The design concept for llmk—Light LaTeX Make

llmk

The Light LaTeX Make

Takuto ASAKURA (wtsnjp)

TUG 2020

Introduction: A demonstration

```
$ ls
duck.tex meeting.bib meeting.tex snowman.tex
```

- Which is the main source TEX file?
- Which TEX engine to use? pdfTEX? XƎTEX? LuaTEX?
- What bib program to use? BIBTEX? Biber?

> We use pdfTEX for this document. The main source is `meeting.tex` and the others are `\input` by it. We use BIBTEX for processing its bibliography.

1. Run `pdflatex meeting.tex`
2. Run `bibtex meeting`
3. Run `pdflatex meeting.tex` (for a few times, if necessary)

TeX, LaTeX, and friends: The rich ecosystem

TeX Engines pdfTeX, XeTeX, LuaTeX, (u)pTeX, ...
Bibliography BibTeX, Biber, ...
Indexing Makeindex, xcindy, mendex, ...
DVIware dvipdfm(x), dvips, ...

Workflows by Projects

The *best* workflow is differ from one project to another

Example

- pdfTeX + BibTeX + Makeindex: one of the most popular
- XeTeX and LuaTeX instead of pdfTeX: reasonable to use system fonts
- (u)pTeX + dvipdfmx: de facto standard for Japanese documents

Telling workflows

A person may use different tools depending on purpose

Example (In my case)

- pdfLaTeX for English documents as the first choice
- XeTeX if I want to use system fonts
- upTeX + dvipdfmx for Japanese documents
- LuaTeX when I want to use its Lua features

Telling which workflow to use in a project to

- human E.g., co-authors, editors, . . .
- systems E.g., text editors, IDEs, build tools, . . .

It would be ideal if we can do this in an easy and uniformed way for both human and systems.

Using generic build tools to tell the workflows?

There are numerous existing tools such as GNU Make.

- They are really useful (I have no doubt!)
- They can handle any complex workflow

Example (simple case)

Just telling “We are using pdfLaTeX for this document” is enough.

- Do we always provide `Makefile` for *all* documents?
- Workflows for *typical* small documents are not that complex

Hypothesis

In many cases, just writing `%#!pdflatex` on top of the TeX file (or similar) would be just fine.

→ I’d like to provide an interpreter for it!

llmk: The motivation

llmk

The Light LaTeX Make

Mission

Encourage people to always explicitly show the workflow for *each document* by providing convenient ways to do it!

- It should provide easy ways to specify the workflows
- It should work in various environments
- It should behave exactly the same in any environment

The design concept

1. Convenience

- it supports independent config files (`llmk.toml`)
- *and also* magic comments in TEX file
  E.g., TOML fields, shebang-like magic comments, etc.
- a default config in *do-our-best* style, which should work fine in *typical* and *simple* LATEX documents

2. Portability

- llmk is cross-platform; it works solely with `texlua`
- no user config (such as `~/.llmkrc`)
  ∵) llmk config is a means of communicating workflows

Note llmk is NOT trying to replace existing tools
→ It foucuses on simple cases that people neglect using them

Basic usage (1) llmk.toml and TOML field

Where to write workflow

- `llmk.toml` is loaded if `llmk` is executed without arguments
- TOML field in `*.tex` files specified as arguments

Example (TOML field)

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

TOML: a config format

- A small language designed for config file cf. INI, JSON, YAML
- It is used by several projects E.g., Hugo and Cargo
- Full spec: see `https://toml.io`

The basics of TOML

TOML is basically line-oriented key=value list, kind of INI extension:

- Comments begin with # and continues to EOL
- Indentation is allowed; Defining a key multiple times is invalid
- Basic data-types (types in red are not yet supported in llmk)
  - Strings (basic and literal / single- and multi-line)
  - Integer, Floats, Date-Time
  - Boolean

Example

```
# Strings
key = "value" # basic string (escape sequences are allowed)
my_favorite_primitive = '\expandafter' # literal string

# Integer
answer = 42

# Boolean
online_conference = true
```

Data structures in TOML

- Array: separated by commas; values of the same data-type
- Table: a.k.a. hash table or dictionary; no guarantee for order
- Inline table and array of tables are not yet supported in llmk

Example

```
# Array
tug = [ "Bachotek", "Rio de Janeiro", "Palo Alto", "Online" ]

# Table
[snowman] # until the next table or EOF are the key/values of this table
hat = "green"
snow = true

# Nested table
[duck.queen]
color = "pink"
# equivalent in JSON: { "duck": { "queen": { "color": "pink" } } }
```

Basic usage (2) Simple keys

- `latex` (*string*): LaTeX command to use (default: `"lualatex"`)
  → `dvipdf`, `bibtex`, etc. are similar
- `max_repeat` (*integer*): to solve cross-reference (default: 5)
- `source` (*string* or *array of strings*): source TeX files
  → only valid and required in `llmk.toml`

Example

```
1 # source TeX files
2 source = [ "test1.tex", "test2.tex" ]
3
4 # software to use
5 latex = "xelatex"
6 bibtex = "biber"
7
8 # misc
9 max_repeat = 7
```

Flexible control (1) Array sequence and Table programs

- sequence (*string array*): program names in the order of execution
- programs (*table of tables*): detailed config for each program

