commit id放在’--hard’后面即可，如**git reset –hard e14da**



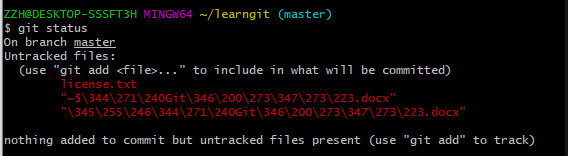
可以看到这一步的commit id和他所要前往的修改节点的commit id是一样的，都是’e14da0d’开头



因为我们创建gitt版本库时，Git自动为我们创建了唯一一个**master**分支

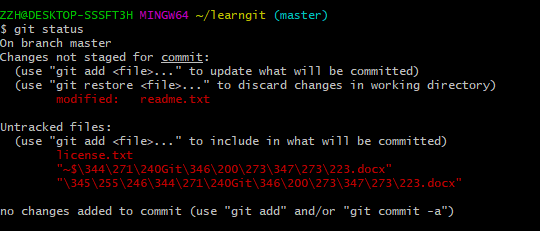
所以， git commit就是往**master**分支上提交更改。  
【git add+ 文件 🡺可以让git跟踪track文件】 每次add + commit都是对文本的一次游戏存档

当我在learngit文件夹中创建了一个新的license.txt文件时，在git add 之前，我输入git status查看一下工作区的状态



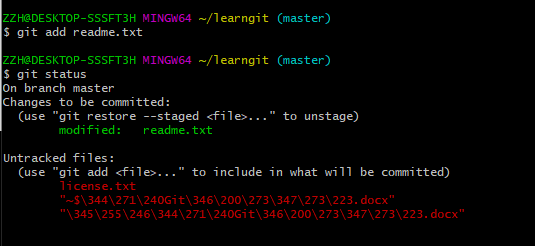
可以看到在untracked files下有license.txt

然后我修改了一下readme.txt，在里面加了一句话，然后再次查看status:



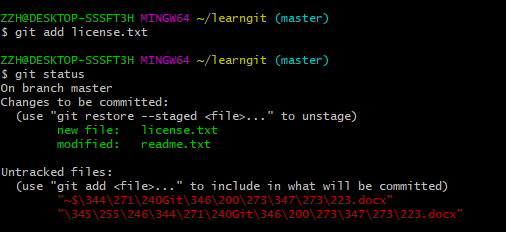
可以看到这个readme.txt内容变更后，git将他放在了‘changes not staged for commit’中

这是一个非常棒的记录，我们便开始用add将readme.txt放到一个**暂存区（stage）**中准备用commit做一次存档

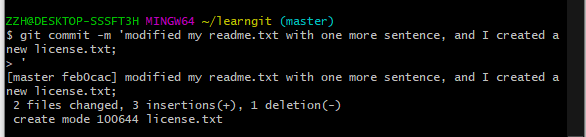


这里readme.txt归在了‘changes to be committed’下，并且变成了绿色，似乎意味着进展不错

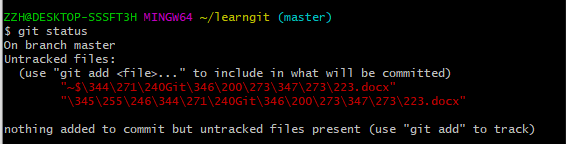
同样add了license.txt后：



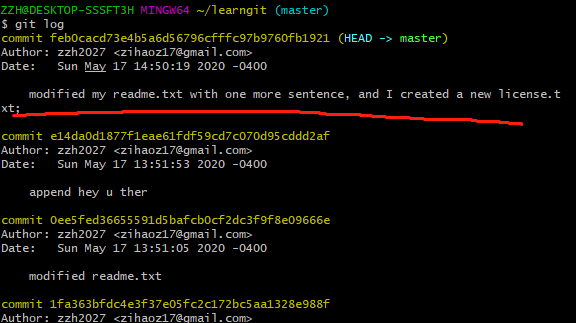
然后一起commit 上去



Status这里也清空了



日志显示了我的整个message

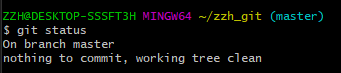


|  |  |  |  |
| --- | --- | --- | --- |
| 工作区（git init） |  | 暂存区（Stage） | Master branch |
| readme.txt  license.txt |  | **git add** |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 工作区（git init） |  | 暂存区（Stage） | Master branch |
| readme.txt  license.txt |  | readme.txt  license.txt | **git commit** |

|  |  |  |  |
| --- | --- | --- | --- |
| 工作区（git init） |  | 暂存区（Stage） | Master branch |
| readme.txt  license.txt |  |  | License.txt  Readme.txt |

