密码: mysql:123456 ssh:root a1b2c3d4E5

版本管理 查文献

多人合作 github 的branch方法

项目进度管理

历史项目的学习，持续改进 一个综合页面，都可以查看

可以考虑加入issues板块

可以考虑下载或者加载的时候加入进度条

代码不够的话：搜索功能区分，全文搜索或者文档匹配，全部匹配或跳字匹配，issues板块。

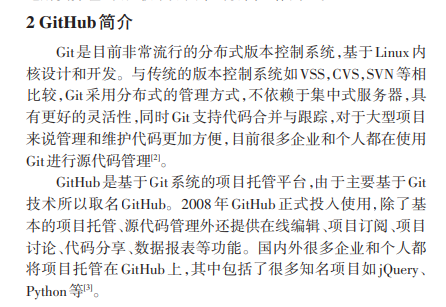
看可不可以把一些人工智能算法，机器学习算法和大数据算法融合到毕业设计中去。

界面美化。

可以将远程服务器连接和配置做成一个界面，用于自定义将repo传到哪个服务器。

**前言**

现在有些老师上课就会将课程作业和课程项目等布置到github上面，可以直接面对这一需求做一个更符合的软件。

可以引用这两篇论文中的引言，以及git相比于其他软件的有点，比如

1.首先做注册登录界面，注册登录界面主要是需要链接数据库和实现界面跳转。界面跳转是新建了一个static变量all\_frame得到run()方法里面的frame，然后在需要跳转的地方将当前frame(即all\_frame)dispose掉，然后跳转到目的界面( setVisible(true) )

看历年的论文中具体的内容和思路。

看看学校图书馆电子图书里面的文献有些什么用处。

然后需要考虑进去之后的界面。有一个问题就是最后从成品是一个电脑软件，每个用户的作品文件需要上传到指定的服务器，怎么将文件传到服务器而不是本地是一个问题。

3.17

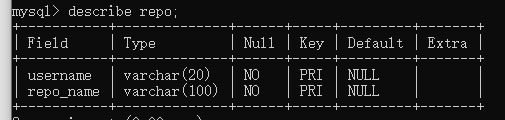
首先考虑登录之后界面的制作，需要固定界面大小。借鉴qq音乐和Github桌面版软件大小之后，选定参数为setBounds(100,50,1050,600)

之后的界面想展示个人repo，需要数据库记录个人之前的项目，先尝试用Table实现(Table 需要自己用手写代码插入行)。之后要实现搜素框,搜索框先做了一个基本的样式，之后具体搜索算法看时间和设计难度按照KMP或者BM算法实现。

然后看怎么在远程服务器上创建repo.设想用数据库，利用username和项目名称作为primary key，将repo上传到GitHub上，用GitHub模拟远程的服务器。

发现之后的项目进度功能可以使用JCheckBoxMenu实现。JMenu如果有需要也可以使用。

3.18

在数据库中床架一个repo的表格，

在网上查找是否有免费的git服务器。

具体是在一个账户里面创立所有的项目，再从数据库中将创建的项目进行分类，最后是否可以实现

先做成一个框架，不能直接在软件上查看项目中的文件，只能上传文件或者下载文件，做完框架之后之后可以稍微是否可以实现一些基本的代码可以在软件中展示和txt等简单文件在线展示，word和pdf文件等可以不用在线展示。

软件大小改成了屏幕大小。获取屏幕长宽，然后setBounds为整个屏幕的大小。

版本管理查看一下word的crtl + z的原理。

需要友情提示的是，你看到的一大串类似1094adb...的是commit id（版本号），和SVN不一样，Git的commit id不是1，2，3……递增的数字，而是一个SHA1计算出来的一个非常大的数字，用十六进制表示。为什么commit id需要用这么一大串数字表示呢？因为Git是分布式的版本控制系统，后面我们还要研究多人在同一个版本库里工作，如果大家都用1，2，3……作为版本号，那肯定就冲突了。