Example sequence

"latex" → "bibtex"

Example programs

latex

```
command: "xelatex"
auxiliary: "foo.aux"
opts: "-recorder"
```

bibtex

```
command: "bibtex"
target: "foo.bib"
postprocess: "latex"
```

Flexible control (2) Table programs

Available keys in `program` (summary)

- `command` (*string*): command to execute
- `target` (*string*): the command is run, only if the target file exists
- `opts` (*string* or *array of strings*): command-line options
- `args` (*string* or *array of strings*): command-line arguments
- `auxiliary` (*string*): the file to monitor (for cross-referencing)
- `postprocess` (*string*): the program will be run after, only if it runs

Special specifiers

The following specifiers are available in values for some keys:

- `%S`: source file which is processed
- `%T`: target file for each program
- `%B`: basename of `%S`

Default config (1)

Design concept

- Writing all config from scratch every time is meaningless
  → Providing *do-our-best* style default config, which should work for typical simple LaTeX documents
- Users only need to write *diff* from the default
- No user config (such as `~/.llmkrc`)
  → A TeX file should be processed exactly as the same anywhere

Default sequence

"`latex`"→"`bibtex`"→"`makeindex`"→"`dvipdf`"

Default config (2)

Default programs (summary)

latex

```
command: "lualatex"
auxiliary: "%B.aux"
```

dvipdf

```
command: "dvipdfmx"
target: "%B.dvi"
```

bibtex

```
command: "bibtex"
target: "%B.bib"
postprocess: "latex"
```

makeindex

```
command: "makeindex"
target: "%B.idx"
postprocess: "latex"
```

- There are default config also for dvips, ps2pdf, etc. cf. README
- The default programs table will be extended on demand

Sample use cases

Case 1: I want to use dvips instead of dvipdfmx

There is already config for dvips and ps2pdf in default `programs`
→ just modifying `sequence` is enough:

```
# pLaTeX produces DVI (not PDF)
latex = "platex" # this is shorthand for "command" in [programs.latex]

# using dvips + ps2pdf instead of dvipdf
sequence = [ "latex", "dvips", "ps2pdf" ]
```

Case 2: I want to use my own awesome program

You can use arbitraly command:

```
sequence = [ "awesome" ]

[programs.awesome]
command = "awesome"
```

Cleaning actions

Cleaning actions are available (thanks @hidaruma):

- `--clean` (`-c`) removes temporary files such as `*.aux`, `*.log`, and `*.toc`
- `--clobber` (`-C`) removes all generated files including `*.pdf` and `*.synctex.gz`

Usually, the default config should work, but you can customize:

```
# specifier %B represents the basename of source TeX file
clean_files = ["%B.log", "%B.aux", "%B.duck"]
```

Example

Executing the `--clean` action by using config in `foo.tex`:

```
$ llmk --clean foo.tex
```

Supports for other formats

Shebang-like format used by Emacs/YaTeX

```
%#!uplatex
```

```
latex = "uplatex"
```

Magic comment used by TeXShop, TeXworks, and TeXstudio

TODO

```
%!TEX program = pdflatex
%!BIB program = biber
```

```
latex = "pdflatex"
bibtex = "bibtex"
```

Why you want to write config in `*.tex` files?

- sometimes it is annoying to open another file
  (especially for small and casual use cases)
- compatible with TeX-specific IDEs and Web-based editors

Frequent Q&A (1)

How does llmk differ from latexmk?

Our goals are similar but not exactly the same

Mission for llmk

Encourage people to always explicitly show the workflow for *each document* by providing convenient ways to do it!

Thus, there are some differences in design concept:

- llmk allows users to write config in `*.tex` files
- No user config
- Less implicit decision for workflows

Does it give clear error messages?

I tried my best: llmk has typechecker and own TOML perser for this.

Frequent Q&A (2)

What make llmk LaTeX-specific?

Using it for general-purpose is possible in theory, but meaningless:

- Magic comment features are TeX-specific E.g., % is fixed
- The default config is for typical LaTeX documents
- LaTeX-oriented rerun feature until all cross-references are solved

How about security concerns?

Same as other build tools. But llmk requires explicit config.

Warning

Do not process unreliable TeX documents with llmk, especially those you get from Internet, without checking their contents!

Current status and future plan

Current version: pre-0.1.0

- No public release, even v0.1.0, yet
- You have to install it manually; Don't worry, it's a single file
  → Please visit `https://github.com/wtsnjp/llmk`
- In consideration of backword-compatibility: `llmk_version`
  → If the compatibility is broken in the future, you'll get warning

```
llmk_version = "0.1.0"
```

Future plan

- It needs reference manual; at this moment we have only README
  → I will make it ASAP and upload to CTAN
- Supporting other magic comment formats

Conclusion

Mission for llmk

Encourage people to always explicitly show the workflow for *each document* by providing convenient ways to do it!

For the above mission, llmk is designed to:

- provide several easy ways to describe the workflows
- work in various environments; it only requires LuaTeX in principle
- behave exactly the same in any environment

No more documents that no one but authors knows how to process!
Please visit `https://github.com/wtsnjp/llmk`

Thank you! Questions and comments?