规范ver0.5

**目录**

[规范ver0.5 1](#_Toc171932679)

[1.编码规范 2](#_Toc171932680)

[1.1关于注释 2](#_Toc171932681)

[1.2关于接口定义和合法性约束 2](#_Toc171932682)

[1.3关于缩进和换行 3](#_Toc171932683)

[1.4关于空格 3](#_Toc171932684)

[2.Git规范 4](#_Toc171932685)

[2.1关于提交和拉取 4](#_Toc171932686)

[2.2关于fork 4](#_Toc171932687)

[2.3关于提交注释和git使用 4](#_Toc171932688)

[2.4关于git的本地管理 4](#_Toc171932689)

[2.5关于git远程仓库管理 5](#_Toc171932690)

[2.6关于git远程仓库的合并管理 6](#_Toc171932691)

[2.7关于本地仓库的更新 7](#_Toc171932692)

[2.8关于流程介绍 7](#_Toc171932693)

[2.9关于仓库的目录结构 8](#_Toc171932694)

[3.模型的使用 8](#_Toc171932695)

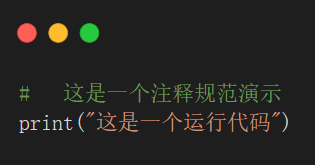
[3.1模型的绘图 8](#_Toc171932696)

1.编码规范

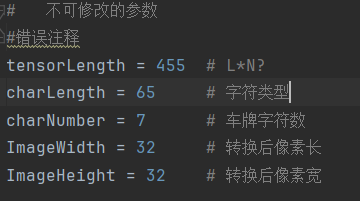
1.1关于注释

所有编码都需要进行注释，没有注释的代码经审核后不会同意合并分支并且拒绝push请求。

所有的注释使用#进行注释，在注释时使用tab进行对齐。不可以使用空格对齐，注释需要写在代码的上方，当一行不够的时候可以追加且需要注意对齐。

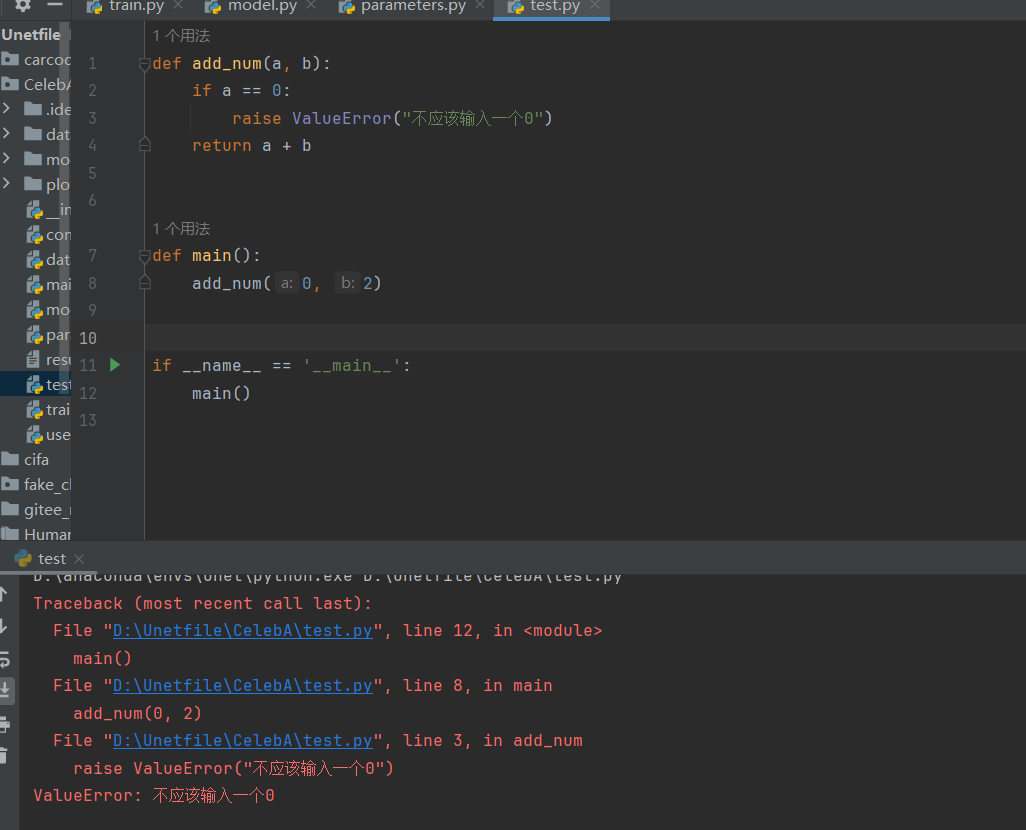


如图下图所示，一个好的注释应该做到对齐，否则下划线会标黄线。在对齐时，使用tab即可。会自动对齐的。假如不对齐，不使用制表符则会亮黄线，因为不符合PEP8编码规范。为了编写代码的时候眼睛舒服，尽量规范注释。