因为之前删除了文件，git log里不是很干净，所以新建了一个zzh\_git的文件夹，作为新的工作区

将学习总结文件保存在里面，并存档了这一版的docx，然后查看git status，终于出现了这个这个干净界面

***Nothing to commit, working tree clean***

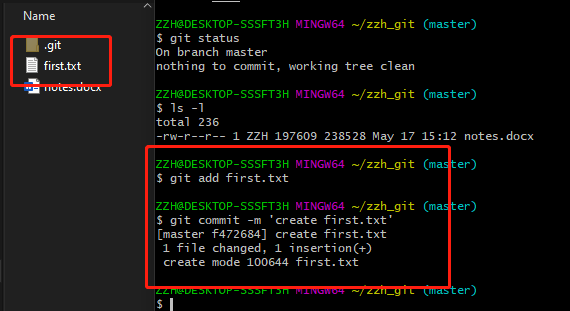
根据上面的图示便自然可以理解以下这种情况：

当我修改了一个文件，然后git add到了暂存区，然后我又回去修改了第二次,然后git commit。

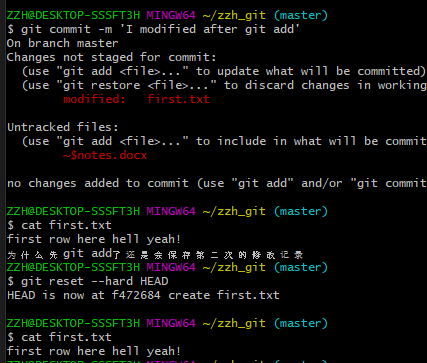
这时候第二次修改不会被存档，而是会被归到一个‘changes not staged to commit’中去

**来试试！**

创建一个新的first.txt，只写了一行话，然后上传存档！



接下来做第一次修改

然后在git add后又写了一行，才commit上去

可以看到cat first.txt那里显示的是第二次修改后保存的本文内容，而当我们reset到head时，会发现first.txt的内容已经变化了，原本一个正常存档的文本在reset到head node时都是不该有变化的。

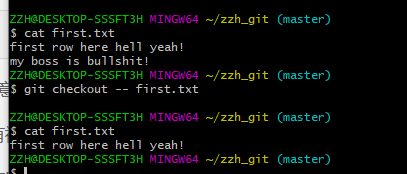
**我们可以理解为git commit只负责把暂存区的修改提交了，也就是第一次的修改被提交了，第二次的修改不会被提交。**

**或者**

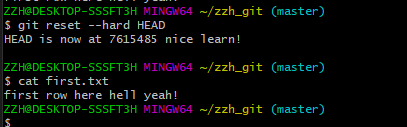
**git管理的是修改，而不是文件**

1. **如果文本文件里写了胡话，怎么办**

有两个办法

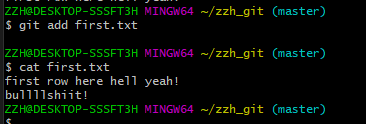


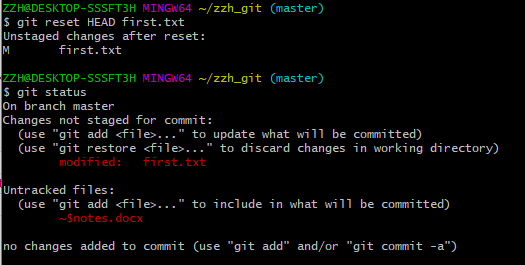
🡪git checkout -- file可以丢弃工作区的修改



🡪用reset把当前HEAD Node的记录覆盖当前工作区的文件

1. 如果写了胡话还add到了暂存区，怎么办



🡪这里用reset的方法

**git reset HEAD 文件名**

用git status查看到文件不在暂存区，**unstaged changes after reset!**

**这样子就回到了第一种情况，我们可以用git checkout -- 文件名的方式将文件修改**

**这里notes.docs是untracked，因为我目前在编辑**