

てふてふ

Sunday, September 17th, 2023

Systems and Control Laboratory, DEPARTMENT OF AERONAUTICS AND ASTRONAUTICS,
Graduate School of Engineering, **Kyoto University**

H. Asakura*

* GitHub: <https://github.com/ha-a>

Table of Contents

1. はじめに	3
2. $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ と $\mathrm{L}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$	3
2.1. $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ とはなにか	
2.2. $\mathrm{L}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ とはなにか	
2.3. その他のナツカ $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ とか $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ ナツカとかナツカ $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ ナツカとはなにか	
2.4. 番外編：文書クラス	
3. 再三のコンパイルと自動化	9
3.1. tex ファイルから pdf ファイルまで	
3.2. 自動化	
4. 結局なにをどうするのか	14
5. ちょっとした Tips	15
5.1. $\mathrm{LuaL}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ でなんとやら	
5.2. $\mathrm{L}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ で使える長さについて	
6. おわりに	19
References	19
Appendices	23
A. はじめてのらてふかんきょうづくり	
B. なぜ $\mathrm{L}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ なのか	

1. はじめに

本資料は $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ や $\mathrm{L}_{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$, `latexmk` の中身やコンパイルとはなにかについて、備忘録の意味も込めて概説するものです。インストールの方法などについて解説するわけではなく、これを読んだら環境構築が楽になるわけでもなければ、`tex` が上手に書けるようになるわけでもないです。これまで気にせずに使っていたけど、よく考えると何も知らなくて、たまに理不尽に言うことを聞いてくれないから怖い“アイツ”についてほんの少しでもその中身と歴史を知ろうというわけです。実際ネット上には、 $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ と $\mathrm{L}_{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ の区別がついてないものや、使ってはいけないもしくは強く推奨されないコマンドを紹介しているようなブログや Qiita 記事がたくさんあります。というか、当時正しくても今となっては正しくないものも多かったりします。そんなことを念頭に本資料ではなるべく、少なくとも執筆当初は嘘ではないこと、を書くようにしているつもりです。が、私のが π ではないので、間違えてることもあるかもしれません。もし間違いを見つけた場合はお近くの私までご連絡をお願いします。

2. $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ と $\mathrm{L}_{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$

まず、 $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ と $\mathrm{L}_{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ は異なるものである。しかし、そもそも名前が似てるし、両者の区別が曖昧のままでも所望の文書を作成することはとりあえずできてしまうので、別にどっちでもいいという考え方もできる。が、違いがあるから名前が違うのである。ここでは $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ と $\mathrm{L}_{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ の何が違うのかを説明し、言葉として $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ がつく様々なものについて、今後混乱・誤用しないようにざっくりと解説していく。

ところで、 $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ は $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ であり、`tex` でも `TEX` でもなければ `Tex` でもない。しかし、場合によって `TeX` と表記されることはある。また、本資料では拡張子が `*.tex` となっているものについては `tex` ファイルなどと表記することにする。 $\mathrm{L}_{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ も同様で、 $\mathrm{L}_{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ かもしくは `LaTeX` である。ちなみに、 $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ の発音はアルファベットの `X` /eks/ とし読むのではなく、無声軟口蓋摩擦音 /x/ と発音するのが正しい（諸説あるが）*。

2.1. $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ とはなにか

$\mathrm{T}_{\mathrm{E}}\mathrm{X}$ とは 1978 年に Donald E. Knuth^[1] が発表した“組版[†]ソフトウェアと組版言語”である^{[3]‡}。つまり、“文章そのもの”と“文章の構造を指定する命令”が書かれたテキストファイル（この命令を記す言語も $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ という）を読み込み、その命令に従って文章を組版するソフトウェアが $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ である。組版結果は DVI 形式 (DeVice-Independent) に書き出されるが、この `dvi` ファイルは文書の見目のレイアウト（紙面のどの位置にどの文字を配置する、などの情報）を画像形式・表示デバイス・プリンタにまったく依存しない形で記録している^[4]。結局 $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ をとてもざっくりまとめると（Knuth の思う）美しい組版がデバイスによらない形式で出力される、ということになる。

*日本語で表記するなら「てふ」であり、歴史的仮名遣いにその発音は /tʃo:/ である。

†原稿及びレイアウトの指定に従って、文字・図版・写真などを配置する作業の総称^[2]。

‡由来はギリシャ語で美術や工芸を意味する “τέχνη” /téxni/

2.2. $\mathrm{L}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ とはなにか

$\mathrm{T}_{\mathrm{E}}\mathrm{X}$ での命令は「横方向に何センチメートルの空白を作る」のように非常に原始的なもので、使い勝手が良くない。そこで $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ を使ってもっと簡単に論文やレポートを作成したいという要望から $\mathrm{L}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ は開発された^[5]。 $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ の原始的な命令を組み合わせて新しい命令（マクロ）を作ることができる（`\def` みたいなやつ）が、 $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ で文書作成するには多量のマクロが必要になるため、それらの必要なマクロが一通り揃えられた“マクロ体系”の 1 つが $\mathrm{L}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ である。

繰り返しになるが、この世にいくつも存在する $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ を楽に使うためのマクロ体系のひとつが $\mathrm{L}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ である。現代の多くの `tex` ファイルはマクロが準備された $\mathrm{L}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ で処理されることが前提であり、それを $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ で処理しようとして目的のものが得られることはまずない。次節で述べる通り、 $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ にも $\mathrm{L}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ にも沢山の種類がある。つまり、同じ $\mathrm{L}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ でも $\mathrm{pL}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ と $\mathrm{LuaL}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ とでは用意されているマクロや定義に違いがあるため、`tex` ファイルが想定するそれと処理する $\mathrm{L}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ が食い違っていればエラーが出るのも当然である。また、 $\mathrm{L}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ で定義されたマクロのおかげで成り立つ事象を $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ 全体で成り立つと理解するのは危険なのである。

