Lab 0 实验文档

Lab 0: Hajime to git

实验目标

- 配置好 PyCharm 或 VSCode 的 python 开发环境
- 熟悉命令行工具的使用
- 了解 Git 的基础操作
- 利用 git-cz 工具自动生成符合规范的 commit message
- 能够使用 Github 作为代码托管的平台

实验内容

开发环境配置

安装 VSCode 作为文本编辑器

在这里获取 下载地址.

推荐插件

- Chinese (Simplified) (简体中文) Language Pack for Visual Studio Code: 中文包
- Markdown Preview Enhanced: markdown 的增强渲染
- Python: python 语言支持
- 用 vscode 写 C/C++, 只要配置好编译环境

你还可以安装一些视觉主题插件, 更细节的配置大家可以参考网上教程.

一些方便的操作

常用的快捷键(Cheatsheet):

- 打开内置终端: `Ctrl + ` (Mac: `Command + J`)
- 将代码向上移动: `Alt + ↑`
- 添加单行注释: `Ctrl + /`

在菜单栏选择「文件-自动保存」, 这样可以省的你需要 `Ctrl+s` 手动保存.

使用命令行在当前目录打开 VSCode:

```
code .
```

你可以直接在 VSCode 里从 URL 克隆仓库, 也可以在命令行里使用 `git clone` 命令克隆仓库.

安装 PyCharm

当你获取了 Github 学生认证之后, 可以在 免费教育许可证 中申请 PyCharm 专业版.

如果你没有完成 Github 学生认证, 也可以在 JetBrains 这里直接进行认证: 它的验证流程会比 Github 更快速, 同时简便.

