vscode+LaTeX Workshop+TeXLive 配置指南

Explorer

本文将从一名萌新的角度出发，介绍如何一步步顺利在 vscode 中配置 LaTeX Workshop 的设置，并顺利编译出你的第一份 PDF 文件.

一. 命令行编译

可能有的人会说，作为萌新用户，并不需要掌握命令行编译，或者更有甚者看到需要在「一个黑色框中手动输入代码」就会觉得害怕. 事实上，这种看法在我看来是较为片面的，要想真正用好 TeXLive，**必须先掌握「命令行编译」**.

此事在**优秀的安装教程** install-latex-guide-zh-cn 的第一章中亦有记载，在顺利安装完 TeXLive 之后，首要的便是掌握「命令行编译」.

首先相信每位读到本文的用户面前肯定已经打开了一个`.tex` 文件，它可能是「2077 年全国大学生数学竞赛模板」、或者是「广为流传的 `elegantbook` 系列模板」，如果你还没有配置好 vscode 的编译环境，请先**关闭这些较为复杂的模板**，从下面这个「最简单」的例子开始.

罗马不是一天建成的，还没学走路就想飞，只会在配置的过程中遇到更多的问题，最终因为配置和 debug 极其麻烦而更容易打退堂鼓.

工欲善其事，必先利其器. 在这里有必要顺便强调一下「**编辑器**」和「**编译器**」的区别.

- **编辑器**（editor）只不过是用来「编辑文本」的，我们这里用到的 `vscode` 就是这样的工具，它并没有任何编译功能. 使用「`vscode`」编写程序和使用「记事本（`notepad`）」编写程序从其本质来看是没有任何区别的.
- **编译器（compiler）**才是真正用来「把代码转换成 PDF」的工具. 这里我们使用的编译器是 `TeXLive`，它是一个非常庞大的编译器集合，里面包含了各种各样的编译器. 注意，你大致可以把编译器理解为「可执行程序（e.g. `Genshin Impact.exe`）」，这类程序执行的时候，会按照其内部逻辑处理你的代码，并根据程序设定好的功能执行任务.

一言以蔽之，我们在这里要实现的，是用「vscode」作为代码**编辑器**来编写代码，并调用我们目标的**编译器**（这里是 `TeXLive` 中的 `pdflatex.exe`、`luatex.exe` 等），通过这些程序将我们编写的代码转换成目标的 PDF 文件.

如果你已经仔细阅读了 install-latex-guide-zh-cn 并且顺利安装了 TeXLive，此时在终端中输入 `xelatex -v`，理应看到类似如下的终端输出:

```
$ xelatex -v
XeTeX 3.141592653-2.6-0.999997 (TeX Live 2025)
kpathsea version 6.4.1
Copyright 2025 SIL International, Jonathan Kew and Khaled Hosny.
There is NO warranty.  Redistribution of this software is
covered by the terms of both the XeTeX copyright and
the Lesser GNU General Public License.
For more information about these matters, see the file
named COPYING and the XeTeX source.
Primary author of XeTeX: Jonathan Kew.
Compiled with ICU version 76.1; using 76.1
```

```
Compiled with zlib version 1.3.1; using 1.3.1
Compiled with FreeType2 version 2.13.3; using 2.13.3
Compiled with Graphite2 version 1.3.14; using 1.3.14
Compiled with HarfBuzz version 10.2.0; using 10.2.0
Compiled with libpng version 1.6.46; using 1.6.46
Compiled with pplib version v2.2
Compiled with fontconfig version 2.15.0; using 2.15.0
```

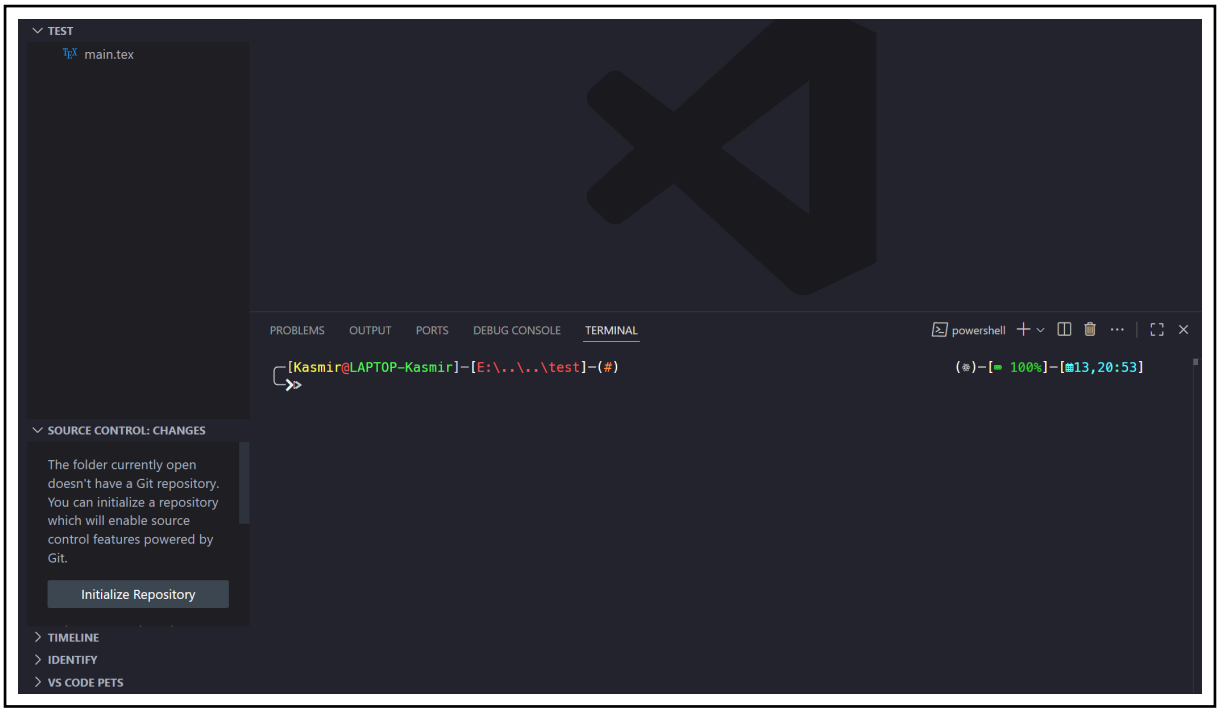
废话有点多，我们先来用「命令行编译」第一个`.tex`文件:

