



中国海洋大学
OCEAN UNIVERSITY OF CHINA

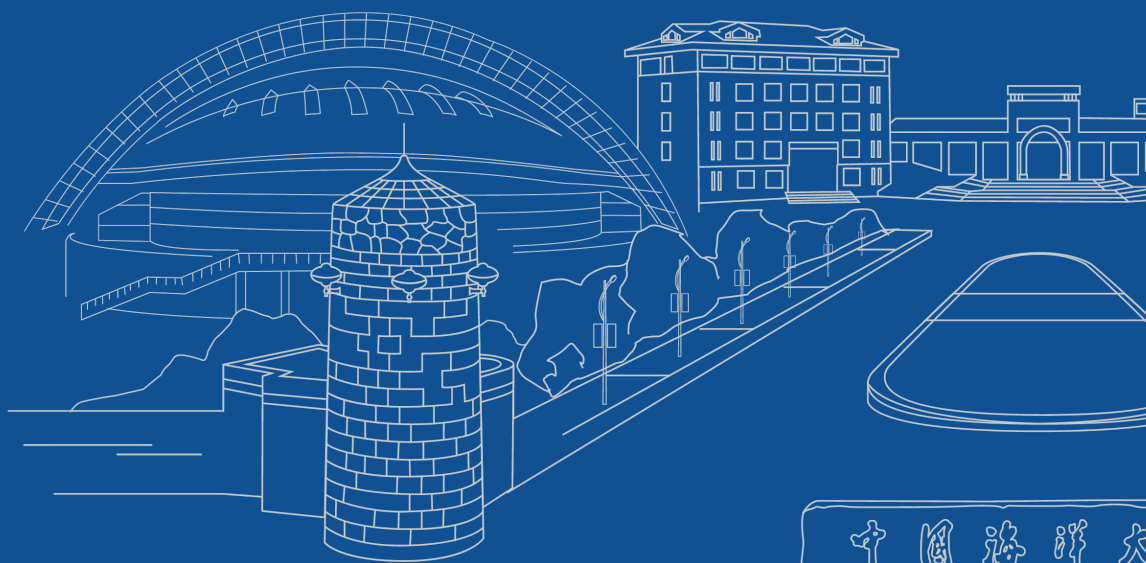
系统开发工具基础第一次实验报告

了解实践 Git 版本控制 学习使用 \LaTeX 文档编辑

于景一

23090032047

信息科学与工程学部
计算机学院
计算机科学与技术专业



中国海洋大学

JStar, August 2024



中国海洋大学
OCEAN UNIVERSITY OF CHINA

系统开发工具基础第一次实验报告

了解实践 Git 版本控制 学习使用 \LaTeX 文档编辑

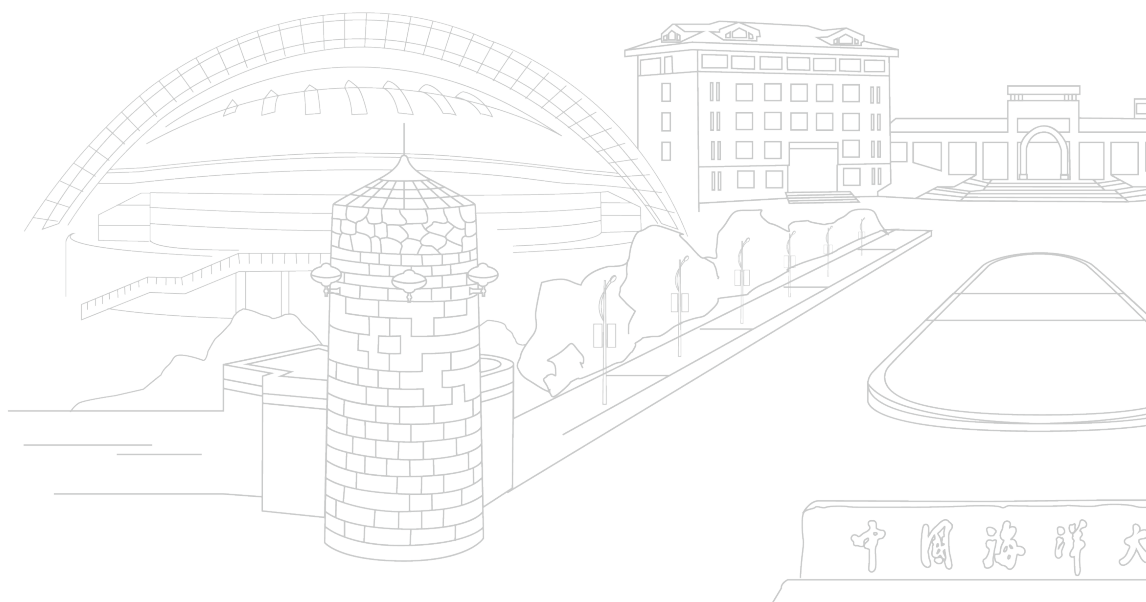
于景一

23090032047

指导老师: 周小伟

系统开发工具基础, 计算机学院

信息科学与工程学部
计算机学院
计算机科学与技术专业



JStar, August 2024

概述

本实验报告使用由于景一制作的 \LaTeX 模板完成。关于此模板的信息,您可以前往GitHub模板仓库具体了解。¹²³

本实验报告是系统开发工具基础课程的第一次实验报告,主要关于使用 **Git** 进行版本控制,以及掌握 \LaTeX 模板的基本使用方法,总结个人经验,记录心得体会。

关于 **Git** 的部分,我们主要学习了 **Git** 的基本概念,并通过实际操作,掌握了它的基本使用方法,包括但不限于初始化仓库、添加文件、提交更改、查看历史、分支管理等。特别的,由于我们在程序设计基础实践课程进行了小组协作(使用 **GitHub** 进行代码托管和版本管理),我们在实操中练习了使用 **Git** 进行远端操作、版本控制,以及如何解决冲突等问题。

至于 \LaTeX 部分,我在制作个人模板、完成计算机工程伦理的课程报告⁴的过程中,对 \LaTeX 有了更深入的了解。在实操中,我学习了 \LaTeX 的基本语法,包括但不限于文档结构、文本格式、数学公式、插图表格、参考文献等。例如本模板,就是学习的成果,使用交叉编译,采用 $\text{XeLaTeX} \rightarrow \text{BibTeX} \rightarrow \text{XeLaTeX} \rightarrow \text{XeLaTeX}$ 的编译顺序,支持中文、英文混排,支持引用文献,兼具较好的兼容性和美观性。

NOTE