开展实验

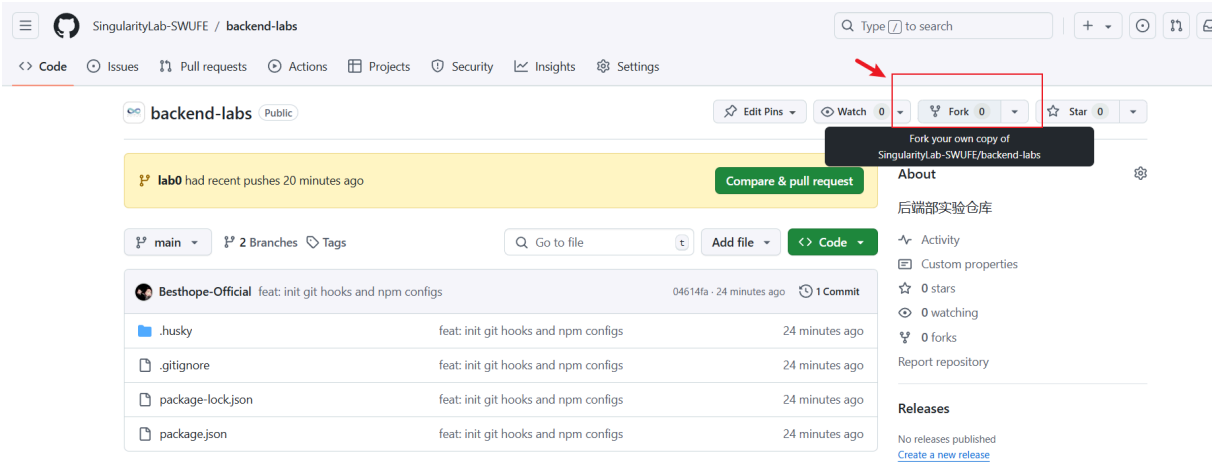
安装 node 环境

安装 node 环境. 根据安装指示进行下去, 它应该就会添加到你的系统变量路径中

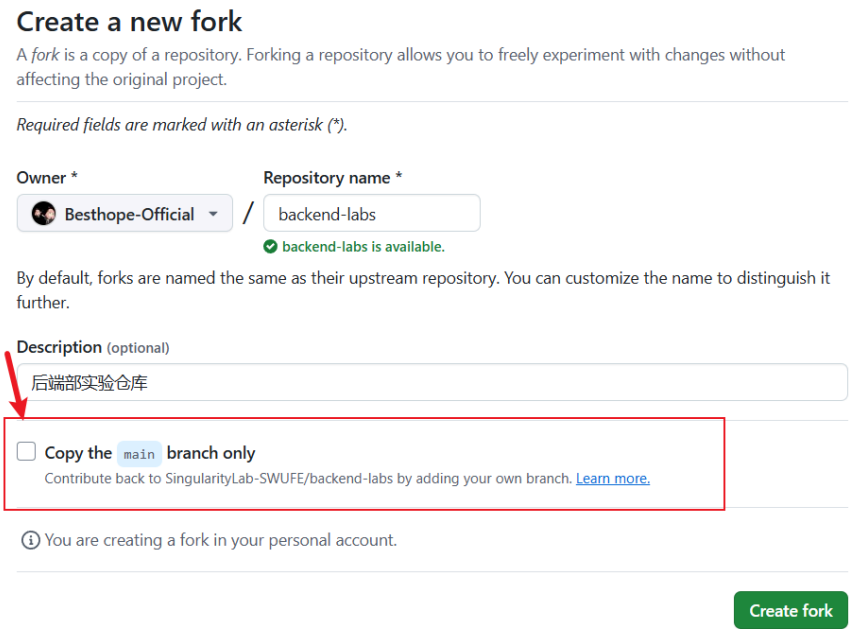
```
$ node -v
v20.18.0
```

克隆开发仓库

进入实验仓库, 点击右上角的 `Fork` 按钮



注意: fork 里将“只复制主分支”的选项关闭



之后使用 `git clone` 命令克隆你 fork 的仓库(使用 HTTPS 地址)到本地.

切换到 `lab0` 分支来开展你的第一次实验:

```
git switch lab0
```

你也可以在 VSCode/PyCharm 里直接通过图形化界面来切换分支.

配置仓库

配置好 npm 镜像源或者开启代理

```
npm config set registry https://registry.npmmirror.com/
```

如果你想检查是否配置成功, 使用 `npm config get registry` 命令.

全局安装 `commitizen`, 它是 `cz-git` 的一个依赖包:

```
npm install -g commitizen
```

这会帮助你之后在其它项目不必再安装 `commitizen` 才能使用 `cz-git`.

注: npm 安装包的过程可能需要一段时间, 请耐心等待, 同时不要进行多余的操作. `Ctrl+C` 可以终止安装过程.

在项目根目录安装好 `husky` 和 `cz-git` 插件.

- husky: 是一个便捷的 Git Hook 工具, 使现代的原生 Git 钩子变得简单
- cz-git: 是一个自动化规范 Git 提交消息的工具.

我们提供了 `package.json` 文件, 你可以直接使用 `npm install` 命令安装依赖:

```
npm install
```

如果你希望手动激活 husky 钩子的话

```
npm run prepare
```

任务 1: 实现 `lab.py`

`lab0` 目录存放着本次实验的内容:

```
lab0
├── images/
├── lab.py
├── README.md
└── test_lab.py
```

- `README.md`: 实验文档
- `lab.py`: 实验代码
- `test_lab.py`: 测试代码

你需要完成对 `lab.py` 中三个函数的编写, 具体要求如下:

- `middle` 返回三个数中第二大的数
- `sum_digits` 返回各数位之和
- `double_eights` 判断一个数是否包含 2 个连续的 8

```
def middle(a, b, c):
    """Return the number among a, b, and c that is not the smallest or largest.
    Assume a, b, and c are all different numbers.
    """
    return ____


def sum_digits(y: int) -> int:
    """
    Sum all the digits of y.
    """
    pass


def double_eights(n: int) -> bool:
    """
    Return True if n has two eights in a row.
    """
    pass
```

这三个函数的实现并不困难(肯定没有你程序设计的题目有挑战).

`lab.py` 的函数有 docstring 更详细的注释; `test_lab.py` 里的测试用例可以帮助你理解函数的输入输出.

最后, 使用 `pytest` (一个 python 的测试框架)对你的代码进行测试:

利用 `pip` 安装

```
pip install pytest
```

然后执行

```
pytest
```

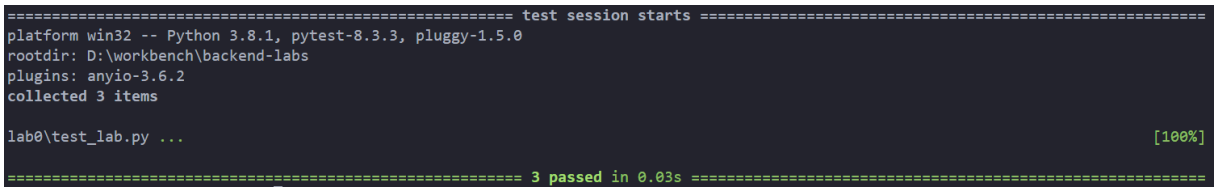


Figure 3: 如果成功, 你会看到终端如下的输出

任务 2: Git 操作

使用 Git 的命令行工具(在终端)完成以下操作:

- 在 `lab0` 分支的基础上新建一个 `lab0-dev` 分支.
- 在 `lab0-dev` 分支上发起两条新的提交.
- 将 `lab0-dev` 分支推送到你 fork 的远程仓库中.
- 将 `lab0-dev` 分支合并到 `lab0` 分支下.
- 将 `lab0` 分支推送到远程仓库.
- 在你的项目远程仓库上(Github), 发起一条 Pull Request 到源仓库的 `lab0` 分支.