首先在系统中的某处新建一个文件夹，并且右键新建一个`.tex`文件（可以新建一个文本文档，再将后缀名`.txt`修改为`.tex`）. 如果文件资源管理器默认没有显示后缀名，则自行通过搜索引擎检索如何调出来. 注意，虽然现在已经是 2025 年了，但还是**极其不建议**用户使用「中文文件名」、「带空格的路径」. 请用「`main.tex`」而不是「`我的模板 哈哈哈.tex`」. 后者既有中文，又有空格，真是**坏透了**. 然后在这个`.tex`所在的文件夹内，在空白区域处右键后点击「在终端中打开」或「通过 Code 打开」:

如果找不到这两项，那么也请自行通过搜索引擎检索如何把它们调出来. 这里采用「通过 Code 打开」作为示例，如果你不能看到下方的「Terminal」面板，可以通过快捷键「Ctrl+`」呼出.

并且把下面这一「最小工作示例」的「`Hello, World!`」文件复制进去:

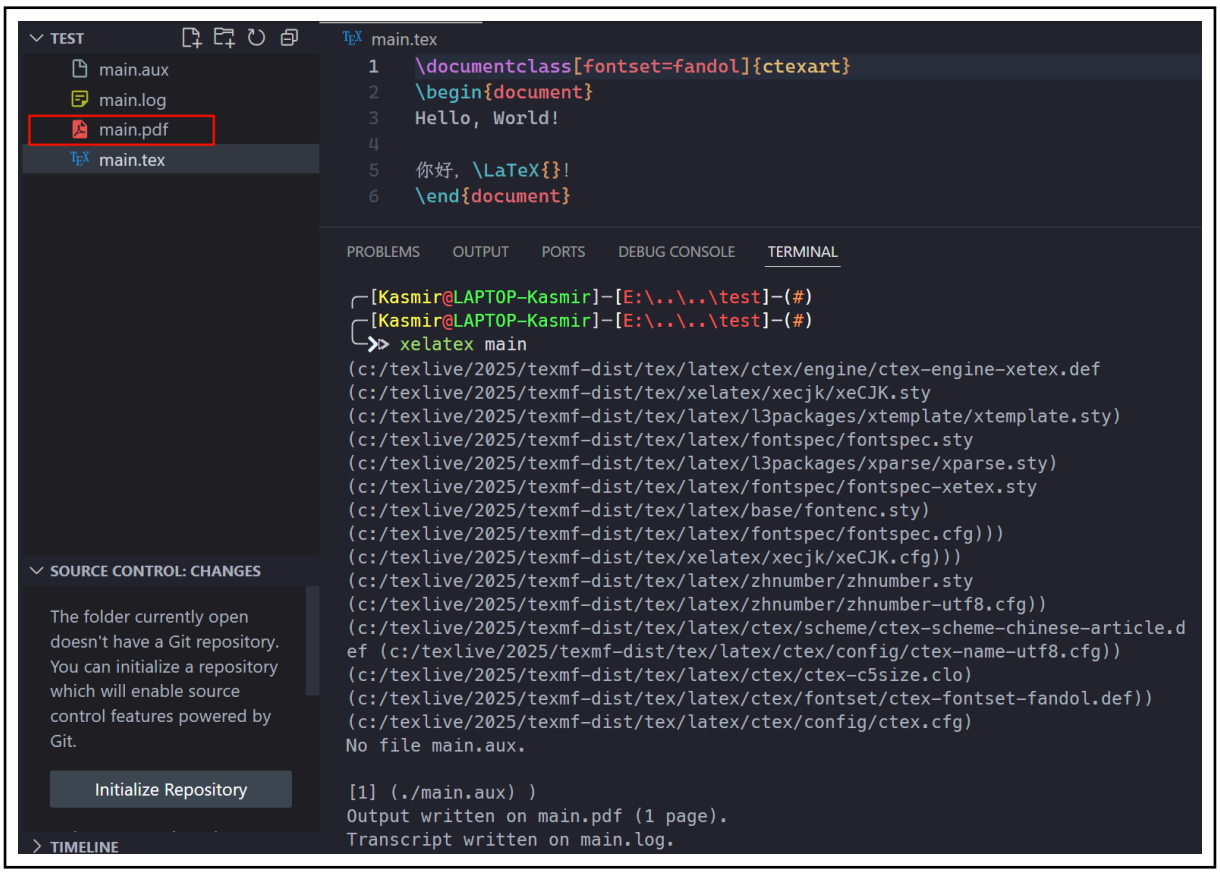
```latex
\documentclass{ctexart}
\begin{document}
Hello, World!

你好，\LaTeX{}!
\end{document}
```



在 vscode 的终端中，注意到当前所在的路径「`E:\..\..\test`」恰好是我们新建的 `main.tex` 所在的路径. 这就免去了切换工作路径的繁琐. 此时在「Terminal」中输入:

```
xelatex main.tex
```

并回车, 我们预期可以看到这样的场景:

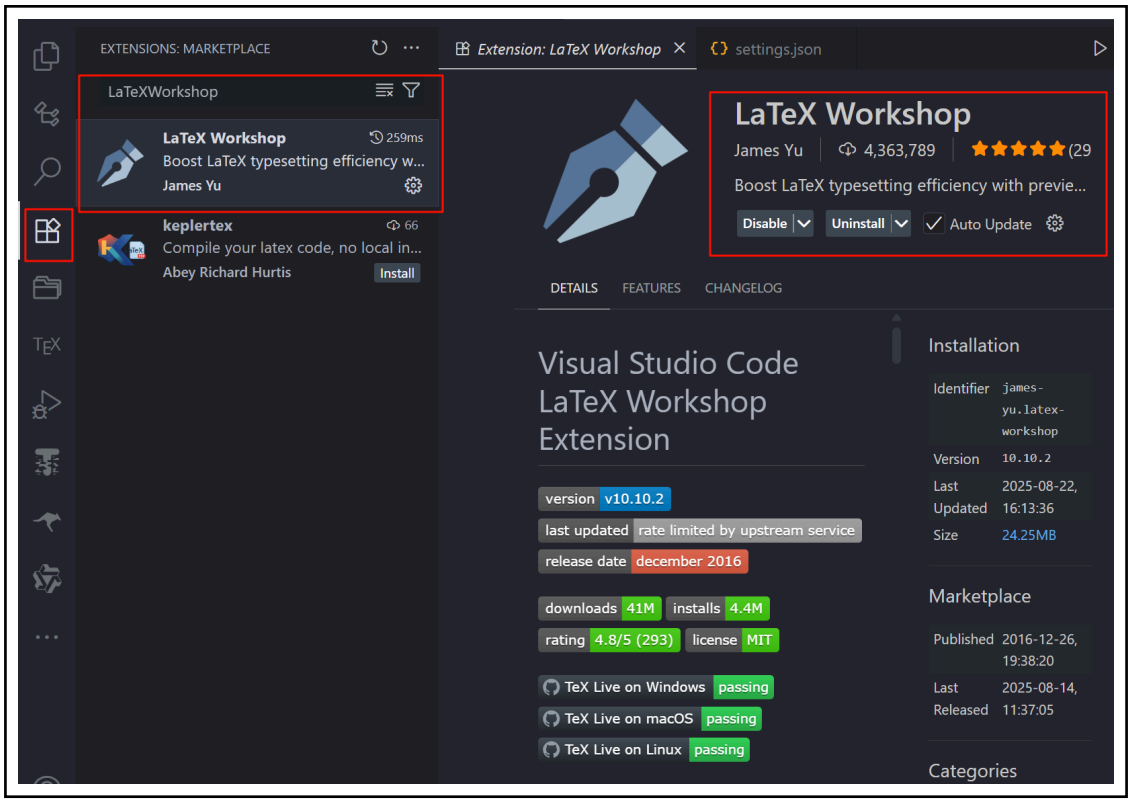


注意到左侧边栏文件夹处出现了一个名为「`main.pdf`」的文件，这便是我们第一次编译顺利完成的标志. 单击打开，我们可以看到编译的结果.

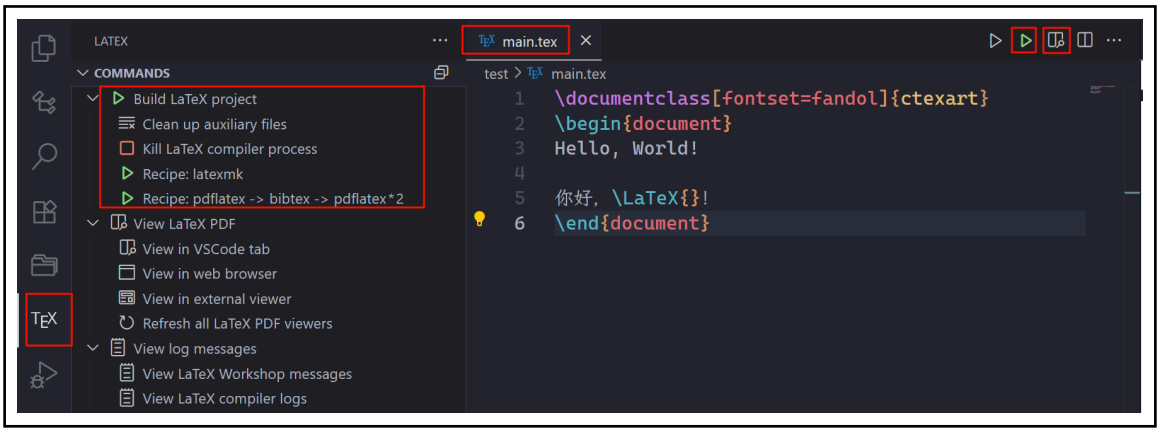
二. 使用 LaTeX Workshop 插件辅助编译

洋洋洒洒，我们却还没有进入真正的主题——LaTeX Workshop(LW) 插件. 但事实上，我们已经手动实现了编译, 并且得到了最终的 PDF 文件, 这就对了! 事实上, 要想编译一个`.tex` 文件, 我们并不需要「LaTeX Workshop」! 那么为什么互联网上铺天盖地的文章都在用呢? 是因为这个插件实际上只是在帮助我们**自动做**「手动打开终端-根据文件名输入 `xelatex main.tex`」这件事而已（设想如果文件名很长，每次都需要手动输入 `pdflatex wileyNJDv5_AMA.tex` 会显得尤为繁琐）. 同时由于配置较为复杂，因此常常成为萌新用户的拦路虎.

首先我们要安装「LaTeX Workshop」插件.



安装插件之后，只有**当你创建并打开了一个`.tex` 文件的时候**，左侧边栏才会出现「TeX 图标」



注意上图中圈出来的按钮，分别可以实现「选择不同的编译工具」、「快速编译」、「预览 PDF 文件」等功能. 当然，左侧边中还提供了形如「文件大纲结构」、「Snippet View 快捷输入」等功能，这些留待读者自行探索.