2.3. その他のナツカ $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ とか $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ ナツカとかナツカ $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ ナツカとはなにか

$\mathrm{T}_{\mathrm{E}}\mathrm{X}$ と $\mathrm{L}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ 以外にも名前に $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ がつくものは数多くある。違いがわかるように簡単にまとめておこう^[6]。

(1) $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ の仲間

組版を行うソフトウェアとして、もともとの $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ の機能が様々に拡張されたものがある。例えば、

- $\varepsilon\text{-}\mathrm{T}_{\mathrm{E}}\mathrm{X}$: オリジナルの $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ と 100% の互換性を保ちながら $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ の少々不便なところを補う実用性の高い拡張。
- $\mathrm{pT}_{\mathrm{E}}\mathrm{X}$: 縦組みに対応し、和文組版できる拡張 $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ 。
- $\varepsilon\text{-}\mathrm{pT}_{\mathrm{E}}\mathrm{X}$: $\varepsilon\text{-}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ を取り入れた $\mathrm{pT}_{\mathrm{E}}\mathrm{X}$ 。
- $\mathrm{upT}_{\mathrm{E}}\mathrm{X}$: $\mathrm{pT}_{\mathrm{E}}\mathrm{X}$ の Unicode*版。つまり、 $\mathrm{pT}_{\mathrm{E}}\mathrm{X}$ の内部コードを Unicode 化している。簡単に言ってしまうと扱える文字が増える。例えば Fig. 2.1 では同じ日本語で書かれた `tex` ファイルを $\mathrm{pT}_{\mathrm{E}}\mathrm{X}$ と $\mathrm{upT}_{\mathrm{E}}\mathrm{X}$ で処理した際の出力の違いである（正確には $\mathrm{pT}_{\mathrm{E}}\mathrm{X}$ に対応したマクロ体系 $\mathrm{pL}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ を利用した場合と $\mathrm{upL}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ を利用した場合の違い）。 $\mathrm{pT}_{\mathrm{E}}\mathrm{X}$ の方では一部の文字（例えば踊り字「ゝ」や非常用漢字「沝漚」など）が出力されていないことがわかる。Unicode を用いることでこの世に存在するより幅広い文字を扱えることになる。
- $\mathrm{pdfT}_{\mathrm{E}}\mathrm{X}$: $\varepsilon\text{-}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ の拡張。DVI ではなく、PDF を直接出力する。少しだけ触れたが、`dvi` ファイルとはデバイスに依存しない、 $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ 固有の出力形式である。人間が読めるようになっているわけではないため、別に PDF などに変換する操作（`dvipdfmx` とかそういうの）が必要である。 $\mathrm{pdfT}_{\mathrm{E}}\mathrm{X}$ は出力としてこの `dvi` ではなく `pdf` ファイルを直接出力するため、そういった変換も不要なほか、ハイパーテキストリンクや目次といった PDF

*様々な言語の文字を単一の文字コードで取り扱うために開発されている文字コード^{[7],[8]}

の機能を直接扱うことができる。

- $\text{Xe}\text{\TeX}$ (/zi:tex/)^{[9],[10]}: $\varepsilon\text{-}\text{\TeX}$ の拡張。符号空間を Unicode 全体に拡大したもの（上述 $\text{up}\text{\TeX}$ の説明参照）。また、OpenType フォントに対応している。これはつまり、OS にインストールされているフォントをそのまま利用することが可能ということである^[11]。もともとの \TeX では組版を行うとき、組み立てる文字の情報を **tfm** ファイルと呼ばれるものから参照する。したがって好きなフォントを使うためにはそのための **tfm** ファイルを用意する必要があったのだが、 $\text{Xe}\text{\TeX}$ ではその必要はなく、インストールさえしてしまえば好きなフォントが使える。
- $\text{Lua}\text{\TeX}$: $\text{pdf}\text{\TeX}$ の後継。OpenType フォント、Unicode に対応しているほか、Lua という軽量スクリプト言語を利用できる。この Lua の利用によりあらゆるものがカスタマイズ可能となり、究極の柔軟性が得られる^[12]。

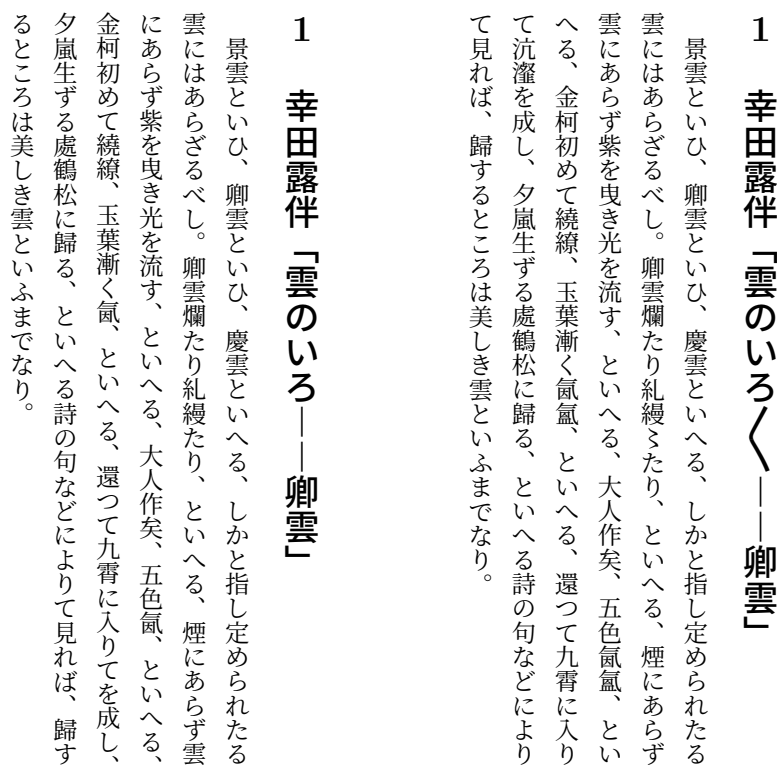


Fig. 2.1. $\text{p}\text{\TeX}$ engine (left) vs $\text{up}\text{\TeX}$ engine (right).^[13]

October 27, 2022 ConTeXt template

少し話は変わるが、 $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ のバージョンについても名前がついている：

- $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X} 2_{\varepsilon}$: 現在 (2023-09-17) の主流.
- $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X} 2.09$: 古いバージョン.
- $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X} 3$: 開発中の次期バージョン.

