本教程提供在不同系统上安装 LaTeX 和 Visual Studio Code，进行配置并使用的教程。

这是教程的简体中文网站版。

安装 LaTeX

比较推荐的 LaTeX 安装方式是安装 TeX Live。在其官网提供了安装的完整指导。

以下是一些简略的指引：

- 对于 **Windows**，建议直接下载 ISO 文件，通过挂载该镜像，执行其中的安装程序来进行安装；
- 对于 **RHEL 系 Linux 发行版** (如 RHEL, CentOS Stream, Rocky Linux, AlmaLinux 等)，可以使用 ISO 镜像进行安装；
- 对于 **Debian 系 Linux 发行版** (如 Debian, Ubuntu, Kubuntu, Xubuntu, Linux Mint 等)，可以直接在终端执行 `sudo apt install texlive-full`{.bash} 进行安装；
- 对于 **MacOS 和 MacOSX**，建议的安装版本为 MacTeX，它包含全部的 TeX Live，以及一些专用于 Mac 的附加内容；

虽然 TeX Live 官方建议通过在线安装程序来安装，但是因为网络原因，除非你位于海外地区，否则使用 ISO 镜像可以有效节省安装时间。

在安装完 TeX Live 后，后续的升级就可通过 `tlmgr update --all` 进行更新了。如果需要跨版本升级 (如从 TeX Live 2024 升级到 TeX Live 2025)，则最好删除或卸载已有的更新，然后直接安装新版本。一般来说 TeX Live 几乎不需要升级。

在 Windows 下安装 LaTeX

在下载上文提到的 ISO 镜像后，双击该 ISO 文件即可挂载。切换到挂载的镜像所在的目录，运行 `install-tl-Windows.bat`，指定安装目录后采用默认设置安装即可。

依次使用以下指令查看是否有版本信息输出，从而确认安装是否成功：

```
xelatex --version
pdflatex --version
lualatex --version
latexmk --version
```

在 Linux (RHEL 系发行版) 下安装 LaTeX

在下载上文提到的 ISO 镜像后，双击该 ISO 文件即可挂载。切换到挂载的镜像所在的目录，右键选择在该位置打开终端进入命令行。输入如下命令：

```
sudo perl ./install-tl --no-interaction
```

按照命令行提示输入密码，即可以管理员权限开始安装。

安装完成后，我们需要将 TeX Live 的路径添加到 `PATH` 以便系统和其他程序能够使用，一个可靠的方式是通过 `sudo nano ~/.bashrc` 编辑当前用户的环境变量，在其下新增一行，内容为：

``` export PATH=<TeX Live 的安装路径>:$PATH export MANPATH=/usr/share/man:<TeX Live 的 man 路径>:$MANPATH export INFOPATH=<TeX Live 的 info 路径>:$INFOPATH ```

以在 Intel x86-64 处理器的计算机上默认安装为例，则该内容具体为：

```
export PATH=$PATH:/usr/local/texlive/2025/bin/x86_64-linux:$PATH
export MANPATH=/usr/share/man:/usr/local/texlive/2025/texmf-dist/doc/man:$MANPATH
export INFOPATH=/usr/local/texlive/2025/texmf-dist/doc/info:$INFOPATH
```

随后使用 `source ~/.bashrc` 刷新环境变量，然后使用 `sudo dnf install perl-core perl-Time-HiRes perl-Unicode-Normalize perl-LWP-Protocol-https` 安装必要的 perl 包，并依次使用以下指令查看是否有版本信息输出，从而确认安装是否成功：

```
xelatex --version
pdflatex --version
lualatex --version
latexmk --version
```
```

在 Linux (Debian 系发行版) 下安装 LaTeX

直接在终端执行 `sudo apt install texlive-full`{.bash} 进行安装，并依次使用以下指令查看是否有版本信息输出，从而确认安装是否成功：

```
xelatex --version
pdflatex --version
lualatex --version
latexmk --version
```

安装 Visual Studio Code

由于 FlatPak (或 Snap) 的限制，如 Visual Studio Code (VSCode, VSC) 和 JetBrains 系列产品等代码编辑器或 IDE 最好直接安装在 Linux 系统中。以下安装流程也将按照这一方式安装来提供指导。

