Project Zero

2025 年 9 月 8 日

1 写在前面

Project Zero 是一个 **不计分，也不强制要求提交**的 Project，同时也不设置 Deadline。尽管如此，我们仍然推荐你完成这个 Project。

这个 Project 的主要目的是：检验同学们是否已经按照 Lab 1 的要求配置好开发环境，以及学习使用 Makefile 工具来提交 Project。

2 环境配置

请参照第一周的 Lab 课件，配置好基于 Linux （或 macOS 原生环境）的开发环境。我们推荐使用 IDEA IntelliJ 作为开发工具，但是我们也支持使用 VSCode 完成所有 Project 任务。

3 Task 1 - Git Repository

打开一个命令行 Shell，使用 `git clone` 命令，从 `https://github.com/sqlab-sustech/CS323-Compilers-2025F-Projects` 克隆 git 仓库。

```
$ ~> git clone https://github.com/sqlab-sustech/CS323-Compilers-2025F-Projects
Cloning into 'CS323-Compilers-2025F-Projects'...
remote: Enumerating objects: 23, done.
remote: Counting objects: 100% (23/23), done.
remote: Compressing objects: 100% (18/18), done.
remote: Total 23 (delta 0), reused 23 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (23/23), 1.87 MiB | 6.62 MiB/s, done.

$ ~> cd CS323-Compilers-2025F-Projects/
```

4 Task 2 - 切换到 project0-base 分支

使用 `git checkout project0-base` 切换到 project0-base 分支，`-base` 表示该分支为我们提供的基础代码。

```
$ ~/CS323-Compilers-2025F-Projects (main)> git checkout project0-base
branch 'project0-base' set up to track 'origin/project0-base'.
Switched to a new branch 'project0-base'

$ ~/CS323-Compilers-2025F-Projects (project0-base)>
```

5 Task 3 - 创建你自己的分支

在 project0-base 分支下，使用 git branch -C my-project0 创建属于你自己的分支，然后使用 git checkout 切换到该分支下面。

```
$ ~/CS323-Compilers-2025F-Projects (project0-base)> git branch -C my-project0
$ ~/CS323-Compilers-2025F-Projects (project0-base)> git checkout my-project0
Switched to branch 'my-project0'
Your branch is up to date with 'origin/project0-base'.

$ ~/CS323-Compilers-2025F-Projects (my-project0)>
```

6 Task 4 - Hello World

在 IDEA IntelliJ 中打开这个文件夹，你将会看到如下的目录结构：

在 Main.java 中输入任何代码，例如 Hello World。点击 main 函数左侧的绿色箭头，运行它。

随后，在 IDEA IntelliJ 自带的 Git 工具里面创建一个 commit，或者使用以下命令创建一个 commit：

```
~/CS323-Compilers-2025F-Projects (my-project0)> git status
On branch my-project0
Your branch is up to date with 'origin/project0-base'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
```

```
        modified:   src/main/java/Main.java

no changes added to commit (use "git add" and/or "git commit -a")

~/CS323-Compilers-2025F-Projects (my-project0)> git add .
~/CS323-Compilers-2025F-Projects (my-project0)> git status
On branch my-project0
Your branch is up to date with 'origin/project0-base'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   src/main/java/Main.java

~/CS323-Compilers-2025F-Projects (my-project0)> git commit -m "my first commit!"
[my-project0 57f16f3] my first commit!
 1 file changed, 1 insertion(+)
```

我们修改了 `Main.java` 文件，在 `git add .` 前，`git status` 将他列在"Changes not staged for commit"下面；在 `git add .` 后（`.` 表示当前目录，即 `git add` 当前目录下所有文件），它被列在了"Changes to be committed" 里面。

然后，我们使用 `git commit` 创建了一个 commit，表示有一个文件被更改，有一行代码插入。

7 Task 5 - 关联自己的仓库

上述 sqlab-sustech 的仓库是由教学团队控制的、用于释放 Project 基础代码的仓库，你无法向该仓库写入。

在 GitHub 或其他任何平台上，创建一个**私有的**仓库，用来存放你的编译原理课上所有的 Project。注意不要添加任何文件。

使用 `git remote add` 命令，在你本地的 Git 仓库中添加远程仓库的链接：

```
~/D/CS323-Compilers-2025F-Projects (my-project0)> git remote add mygithub git@github.com:
yuk1i/my-cs323.git
~/D/CS323-Compilers-2025F-Projects (my-project0)> git remote -v
mygithub        git@github.com:yuk1i/my-cs323.git (fetch)
mygithub        git@github.com:yuk1i/my-cs323.git (push)
origin  https://github.com/sqlab-sustech/CS323-Compilers-2025F-Projects (fetch)
origin  https://github.com/sqlab-sustech/CS323-Compilers-2025F-Projects (push)
```