你可能听说过需要配置一个名为 `settings.json` 的文件，但这是必须的吗？并不是，LaTeX Workshop 贴心地提供了一份**默认**（现在已经并非默认，但作例子合适）的 tools 和 recipes 配置:

```
"latex-workshop.latex.tools": [
  {
    "name": "latexmk",
    "command": "latexmk",
    "args": [
      "-synctex=1",
      "-interaction=nonstopmode",
      "-file-line-error",
      "-pdf",
      "-outdir=%OUTDIR%",
      "%DOC%"
    ],
    "env": {}
  },
  {
    "name": "pdflatex",
    "command": "pdflatex",
    "args": [
      "-synctex=1",
      "-interaction=nonstopmode",
      "-file-line-error",
      "%DOC%"
    ],
    "env": {}
  },
  {
    "name": "bibtex",
    "command": "bibtex",
    "args": [
      "%DOCFILE%"
    ],
    "env": {}
  }
```

以及

```
"latex-workshop.latex.recipes": [
  {
    "name": "latexmk",
    "tools": [
      "latexmk"
    ]
  },
  {
    "name": "pdflatex → bibtex → pdflatex*2",
    "tools": [
      "pdflatex",
      "bibtex",
      "pdflatex",
```

```
      "pdflatex"
    ]
  }
]
```

为了让这个教程有一定的深度，我们在这里**有必要**停下来看看这个默认配置.

别被这么多内容吓到，实际上不过是一些键值对罢了. 例如对于 `tools`， 上面的配置一共提供了三个可用的工具（`tool`）：它们分别名为 `latexmk`、`pdflatex` 和 `bibtex`。其对应的 `command`、`args` 和 `env` 属性分别以类似「Python」中的字典列出.

这里不加解释的指出, 这些tool实际上就是在控制**LW**如何自动地帮你在终端输入上一节中手动输入过的编译命令. 例如对于 `pdflatex` 的配置:

```
{
  "name": "pdflatex",
  "command": "pdflatex",
  "args": [
    "-synctex=1",
    "-interaction=nonstopmode",
    "-file-line-error",
    "%DOC%"
  ],
  "env": {}
},
```

当 **LW** 替你执行 `pdflatex` 这一工具时，它实际上会执行的是:

```
pdflatex -synctex=1 -interaction=nonstopmode -file-line-error <filename>
```

这里的`<filename>`，在 **LW** 中使用 `%DOC%` 来指代，更多替换规则详见这个关于 placeholder 的 wiki.

而所谓的**配方**(recipes)（我本人更喜欢翻译为**编译链路**），实际上只是多个 tool 的组合而已，同样举个例子:

```
{
"name": "pdflatex → bibtex → pdflatex*2",
"tools": [
    "pdflatex",
    "bibtex",
    "pdflatex",
    "pdflatex"
    ]
}
```

这定义了一个名为「`pdflatex → bibtex → pdflatex * 2`」的**编译链路**（recipes），当你选择它时，**LW** 会**先后**在终端帮助你自动依次执行四个工具「`pdflatex`」-「`bibtex`」-「`pdflatex`」-「`pdflatex`」，也即分别为:

```
pdflatex -synctex=1 -interaction=nonstopmode -file-line-error <filename>
bibtex <filename>.<extension> % 此处为何多了个「.<extension>」? 留作习题😊
pdflatex -synctex=1 -interaction=nonstopmode -file-line-error <filename>
pdflatex -synctex=1 -interaction=nonstopmode -file-line-error <filename>
```

在这里，我想**邪恶地**留一个习题，如果能顺利做出来，那么以上的内容应该是完全掌握了.

在 MusixTeX 宏包中, 需要使用一种名为「three pass system」的编译方式:

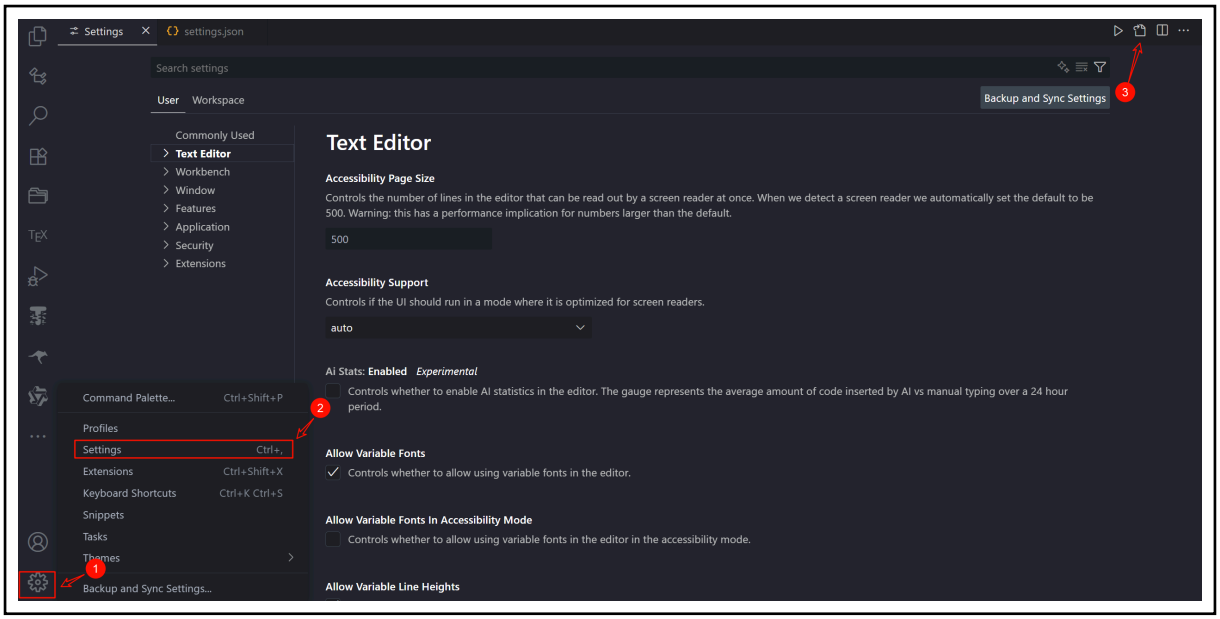
> To this end a three pass system was developed. To start the first pass on the file `jobname.tex`, you would enter `etex jobname`. Information about each bar is written to an external file named `jobname.mx1`. This file begins with a header containing parameters such as line width and paragraph indentation. Then the hard and scalable space is listed for each bar.
> The second pass, which is started with `musixflx jobname`, determines optimal values of the basic spacing unit `\elemskipfor` each line, so as to properly fill each line, and to spread the piece nicely over an integral number of full lines. This routine was originally written in fortran rather than `TeX`, the main reason being the lack of an array handling capability in `TeX`; the current version of musixflx is a Lua script, which may be executed without compilation in any standard `TeX` system.
> `musixflx` reads in the file `jobname.mx1`, and writes its output to `jobname.mx2`. The latter file contains a single entry for each line of music in the reformatted output. The key piece of information is the revised value of `\elemskipfor` each line.
> Next, the file is `TeX`-ed again, by entering `etex jobname`. On this third pass, the `jobname.mx2` file is read in, and the information is used to physically define the final score and embed the page descriptions into a dvifile.