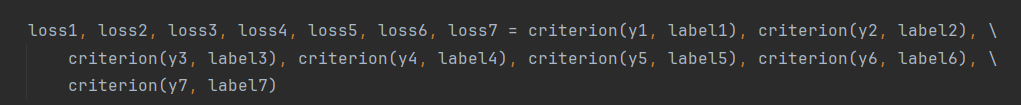
1.2关于接口定义和合法性约束

在接口编写时，需要进行合法性检查。只有接口输入的数据合法才能进行下一步运行，否则抛出报错。规范如图所示。



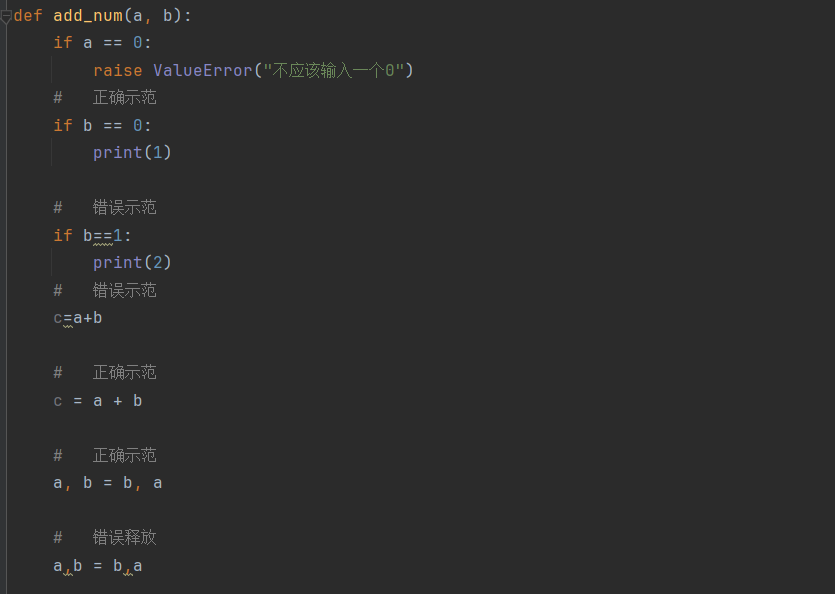
1.3关于缩进和换行

Python重视缩进，不要使用空格。当一条代码过长时，不要写在一行中，应该使用换行符换行。或者直接换行。



1.4关于空格

在变量之间，赋值之时，判断之刻，应尽量使用空格。下图是一个示范。



2.Git规范

2.1关于提交和拉取

对于master分支在未经允许时不可以进行修改。也就是不可以对master分支进行写操作，只允许读，也就是只允许抓取克隆到本地。

只有在单码经过审核后，也就是merge合并请求后且代码审核没有问题时，才可以进行合并。

2.2关于fork

在开发时需要将代码fork出来进行开发，这个概念会在日后进行更新并讲解。写好时会更新文档。在本地执行pull即可。

2.3关于提交注释和git使用

每次git提交都需要进行注释.并且git add时最好不要git add . 也就是不要将所有的文件都add到缓存区，这样提交到git仓库时是所有文件都提交到代码仓库的。这样的习惯不好，也危险。很容易造成本地错误的文件覆盖仓库的基线文件。因此git add时特定对应的文件，也就是git add filename。

2.4关于git的本地管理

1、你想创建新的项目库

创建一个文件夹（整个目录不能有中文，windows系统编码问题），然后在此文件中，右键git bash 输入git init

2、你想用git管理你的项目

把项目复制到这个文件夹下，输入git add . 再输入 git commit -m "这是我项目的第一个版本"

3、你想更新了你的代码

输入git add 你的代码文件 ，再输入git commit -m "我刚修改了x"

4、你想删除你的没用的文件

输入git rm 没用的文件路径和名称， 再输入git commit -m "我刚删除了x"

