Still a work in progress...

There are many ways to set-up LaTeX on your local machine. My favorite is to use VS Code with Latex Workshop, but TexStudio is probably the more beginner friendly option. There are several guides on the internet for the basic set-up of Latex. But here are a couple things I have learned and found useful specifically for tagging documents.

1 TexStudio Option

To use TexStudio, you will need to download Latex (from the internet) - it is free. You can choose which distribution of Latex to download, the most popular are TexLive, MikTex, or MacTex (I prefer MikTex for Windows). You will also need to download TexStudio (from the internet) - it is also free.

Then, we want to use LuaLatex with this project for tagging. Therefore, in TexStudio go to Options- Configure TexStudio- Build. Use the dropdowns to change the default compiler to LuaLatex and the bibliography tool to Biber. When you want to compile, use the double play button icon. When you change the bibliography (by changing the bib file or by adding references), you will want to compile the latex (if you have not already done so), then run the bibliography tool by going to Tools - Bibliography, and then compile the Latex again.

If you get errors, the first things you should do is re-compile the latex, run the bibliography tool, and re-compile the latex again and see if the errors persist. A warning about cross-references may persist across several runs - you do not need to worry about it while drafting. Before final submission, keep recompiling until it goes away (and it will). If the error ".. doesn't match definition.." does not resolve by running the bibliography tool and recompiling, you can also try Tools - Clean Auxiliary Files, and then compiling - bibliography tool - compiling again.

2 VS Code Option

2.1 Using LuaLatex without using latexmk

By default, Latex Workshop mostly uses latexmk - a useful tool. However, with tagging enabled, latexmk does several runs every time you make any change, which can be frustratingly slow. Most of the time, it is better to do a single compilation between minor changes as you edit and then a couple more re-compiles to make sure everything is settled right before submission. I also like auxillary files to be in a separate folder and therefore provide instructions to that end. Putting the auxillary files in a separate folder is only (easily) possible with MikTex, so there will also be instructions for leaving the auxillary files alone.

Regardless, to do this, we need to add extra recipes in order to have an option to just run LuaLatex or possibly LuaLatex + Biber + LuaLatex (for

after bibliography changes). Go to Settings, search for Recipes, then choose the option to open Settings.json.

2.1.1 Auxillary Files in a Different Folder with MikTex

Search for "latex-workshop.latex.tools" and **add** to the list there:

```
{
    "name": "lualatex",
    "command": "lualatex",
    "args": [
        "--aux-directory=%DIR%/LatexAux",
        "--interaction=nonstopmode",
        "--synctex=1",
        "--c-style-errors",
        "--output-directory=%DIR%",
        "%DOC%"
    ],
    "env": {}
},
{
    "name": "biber",
    "command": "biber",
    "args": [
        "%DIR%/LatexAux/%DOCFILE%",
        "--output-directory=%DIR%/LatexAux",
        "--input-directory=%DIR%"
    ],
    "env": {}
},
```

2.2 Auxillary Files in same folder (with any latex)

Search for "latex-workshop.latex.tools" and **add** to the list there:

```
{
    "name": "lualatex",
    "command": "lualatex",
    "args": [
        "--interaction=nonstopmode",
        "--synctex=1",
        "--c-style-errors",
        "--output-directory=%DIR%",
        "%DOC%"
    ],
    "env": {}
},
{
    "name": "biber",
    "command": "biber",
    "args": [
        "%DOC%,
        "--output-directory=%DIR%",
    ],
    "env": {}
},
```

2.3 Continued Instructions for both

Then, search for "latex-workshop.latex.recipes" and add to the list there:

```
{
    "name": "lualatex-plain",
    "tools": [
        "lualatex"
    ]
},
{
    "name": "lualatex with biber",
    "tools": [
        "lualatex",
        "biber",
        "lualatex"
    ]
},
{
    "name": "biber",
    "tools": [
        "biber"
    ]
},
```

Now, in the TeX menu there will be the option to run just "lualatex-plain", "biber", or "lualatex with biber." You can change the default used for your file by adding to the top of thesis.tex

```
%!LW recipe=lualatex with biber
```

2.4 Using Version Control

Using version control is a great way to back-up your work and be able to see the history of the document. VS Code has built-in support for GitHub, which you can find by choosing the version control tab on the left. You will need a GitHub account (free; GitHub is owned by Microsoft).

For version control to be at its best, however, you don't any one line to be too long. One way to accomplish this is to have every sentence on its own line (as LaTeX ignores single new-lines). You can do this manually, or you can configure VS Code to automatically run a script called "latexindent.pl" (https://github.com/cmhughes/latexindent.pl) when you save the file to format your file for you. Occasionally, this adds line breaks where they should not be, but it is usually a minor problem.

If you want to use latexindent, you need to adjust the settings to allow it to modify the line breaks. You can do this e.g. with the settings in localSettings.yaml. Further, in VS Code, you need to modify the setting "Latexindent: Args" and add "-l" (use local settings) and "-m" (modify line breaks).