随后，将你新创建的 commit 推送到你自己的远程仓库中：

```
~/D/CS323-Compilers-2025F-Projects (my-project0)> git push mygithub my-project0 --set-upstream
Enumerating objects: 29, done.
Counting objects: 100% (29/29), done.
Delta compression using up to 16 threads
Compressing objects: 100% (21/21), done.
Writing objects: 100% (29/29), 1.87 MiB | 5.76 MiB/s, done.
Total 29 (delta 3), reused 21 (delta 1), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), done.
To github.com:yuk1i/my-cs323.git
 * [new branch]      my-project0 -> my-project0
branch 'my-project0' set up to track 'mygithub/my-project0'.
```

然后，你应该可以在 GitHub 上看到代码了。

8 Task 6 - 合并来自教学团队的代码更新

编译原理课程的 Project 是连续的，你需要在你上一个 Project 的基础上完成下一个 Project。教学团队会给出一部分的框架代码，你需要在你上个 Project 的代码上来合并下一个 Project 的初始代码。

在本次 Project Zero 中，我们假设你自己完成了 Project Zero 并 commit 了自己的代码（即上述的 Hello World）；现在，Project 1 (Fake) 释出了，你需要在你的 `my-project0` 上创建 `my-project1` 分支并合并来自教学团队的基础代码（`project0-fake1`）。

在实际情况下，以 Project 1 和 Project 2 举例，教学团队提供的代码位于 `project1-base` 分支，你从该分支分叉得到了你自己的分支 `my-proj1-balabala` 并提交了自己的代码。在 Project 2 释出后，你需要合并 `project2-base` 分支，其中包含 Project 2 的基础代码。

首先,在你的本地仓库中,切换到 `my-project0` 分支下,并创建一个为了 Project 1 (Fake) 的分支 `my-project1-fake`，最后，切换到这个新分支上。

```
~/D/CS323-Compilers-2025F-Projects (my-project0)> git checkout my-project0
Already on 'my-project0'
Your branch is up to date with 'mygithub/my-project0'.

~/D/CS323-Compilers-2025F-Projects (my-project0)> git branch -C my-project1-fake
~/D/CS323-Compilers-2025F-Projects (my-project0)> git checkout my-project1-fake
Switched to branch 'my-project1-fake'
Your branch is up to date with 'mygithub/my-project0'.
```

然后，使用 `git fetch origin` 拉取来自教学团队的代码变更：

```
~/D/CS323-Compilers-2025F-Projects (my-project1-fake)> git fetch origin
remote: Enumerating objects: 12, done.
remote: Counting objects: 100% (12/12), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 7 (delta 1), reused 7 (delta 1), pack-reused 0 (from 0)
Unpacking objects: 100% (7/7), 699 bytes | 699.00 KiB/s, done.
From https://github.com/sqlab-sustech/CS323-Compilers-2025F-Projects
 * [new branch]      project0-fake1 -> origin/project0-fake1
```

Note: 在你按照本文顺序实际上运行这条命令时，你应该不会看到"[new branch]..." 等等提示，因为你在第一次 `git clone` 时就已经得到了这个 branch。而在后续正式的 Project 中，你应该才会看到这条提示。

然后，使用 `git branch --remotes --all` 列出所有远程分支，`origin/` 下的分支即是来自教学团队的初始代码。

```
~/D/CS323-Compilers-2025F-Projects (my-project1-fake) > git branch --remotes --list
  mygithub/my-project0
  origin/HEAD -> origin/main
  origin/base
  origin/main
  origin/project0-base
  origin/project0-fake1
```

最后，使用 `git merge origin/project0-fake1` 在 `my-project1-fake` 上合并远程分支。

```
~/D/CS323-Compilers-2025F-Projects (my-project1-fake)> git merge origin/project0-fake1
Auto-merging src/main/java/Main.java
CONFLICT (content): Merge conflict in src/main/java/Main.java
Automatic merge failed; fix conflicts and then commit the result.
```

git 会尝试自动合并,如果出现 CONFLICT,则表示你的分支(my-project1-fake)和要合并进来的分支(origin/project0-fake1)都修改了这个文件，你需要手动解决 Conflicts。在 Main.java 中，会有以下内容：

```
public class Main {
    public static void main(String[] args) {
<<<<<<< HEAD
        System.out.println("Hello World!");
=======
        AnExampleClass exmaple = new AnExampleClass("qwq");
        System.out.println(exmaple);
>>>>>>> origin/project0-fake1
    }
}
```