本模板仍非特别完善,仍在活跃维护中,接下来将会进一步提升本地化水平,欢迎您提出宝贵意见。

¹ 或您可直接搜索 GitHub 账号 `@jstar0` 了解更多

² 您请注意,本模板基于 LPPL v1.3c 分发,本项目在原模板Polytechnic University of Leiria: LaTeX Thesis Template的基础上进行了合法地大量二改,包括但不限于自定义风格、中文化支持、样式重定义、功能增加等。

³ 模板提供两种样式,一种为学术论文样式,另一种为实验报告样式,具体区别请检查 GitHub 仓库上的两个分支。

⁴ 如果您对此感兴趣,可以点击此链接进行预览。

目录

目录	1
1 使用 Git 进行版本控制	2
1.1 理论基础	2
1.1.1 Git 的基本模型	2
1.1.2 整理 Git 的基本操作	2
1.2 实践操作	3
1.2.1 初始化一个仓库	3
1.2.2 添加、提交和查看	4
1.2.3 推送到远端、拉取最新版本	5
1.2.4 分支操作	6
1.3 总结与感悟	7
2 使用 L^AT_EX 进行文档编辑	8
2.1 基本常识	8
2.2 基本结构	9
2.3 设置文档、调节字体、字号等	10
2.4 文本格式控制	10
2.5 命令和宏包	10
2.6 样式设定	11
2.7 插图表格	11
2.8 引用参考文献	12
2.9 总结	12

使用 Git 进行版本控制

NOTE

本人多年前就接触过 Git 的使用方法，相关记录可以查看本人的 GitHub 账号 (@jstar0)，并用它进行过代码托管、持续集成（自动化构建和部署）、项目管理、提 PR、SaaS 对接等操作。因此在本报告中，对 Git 相关操作描述可能相对简略。当然，经过系统学习，我对 Git 的掌握更加熟练和深入了。

1.1 理论基础

1.1.1 Git 的基本模型

Git 使用对象和引用的方式来管理版本控制。对象是 Git 中最基本的单位，包括三种类型：**blob**、**tree** 和 **commit**。其中，blob 是文件内容，tree 是目录结构，commit 是提交信息。而引用是指向对象的指针，包括分支、标签等。通过对象和引用的方式，实现了 Git 的基本模型，包含三个基本区域：**工作区**、**暂存区**和**版本库**。工作区是指用户进行编辑的区域，暂存区是指用户准备提交的地方，版本库用于存储历史记录。