ここまで、組版ソフトウェアの $\text{T}_{\text{E}}\text{X}$ とその拡張、 $\text{T}_{\text{E}}\text{X}$ を楽に使うためのマクロ体系の $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ とその拡張および仲間について説明した。ここからは $\text{T}_{\text{E}}\text{X}$ の名前がついていて、 $\text{T}_{\text{E}}\text{X}$ とともに使うもの*について見ていこう。

(3) 文献管理ソフト

`\cite` 等を使って文献の引用をする場合、そのタイトルや著者、出版年、出版社等々の文献情報を管理しなくてはならない。thebibliography 環境でひとつずつ (`\bibitem`) 自分の手で文献情報を入力することも可能だが、数が増えると大変である上、表記に一貫性を持つための調整も手間である。そんなときに、 $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ と組み合わせて文献データベースから自動的に参考文献リストを作るためのツールが文献管理ソフトである。よく耳にするのは

- Bib $\text{T}_{\text{E}}\text{X}$: 書誌情報のデータベース (bib ファイル) を用いて、自動で参考文献リストを作成する。このデータベース bib ファイルのなかで各書誌情報はまずエントリごとに種別分けされ (@article, book, proceedings, etc.)、それぞれの情報を保存する (title, author, journal, etc.)。
- pBib $\text{T}_{\text{E}}\text{X}$: p $\text{T}_{\text{E}}\text{X}$ に特化。日本語の文献を扱える。
- upBib $\text{T}_{\text{E}}\text{X}$

などがあるが、最近是他にも

- Bib $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ (+Biber): $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ で用いるパッケージ。そもそも Bib $\text{T}_{\text{E}}\text{X}$ は参考文献のフォーマットは bst ファイルというもので設定されるのだが、これまた編集が大変である。そのため、よりカスタマイズが容易なのが、本パッケージである。Bib $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ のバックエンド (文献のソートを担当) として biber か bibtex を用いることになるが、biber のほうが bibtex を用いるよりさらにカスタマイズの柔軟性が上がる^[15]。

が次世代文献管理ソフトとして挙げられる。

(4) 統合開発環境

$\text{T}_{\text{E}}\text{X}$ 周りの作業がしやすいように整備されたエディタ。これさえあれば $\text{T}_{\text{E}}\text{X}$ がつかえる、というものではない。

- $\text{T}_{\text{E}}\text{X}$ Shop: macOS 専用の $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ 統合環境。シンプルな UI と直感的な操作性が特徴。
- $\text{T}_{\text{E}}\text{X}$ works: $\text{T}_{\text{E}}\text{X}$ Shop をモデルにすべてのシステム上で実現すべく開発された $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ 用統合環境。
- $\text{T}_{\text{E}}\text{X}$ studio: 豊富な機能と高度な LaTeX サポートがあり、拡張性が高い。全システム上で動く。

* $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ は $\text{T}_{\text{E}}\text{X}$ “を” 使うもの

(5) ディストリビューション

$\mathrm{T}_{\mathrm{E}}\mathrm{X}$ がまともに使えるようになるためには $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ さえあればいい，わけではない． $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ なのか $\mathrm{pT}_{\mathrm{E}}\mathrm{X}$ なのか $\mathrm{LuaT}_{\mathrm{E}}\mathrm{X}$ なのか， $\mathrm{L}_{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ なのか $\mathrm{ConT}_{\mathrm{E}}\mathrm{Xt}$ なのか， $\mathrm{BibT}_{\mathrm{E}}\mathrm{X}$ なのか Biber なのか，統合開発環境は使うのか，使うならどれか，さらに言えば，自分の文書作成では $\mathrm{LuaL}_{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ のみ使うと決心したとしても，学会によっては $\mathrm{upL}_{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ で動くことのみ想定したフォーマットを配っている場合もあり，そのフォーマットを $\mathrm{LuaL}_{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ 用書き換えるのはただ面倒で生産性があまりなく，こうなると最初から $\mathrm{upL}_{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ を準備しておくのが望ましい．これ以外にも，特定のマクロ体系を更に拡張するための“パッケージ”が必要なことがほとんどで，それらも新たに必要になるたびインストールしなくてはならない．結局， $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ を使えるようになるためには非常に多くのものをインストールしなくてはならないが，一つ一つ個別にインストールするのは大変であり，これらを一括でインストールできるのがディストリビューションである．

日本国内で多く用いられているディストリビューションは

- $\mathrm{T}_{\mathrm{E}}\mathrm{XLive}$ ^[16]

であるが，他にも

- $\mathrm{MacT}_{\mathrm{E}}\mathrm{X}$: $\mathrm{T}_{\mathrm{E}}\mathrm{XLive}$ をベースにした macOS 専用のディストリビューション．
- $\mathrm{BasicT}_{\mathrm{E}}\mathrm{X}$: $\mathrm{L}_{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ の文書作成に必要な最低限なものが含まれる macOS 専用ディストリビューション． $\varepsilon\text{-}\mathrm{pT}_{\mathrm{E}}\mathrm{X}$ などが含まれない．
- $\mathrm{MiKT}_{\mathrm{E}}\mathrm{X}$: Windows, macOS, Linux で動作するディストリビューション．日本語 $\mathrm{pT}_{\mathrm{E}}\mathrm{X}$ が含まれない．