上面的内容（«< HEAD）即你的修改，下面的内容（»> origin/xxx）即别人的修改。

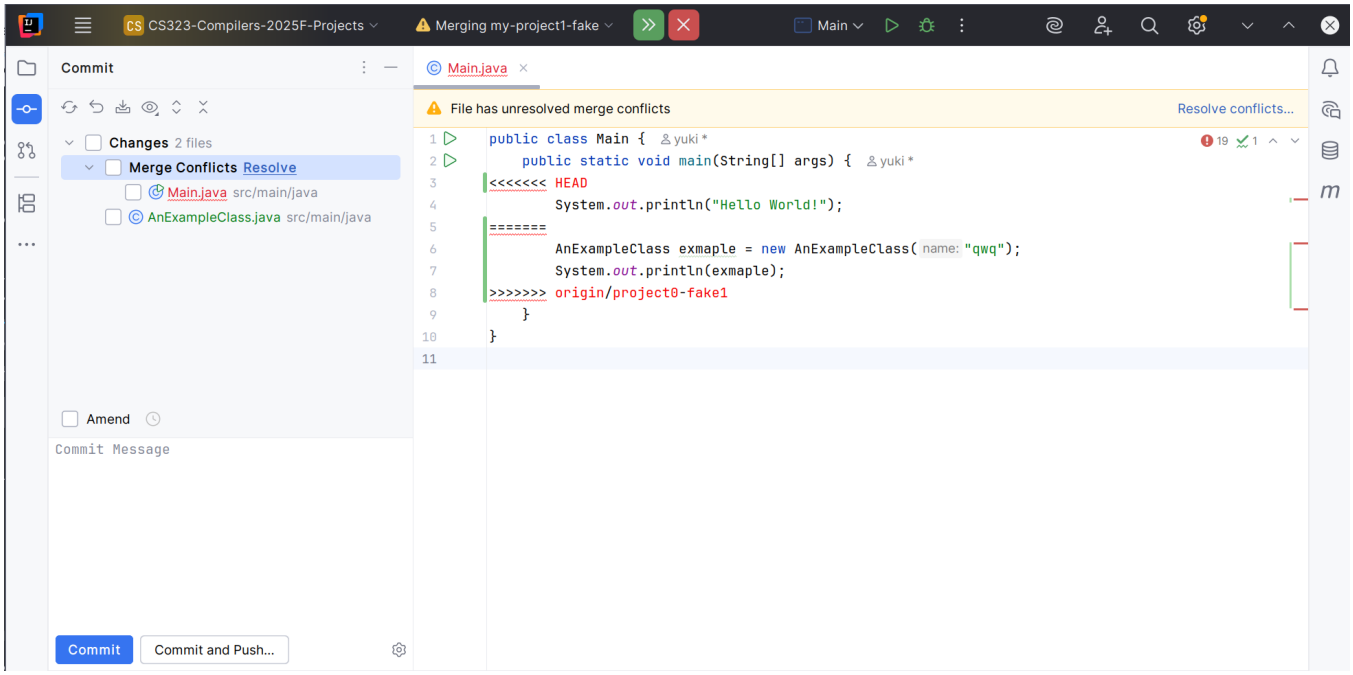
我们要处理 Conflicts，Git 的手册有以下推荐：HOW TO RESOLVE CONFLICTS

After seeing a conflict, you can do two things:

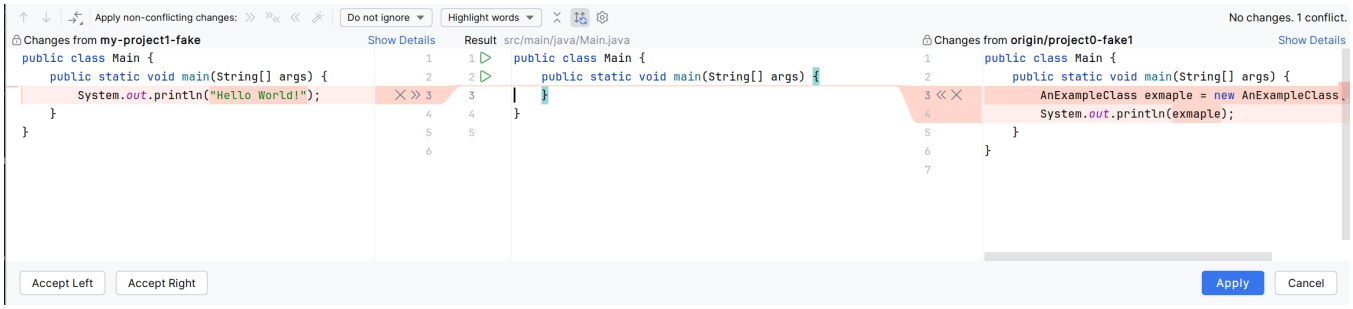
- Decide not to merge. The only clean-ups you need are to reset the index file to the HEAD commit to reverse 2. and to clean up working tree changes made by 2. and 3.; `git merge --abort` can be used for this.
- Resolve the conflicts. Git will mark the conflicts in the working tree. Edit the files into shape and `git add` them to the index. Use `git commit` or `git merge --continue` to seal the deal. The latter command checks whether there is a (interrupted) merge in progress before calling git commit.

总之，要解决冲突的话，你需要修改文件、使用 `git add` 标记该文件已被解决；在解决完所有文件后，你需要使用 `git commit` 或 `git merge --continue` 来完成 merge。

在实践上，我们推荐使用 IDEA 来解决冲突。在 IDEA 中打开 Commit 面板，你会清晰的看到 Changes 下有一大分类 Merge Conflicts，里面即是所有有冲突的文件。



点击 Merge Conflicts 后的 Resolve，选择某个文件开始处理冲突，你会看到如下窗口。箭头表示 Accept 接受更改，叉号表示 Reject 拒绝更改，中间的代码是最终的结果。



在完成修改后，点右下角 Apply 即可完成这个文件。

在你处理完所有冲突后，在 IDEA 的 Commit 面板中输入 Commit Message，然后点击 Commit 按钮，完成 commit。

至此，你就完成了合并，你在你的 Project0 的基础上合并了 Project 1 (Fake) 的代码，可以开始继续下一个 Project 了。

本学期所有的 Project 基础代码都将被公布在 `https://github.com/sqlab-sustech/CS323-Compilers-2025F-Pro`

9 Task 7 - Submission

Makefile 是一个用于自动化构建和管理项目的工具。它通过定义一系列规则和依赖关系，使得开发者可以方便地编译、链接和安装软件。在我们的多个 Project 中，它不会扮演什么重要的角色，你可以将它视为一个大号的脚本管理器。

在项目根目录中，输入 `make handin` 开始提交流程。在提交代码时，你需要保证：

- 你的项目是在一个 Git 仓库里面。
- 所有需要提交的文件都被 Git 追踪。
- 你已经 commit 了所有更改。

若你的项目目前尚未满足以上条件，脚本会提示你。

注意：你所提交的内容仅包含被 Git 追踪的文件。如果有文件未被 Git 追踪，我们不会在你的提交中收到这些文件。例如：你修改了被 `.gitignore` 忽略的文件，这些文件通常是 ANTLR 生成的文件，你不应该修改它们。

在第一次提交时，`make handin` 会要求你输入一个 API Key。我们会在你选课后将该 API Key 私密地发送给你。若你尚未收到，请邮件联系我们。请注意：**API Key 代表你个人的身份，一定不要将它交给他人。**

随后，脚本会将你的代码打包发送到教学团队的服务器上，你将会收到如下信息：

```
~/D/CS323-Compilers-2025F-Projects (my-project1-fake)> make handin
  main
  my-project0
* my-project1-fake
  project0-base
You are going to hand-in the current branch 'my-project1-fake'. Is it correct? [y/N] y
A compressed tarball of your submission is generated at Project0-handin.tar.gz

Enter your API key. You should receive it by email or by blackboard.
Please enter your API key: 12345678AAAAAAAABBBBBBBBCCCCCCCC

================================
You just finished a submission for Project XXX.

Submission ID: 12345678_97d83ff858f14fe599bfcd9a40411d67
SHA1: eca140f4eb295d75623d35f7ecf5c6cbb314fbda

Please upload this message to blackboard
if you want to make it as your final submission.
================================
```

我们将使用你在 Blackboard 上最后一次提交的该消息作为评分所用的代码版本。