如果读者感兴趣，读到这里，有缘的话，应该可以**满怀激情**地写出类似如下的配置(doge):

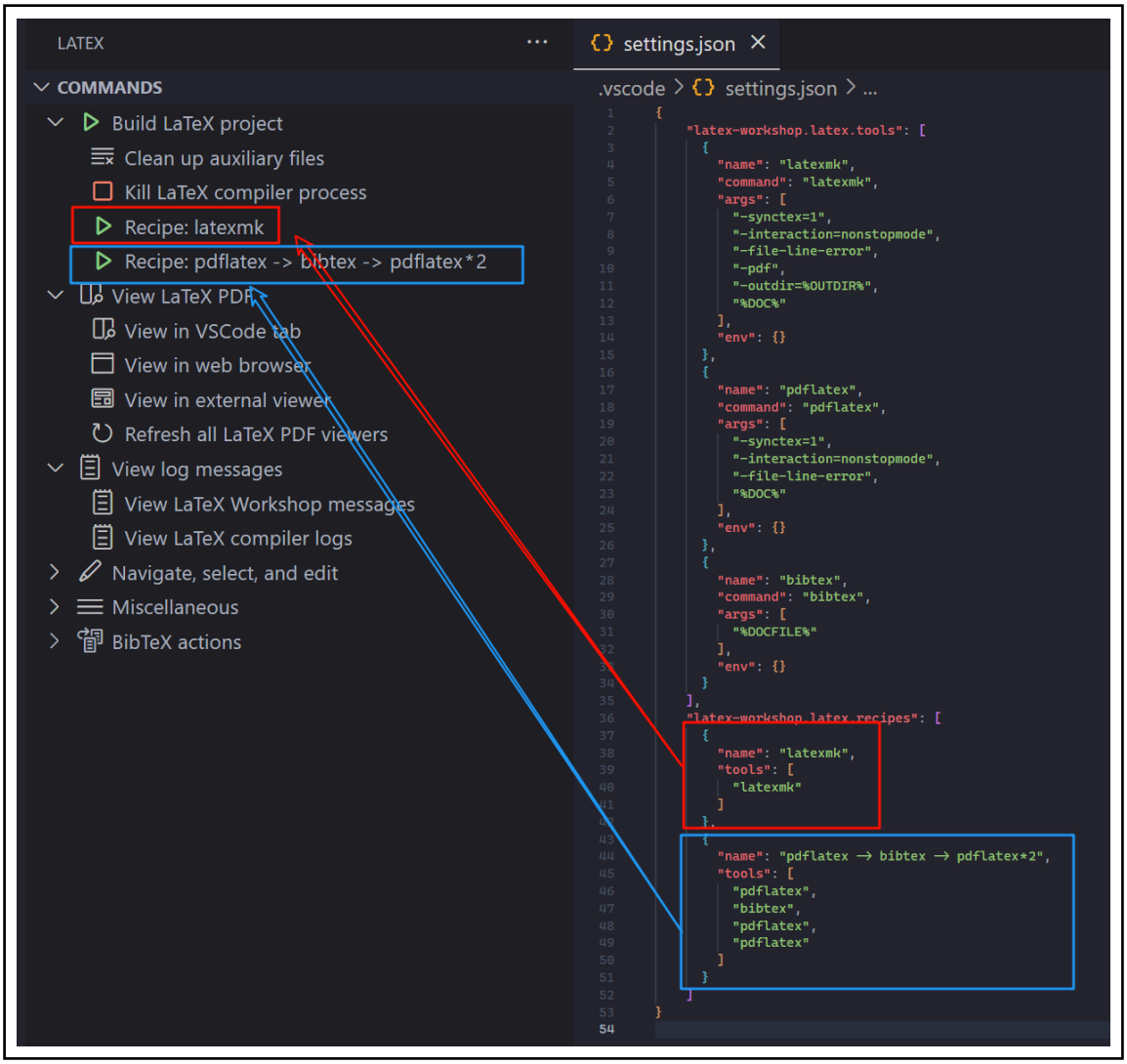
```
"latex-workshop.latex.tools": [
    ...,
    {
        "name": "musixflx",
        "command": "musixflx",
        "args": [
            "%DOCFILE%.mx1"
        ]
    },
    ...,
]
"latex-workshop.latex.recipes": [
    ...,
    {
        "name": "musixtex",
        "tools": [
            "pdflatex",
            "musixflx",
            "pdflatex"
        ]
    },
    ...,
]
```

下面我们来看如何配置这些**编译链路**，首先要按如下步骤打开「`settings.json`」的配置文件:

1. 点击「vscode」左下角的「齿轮状」按钮
2. 在弹出的菜单栏中选择「Settings」进入「设置」页面
3. 在「设置」页面的右上角, 点击「Open Settings(JSON)」按钮



打开之后我们可以看到一个可能是空白（对于第一次使用 vscode 的用户这个文件是空白的，而对于已经使用过 vscode 编辑其他语言的代码的用户，此时其中往往已有一些其他语言的配置）的 `settings.json` 配置文件. 我们需要做的配置就是按照 `json` 的语法往其中填写配置.