5、你做了新代码，发现新功能不 好，然后回溯到某个版本

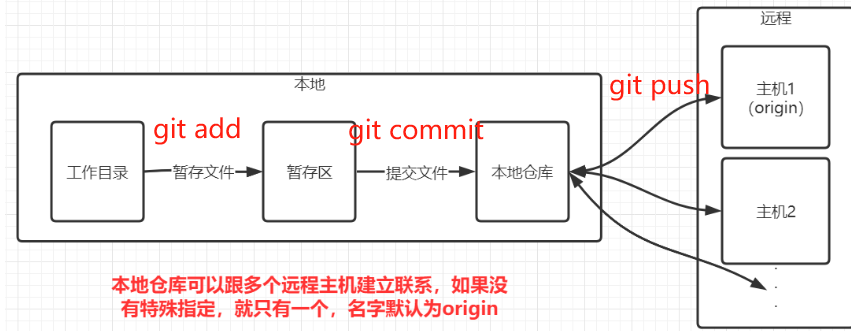
输入git log，再输入git reset --hard 版本号（通过git log看到的hash值就是版本号）

6、如果回溯也错了，不记得是哪个版本了

输入git reflog，可以看到所有的版本，如A版本和B版本还有C版本，你回溯到了B版本，那么A版本的id不记得了，通过reflog就可以看到回溯的操作和所有版本的commit，再通过git reset --hard 版本号 回来。

7、如果你想看git的步骤到哪了，有没有需要提交的东西

输入git status，可以看到目前各个区域的情况，如果想要对比这个文件和之前的文件的情况，可以通过git status发现哪些文件和版本库的分支不一样。在通过status列出的文件，进行git diff 文件名 对比。



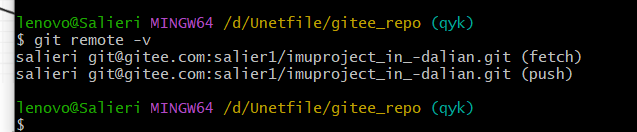
## 2.5关于git远程仓库管理

远程主机名指本地连接的git仓库，可以通过git remote -v查看。比如我这里将远程仓库命名为salieri，所以远程仓库名就是salieri。首先，需要在本地创建自己的分支，也就是git branch yourname。然后checkout到那个分支，git checkout yourname。然后就可以继续你的编码。

当然，你也可以不在本地创建新的分支。只使用本地的master分支，但是在push的时候需要push到你自己的分支。也就是git push reponame master:yourforkname。

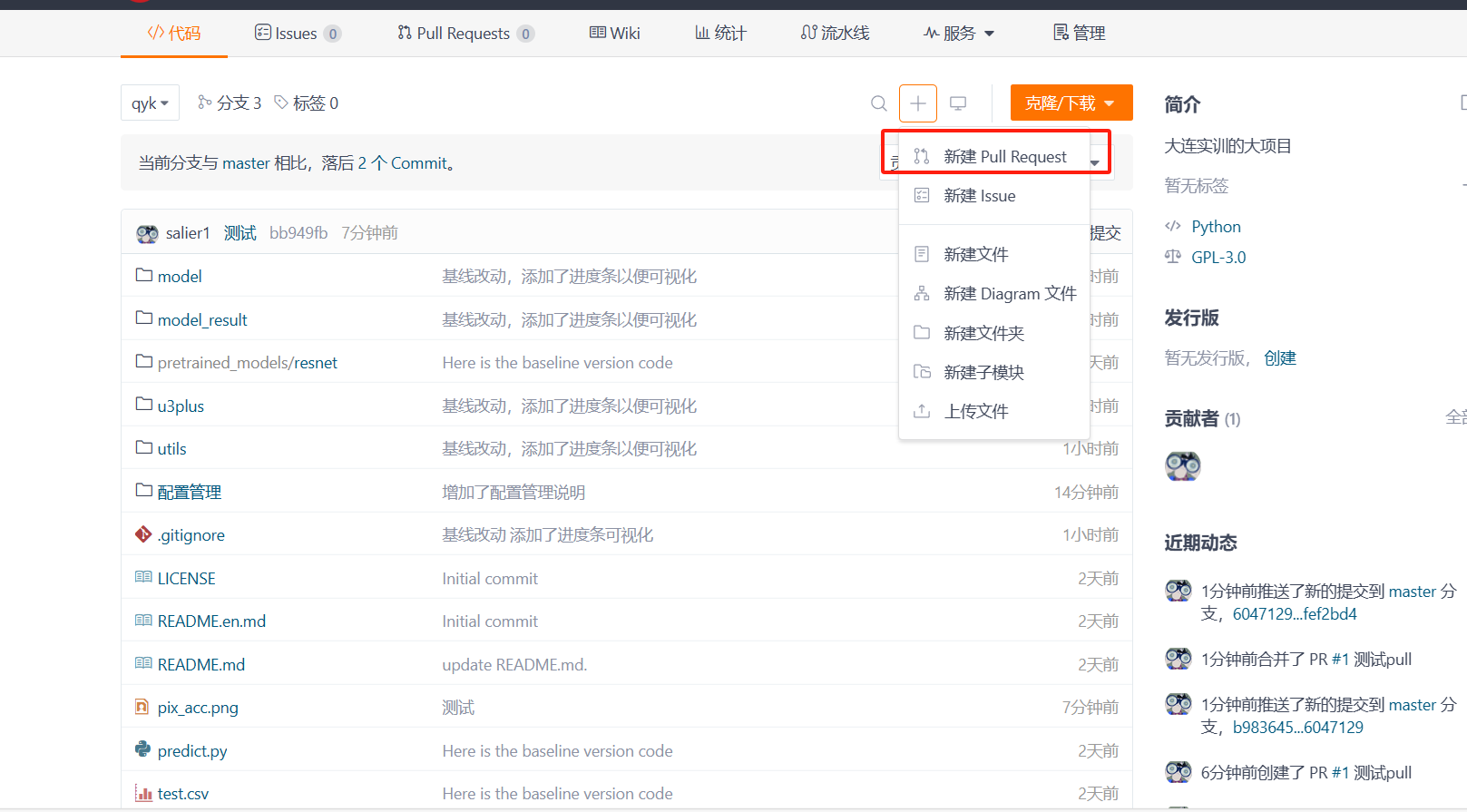
假如远程没有对应分支会自动创建一个分支。所以自己执行就行。最好以自己的名字缩写命名远程分支。因为到时候仓库会统计提交数，统计的时候会看的比较方便。