などが存在する．

2.4. 番外編：文書クラス

さて， $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ と名のつくものについて説明してきたが，ここで $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ を名前に含まないものの，種類がいくつかあってややこしいものとして，文書クラスについて解説する^[17]． tex ファイルの一番最初に `\documentclass` とか書くあれである．

- **article**: 論文やレポートなどの一般的な文書．文章の構造は **part**, **section**, **subsection**, **subsubsection**, **paragraph**, **subparagraph** の 6 つのレベルに分けられる．
- **report**: 論文やレポートのより長い文書．構造としては **article** に **chapter** が追加される (**part** と **section** の間)．また，(デフォルトの設定では) **part** が 1 ページ使って表示される (**Part I** ○○とだけ書かれたページができる)．
- **book**: 書籍．構造は **report** と同じだが，奇数ページと偶数ページのレイアウトが変わる．例えば，**part** の示すページは必ず奇数ページ (見開きの右側) になる．

- `jarticle`: 和文横書き対応の `article`.
- `jreport`
- `jbook`
- `tarticle`: 和文縦書き対応の `article`.
- `jsclasses` (`jsarticle` etc.): 日本語 (u)p \LaTeX 用の新しい文書クラス^[18]. JIS フォントメトリックを使用している. 簡単に言えば「文字の組み方」が綺麗に, 読みやすくなる. 例えば「ちょっとチャックをチェック」のように小さい文字を並べて表示する際 `jarticle` などの min10 フォントメトリックでは不自然に詰められ, 美しくなかったり読みづらかったりするものが `jsclasses` では解消される^[19]. 他にも日本語中の英語の組み方などに差が出る.
- `ltjsclasses` (`ltjsarticle` etc.): `jsclasses` を Lua \LaTeX 用に改変したもの.
- `bxjsclasses` (`bxjsarticle` etc.): `jsclasses` の派生で, (u)p \LaTeX 依存の部分を切り離したもの. つまり, Lua \LaTeX など他のエンジンを用いても綺麗な日本語文書を作成できる.
- `jlreq`: 「日本語組版処理の要件」*を満たす Lua \LaTeX , (u)p \LaTeX 用の文書クラス.
- `beamer`: スライド. 「Beamer のすゝめ」も読んでね.

3. 再三のコンパイルと自動化

ここまで `tex` ファイルから文書出力を得ることを曖昧に“処理”と表現してきた. この“処理”とは実際にはどのような“処理”を行っているのかについて説明していく. 具体的には, pdf \TeX 系なのか否かや相互参照・文献情報の有無によって必要な処理の種類・順番や回数が変わる. 最近では `latexmk` というコマンドのおかげで, このことを意識しなくてもいいことが多いが, この処理について知っておくことは `latexmk` 丸投げよりも効率がいい処理も可能で, デバッグの役にも立つので, 知っておいて損はないと思う. なお, `latexmk` という名前からも分かる通り, 今後の話は \LaTeX まわりの話である.

3.1. tex ファイルから pdf ファイルまで

`tex` ファイルから pdf ファイルを出力するまでの処理の流れについて簡単に説明する^[22]. なお, 以下では作成した `tex` ファイルの名前を `hoge.tex` とする. ユーザは `hoge.tex` の中身に従い, 適切なマクロ体系を選択する. というよりも, 日本語を使うのか, Unicodeを使うのか, 好みのフォントを選びたいのか, といった目的に応じて選択すべきマクロ体系があり, それに合わせて `hoge.tex` を作成する. 本節では, 例として Lua \LaTeX +upB \TeX を用いることにする.

*標準化団体 W3C (World Wide Web Consortium) の技術ノート “Requirements for Japanese Text Layout”^{[20],[21]} のことで, 字詰めやルビ, ページ構成や図表に関してなどあらゆる規定が書かれている.

(1) はじめてのらてふこんぱいる

選択したマクロ体系で処理をする:

```
lualatex hoge.tex
```

これにより、次のファイルが生成される:

- 出力ファイル
 - `hoge.dvi`: \LaTeX , $(\text{u})\text{\pLaTeX}$ などを用いた場合
 - `hoge.pdf`: `pdf \LaTeX` , `X \LaTeX` , `Lua \LaTeX` などを用いた場合
- 補助ファイル `hoge.aux`: 相互参照・ラベルの情報を保存する.
- ログファイル `hoge.log`: コンパイル時の記録. エラーや警告も保存される.

このとき、もともと相互参照に関する `hoge.aux` を読み込まずに `hoge.tex` のコンパイルを行ったため、今回の出力で適切な相互参照はできていない. 相互参照するべきところ (`\ref`) では “??” が表示され、`hoge.log` には参照が見つからなかった旨の警告*が記録される. これは一度の処理を高速・省メモリで終えるために、1 回あたりのコンパイルでは `tex` ファイルを最初から最後まで順に読み込みながら出力を行っているからである. このコンパイルの最中に補助ファイル (`hoge.aux`) にラベルを前から順に保存していく.

