The design concept for llmk — Light LaTeX Make

Takuto Asakura

Abstract

It is a matter of joy that we have many options for processing a LaTeX document. We can choose the most suitable TeX engine and external programs, such as for bibliography and for indexing, depending on one's needs. However, now it is hard or even impossible in some cases to know what is the 'right' workflow to process a document only by seeing document sources.

Light LaTeX Make (llmk) is yet another build tool specific for LaTeX documents, intended to remove such ambiguity in the workflows. Its aim is to provide a simple way to write down a workflow for document authors and encourage people to always explicitly show the right workflow for each document. For this goal, the design of llmk gives primary consideration to convenience and portability. For example, it supports multiple magic comment formats to enable users to easily write the workflows and it requires only `texlua`, so that it will work under any environment which has LuaTeX.

1 The goal and design concept

TeX, LaTeX, and their friends have a long history and a variety of related software has been developed, including variations of TeX engines, DVIware, and supporting programs such as BibTeX, MakeIndex, and their alternatives. Thanks to such a rich ecosystem, we have numerous options for TeX workflow to create a document. However, on the other hand, there are so many possible *workflows* for processing a LaTeX document, and therefore it is not necessarily easy to detect the right workflow only from the document sources. In addition, there is no ultimate general workflow that can be used for every purpose. Using pdfTeX is one of the typical choices for creating a document in English, but in some cases, it is reasonable to choose XeTeX or LuaTeX, e.g., if you want to use fonts installed in your system independent of TeX systems. For these reasons, LaTeX users should clearly specify the workflow for each document, at least for those documents where the sources will be seen by someone else.

There are a number of existing well-established generic build tools, such as (GNU) Make, that can be used to explicitly specify the workflows. However, for many simple LaTeX documents, such as those that require only a single pdfTeX run, it might be rather overkill to utilize such tools. As a matter of fact, people often neglect using them for small documents and leave the right workflows as mysteries. Also, it is difficult to regard knowledge of such generic build tools as an essential skill for all LaTeX users, especially considering light users and non-programmers.

Focusing on such casual use cases, I began a new project named "Light LaTeX Make" (llmk). Its goal is to encourage people to always explicitly show the workflow for each document by providing convenient ways to do it. The design of the tool is all about this purpose. First, it supports multiple magic comment formats to specify the workflows in addition to external configuration files. Magic comments are an easier way than external files, though the difference is small. It should be compatible with most LaTeX use cases, including using it on cloud services and LaTeX-specific IDEs.

Second, it is fully cross-platform. It requires only `texlua`, and thus it should work in almost all TeX environments. For instance, one does not need to install any dependency other than the TeX Live distribution.

Third, it behaves exactly the same in any environment. At this moment, llmk intentionally does not provide any method for user configuration, so that a LaTeX document with a supported workflow specification should be processed exactly in the same way, no matter where you run the program.

Overall, llmk is a tool to provide a convenient way to describe the workflow for an individual LaTeX document. It is designed more or less for simple documents and might not be suitable for large projects that require complicated workflows. For such cases, more sophisticated tools are better suited. A well-written document that already has a `Makefile` or similar is not the target of this project. In such a document, the right workflow is already explicitly shown. The major targets of llmk are small documents without unusual requirements.

There are other LaTeX-specific build tools with aims similar to llmk. The differences from such tools will be discussed later (Section 3).

2 How llmk works

In this section, only a brief summary of the usage and the mechanism in llmk is given. The details are shown in the bundled documentation.

2.1 How to write the workflows

A user of llmk can write a document's workflow in a special external file (`llmk.toml`) or in the TeX file (`*.tex`) itself. When the `llmk` command is executed without any argument, it loads the `llmk.toml` file in the working directory. If one or more names of TeX

files are specified as arguments for `llmk`, it reads the *TOML fields* in the files — these are special comment areas that are given by comment lines containing only three or more consecutive `+` characters:

```
% +++
% latex = "xelatex"
% +++
\documentclass{article}
```

Either way, you can write the workflow in the TOML format [5] — a small configuration-oriented language. This language is designed to be human-friendly and is used in numerous projects.[1]

General-purpose programming languages, such as Perl and Lua, can also be used for writing workflows and are in fact used in some TEX-related build tools, but they are too powerful and have large specifications. Smaller languages designed specifically for configuration, which are easier to learn, are better for llmk. Among the various configuration languages, including JSON and YAML, TOML is easy to parse and thus a built-in parser can be written in reasonable lines of code in pure Lua. These are the reasons why TOML was chosen for llmk.

2.2 Simple keys

There are only a few important keys for llmk configuration for casual users. For simple documents where the default configuration is applicable, using some of these keys should be enough:

`latex` (string) specifies the LATEX command to use. The default value is `"lualatex"`. Since llmk runs on `texlua`, the installation of LuaTEX is guaranteed. This is the reason that LuaTEX is chosen for the default engine. Similar keys `dvipdf`, `bibtex`, etc., are also available.

`max_repeat` (integer) sets the maximum number of repetitions. For various reasons, such as solving cross-references, llmk has a feature to repeat command executions. This key exists to prevent potential infinite loops. The default value is `5`.

`source` (string or array of strings) sets the source TEX files to process. This key is effective, and required, only in `llmk.toml`.

The following is a small example of a configuration for llmk which overrides the defaults:

```
# source TeX files
source = [ "test1.tex", "test2.tex" ]
# software to use
latex = "xelatex"
bibtex = "biber"
# misc
max_repeat = 7
```

When a value of a wrong type is given for a key, it will result in a type error *before* llmk tries actual document processing. It is designed to produce helpful error messages as much as possible, not to add confusing errors in addition to those produced by TEX engines.