此部分参考了 ([计算机教育中缺失的一课, 2020](#)) 的概念，仅作简述。

1.1.2 整理 Git 的基本操作

我们将远程操作和本地操作分开研究，这是基于我们实用性的考量。在远程操作中，我们主要关注**克隆**、**推送**、**拉取**等操作，而在本地操作中，我们主要关注**初始化库**、**添加**、**提交**、**查看**、**分支**等操作。我们给出两个表格[Table 1.1](#)和[Table 2.2](#)

远程操作：

表 1.1: Git 远程操作一览表

操作名	命令	简介
克隆	<code>git clone <url></code>	从远程仓库克隆到本地
推送	<code>git push</code>	将本地提交推送到远程仓库

表 1.1: Git 远程操作一览表（续）

操作名	命令	简介
拉取	git pull	从远程仓库拉取最新版本
获取	git fetch	从远程仓库获取最新版本
指定远程	git remote	查看远程仓库
分支	git branch	查看分支

本地操作：

表 1.2: Git 本地操作一览表

操作名	命令	简介
初始化	git init	初始化一个 Git 仓库
添加	git add <file>	将文件添加到暂存区
提交	git commit -m <message>	将暂存区的文件提交到版本库
查看	git status	查看当前状态
分支	git branch	查看分支

1.2 实践操作

1.2.1 初始化一个仓库

要初始化一个仓库，我们可以选择直接从本地初始化或者从远程仓库克隆。从本地初始化的操作如下：

```
1 git init
```

代码 1.1: INITialize(初始化) 一个 Git 仓库

而若要从云端创建一个仓库并克隆到本地，以 GitHub 为例，我们先在云端创建一个仓库，如下图：

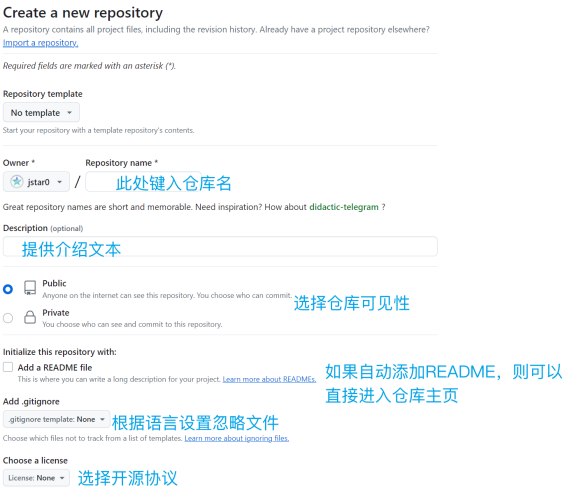


图 1.1: 在 GitHub 上创建一个仓库

然后，我们可以使用 `git clone` 命令将仓库克隆到本地：

```
1 git clone <url>
```

代码 1.2: CLONE(克隆) 一个 Git 仓库

1.2.2 添加、提交和查看

在仓库中，最基本的操作就是，添加文件、提交文件，以及查看当前状态。添加文件会将文件添加到暂存区，而提交文件会将暂存区的文件提交到版本库。添加文件的操作如下：

```
1 git add <file>
```

代码 1.3: ADD(添加) 文件到暂存区

如需提交所有文件，可以使用 `*` (通配符)，或者在不使用此指令而是使用 `git commit -a` 命令。但注意，使用此指令或许会造成不希望的结果。提交文件的操作如下：

```
1 git commit -m <message>
```

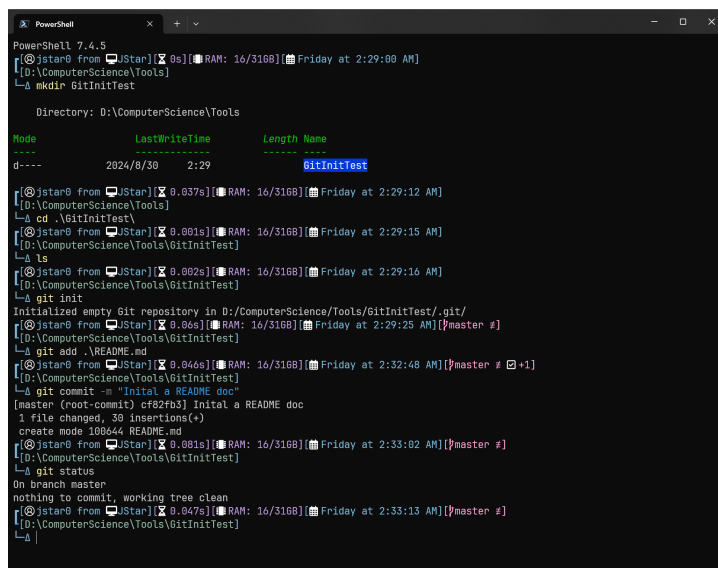
代码 1.4: COMMIT(提交) 文件到版本库

注意，上述指令中 `-m` 参数可以留空，不过后期会自动打开一个文件要求输入 Commit 信息。查看当前状态的操作如下：

```
1 git status
```