次のステップは文書の中に相互参照が含まれるかどうかによって以下のように場合分けされる:

	次の処理
参照したい文献 <code>\cite</code> がある場合	(2) 参考文献処理
参考文献以外の相互参照 <code>\ref</code> のみある場合	(3) 相互参照を処理する \LaTeX コンパイル
相互参照も参考文献もない場合	(5) 文書の pdf 出力を得る

(2) 参考文献処理

所望の形に整形された参考文献の一覧を作る:

```
upbibtex hoge.aux
```

具体的には、(1) はじめてのらてふこんぱいる によって `hoge.aux` ファイル内に保存された文献リストの雛形から所望の形の文献リストが `hoge.bbl` に生成される^[23]. `hoge.bbl` の中身は `thebibliography` 環境であり、`BibTeX` 等を使わずに手動で `\bibitem` などとちまちま書いていくアレである. つまり、`hoge.bbl` にはそれぞれの情報を含めた文献項目 (`bibitem` とか `entry`) が所望の形式で出力できるように収められている.

次のステップは (3) 文献リストを出力する \LaTeX コンパイル.

* \LaTeX Warning: There were undefined references.

(3) 文献リストを出力する L^AT_EX コンパイル

hoge.bbl を使って、文献リストを出力しつつ、補助ファイル hoge.aux に文献に関わる相互参照情報を保存する。

```
lualatex hoge.tex
```

ちなみに、(1) で作られた相互参照は今回の L^AT_EX コンパイル時に読み込まれた hoge.aux を使って、解決される。なぜならこのコンパイル時にはすでに 1 回目のコンパイルで作成された hoge.aux があり、そこに文献参照以外の必要な相互参照の情報が保存されているからである。

次のステップは (4) 相互参照を処理する L^AT_EX コンパイル。

(4) 相互参照を処理する L^AT_EX コンパイル

hoge.aux の相互参照を解決し、hoge.dvi か hoge.pdf を出力する：

```
lualatex hoge.tex
```

補助 aux ファイルの中に前回保存したラベルがあるため、その番号を採用・出力していく。御存知の通り、出力が dvi ファイルになるか pdf ファイルになるかはマクロ体系として何を使うかに依存する。直接 pdf ファイルが出力される系統を用いているのであれば、ここで文書は完成である。

マクロ体系	出力	次の処理
L ^A T _E X, (u)pL ^A T _E X	hoge.dvi	(5) 文書の pdf 出力を得る
pdfL ^A T _E X, XeL ^A T _E X, LuaL ^A T _E X	hoge.pdf	-

(5) 文書の pdf 出力を得る

すでに述べたが、dvi ファイルとは DeVice-Independent file のことで、文書の見た目のレイアウトを画像形式・表示デバイス・プリンタなどにまったく依存しない形で記録している^[24]。これは人間が読むように設計されておらず、DVI ドライバと呼ばれる別のプログラムに入力することで、画像として表示したり、文書形式に変換したりできる。主な DVI ドライバとしては次のようなものがある：

- dvipdfmx: dvi→pdf. 日本語文書作成においては最もよく使われている。そもそも海外では pdfL^AT_EX とその周辺が主流であり、dvi ファイルは生成されないのである。
- dvips: dvi→ps (PostScript). この ps ファイルとは pdf ファイルの基礎となったものである^[25]。
- dviout: (u)pL^AT_EX 対応の DVI プレビューア。最近では推奨されない。

無限ループ

文書作成の具体的な流れはここまでで説明したとおりである。基本的な相互参照にしろ、文献の参照にしろ、ラベルを適切に補助 `aux` ファイルに保存できれば、その後もう一回コンパイルすることですべての参照がきちんと対応して出力される。つまり `\label`、`\ref` があって、文献参照が必要ない（もしくは `bbl` ファイルに変更がない）場合には2回のコンパイルによって出力が得られることになる。これは基本的に間違っていないが、実際には2回では済まない場合がある。具体的には、参照したいラベル先が不明でとりあえず“??”と出力されていた部分に、ラベルがついている数字や文献を表す記号などが正しく代入されることで改ページの場所や図表の位置が変わってしまった場合、ラベルと番号の対応付けが変わってしまい、参照番号が正しく振られていないことになってしまうことがある*。そこで次のような収束しないおもちゃを作ることができる^[26]：

```

1 \documentclass{article}
2 \pagenumbering{roman}
3 \setlength{\textwidth}{5em}
4 \setlength{\textheight}{2\baselineskip}
5 \begin{document}
6   \setcounter{page}{999}
7   \noindent The Page Number \pageref{foo} is\label{foo} shaking up and down.
8 \end{document}

```

この文書は、999 ページ始まりで、横 5 文字強、縦 2 行の本文スペースに文章を書き連ねている。`\label` は“is”の場所を示し、そのページ数を `\pageref` でローマ数字で表記している。実際にコンパイルすると、

- i. はじめてのコンパイルでは、補助 `aux` ファイルが存在しないため、`\pageref` で参照すべきものがなく、“??”が出力される。その後 `\label` で“is”のページ番号 999 が補助ファイルに保存される。
 - ii. 参照が解決していないので、2 度目のコンパイルを行う。今度は先程作った補助ファイルがあるので、`\pageref` 部分で適切に“is”の位置 (999) がローマ数字 `cmxcix` で表示される。その結果、“is”は次の 1000 ページ目に移動してしまい、`\label` でこの 1000 ページの情報が再び補助 `aux` ファイルに保存される。
 - iii. “??”はないが、参照先と参照元があってないので、3 度目のコンパイルを行う。今度は `\pageref` では先程保存した 1000 ページの情報をローマ数字 `m` で表示する。今度は“is”が 999 ページ目に戻ってしまい、補助ファイルには再び 999 ページの情報が保存される。
- ということで以下 ii. と iii. 繰り返す。

この `tex` ファイルをコンパイルした結果（のうち最初のページ、つまり 999 ページ）を Fig. 3.1 に示す。 $n \in \mathbb{Z}_+$ 回コンパイルしたあとの出力において、ラベルが付けられる“is”のページ番号とそれを参照しているはずの `\pageref{foo}` の出力は必ず一致しない。このおもちゃに面白い以上の意味は見いだせないが、相互参照のために `aux` ファイルが必要で、同様に複数回のコンパイルが必要なこともわかるであろう。

*LaTeX Warning: Label(s) may have changed. Rerun to get cross-references right.