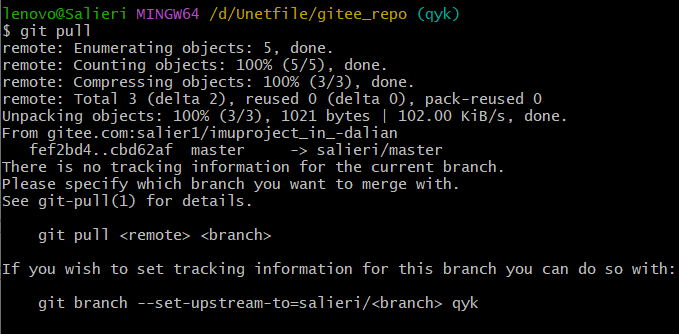
## 2.6关于git远程仓库的合并管理

在你对远程仓库自己的分支进行更新后，点击仓库的pull request可以申请远程仓库的合并。申请了的话我会查看代码，如果没有问题就合并到主分支。再次注意，提交到远程仓库时，提交到自己的分支上。不要提交到master分支，理论上来说也没有权限提交到master分支。更多的信息可以看[使用 Pull Request 功能进行代码审查 - Gitee.com](https://gitee.com/help/articles/4304#article-header0)。



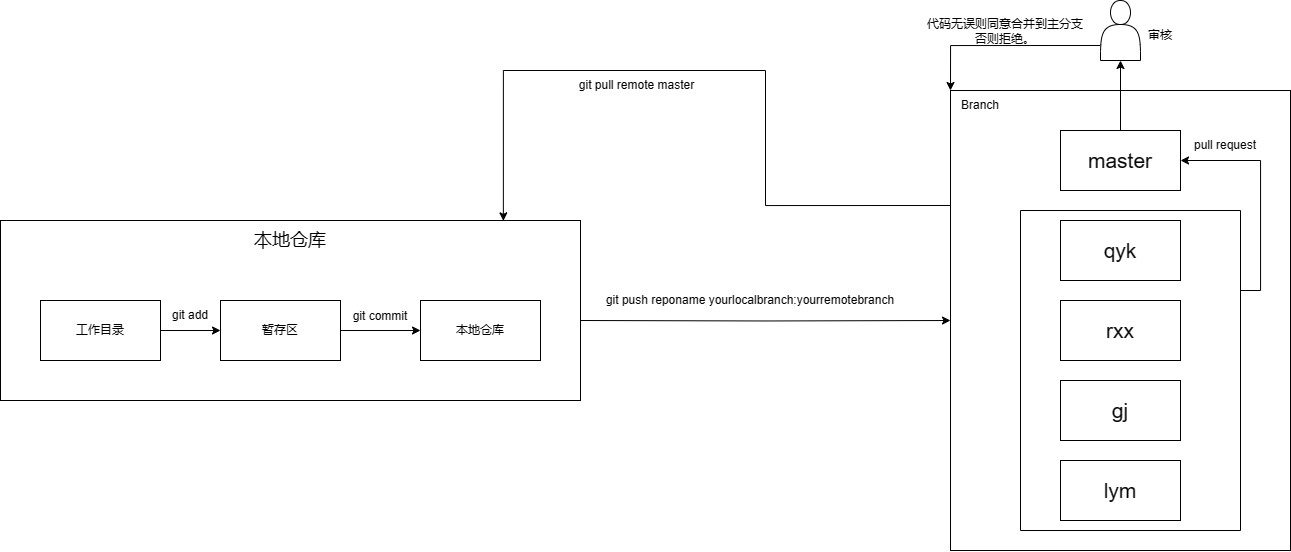