突然发现有Git的GUI版本。借鉴一下其中的功能。例如可视化branch, pull等过程。

还可以借鉴一下Gitee的功能。比如存在Members的分工，之后考虑这个功能是否实用。

由于Git中的版本回退功能只能用在一个项目中，所以暂时远程服务器如果还是gitee的话，需要创建许多个项目，swing软件创建的名字需要在gitee中存储为username+项目名字，然后分析的时候必须将第一个+号前面的当作用户名。并且考虑到需要创建很多个项目，所以必须用java实现web上表单的获取等，从而实现在java上直接访问到gitee网站，然后创建项目，即java爬虫登录。

准备使用jsoup爬虫api。但是好像爬虫这里不是很好写，所以可能采用自己搭建的一个git远程仓库。

3.19

尝试用另外一种方法，也就是用阿里云购买的服务器，自己搭建Git的远程仓库。

突然发现码云的API文档，尝试查看如何使用，实在不行就采用阿里云自己购买服务器。

为了不适用密码就可以直接clone和push，需要设置ssh的密匙。服务器配置和ssh密匙参考:

<http://www.findme.wang/share/detail/id/435.html>

<http://www.findme.wang/blog/detail/id/376.html>

之后打算服务器上对应的目录结构为/username/projects. 在创建用户的时候，直接在远程创建一个用户民的目录，之后该用户创建的所有项目都放在目录结构当中。Java使用ssh(Jsch)连接eclipse参考这个网站:

<https://blog.csdn.net/u013256816/article/details/52701563>

之后的学习参考都可以考虑找一些参考论文上去。

现在更新软件到每次注册用户的时候，除了向数据库中写入相应数据之外，还可以直接远程ssh到服务器上创建一个相应的以用户名作为名称的文件夹。

3.20

准备完成创建项目这个按钮，按钮按下之后可以创建一个项目，在数据库的repo中插入数据，创建项目过程需要一个新的小窗口，用于输入项目名称，默认项目都是public的。

界面跳转时容易空指针异常。比如在login里面创建了after\_login的对象window,然后用window打开了after\_login这样的界面，但是after\_login的main函数没有被运行，直接运行的构造函数，所以还需要在login中把after\_login window = new after\_login()中的window交给after\_login，否则在after\_login中dispose()关闭窗口会出现空指针异常。

当前进度:可以创建一个新的项目，并且之后可以在远程服务器git init初始化项目。明天考虑怎么实现多人合作，比如pull 和merge,fork等。

3.21

首先需要将已经登录的个人有哪些项目都展示出来，之后可以考虑给项目加stars的方法，首先没有考虑这一点。想要将内容用jtable展示出来。

数据库现在采取的方式是从一开始Login就打开，直到整个程序结束后才自动结束，所以如果后面插入或者访问数据库出问题，有可能是这个原因。

3.22

后期可以将表格的竖线模糊掉，现在暂时不用去考虑。

Branch功能参考:

<https://www.liaoxuefeng.com/wiki/896043488029600/900003767775424>

因此，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！

由于模拟的过程其实是在远程服务器创建的不同用户名的文件夹，所以后期如果有权限要求的时候，需要单独的一个数据库来实现权限的要求。

最终采取的多人协作方式是:

<https://www.cnblogs.com/onelikeone/p/6857910.html>

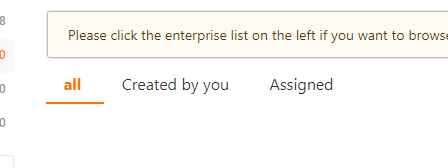
暂时没有采取Github那种pull request的方式。而是采取的直接分工合作的方式。

3.24

先设计一下界面，准备先实现多人协作这个功能。本来想的多人协作是没有pull request,直接克隆到本地，然后后来想到这样可能会有人恶意上传代码，虽然可以回退到某个版本，但是安全考虑还是加上有一个选项，按钮之类的，让项目创建者选择是否每次上传需要一个项目创建本人的确认，关闭选项就是所有人都可以进行代码上传和修改，错了可以使用版本回退回到特定版本，开启选项就是所有操作都需要有项目设计人确认之后才能最终实现代码上传。