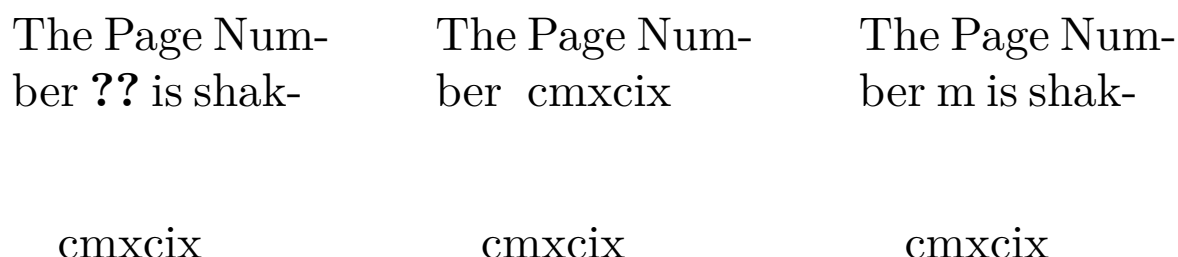


Fig. 3.1. The example of not converging cross-referencing. Left: former page (page 999) of first compilation results. Center: That of even numbered compilations. Right: that of odd numbered compilations.

3.2. 自動化

ここまででわかるように、文献を含む相互参照をきちんと機能させるためには、補助ファイル `hoge.aux` の存在も、それを使って複数回コンパイルすることも必要不可欠である。一方で、そのコンパイル回数はある程度目処がつくとは言え、正確に予測することは困難である。このような状況で、変更のたびに参照が失敗している箇所がないか確認しながら自分の手でコンパイルしていくのは大変面倒である。

この「複数回コンパイル等に起因する煩雑さ」を解決・軽減する方法として有名なのはもちろん

- **latexmk**: 現在最も有名な自動化ツール。複数回の実行に加え、**BibTeX** 等の関連ツールの実行も自動で行える。**latexmk** は `aux` ファイルの内容が変化したときにコンパイルし直すことになっている。例えば `aux` ファイルに時刻の情報を含むようなコードを書けば、コンパイルのたびに `aux` ファイルが変化することになる*。

であるが、自動化ツールも何種類か存在する。いくつかを下記に示すが「煩雑だな」と感じるのは同じでもそれを解消するための設計指針や思想は異なっているものも多い。

- **arara**: 手順を書いたファイルを事前に用意して、ビルドする。有名どころ (`pdflatex` や `biber` などなど) は事前に用意されている。
- **ptex2pdf**: `(u)platex` と `dvipdfmx` をまとめて実行する。
- **l1mk**: Light Latex MaKe.

*`latexmk` では上限回数が設定されており、これにより無限の彼方に飛ばされることはない^[27]。

4. 結局なにをどうするのか

ここまでで、 $\text{T}_{\text{E}}\text{X}$ があって、その拡張としていくつか種類があり、 $\text{T}_{\text{E}}\text{X}$ たちを楽に使うための $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ も何種類が存在することがわかった。tex ファイル中では文書クラスも適宜選ばなくてはならない。文献管理の方法も一つではなく、コンパイルを自動化してくれるツールも（一つシェア率が高いものがあるとは言え）他にも選択肢がある。もちろん、日本語を扱えないものや明らかに目的が違うものなどもあるため、その中から選択していく必要はあるが、それでも選択肢が何個もあるのは確かである。では、我々はなにを選択するべきなのか。

結論

$\text{LuaL}_{\text{A}}\text{T}_{\text{E}}\text{X}$ + $\text{upBibT}_{\text{E}}\text{X}$ (or $\text{BibL}_{\text{A}}\text{T}_{\text{E}}\text{X}$) + bxjclasses + latexmk

これは結局カスタマイズ性を優先している方法である。その中で個人的には一番「特別なにも考えなくてもいいレシピ」である。 $\text{LuaL}_{\text{A}}\text{T}_{\text{E}}\text{X}$ を使う理由としては

- Unicode に対応しているので、出力できない文字を気にしなくて良い。
- dvi ファイルという $\text{T}_{\text{E}}\text{X}$ 固有のよくわからないものを生成する必要もない。

などが挙げられるが、さらにもう少し $\text{T}_{\text{E}}\text{X}$ を自在に扱いたいと思ったとして

- OpenType 対応なので、フォントの設定が楽。
- 扱いが難しいとされる原始的な $\text{T}_{\text{E}}\text{X}$ 言語を扱わなくても Lua によって柔軟なカスタマイズが可能。

などもメリットであろう^[28]。一方で、デメリットとして $\text{LuaL}_{\text{A}}\text{T}_{\text{E}}\text{X}$ は処理が遅いことが挙げられるが、以前は 20 倍程度遅かったところ、最近では 2 倍程度に抑えられており^[29]、今後さらに速くなっていくと考えられる。

逆に (u)p $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ を使わない理由としては、欧文と和文でフォントの扱いが違うためにせっかくの文書が美しくならなかったり、近いうちにまともに動かなくなるなんて話もあったり^{[30],[31]}*するからである。

一方で $\text{upBibT}_{\text{E}}\text{X}$ を用いるのは簡単に

- 学会であれば基本的に参考文献のフォーマットは決まっている (bst ファイルが配布される)。
- 自分で参考文献のところを必要以上に凝りたいと思うことが少ない。