代码 1.5: STATUS(状态) 查看当前状态

上述操作是 Git 中最基本的操作，我们在本地创建一个仓库作为示例，如下图：



```
PowerShell 7.4.5
[0:~jstar0 from ~][X 0s][RAM: 16/3168][Friday at 2:29:00 AM]
[D:\ComputerScience\Tools]
~> mkdir GitInitTest

Directory: D:\ComputerScience\Tools

Mode                LastWriteTime         Length Name
----                -
d-----            2024/8/30          2:29             GitInitTest

[0:~jstar0 from ~][X 0.037s][RAM: 16/3168][Friday at 2:29:12 AM]
[D:\ComputerScience\Tools]
~> cd .\GitInitTest\
[D:\ComputerScience\Tools\GitInitTest]
~> ls
[D:\ComputerScience\Tools\GitInitTest]
~> git init
[D:\ComputerScience\Tools\GitInitTest]
initialized empty Git repository in D:\ComputerScience\Tools\GitInitTest\.git\
[0:~jstar0 from ~][X 0.06s][RAM: 16/3168][Friday at 2:29:25 AM][?master #]
[D:\ComputerScience\Tools\GitInitTest]
~> git add .\README.md
[D:\ComputerScience\Tools\GitInitTest]
~> git commit -m "Initial a README doc"
[0:~jstar0 from ~][X 0.046s][RAM: 16/3168][Friday at 2:32:48 AM][?master #]
[D:\ComputerScience\Tools\GitInitTest]
~> git status
[0:~jstar0 from ~][X 0.081s][RAM: 16/3168][Friday at 2:33:02 AM][?master #]
[D:\ComputerScience\Tools\GitInitTest]
~>
nothing to commit, working tree clean
[0:~jstar0 from ~][X 0.047s][RAM: 16/3168][Friday at 2:33:13 AM][?master #]
[D:\ComputerScience\Tools\GitInitTest]
~>
```

图 1.2: 本地操作创建、提交示例

如果要查看提交的历史记录，可以使用 `git log` 命令。此命令会列出所有的提交记录，包括提交者、提交时间、提交信息等。如果要查看某个文件的提交历史，可以使用 `git log <file>` 命令。

一般的,我们都会使用`.gitignore`文件来忽略一些不必要的文件,如`.DS_Store`、`.vscode`等。我们可以在`.gitignore`文件中添加这些文件, 比如我们:

```
1 echo ".vscode/" >> .gitignore
```

代码 1.6: 添加`.vscode`文件夹到`.gitignore`

就可以忽略`.vscode`文件夹了。此处忽略的意思是, Git 不会将这些文件夹或文件添加到版本库中。具体的正则、匹配写法, 可以参考Git 官方文档。

1.2.3 推送到远端、拉取最新版本

在本地操作完成后, 我们可以将本地的提交推送到远程仓库, 或者从远程仓库拉取最新版本。

通常, 我们直接编辑从云端 clone 下来的仓库, 在这种情况下的拉取和推送操作如下:

```
1 git pull # 拉取最新版本
2 git push # 推送到远程仓库
```

代码 1.7: *PULL*(拉取) 和 *PUSH*(推送) 操作

然而, 如要将本地的仓库推送到远程仓库, 我们需要先指定远程仓库, 然后再推送。指定远程仓库的操作如下:

```
1 git remote add origin <url>
2 git remote -v # 验证远程仓库
```

代码 1.8: *REMOTE*(远程) 添加远程仓库

然后, 我们可以推送到远程仓库:

```
1 git push -u origin master # 关联远程仓库分支并推送
```

代码 1.9: *PUSH*(推送) 到远程仓库

其中, `-u` 参数表示 *upstream*, 即将本地的 `master` 分支与远程的 `master` 分支关联起来。这样, 我们在后续的推送和拉取操作中, 就可以省略 `origin master` 参数了。