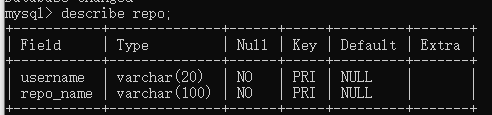
简单的实现方式是(A是项目负责人，B是想要修改者)。B 想要push，只能提交一个push申请，A确认B可以push之后，再告知B可以提交，最好的设计方式是可以直接提交。但是由于每个人的项目存储在自己的服务器上面的，所以必须要将内容先传到服务器上面去，所以最终还是想到了用github的fork之后pull request的方法。但是由于一个项目上传次数会很多，所以还是可以在最开始的时候添加一些信任的用户，让这些用户的上传不会受到阻碍，其他的人如果想要进行代码的修改就必须进行fork相应的仓库到自己的本地，然后修改之后想要合并再进行Pull request.

Pull request的设计可以参考gitee

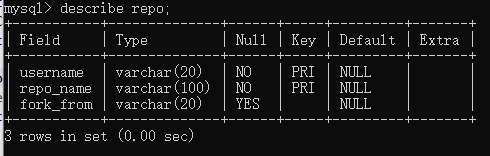


不仅有别人申请的，还要有自己申请的，这里的数据库需要稍微设计一下。

首先实现fork功能，服务器需要新创建一个项目，数据库中的repo进行修改，从



变成:



在project处保存进度。

3.25

完成Fork功能。

由于这个软件完全不让用户用到git命令，所以自己实现Git GUI的基本功能。

准备实现文件下载的功能，或者clone的功能，得先有一个保存的文件路径。

Swing 中保存文件图形界面可以参考下面链接:

<https://blog.csdn.net/weixin_44512194/article/details/94391842>

由于SSH传输的限制，所以还需要让用户添加SSH的公钥和密钥。由于使用的人数比较少，所以暂时采用的是



全部放到authorized\_keys里面进行直接的公钥管理。

3.26

暂时是直接通过密码连接的ssh，之后可以改进通过公钥进行连接。(代码里面)

3.27

暂时git clone 利用ssh协议时是使用的密码登录方式，没有用公钥的方式，争取之后使用公钥的方式。

3.28

尽量完成Repo\_manager界面。

Git add功能暂时只实现add all，分文件add后面改进再来。

给 addfilepattern() 的路径必须相对于工作目录的根。如果一个路径没有指向一个现有的文件，它就被忽略了。

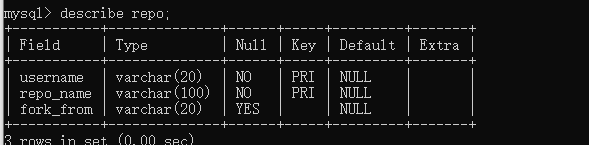
暂时完成了 同步 刷新状态 commit 三个功能，之后还要再进行Push功能。

git.rm.filepattern 参数不能为 “.” 必须指定一个文件一个文件进行改变。

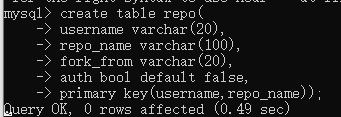
3.29

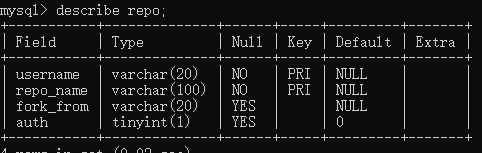
试着完成push.