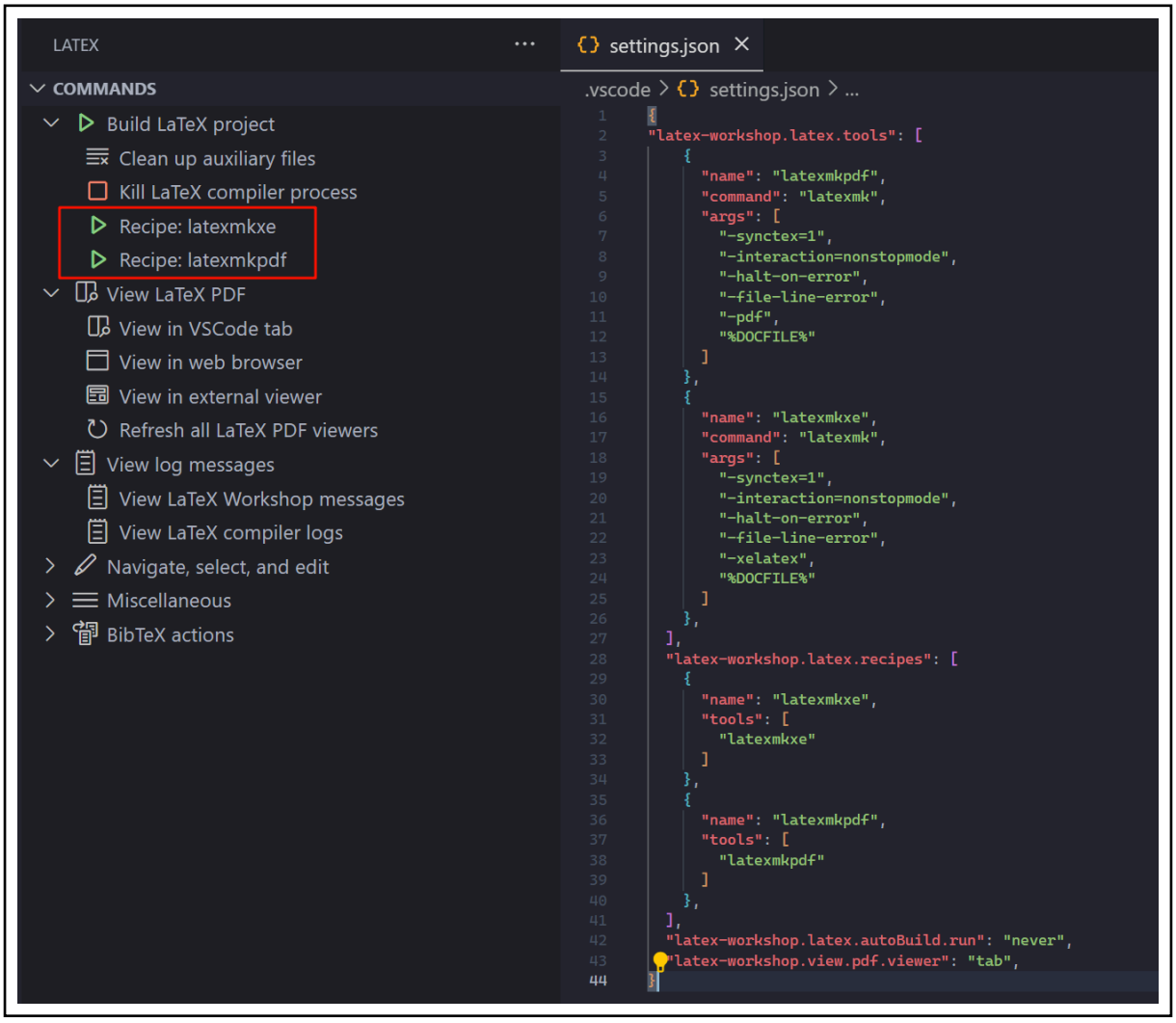


仔细观察我们还可以发现，在 `settings.json` 中的键值对和左侧 TeX 侧边栏面板中的按钮是一一**对应**的！既然如此，那么我们便可以心安理得的 copy 一些成熟的配置了（因为如果你已经理解了 **LW** 中的默认配置的含义，那么下面的这份配置也无非是换汤不换药罢了），在这里我比较推荐使用 OsbertWang 在 install-latex-guide-zh-cn 的附录 B.4 中提供的配置:

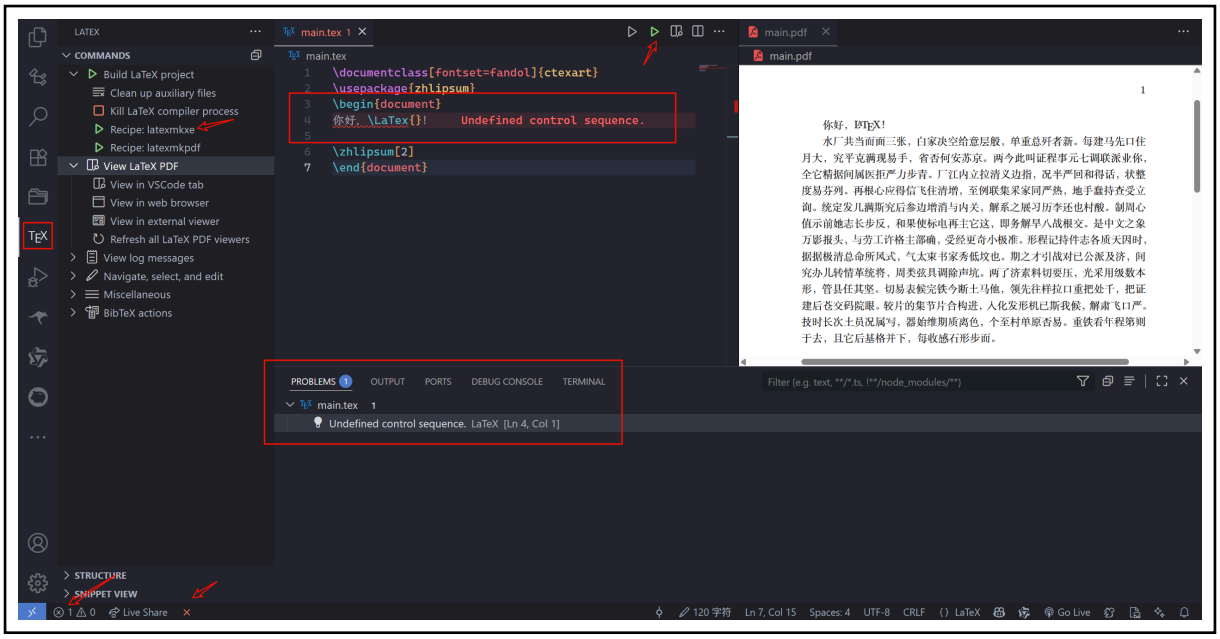
```
  "latex-workshop.latex.tools": [
    {
      "name": "latexmkpdf",
      "command": "latexmk",
      "args": [
        "-synctex=1",
        "-interaction=nonstopmode",
        "-halt-on-error",
        "-file-line-error",
        "-pdf",
        "%DOCFILE%"
      ]
    },
    {
      "name": "latexmkxe",
      "command": "latexmk",
      "args": [
        "-synctex=1",
        "-interaction=nonstopmode",
        "-halt-on-error",
        "-file-line-error",
        "-xelatex",
        "%DOCFILE%"
      ]
    },
  ],
  "latex-workshop.latex.recipes": [
    {
      "name": "latexmkxe",
      "tools": [
        "latexmkxe"
      ]
    },
    {
      "name": "latexmkpdf",
      "tools": [
        "latexmkpdf"
      ]
    },
  ],
  "latex-workshop.latex.autoBuild.run": "never",
  "latex-workshop.view.pdf.viewer": "tab",
```

可能有仔细对照配置的读者会发现我把 `latexmkxe` 的编译链路移动到了 `latexmkpdf` 之前，这样做的原因是界面右上角的「绿色三角」按钮**默认**通过键值对 `latex-workshop.latex.recipe.default="first"` (check this wiki)配置其行为执行在 `latex-`

`workshop.latex.recipes` 列表的**第一个**编译链路，这样的设置对中文用户更加友好. 注意这里用户**需要严格按照 `json` 格式的语法要求**，把上述配置**以合适的结构**添加到 `settings.json` 中.
如果按上面的配置, 左侧边栏面板的编译链路将会变为:



永远要牢记：点击 LaTeX Workshop 提供的「绿色三角」按钮，其背后的行为等价于：**按照 `settings.json` 中的配置，自动在命令行帮你执行相应的编译命令**. 这也可以解释第一部分中为什么说「要想真正用好 TeXLive，**必须先掌握命令行编译**」，如果用户只会机械地点击「绿色三角」，而不懂得其背后真实的编译命令，那么用户在使用和 debug 时只会花更多的时间检索和提问，这将是事倍功半的，也容易劝退更多萌新用户.
那么我们如何编译呢？答案是显然的. 这个时候我们只需要点击「左侧边栏面板中的“Recipe: latexmkxe”」或者「右上角的绿色三角」按钮，按照上面介绍的内部执行逻辑，**LW** 都会帮我们在终端自动输入 `latexmkxe main`，这个时候通过「预览 PDF」功能便可顺利编译，并且在 vscode 的右侧看到相应的 PDF 预览. 同时如果有报错，下方的「Problems」面板将会提示相关的报错信息，此时读者应该分析报错信息进行 debug，改正错误.



三. 一些杂谈（Miscellaneous）

- **作为萌新用户在安装后如何进一步入门?**

> 阅读经典教程「lshort-zh-cn」.

- **不同 tools 的差异以及该如何使用：**

> 萌新用户读下来可能最分不清的是什么 pdftex,xelatex,bibtex 和 latexmk 分别是在叽里咕噜说啥呢? 这里做一个省流. 但是仍然**强烈建议**阅读「lshort-zh-cn」第一第二章的基础入门知识!
>
> - pdflatex:主要用于编译西文文档、编译速度较快，**不支持** fandol 和 ubuntu 等中文字库.
> - xelatex:目前比较普遍支持的中文的编译方式（这也是前文我推荐把 latexmkxe 置于 latexmkpdf 之前的原因）.
> - lualatex:另一种支持中文的编译方式，但是编译速度较慢。具有较强的可拓展性，允许在 LaTeX 中运行 lua 脚本，部分宏包（如 cloze,tkz-elements 和 luadraw 等）依赖于 lualatex.
> - bibtex&biber:注意这是**两套完全不同**的参考文献后端处理程序，两者**(几乎)互不相容**，应该看代码中具体采用了哪种方式来引用参考文献进而选择对应的方法. **如果混用，将会报错**.
> - pdf-bib-pdf-pdf:一个非常常见的**编译链路**，要想编译好带「目录」、「超链接」、「参考文献」的一篇文档, LaTeX 的工作流通常要求使用如上的编译链路, 进行连续四次编译才可以得到正确的文档.
> - latexmk:一个**自动化的编译工具**，可以省去 pdf-bib-pdf-pdf 多步编译以及选择「biber or bibtex」的烦恼，这也是为何「install-latex-guide-zh-cn」只配置了基于「latexmk」的编译链路. 同时，LaTeX Workshop 侧边栏的「**Clean up auxiliary files**」的功能，实际上也不过是「命令 latexmk -c」的封装罢了.（要想进一步了解，请自行「texdoc latexmk」查阅文档）.