特别的, 我遇到的问题是, 如果本地未配置 SSH 密钥, 那么在推送时会出现 `Permission denied` 的错误。这时, 我们需要配置 SSH 密钥, 参考GitHub 官方文档。

使用 Git Bash 进行如下操作:

```
1 ssh-keygen -t rsa -b 4096 -C "MyEmail@example.com" # 生成 SSH 密钥
2 eval $(ssh-agent -s) # 启动 ssh-agent
3 ssh-add ~/.ssh/id_rsa # 添加 SSH 密钥
4 # 然后将 SSH 密钥添加到 GitHub 账号中
5 cd ~/.ssh
6 cat id_rsa.pub # 查看公钥
```

代码 1.10: 配置 SSH 密钥

此时再行推送，一切正常。

1.2.4 分支操作

分支是一个相当重要的概念。我们可以通过分支来实现并行开发，在不同的分支上进行不同的开发，然后再合并到主分支上。

首先我们提供分支相关的基本操作。

```
1 git branch # 查看目前的所有分支
2 git branch <branch-name> # 创建一个分支
3 git checkout <branch-name> # 切换到一个分支
4 git merge <branch-name> # 从 <branch-name> 合并到当前分支
5 git branch -d <branch-name> # 删除 <branch-name> 分支
```

代码 1.11: BRANCH(分支) 操作

而在实际操作中，如果要快速创建并切换分支，可以使用如下命令：

```
1 git checkout -b <branch-name> # 创建并切换分支
```

代码 1.12: BRANCH(分支) 创建并切换分支

一般的，我们会在 `master` 分支上创建一个新的分支，然后在新的分支上进行开发。开发完成后，我们可以将新的分支合并到 `master` 分支上。

如果遇到冲突，我们需要手动解决冲突。此时，Git 会在冲突的文件中标记出冲突的地方，我们需要手动解决这些冲突。解决冲突后，我们需要将文件标记为已解决，然后再提交。

而如果我此刻在处理一些文件，突然遇到紧急的问题，需要切换到 `master` 分支，可以使用 `git stash` 命令，将当前的工作区保存到 `stash` 中，然后切换到 `master` 分支。在处理完紧急问题后，我们可以使用 `git stash pop` 命令，将 `stash` 中的工作区恢复。这个过程称作**贮藏**

如果要**清理**掉所有未被跟踪的文件，可以使用 `git clean -f` 命令，以清理掉所有未被跟踪的文件，包括 `.gitignore` 中忽略的文件。如果要清理掉所有未被跟踪的文件和文件夹，可以使用 `git clean -fd` 命令。具体参见 ([Git 官方文档, 2nd Edition](#)[b])。

下面给出一个分支操作的实例：(见??)

```
1 # 目前在 master 分支上
2 git branch
3 # * main
4 git branch hotfix
5 git checkout hotfix
6 # Switched to branch 'hotfix'
7 # 或者直接使用 git checkout -b hotfix
8 git status
9 # On branch hotfix
10 # nothing to commit, working tree clean
```

```
11      # 我们发现此时已经在 hotfix 分支上了, 对 hotfix 分支进行一些修改
12      echo "some changes" > hotfix.txt
13      git add .\hotfix.txt
14      git commit -m "Add hotfix.txt"
15      # [hotfix 82b935a] Add hotfix.txt
16      # 1 file changed, 1 insertion(+)
17      # create mode 100644 hotfix.txt
18      git checkout main
19      # Switched to branch 'main'
20      git merge hotfix
21      # Updating 57433c9..82b935a
22      # Fast-forward
23      # hotfix.txt | 1 +
24      # 1 file changed, 1 insertion(+)
25      # create mode 100644 hotfix.txt
26      # 我们发现, hotfix 分支已经合并到 main 分支上了
27      # 这只是最简单的情况, 两个分支并没有文件冲突
      ↪ (即hotfix分支中的文件和main分支中的文件在分支后没有同时被修改)
```

代码 1.13: Git 分支操作示例其一

进行 merge 操作前后的区别如下图:

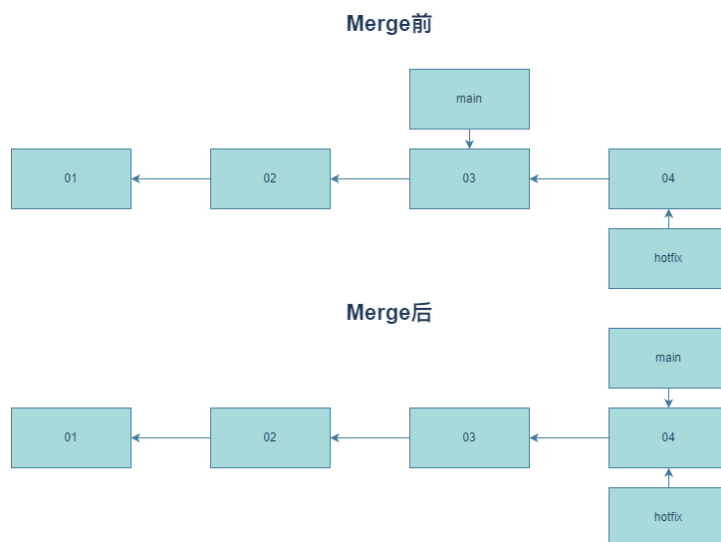


图 1.3: 分支合并前后的区别

现实情况往往更加复杂, 如果在分支出现后, 两者同时修改了同一个文件, 就会出现冲突。此时, 我们需要手动解决冲突, 然后再提交。解决冲突可以是手动修改文件, 也可以使用 `git mergetool` 命令等, 具体参见 ([Git 官方文档, 2ndEdition\[a\]](#))。

1.3 总结与感悟

Git 是非常强大的版本控制工具, 在实际开发中, 我们通过使用 Git 来进行版本控制, 实现并行开发、版本回退、代码合并等操作。实际上, VS Code 也提供了强大的 Git 支持, 在使用 VS Code 进行代码编辑时, 也可以体会这种带 GUI 界面的便捷性。

通过本次实践, 我对 Git 的操作有了更深入的了解, 包括初始化仓库、添加、提交、查看、推送、拉取、分支操作等。工欲善其事, 必先利其器。如果能熟练运用 Git, 我们能够更好地维护代码, 提高开发效率, 减免错误发生。

使用 L^AT_EX 进行文档编辑

2.1 基本常识

L^AT_EX 是一种基于 T_EX 的排版系统，提供了一种类似于标签语言的文档编辑方式，我们通过源代码来控制文档的排版效果，此外，通过命令则可以控制文档的格式、结构、样式等，预定义好了这些设定（或使用编写好的模板），我们便不需要关心具体的排版细节。L^AT_EX 的结构严谨性及对公式表达的良好支持，使它科学、学术和出版等领域应用广泛。

正如概述中所述，在对 L^AT_EX 的实践中，我深入了对该语言的认知。在编辑器选择上，我偏向于使用本地编译¹，采用 TeX Live 引擎，利用 VS Code 进行。

本章所有对 L^AT_EX 的解释，都出自于我逐步积累的经验（例如本文章的写作就积累了不少经验）。我将基本概括 L^AT_EX 的基本知识，包括基本结构、命令和宏包 (Packages)、样式设定、插图表格、参考文献等。

对本模板的维护历史，请您前往我的GitHub 仓库查看。下面提供 Commit 记录截图：

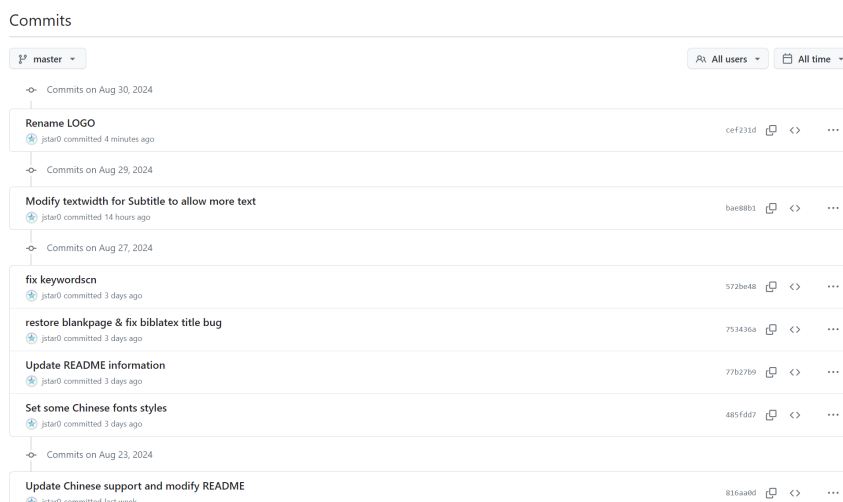


图 2.1: 模板维护历史

¹ Overleaf 提供的在线编辑器常出现编译时间过长问题，可能是由于我的模板较为复杂，另外免费版无法托管至 GitHub，故而弃用。在本地编写编译更方便，还可以托管到 GitHub 库中进行版本控制。