要求/提示

显式做出的两次提交都是针对 `lab.py` 文件的修改.

你的第一次提交应该包含:

- 内容: 对 `middle` 函数的修正
- 提交消息格式: `fix` 类型

第二次提交应该包含:

- 内容: 实现剩下的两个函数
- 提交消息格式: `feat` 类型

详细的提交格式在我们的 Git 文档中已经给出.

你可以使用 `git add -p` 将文件分段暂存, 然后使用 `git commit -m "<message>"` 来提交.

你可以使用 `git cz` 来自动生成符合规范的提交消息:

```
(.venv) besthope:~/workbench/backend-labs$ cz
cz-cli@4.3.1, cz-git@1.10.1

? Select the type of change that you're committing: feat:     A new feature
? Denote the SCOPE of this change (optional): empty
? Write a SHORT, IMPERATIVE tense description of the change:
 [Infinity more chars allowed]
 add python lab materials
? Provide a LONGER description of the change (optional). Use "|" to break new line:

? List any BREAKING CHANGES (optional). Use "|" to break new line:
```

如果你的提交信息不符合规范, 则会被 git hook 拦截, 并提示你修改.

推送到远程仓库(origin)会检查最近的两次提交信息中是否包含 `fix` 和 `feat`. 如果不包含, 则会被 Git Hook 拒绝.

第一次提交到远程会提示你设置分支的 upstream, 这是因为 `lab0-dev` 还没有远程的一个引用. 在 push 的时候你需要使用到 `--set-upstream` 参数, 具体你可以直接复制 Git 给出的解决方案. (这也可以在 VSCode 里选择发布分支解决)

将 `lab0-dev` 合并到 `lab0` 分支的方式是 fast-forwarded 的方式, 因为它是对 `lab0` 分支历史的直接延续, 从提交树来看, 它是线性的:

```
$ git log --all --graph --decorate --oneline
* 54add0e (HEAD -> lab0, origin/lab0-dev, origin/lab0, lab0-dev) feat: implement
lab.py
* d5ccfcc fix: fixed middle
* 75654ca docs: add lab docs
* 8bba6be feat: add python lab materials
| * 98e5c42 (origin/main, origin/HEAD, main) docs: add lab manual
|/
* 9a66800 fix(pre-push): use HEAD~~..HEAD in rev-list
* ca993f3 fix: pre-push hook
* 207d679 fix: dealing with case that remote_ref is empty
* 04614fa feat: init git hooks and npm configs
```

我们希望, 通过你的分支操作, 你的 Git 历史是类似于下面这样的:

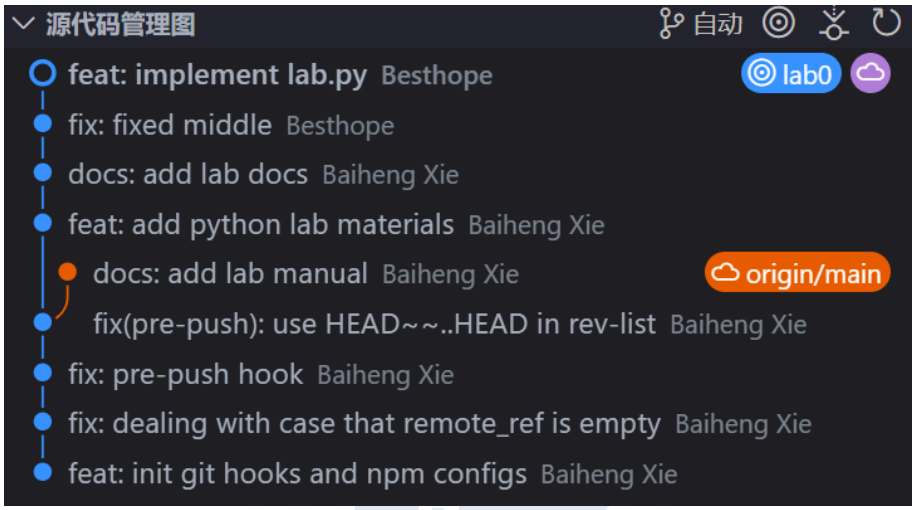


Figure 5: vscode 里的分支图, 图中展示了 lab0 分支的历史树的一个示例

如果你的提交信息不符合规范, 使用下面的命令

```
git push --no-verify
```

来绕开 Git pre-push Hook 的检查.

最后在 Github 仓库里发起 Pull Request 到源仓库的 `lab0` 分支, 并描述你对 `lab.py` 的修改. 类似于

如果你有任何问题, 可以查看上面的实验示例.

Bonus

后端的这些文档也都开源在 Github 上, 如果你有任何补充, 或者有地方纠正(如 typo), 可以按照上面的流程提交 PR 到源仓库.