在 Windows 下安装 Visual Studio Code

在 Windows 下安装 VSCode 有两种不同的方式：通过 **Microsoft Store** 和通过 **安装包**。

通过 Microsoft Store 安装 Visual Studio Code

如果你使用自己的微软账户登录了你的 Windows，那么使用 **Microsoft Store** 安装 Visual Studio Code 可能是最简单便捷的方法：

1. 打开 Microsoft Store；
2. 在搜索区域搜索 Visual Studio Code；
3. 选择蓝色图标的 Visual Studio Code，或者如果你喜欢尝鲜，选择绿色图标的测试版；
4. 点击安装，等待安装完成即可

通过安装包安装 Visual Studio Code

另一种比较传统的方法是使用安装程序来安装，这允许你对安装过程进行一些自定义。

1. 访问 Visual Studio Code 的下载页面；
2. 选择对应的安装包，或者如果你想进一步定制，在下方小字部分选择 "User Installer" 或 "System Installer"，**注意选择和你的机器处理器架构对应的安装包！**
3. 运行下载的安装包，根据指引逐步完成安装即可

在 Linux (RHEL 系发行版) 下安装 Visual Studio Code

在 RHEL 系发行版下，可以通过如下方式将 VSCode 加入软件源，从而便于安装和更新。

首先，将 VSCode 存储库和密钥安装到系统：

```
sudo rpm --import https://packages.microsoft.com/keys/microsoft.asc &&
echo -e "[code]\nname=Visual Studio
Code\nbaseurl=https://packages.microsoft.com/yumrepos/vscode\nenabled=1\nautorefresh
=1\ntype=rpm-
md\ngpgcheck=1\ngpgkey=https://packages.microsoft.com/keys/microsoft.asc" | sudo tee
/etc/yum.repos.d/vscode.repo > /dev/null
```

如果你使用较新的系统（对于 Fedora 则是 Fedora 22 或更高版本），使用这个命令来更新软件源缓存并安装 VSCode：

```
dnf check-update
sudo dnf install code # 如果你喜欢尝鲜，把 code 换成 code-insiders
```

如果你的系统较旧，则需要使用 yum 而不是 dnf：

```
yum check-update
sudo yum install code # 如果你喜欢尝鲜，把 code 换成 code-insiders
```

在 Linux (Debian 系发行版) 下安装 Visual Studio Code

在 Debian 系发行版下，可以通过如下方式将 VSCode 加入软件源，从而便于安装和更新。

首先，打开终端，运行以下脚本来安装签名密钥：

```
sudo apt-get install wget gpg &&
wget -qO- https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor >
microsoft.gpg &&
sudo install -D -o root -g root -m 644 microsoft.gpg
/usr/share/keyrings/microsoft.gpg &&
rm -f microsoft.gpg
```

随后，使用 `sudo nano /etc/apt/sources.list.d/vscode.source` 创建软件源配置，将 VSCode 的软件包仓库加入系统的软件源列表。其中文件内容如下：

```
Types: deb
URIs: https://packages.microsoft.com/repos/code
Suites: stable
Components: main
Architectures: amd64,arm64,armhf
Signed-By: /usr/share/keyrings/microsoft.gpg
```

最后，使用以下命令更新软件源缓存并且安装 VSCode：

```
sudo apt install apt-transport-https && sudo apt update # 更新软件源缓存
sudo apt install code # 如果你喜欢尝鲜，把 code 换成 code-insiders
```

配置 Visual Studio Code

安装完 VSCode 后，即可在已安装的程序中找到 VSCode。在配置好语言、主题、账号和同步等后，即可进入下一步：**针对 LaTeX 的配置**。

LaTeX Workshop

LaTeX Workshop 是 Visual Studio Code 上颇为经典的一款扩展，提供了预览、编译选项管理、自动完成、关键词高亮等功能。这个插件也是我们选择 VSCode 而不是其他工具来编写和编译 LaTeX 文档的最主要原因。

要安装 LaTeX Workshop，只需要打开左侧的扩展选项（或按快捷键 `Ctrl+Shift+X`），搜索 `LaTeX Workshop`，点击安装即可。