2.2 基本结构

\LaTeX 的文档结构主要包括**导言区**和**正文区**，导言区用于设置文档的格式、样式、引入宏包等，正文区用于书写文档内容。导言区一般以 `*.cls` 或 `*.sty` 文件为主，正文区则以 `*.tex` 文件为主。在简单的文档中，我们可以将导言区和正文区写在同一个文件中，但在复杂的文档中，我们可以将导言区和正文区分开，甚至将正文区分为多个文件，通过`\input{}`或`\include{}`命令引入。²这种解耦合的方式，使得文档结构更加清晰，易于维护。

对我的模板进行 `tree /f` 命令，可以发现：

```

1      D:.\
2      |  .gitignore
3      |  MainPage.pdf
4      |  MainPage.tex
5      |  README.md
6      |  ThesisTemplate.cls
7      |
8      --Assets
9      --Bibliography
10     |    Bibliography.bib
11     |
12     --Chapters
13     | |  00-Abstract.tex
14     | |  01-Git.tex
15     | |  02-LaTeX.tex
16     | |
17     | --Annexes
18     --Figures
19     | |  git-branch-1.png
20     | |  git-local-operations.png
21     | |  github-create-repo.png
22     | |
23     | --Theme
24     |    Back-Page-BG.pdf
25     |    Cover-BG.pdf
26     |    Front-Page-BG.pdf
27     |    OUC-Logo-B.pdf
28     |    OUC-Logo-W.pdf
29     |
30     --Matter
31     |    00-Cover.tex
32     |    01-FPage.tex
33     |    02-Declaration.tex
34     |    03-Acknowledgements.tex
35     |    04-Glossary.tex
36     |    05-Acronyms.tex
37     |    08-BPage.tex
38     |
39     --Variables
40     |    Variables.tex
41     |
42     --minted-MainPage
43     ....pygtex (省略)

```

代码 2.1: 模板文件结构

² 本模板就是将导言区和正文区分开的，导言区在 *ThesisTemplate.cls* 中，正文区在 *MainPage.tex* 中。而正文区又引用了其他多个 `*.tex` 文件。

2.3 设置文档、调节字体、字号等

我们可以通过`\documentclass{}`命令设置文档的类型,例如 *article*、*report*、*book* 等。对于字体的设定,我们一般使用 *xeCJK* 宏包,因为他提供了很好的中文支持,并通过`\setmainfont{}`、`\setCJKmainfont{}`等命令设置文档的字体, 具体而言:

```
1 \documentclass[UTF8]{ctexart}
2 \usepackage{xeCJK}
3 \setmainfont{Times New Roman}
4 \setCJKmainfont{SimSun}
5 \setCJKsansfont{SimHei}
6 \setCJKmonofont{FangSong}
7 \setCJKfamilyfont{zhsong}{SimSun}
8 \setCJKfamilyfont{zhhei}{SimHei}
9 \setCJKfamilyfont{zhfs}{FangSong}
10 \setCJKfamilyfont{zhkai}{KaiTi}
11 \setCJKfamilyfont{zhli}{LiSu}
12 \setCJKfamilyfont{zhyou}{YouYuan}
13 \newcommand*\songti{\CJKfamily{zhsong}}
14 \newcommand*\heiti{\CJKfamily{zhhei}}
15 \newcommand*\kaishu{\CJKfamily{zhkai}}
16 \newcommand*\fangsong{\CJKfamily{zhfs}}
17 \newcommand*\lishu{\CJKfamily{zhli}}
18 \newcommand*\youyuan{\CJKfamily{zhyou}}
```

代码 2.2: 设置文档字体

2.4 文本格式控制

一些基本的命令: `\textit{}`、`\textbf{}`、`\underline{}`等命令可以设置文本的格式,而在中文支持中,斜体一般使用楷体,加粗一般使用黑体来显示。具体的控制表格如下:

表 2.1: 文本格式控制

命令	效果
<code>\textit{}</code>	斜体
<code>\textbf{}</code>	加粗
<code>\underline{}</code>	下划线
<code>\emph{}</code>	强调
<code>\texttt{}</code>	等宽字体
<code>\textsc{}</code>	小型大写字母
<code>\textsf{}</code>	无衬线字体
<code>\textrm{}</code>	衬线字体
<code>\textsl{}</code>	斜体
<code>\textsuperscript{}</code>	上标 ^{上标}
<code>\textsubscript{}</code>	下标 _{下标}