2.3 Flexible control

For most simple LATEX documents, just using simple keys described in the previous section should work fine. Though such documents are the main targets of llmk, it has features to process more complicated documents if users desire to do so.

The core of flexible control in llmk is a pair of keys: `sequence` (array of strings) and `programs` (table of tables). The `sequence` array holds the names of programs in the order of execution, and the `programs` table contains detailed configuration for each program in the sequence.

The default configuration of llmk is designed to work without changes for typical LATEX documents. Users are required to write only the differences from the default, so that they do not have to write all configurations from scratch every time. The default value of the `sequence` array is as follows:

```
["latex", "bibtex",
 "makeindex", "dvipdf"]
```

Under this configuration, llmk tries to convert `*.tex` files to `*.pdf`. In case `*.dvi` is generated in the process, the `dvipdf` program (by default DVIPDFMx) is executed to convert it to a PDF. The `bibtex` and `makeindex` programs are executed only if the corresponding files (`*.bib` and `*.idx` respectively) exist, and the `latex` program is set as `postprocess` in order to make sure to rerun the LATEX command after those executions.

2.4 Supports for other formats

For the convenience of the users, llmk supports other existing magic comment formats. At present, the so-called shebang-like magic comment, which is supported by a few existing tools, notably the YaTeX mode for Emacs,[2] is supported by llmk. Writing `%#!pdflatex` in the first line of a `*.tex` file is equivalent to specifying `"pdflatex"` to the `latex` key. Other formats are also planned to be supported.

3 Differences from other tools

In response to the most frequently asked question, I will briefly explain the differences from other similar

[1] You can find the list of projects using TOML in its official wiki: `https://github.com/toml-lang/toml/wiki`.

[2] `https://www.yatex.org/`

LaTeX-specific build tools. Please note that most of these differences are just the result of different design concepts, and I would *not* call them 'advantages'. Though the aims and concepts that each tool prioritizes are a bit different from each other, they all have longer histories than llmk and thus have sophisticated designs and implementation. I have been greatly inspired from them and will continue to learn. I hope llmk can provide another useful option for LaTeX users and some new ideas and inspiration for the developers.

3.1 Latexmk and rubber

Latexmk [2] and rubber [3] are two well-known LaTeX-specific build tools. They have their own characteristics and have stable sophisticated implementations, but their purposes are slightly different from that of llmk. Their goals are to provide easy ways to process LaTeX documents; they guess how to process a document by analyzing the log files, for instance, and implicitly determine the process. In other words, they try hard to 'hide' the specific workflow from users as much as possible. In addition, for both tools, users are allowed to choose some variations, e.g., a favorite TeX engine from pdfTeX, XeTeX, LuaTeX, etc., with the command-line options. It is a useful feature, but this makes it harder to reproduce the same process for colleagues without being told another piece of information, i.e., runtime command-line options, from authors of documents.

On the other hand, llmk takes a different approach: it requires users to explicitly show the workflow to process a document either in an external configuration file (`llmk.toml`) or in a `*.tex` file. Thanks to its default configuration, it appears as if llmk determines the workflow automatically for simple configuration, often consisting of a single `latex` key, but in fact this is just a 'shorthand' for one of the typical workflows and nothing is implicitly determined. Thus, once you want to process a more complex document for which the default configuration is unsuitable, llmk will require you to specify everything explicitly. In this way, we can take advantage of both convenience and portability.

3.2 Arara and spix

Arara [1] is a newer build automation tool for LaTeX documents that has become quite popular. Its aim is close to ours: arara provides a way to describe the workflow explicitly for each document. It has a set of *rules* indicating the ways to process typical LaTeX documents and a user can specify which rules with a *directive*, which is a magic comment in the `*.tex` file. It also enables users to create their own rules by writing the details in external files, in case a suitable built-in rule is missing. Arara is a big project and is capable of processing large documents that need complicated workflows, while llmk is small and more or less focusing on simple documents.

Spix [4] identifies itself as a simpler version of arara. It also follows the idea of explicit workflow description for each document and generally focuses on simple documents. Therefore, the goals of spix and llmk are almost the same, though there are a few differences in concrete syntaxes and specifications.

One apparent strength of llmk as compared to these two tools is that llmk can be executed without installing any dependency other than from TeX systems. While arara and spix are implemented in Java and Python respectively and thus require external programs in order to use them,[3] llmk is written in pure Lua and thus can work with only `texlua` available. The specification and features of llmk are far smaller than those of arara. Instead, llmk prioritizes a uniform way to describe the workflows available for nearly all TeX environments.

4 Acknowledgements

This project has been supported by the TeX Development Fund created by the TeX Users Group (No. 29). I would like to thank all contributors and the people who gave me advice and suggestions for new features for the llmk project. I am grateful to Yusuke Kuroki for helping with the manuscript.

References

[1] Paulo Cereda, et al. *arara — The cool TeX automation tool.* `https://ctan.org/pkg/arara`

[2] John Collins. *latexmk — generate LaTeX document.* `https://ctan.org/pkg/latexmk`

[3] Sebastian Kapfer. *rubber — a building system for LaTeX documents.* `https://launchpad.net/rubber/`

[4] Louis Paternault. *SpiX — Yet another TeX compilation tool: simple, human readable, no option, no magic.* `https://ctan.org/pkg/spix`

[5] Tom Preston-Werner. *TOML: Tom's Obvious Minimal Language.* `https://toml.io/`

◇ Takuto Asakura
The University of Tokyo
Department of Computer Science
`tkt.asakura (at) gmail dot com`

[3] Neither the Java virtual machine nor the Python interpreter are included in TeX Live, or in MiKTeX.