## 2.7关于本地仓库的更新

使用git pull remote branch这样的格式来更新本地仓库。假如仓库名通过git remote -v来查看。一般来说就执行git pull remote master，只把master分支更新到本地。当然，你自己分支更新到本地也可以。但master分支是基线，通过冻结这个基线来保证全部人的代码一致。

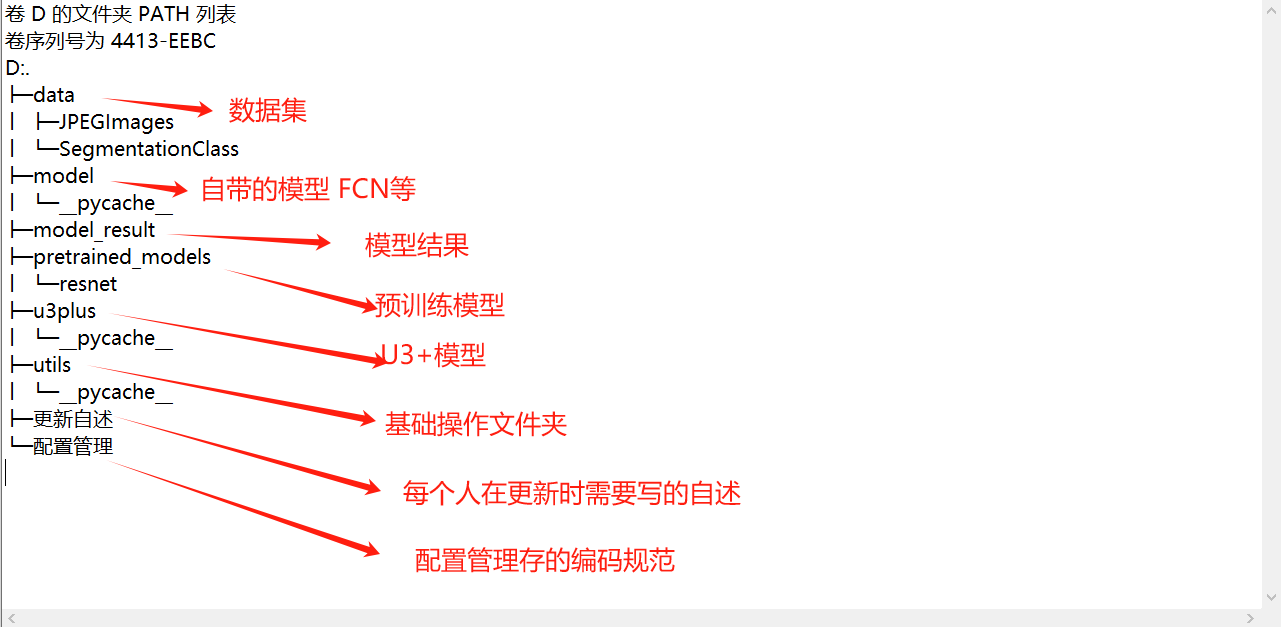


## 2.8关于流程介绍

如果你对上述的过程感到混乱，可以看下边这个图。代码每日进行一次提交，第二天每个人都需要更新自己的仓库。



2.9关于仓库的目录结构



3.模型的使用

3.1模型的绘图

模型绘图要通过Tensorboard进行，相关教学会在后续更新。