2.5 命令和宏包

\LaTeX 的命令以`\`开头,包括控制序列和环境两种。控制序列是 \LaTeX 的基本命令,用于控制文档的格式、样式等,例如`\textbf{}`用于加粗文本, `\textit{}`用于斜体文本等。

环境是一种特殊的控制序列，用于控制文档的结构，例如`equation`环境用于输入数学公式，`table`环境用于输入表格等。譬如：

```
1 \begin{equation}
2   E = mc^2
3 \end{equation}
```

代码 2.3: 数学公式环境

宏包用于扩展 \LaTeX 功能，通过引入宏包，我们可以方便地实现更多的功能，例如 `ams-math` 宏包用于输入数学公式，`graphicx` 宏包用于插入图片等。应当使用`\usepackage{}`命令引入宏包，当然也可以通过`\RequirePackage{}`命令引入。引入后，就可以使用宏包提供的功能了。

2.6 样式设定

您可以发现，本文档的代码块、图片块都已经过高度的客制化。如果想要如此地预先封装好命令和环境，我们可以使用`\newcommand{}`和`\newenvironment{}`命令来自定义命令和环境。例如：

```
1 \newcommand{\note}[1]{\textcolor{red}{#1}}
```

代码 2.4: 自定义命令

```
1 \newenvironment{note}[1]{\textcolor{red}{#1}}{ }
```

代码 2.5: 自定义环境

2.7 插图表格

对于插图、表格的位置，共有 `h`、`t`、`b`、`p`、`!` 五种参数，分别代表 *here*、*top*、*bottom*、*page*、*force*。具体而言：

表 2.2: 位置参数列表

位置参数	介绍
<code>h</code>	当前位置
<code>t</code>	顶部
<code>b</code>	底部
<code>p</code>	单独一页
<code>!</code>	强制

下面给出使用此参数控制插图、表格的例子：

```

1 \begin{longlisting}
2   \begin{minted}{latex}
3     \begin{figure}[h!]
4       \centering
5       \includegraphics[width=0.8\textwidth]{Figures/git-latex-history.png}
6       \caption{模板维护历史}
7       \label{fig:git-history}
8     \end{figure}
9   \end{minted}
10  \caption{插图环境}
11  \label{listing:figure-latex-env}
12
13  \begin{minted}{latex}
14    \begin{table}[h!]
15      \centering
16      \begin{tabular}{|c|c|c|}
17        \hline
18        1 & 2 & 3 \\
19        \hline
20        4 & 5 & 6 \\
21        \hline
22        7 & 8 & 9 \\
23        \hline
24      \end{tabular}
25      \caption{示例表格}
26      \label{tab:example-table}
27    \end{table}
28
29  \end{minted}
30  \caption{表格环境}
31  \label{listing:table-latex-env}
32 \end{longlisting}

```

代码 2.6: 使用参数控制插图位置

2.8 引用参考文献

通常我们使用 *BibTeX* 来管理参考文献，它对应的文件格式 *.bib* 用来存储参考文献信息，通过 `\cite{}` 命令来引用参考文献。此外，还有 `\citep{}`、`\citete{}`、`\citeauthor{}` 等命令，用于引用不同格式的参考文献。例如：

```

1 \begin{thebibliography}{99}
2   \bibitem{lamport94}
3     Leslie Lamport,
4     \textit{\LaTeX: A Document Preparation System}.
5     Addison Wesley, Massachusetts,
6     2nd Edition,
7     1994.
8 \end{thebibliography}

```

代码 2.7: 编写参考文献在 *.bib* 文件中

2.9 总结

\LaTeX 这个排版系统非常强大，在实践中，我熟练了 \LaTeX 的基本知识，包括基本结构、命令和宏包、样式设定、插图表格、参考文献等相关内容，受益匪浅。

在日后的学习生活中， \LaTeX 势必将成为我重要的工具之一，我还会继续深入了解、学习使用，以期为后面的科学研究和论文写作打下坚实的基础。

参考文献

Git 官方文档 (2ndEdition[a]). *Git-分支-解决合并冲突*. Tech. rep. Git Documentation.

Git 官方文档 (2ndEdition[b]). *Git-工具-贮藏与清理*. Tech. rep. Git Documentation.

计算机教育中缺失的一课 (2020). *MIT Missing Semester - Lecture 6: Version Control (git)*. Tech. rep. MIT.