完成一下加项目权限的功能，就是如果该项目是本人的项目。想要实现这个功能，需要将repo表进行修改，变化如下:



变成:





授权添加只能算是发出邀请，然后另外一个人点确认算是默认fork到文件夹里面。

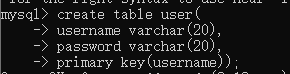
利用数据库中bool类型其实是tinyint，所以可以有除0和 1以外的数字，所以这里准备使用2作为一个人已经被另外一个人授权了，并且fork\_from改为另外一个人的名字，这里需要注意项目里面的名字不要重复。

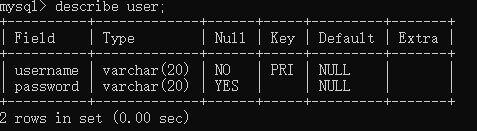
Auth中2表示有授权邀请信息，3表示有pull request信息。

3.30

今天电脑出了点问题，重装电脑。

Mysql数据库另外一个表格user：





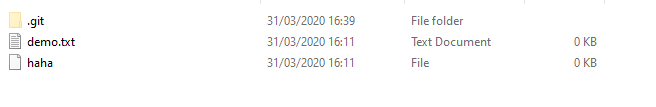
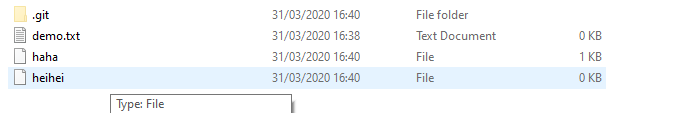
3.31

看了一篇文章，在论文目录中的《基于GIT的软件开发模式探究\_朱守园》。采取的工作流程是其中的2.2 独立远程库之github模式，有pull request和fork。文章表述可以通过git pull实现拉取。

想出了一个实现fork 和 pull request的流程。用git init –bare 和 git clone –bare实现，在服务器上的仓库全部是裸仓库，但是有一个问题就是，如果提交了pull request，另外一个人一直没有回应这个pull request。这一点查看了gitee的操作，如果A fork 了B的项目，A对fork的项目加入一个file A。之后发起pull request，在B发现A发起pull request之前，A又更新了自己fork的项目，这时B如果看pull request会发现，A在pull request发起之后的操作也有记录，如果merge会一并merge过来。从中可以得出几个结论:

1. pull request只是发起通知，并没有将pull request时的状态保存，也就是之后对fork的仓库进行修改，会包含在pull request一起。
2. 解决办法只有用户自己在没有收到别人pull reqest回信的时候，自己创建branch继续完成后续代码，直到pull request接受之后才能在原窗口进行后续操作。

还存在一个fork仓库和贮藏库之间的同步的问题， pull或者fetch实现。试一下pull和fetch的不同之处。在服务端修改之后，下图所示没有变化的(文件数量少的)是fetch，文件数量多的是pull:

所以git fetch不会自动merge，git pull会自动merge.

经过测试之后，gitee的fork之后的仓库进行同步也是直接进行的pull.所以之后准备直接进行pull.

后面是需要修改代码，所有远程服务器的项目名称都应该是projectname.git 有个.git，之前没有，之后都需要进行修改，其中包括ssh操作和数据库操作部分。或者采用另外一种方法，就是重命名fork的项目。但是服务器端的clone操作需要修改代码，变成clone –bare操作。

4.1

数据库中auth显示为0的表示对该项目没有权限。为1的表示对该项目有权限。为2的表示有人请求合作，发送了消息。为3的表示有pull request.

2中repo的fork\_from表示发起邀请的人，username表示被邀请者。

3中fork\_from表示正常的该项目是从谁那里fork过来的。

如果是4表示接受邀请，-4表示拒绝邀请，5表示接受pull request，-5表示拒绝pull request.

由于在查询时要进行字符串分割查询，所以需要对用户名和项目名称进行一些命名限制，用户名只能是英文或者数字或者下划线，不能有空格，项目名称也是

项目重命名。

完成到Message的185行。保存进度。

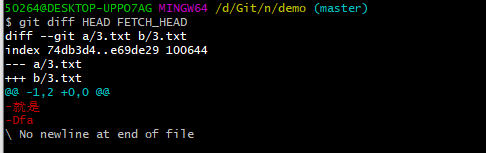
4.2

准备完成pull request部分。之后repo\_manager需要进行历史或者分支可视化，如果不实现上述功能，需要另想一个办法方便branch实现。可视化最好当成附加功能，先实现基本功能。