ために、比較的処理の遅い $\text{BibL}_{\text{A}}\text{T}_{\text{E}}\text{X}$ を用いる必要性を感じないからである。しかし、せっかくなら細かいところまで気にしたい、のであれば $\text{BibL}_{\text{A}}\text{T}_{\text{E}}\text{X}$ を用いるのもよいだろう。

先に述べたコンパイルの煩雑さを回避するためにも自動化は必要不可欠であり、結局広く使われている latexmk をここではおすすめしている。というの

*これは先述した開発中の $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ 次期バージョン $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}3$ では p $\text{T}_{\text{E}}\text{X}$ に対応しない可能性があるからである。他の新しい手段があるのに、日本だけでガラパゴス化されたものに対応しないかもしれないというのも理解できる。しかしまあ、おそらく結局誰かがどうにかして実際に使えなくなることはないのだろうが、それでもそのようなリスクを抱えるものをわざわざ使う理由も特にない。

- 広く使われているからこそ、使い方などの情報が豊富に見つかる。
- Visual Studio Code など公式にサポートされている*。
- 一度設定ファイル `latexmkrc` を作ってしまえば、なにも考えずにその後使いまわすこともできる。

からである。

一方で、もしコンパイルの速さを優先するなら、

`up \LaTeX + upBib \TeX + dvipdfmx + jsclasses (+ latexmk)`

なども考えられよう。この場合、いつでも使える `latexmk` で自動化せずに、必要な回数コンパイルするレシピを作って処理するのがより速いだろう。例えばただの授業の課題など、体裁にこだわる必要のない、ただか数ページのものを作るのならコンパイルの速度を優先するのがいい。ただ、どうせコンパイル中も編集は行えることを加味すれば Lua \LaTeX を利用するのがいいように思える。

5. ちょっとした Tips

ここまでで、基本的な \TeX の実行について幾分イメージが付きやすくなったと思う。これより先は「へえ、そうだったのか」と個人的に思ったような \TeX 周りの何事かを完全に備忘録として記していく。

5.1. Lua \LaTeX でなんとやら

(1) フォント

本資料でもフォントの調整は行っている。このように途中で変えるのもお手の物である。

Table 5.1. Fonts used in this document.

Usage	Font Name
English main	TeX Gyre Termes
Japanese main	Hiragino Mincho ProN
English sans serif	Verdana
Japanese sans serif	Hiragino Kaku Gothic Pro
Typewriter	New Computer Modern Mono10
Footer	3270 Narrow
English in section and subsection	American Typewriter

*実は VScode 上でのエラーや警告の拾い方は `latexmk` だけ特別扱いされる^[32]。

(2) Lua コード

Lua¹TeX の大きな特徴の一つはもちろん“Lua”である。Lua は、高速な動作と高い移植性、組み込みの容易さが特徴であるスクリプト言語である^{[33],[34]}。派生の LuaJIT は動的型付けのスクリプト言語で最速言語である（らしい）。

そんな高速な Lua を TeX にならめて使っていく^{[35],[36],[37],[38]}。もちろんプログラム言語なので普通に計算を行うことができ、わざわざ別のなにかを立ち上げることなく、計算して結果を出すことができる。例えば

$$\sqrt{2} = \$ \backslash \text{directlua}\{\text{tex.sprint}(\text{math.sqrt}(2))\}$$

などと書けば

$$\sqrt{2} = 1.4142135623731$$

と表示される。

他にも引数を持つ自作関数を `\newcommand` でつくと、本文中で `\sumSquare{}{}` と書く度に、引数の小さい方から大きい方までの整数二乗和を計算させられる。

```

1 \newcommand{\sumSquare}[2]{%
2   \directlua{
3     function sumSquareFunc(x,y)
4       if x > y then
5         x, y = y, x
6       end
7       local s = 0
8       for i = x, y do
9         s = s + i^2
10      end
11      tex.sprint(math.floor(s))
12    end
13  }%
14  \directlua{ sumSquareFunc(#1,#2) }%
15 }
```

```

\sumSquare{1}{10}    = 385
\sumSquare{-2}{1}     = 6
\sumSquare{50}{-50}  = 85850
```

もちろん文字列も扱える。もし間違えて猫を犬にしまったときは、以下の自作関数 `\displace` を用いて、`\displace{dog}{cat}{dog cat dog cat}` と書けば、“dog cat dog cat”を“cat cat cat cat”と猫だらけにできる。

```

1 \begin{luacode*}
2   function myDisplace(str_before, str_after, sentence)
3     local sentence_after = ""
4     local l = #str_before - 1
5     local i = 0
6     while true do
7       if string.sub(sentence, i, i+1) == str_before then
8         sentence_after = sentence_after .. str_after
9         i = i + 1
10      else
11        sentence_after = sentence_after .. string.sub(sentence,i,i)
12      end
13      i = i + 1
14      if i > #sentence then break end
15    end
16    tex.sprint(sentence_after)
17  end
18 \end{luacode*}
19 \newcommand{\displace}[3]{ \directlua{myDisplace("#1","#2","#3")} }
```

Lua_{TEX} は Lua を L_AT_EX と絡めて使えるため、文書の様式・組版を変えることもできる。

```

1 \documentclass[twocolumn]{article}
2 \usepackage{lipsum}
3 \usepackage{luacode}
4 \setlength{\columnsep}{1cm}
5 \pagestyle{empty}
6
7 \begin{luacode*}
8 vibrateLine = function (head, group, size)
9     i = 1.5
10     for list in node.traverse(head) do
11         i = i + 0.13
12         if list.id == node.id("hlist")
13             then
14             list.shift = list.shift - (1200000 * (math.sin (i)))
15         end
16     end
17     return head
18 end
19 \end{luacode*}
20
21 \directlua{%
22     luatexbase.add_to_callback('vpack_filter',vibrateLine,"s
23 }
24
25 \begin{document}
26     \lipsum[1-10]
27 \end{document}