安装完成后，点击齿轮图标进入管理菜单，选择"设置"进入其配置界面。

配置部分并不复杂。在搜索栏中已有的内容后添加 `tools`，在弹出的第一个栏目下点击 "在 `settings.json` 中编辑"，然后粘贴下述内容即可获得较为通用的配置：

```
"latex-workshop.latex.autoBuild.run": "never",  // 不在保存时自动编译
"latex-workshop.showContextMenu": true, // 启用 LaTeX 上下文菜单
"latex-workshop.intellisense.package.enabled": true,    // 根据加载的包，自动完成命令或包
"latex-workshop.message.error.show": false,
"latex-workshop.message.warning.show": false,
"latex-workshop.latex.recipe.default": "lastUsed",  // 默认将上次使用的编译方法作为默认编译动作
"latex-workshop.view.pdf.internal.synctex.keybinding": "double-click",
// 编译时调用的指令组
"latex-workshop.latex.tools": [
    {
        // 采用插件预设值的 Latexmk
        "name": "latexmk",
        "command": "latexmk",
        "args": [
            "-synctex=1",
            "-interaction=nonstopmode",
            "-file-line-error",
            "-pdf",
            "-outdir=%OUTDIR%",
            "%DOCFILE%"
        ]
    },
    {
        // 全部根据 .latexmkrc 文件设置进行编译的 Latexmk
        "name": "latexmk_auto",
        "command": "latexmk",
        "args": []
    },
    {
        // 根据 .latexmkrc 文件设置进行辅助文件清理
```

```
        "name": "latexmk_auto_clean",
        "command": "latexmk",
        "args": [
            "-c"
        ]
    },
    {
        // LuaLaTeX
        "name": "lualatex",
        "command": "lualatex",
        "args": [
            "-synctex=1",
            "-interaction=nonstopmode",
            "-file-line-error",
            "%DOCFILE%"
        ]
    },
    {
        // XeLaTeX
        "name": "xelatex",
        "command": "xelatex",
        "args": [
            "-synctex=1",
            "-interaction=nonstopmode",
            "-file-line-error",
            "%DOCFILE%"
        ]
    },
    {
        // PDFLaTeX
        "name": "pdflatex",
        "command": "pdflatex",
        "args": [
            "-synctex=1",
            "-interaction=nonstopmode",
            "-file-line-error",
            "%DOCFILE%"
        ]
    },
    {
        // Biber
        "name": "biber",
        "command": "biber",
        "args": [
            "%DOCFILE%"
        ]
```

```
    },
    {
        // BibTeX
        "name": "bibtex",
        "command": "bibtex",
        "args": [
            "%DOCFILE%"
        ]
    }
],
"latex-workshop.latex.recipes": [
    {
        // 完全根据 .latexmkrc 文件进行编译
        "name": "Latexmk (auto)",
        "tools": [
            "latexmk_auto"
        ]
    },
    {
        // 与上一个对应，用于清理输出目录的辅助文件
        "name": "Latexmk Clean (auto)",
        "tools": [
            "latexmk_auto_clean"
        ]
    },
    // 各个编辑器和 Biber 组合，用于含目录和参考文献的场景
    {
        "name": "lualatex -> biber -> lualatex*2",
        "tools": [
            "lualatex",
            "biber",
            "lualatex",
            "lualatex"
        ]
    },
    {
        "name": "xelatex -> biber -> xelatex*2",
        "tools": [
            "xelatex",
            "biber",
            "xelatex",
            "xelatex"
        ]
    },
    {
        "name": "pdflatex -> biber -> pdflatex*2",
```

```
        "tools": [
            "pdflatex",
            "biber",
            "pdflatex",
            "pdflatex"
        ]
    },
    // 各个编译器和 BibTeX 的组合，用于含目录和参考文献的场景
    {
        "name": "lualatex -> bibtex -> lualatex*2",
        "tools": [
            "lualatex",
            "bibtex",
            "lualatex",
            "lualatex"
        ]
    },
    {
        "name": "xelatex -> bibtex -> xelatex*2",
        "tools": [
            "xelatex",
            "bibtex",
            "xelatex",
            "xelatex"
        ]
    },
    {
        "name": "pdflatex -> bibtex -> pdflatex*2",
        "tools": [
            "pdflatex",
            "bibtex",
            "pdflatex",
            "pdflatex"
        ]
    },
    // 单一编译器命令
    {
        "name": "LuaLaTeX",
        "tools": [
            "lualatex"
        ]
    },
    {
        "name": "XeLaTeX",
        "tools": [
            "xelatex"
```

```
            ]
        },
        {
            "name": "PDFLaTeX",
            "tools": [
                "pdflatex"
            ]
        },
        {
            "name": "Latexmk",
            "tools": [
                "latexmk"
            ]
        },
        {
            "name": "Biber",
            "tools": [
                "biber"
            ]
        },
        {
            "name": "BibTeX",
            "tools": [
                "bibtex"
            ]
        }
    ],
```