Push的时候需要注意一下，是本地分支Push到远程分支，需要有界面进行分支选择。

Pull request发起也需要修改，特定分支到特定分支。

Pull request的时候需要git diff一下，暂时输出的是git 程序结果，即

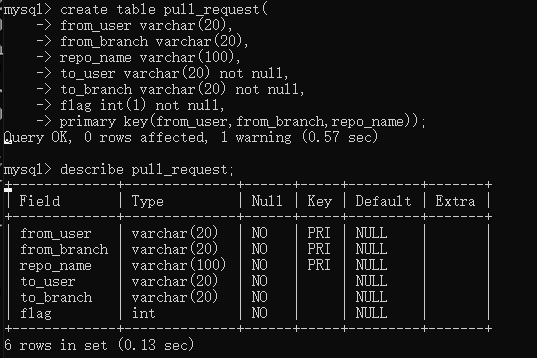


这样，没有具体处理让这个好看。

现在先完成pull request功能，后续完成branch分支功能(在repo\_manager界面)，或者push功能。

到这里发现一个大问题。。Pull request需要进行一个branch到另外一个branch之间的转换，而不是项目到项目转换，这一点也需要在数据库中体现，可能需要新建一个表格。

Pull request的时候像commit一样，必须要添加pull message的信息，所以这个信息还需要进行传递，所以得新建一个pull request的表格。但是可以不需要修改之前对auth位进行的操作。



在SSH()类中添加一个方法用于查询当前有哪些Branch.

由于在远程服务器端git不允许有裸仓库，所以在创建仓库的时候必须加一个选项，是否添加README这样一个文件，如果允许就在服务器端口clone一下再在服务器端口上传，使得裸仓库具有master分支，因为裸仓库默认是不允许有分支的。如果不创建README，就必须将项目上传。所以创建了一个.init，用于clone并且上传内容到服务器端口。

保存进度：暂时写了一半Create\_PR项目，但是有bug,现在debug发现裸仓库没有分支，所以查询分支没有。后续需要完成Create\_PR的比较文件不同和Pull request文件，最后完成Message和本地创建分支的工作，从而完成多人合作。

4.3

可以顺利利用字符串分割读取服务器上的分支。

New\_project已经基本完成，在创建新的项目的时候可以有README选项。

Create\_PR不能正确显示Branch的bug已经被修复。后续完成Create\_PR中 比较不同 功能。

Fork的仓库添加同步origin仓库功能。可以考虑git clone –bare是否可以覆盖原文件。

最后需要对速度进行改进，数据库在一开始就打开，中途不进行任何数据库的连接，服务器连接在一开始就打开，在中途不进行任何服务器的连接，否则软件相应速度太慢。

暂时的pull request等都是没有出错机制。

4.4

数据库暂时是放到本机的，之后需要将数据库全部转到服务器上面去。

接下来工作：

遗留问题：无法克隆远程指定分支，只能默认克隆master分支。所以分支的合并需要在本地完成，需要先fetch远程的分支，再在本地进行merge，之后再将origin远程无用分支删除掉。

Push暂时是让push到和本地同名的分支，比如master->master , dev->dev

还需要做返回到以前版本的功能。

1. 修改pull request接受之后 可以直接push(实现merge) 不用clone之后再push

由于Clone别人的项目再直接上传push是不允许的，所以没有设置这个功能，只能自己push到自己远程服务器上的同名项目，并且是同样的分支。

Git.push.call()暂时没有找到可以指定上传本地某个分支，只能上传当前分支，所以需要配合切换分支来进行指定分支的上传。

4.5

完成了一些和分支有关的基本功能。

4.6

完成fetch功能。默认fetch到FETCH\_HEAD上面，之后再完成merge功能。

Fetch选择fetch的分支，结果在FETCH\_HEAD上面，需要merge。

直接git fetch origin经过测试也是直接fetch的当前分支，比如在dev分支就直接fetch的远程dev分支，在master分支，就直接fetch的远程master分支。

Git clone -–bare是所有分支都有复制下来 在服务器上执行此操作模拟fork

之后的merge模仿GUI版本的界面选项。