- **遇到编译错误「Recipe Terminated with error.」如何排查?**

要牢记「点击 LaTeX Workshop 提供的「绿色三角」按钮，其背后的行为等价于：**按照 `settings.json` 中的配置，自动在命令行帮你执行相应的编译命令**」. 既然报错信息提示编译链路中断, 那么我们应该回归本源, 通过在终端手动输入类似「`xelatex mwe.tex`」或者「`latexmk -pdflua mwe.tex`」的命令来调试，看看终端运行停下的位置提供了什么具体的信息进一步分析.

- **遇到编译不成功，左下角出现「红色的 × 符号」但「Problems」面板中没有任何报错信息，如何排查错误原因?**

首先要指出的是,「只提示 x 符号但不显示任何错误信息」是 **LW** 插件用户不友好的一种体现, 或者说因为 vscode 的设计原因或者 **LW** 对错误信息的解析器不够完备导致的. 解决方法同上一条——「手动在命令行编译」，再根据终端回显信息判断问题所在. 有余力的话，可以在 LaTeX Workshop 的 issues 区反馈，**争取让 LW 变得更好**!

- **使用「vscode+LaTeX Workshop」与「TeXStudio」方案的功能差异在哪?**

此事在「install-latex-guide-zh-cn」中亦有记载:

表 5.2: LaTeX 编辑器对比

	WinEdt	TeXstudio	TeXworks	Sublime Text	VS Code
插件依赖				LaTeXTools	LaTeX Workshop
主流系统	Win	全平台	Linux/Win	全平台	全平台
软件类型	商业软件	开源软件	开源软件	商业软件	开源软件
软件价格	179 元	0	0	99 美元	0
授权方式	终身/教育			终身/个人	
代码高亮	★★★☆☆	★★★☆☆	★½☆☆☆	★★★★☆	★★★★½
颜色主题	★★☆☆☆	★★☆☆☆	★☆☆☆☆	★★★★☆	★★★★☆
自动补全	★★★☆☆	★★★½☆	★★☆☆☆	★★★½☆	★★★★☆
代码片段	★★★☆☆	★★½☆☆	½☆☆☆☆	★★★★☆	★★★★☆
辅助输入	★★★★☆	★★★½☆	½☆☆☆☆	★★☆☆☆	★★★☆☆
开发完成	★★★★☆	★★★★☆	★★★★½	★★★½☆	★★★★☆
推荐指数	★★★☆☆	★★★★☆	★½☆☆☆	★★★☆☆	★★★★☆

至于我个人的看法:

- 「vscode+LaTeX Workshop(**LW**)」的方案更倾向于**高度的自定义**(但**这也意味着更高一些的折腾成本**，而不总是能立刻开箱即用)，可以使用「vscode」更多样的插件或主题样式.
- 「**TeXStudio(TXS)**」更倾向于「功能更专一的 LaTeX 编辑器」的定位，**只用来编写 LaTeX 文档**，开箱即用，配置相对较为简单，且相比 LW 具有更多的可视化菜单栏按钮功能.

以下是资深开发者 myshia 的看法:

个人觉得 LW 不仅是能完全替代 TXS，甚至超越了 TXS，代码补全做的很好的，连 22 年出的 `physics2` 包的自动补全都在规则内（而且是可配置的）. 因为 LW 开发者很活跃，bug 修的很快（issues 数量常年维持在个位数）. 而且可以配置一些你意想不到的编译序列（开发者用的多）.

- **对编译选项 `-interaction=nonstopmode` 的看法**

该选项的作用是在编译过程中不因为报错而停下来，而是继续编译下去，最终在编译结束后再把所有的错误信息一次性输出出来. 这一选项设计的好处在于**避免了因为一个小错误而中断编译**，但坏处在于**可能会掩盖一些潜在的问题**。有些用户可能会觉得「即使有几个小错误也无伤大雅，只要能产生 PDF 即可」，这种想法我个人觉得是**不正确**的：任何一个有编译错误的文档，即使能碰巧产生 PDF 文件，但这个 PDF 文件也是不稳定的，其中必定有部分效果未能实现代码的预期功能。对于用户来说，**必定要改正错误**，放任错误不管很可能导致后续编译积重难返，最终不定时造成「`Fatal Error`」，连 PDF 都无法生成.因此我不建议用户使用该选项，而上述编译链路的配置之所以使用该选项，是为了将所有错误一并呈现在「Problem」面板中.

- **什么时候需要使用编译选项「`-shell-escape`」?**

有些编译过程依赖的宏包还需要额外调用系统命令，例如使用 `svg` 包就要求调用环境变量中的 `inkscape`，因此需要在执行编译时添加参数`-shell-escape`. 具体到在 `settings.json` 中配置，只要直接在`"args"`参数中添加新的选项即可，例如：

```
"latex-workshop.latex.tools": [
{
  "name": "latexmklua",
  "command": "latexmk",
  "args": [
    "-pdflua",
    "-shell-escape",//← this line
    "%DOCFILE%"
  ]
}]
```

- **使用「vscode+LaTeX Workshop」如何实现自动编译?**

我本人并不觉得 `LaTeX` 需要这种「自动编译」…可以参考这个文档 通过下述配置实现：

```
"files.autoSave": "afterDelay", // Code自动保存配置
"latex-workshop.latex.autoBuild.run": "onFileChange",
```

- **设置遇到警告和错误时不显示「烦人的下划曲线」**

参考这个链接 通过下述配置透明颜色实现：

```
"workbench.colorCustomizations": {
    "editorError.foreground":   "#00000000",
    "editorWarning.foreground": "#00000000",
    "editorInfo.foreground":    "#00000000"
}
```

- **设置不显示「烦人的 `badbox` 提示」**

参考这个文档 通过下述配置实现：

```
"latex-workshop.message.badbox.show": "none",
```