如此，在编写好你的 LaTeX 文档后，只需转到 LaTeX Workshop 的侧边栏页面，执行一次编译指令，后续点击 `.tex` 文件编辑窗口右上角的构建运行按钮（`Ctrl+Alt+B`），即可一键编译为 PDF 了。

进阶技巧：Latexmk

就如同前文中设置 Recipe 时所展现的一样，如果文章有目录，你需要连续用同一个编译器编译两次才能获得正确的 PDF；而如果文章有参考文献，还需要先编译一次，再经过 Bibier 或 BibTeX，再由同一个编译器编译一次（如果文章含目录，则需要再编译两次）才可以。有时，为应对复杂的功能需求，甚至还需要在编译时运行一些外部程序。随着需求越来越复杂，显然只是配置 Recipe 会让选项越来越多且不一定实用。

Latexmk 是一个 Perl 脚本，它可以为你处理所有这些事情。这里是我发现的一个介绍页面，介绍了如何直接使用 latexmk。

Latexmk 还可以配合对应的 `.latexmkrc` 文件来管理运行选项。这也是之前的配置文件中我们添加了 `Latexmk (auto)` 和 `Latexmk Clean (auto)` 这两项的原因。前者将另 latexmk 完全根据你的项目目录的（如果没有，则会参考当前用户的或者系统的）`.latexmkrc` 文件中的设置来编译生成文档；后者则用于对应的执行清理辅助文件的工作。

以下是关于 `.latexmkrc` 中常用的几种设置的介绍。可以参考本项目中采用的 `.latexmkrc` 来进一步了解各个设置的具体用法。

设置输出和编译器

使用 `$pdf_mode` 设置是否输出 PDF 文件：

- 0： 不生成 PDF
- 1： 使用 PDFLaTeX 编译
- 2： 使用 PS2PDF 编译
- 3： 使用 DVIPDF 编译
- 4： 使用 LuaLaTeX 编译
- 5： 使用 XeLaTeX 编译

其中，PDFLaTeX 生成很快， XeLaTeX 和 LuaLaTeX 对 UTF-8 支持更好，都是比较常用的选项。

对于参考文献工具，使用 `$bibtex_use` 选项来控制。

- 0： 不使用 BibTeX 和 Bibier，也不会清除 `.bbl` 文件
- 1： 仅当 `.bib` 文件存在时使用 BibTeX 或 Bibier，不会清除 `.bbl` 文件
- 1.5： 仅当 `.bib` 文件存在时使用 BibTeX 或 Bibier，仅当 `.bib` 文件存在时清除 `.bbl` 文件
- 2： 仅当 `.bib` 文件存在时使用 BibTeX 或 Bibier，无论 `.bib` 文件是否存在，都会清除 `.bbl` 文件

此外，对于 Bibier，一般使用如下的参数配置：`biber = "biber %O %S";`

设置主文件

可以通过 `@default_files` 设置需要编译的主文件.需要忽略的文件则可通过 `@default_excluded_files` 设置。一个可供参考的范例如下：

```
@default_files = ("build-en-GB.tex", "build-zh-CN.tex");
@default_excluded_files = ();
```

输出路径和清理

可以通过 `$out_dir` 设置输出文件所在的目录。编译中产生的辅助文件也会放在那里。

通过 `$out_dir` 设置需要清理的中间文件。选择性的保留中间文件对需要发表到期刊的论文可能很有帮助。