```

[illegible]

Fig. 5.1. Example of Lua coding^[36]

5.2. L^AT_EX で使える長さについて

(1) 单位

長さの単位としてはオーソドックスでなものが多い。ここで示したものの以外にも様々あるが、使いたい場面がない。

(2) 環境に依存する長さ

一方、環境依存で定義された長さもいくつかある^[39]。特に図の挿入時には紙面の横幅に対する割合などを指定すれば、見た目の調整が楽である。定義から分かる通り、基本的な長さとしては`\paperwidth>\textwidth≥\linewidth`である。実際、これらの長さ（の半分）の線を引いてみる

_____	.5\paperwidth
_____	.5\textwidth
_____	.5\linewidth (in one column)
_____	.5\linewidth (in 2 cols)
_____	.5\columnwidth

Table 5.2. List of length units available in L^AT_EX.

unit	name	description
pt	point	1 pt \simeq 0.35 mm
pc	pica	1 pc = 12 pt
mm	millimeter	1 mm \simeq 2.85 pt
in	inch	1 in = 25.4 mm = 72.27 pt
em	/em/	the width of an “M” in the current font
ex	/exs/	the height of an “x” in the current font
zw	zenkaku width	the width of a zenkaku-kanji in the current font
zh	zenkaku height	the height of a zenkaku-kanji in the current font

Table 5.3. List of common length macros.

macros	description
<code>\paperwidth</code>	width of the page
<code>\paperheight</code>	height of the page
<code>\textwidth</code>	width of the text on the page
<code>\textheight</code>	height of the text on the page
<code>\linewidth</code>	width of the line in the current environment
<code>\columnwidth</code>	width of the column
<code>\columnsep</code>	distance between columns
<code>\parindent</code>	indentation of paragraphs
<code>\parskip</code>	extra space between paragraphs
<code>\baselineskip</code>	vertical distance between lines in a paragraph

Table 5.4. List of space control macros.

macros	definition
<code>_</code>	0.33 em
<code>~</code>	0.33 em (non-breakable)
<code>\,</code>	3/18 em
<code>\:</code>	4/18 em
<code>\;</code>	5/18 em
<code>\quad</code>	1 em
<code>\qquad</code>	2 em
<code>\!</code>	-3/18 em
<code>\hspace{}</code>	flexible horizontal space
<code>\vspace{}</code>	vertical

(3) 空白

空白の制御を行うときにいつも忘れるのでメモしておく^[40]。「カンマ < コロン < セミコロン」の順に縦に伸びていくので、これで覚えればいいか。

空白関連でもう一つ。T_EX 処理の際には基本的にピリオドがあると終止符だと認識し、その後ろには大きめの空白が入る。しかし例外があり、大文字の後ろのピリオドは終止符認識しない（略語などに対応するため）。したがって、大文字の単語が文末にくと、終止符にも関わらず例外として処理されてしまい、次の文との間の空白が本来より小さくなってしまふ。そこでピリオドの直前に大文字以外のなにかを挟めばよいため、例えば`\@.`などが用いられる。

6. おわりに

$\text{T}_{\text{E}}\text{X}$ と $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ の違いから始まり, その他の $\text{T}_{\text{E}}\text{X}$ と名のつくものについて整理してきた. さらに `tex` ファイルから `pdf` ファイルを作るまでの処理の流れも概説した. 確かに, 文章を書いてるだけなのに (Microsoft Word と違って), よくエラーが出る. しかしそこで苦手意識を持って諦めてほしくない. 少しでも $\text{T}_{\text{E}}\text{X}$ やらその周辺の仕組み・大枠を知ること, よくわからん謎のエラーに対して, そのエラーがとりあえずどこらへんの絡まりが原因なのかを推測できるようになっているはずである. 私の個人的な願いとしては, せっかく $\text{T}_{\text{E}}\text{X}$ を使うなら, 満足できる美しさを目指してぜひとも色々カスタマイズして自分好みになるようにして欲しい.

そんなこれからの楽しい $\text{T}_{\text{E}}\text{X}$ ライフの一助となれば幸いです.

References

- [1] “Don Knuth’s home page.” URL: <https://www-cs-faculty.stanford.edu/~knuth/> (visited on Oct. 26, 2022) (cit. on p. 3).
- [2] “組版 - Wikipedia.” URL: <https://ja.wikipedia.org/wiki/%E7%B5%84%E7%89%88> (visited on Oct. 25, 2022) (cit. on p. 3).
- [3] Arthur Reutenauer. “A brief history of $\text{T}_{\text{E}}\text{X}$, volume II.” In: *EuroBachoTEX 2007* (2008), p. 68 (cit. on p. 3).
- [4] “ $\text{T}_{\text{E}}\text{X}$ と $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ の違い - ラング・ラグー.” URL: <https://blog.wtsnjp.com/2016/12/19/tex-and-latex/> (visited on Oct. 26, 2022) (cit. on p. 3).
- [5] “ $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ - $\text{T}_{\text{E}}\text{X}$ Wiki.” URL: <https://texwiki.texjp.org/LaTeX#f2915fa5> (visited on Oct. 26, 2022) (cit. on p. 4).
- [6] “ $\text{T}_{\text{E}}\text{X}$ Wiki.” URL: <https://texwiki.texjp.org/> (visited on Oct. 25, 2022) (cit. on p. 4).
- [7] “Unicode - Wikipedia.” URL: <https://ja.wikipedia.org/wiki/Unicode> (visited on Oct. 26, 2022) (cit. on p. 4).
- [8] 師茂樹. “Unicode とのつきあい方—漢字文化圏を中心に—.” In: コンピュータ & エデュケーション 27 (2009), pp. 12–17 (cit. on p. 4).
- [9] “ $\text{X}_{\text{E}}\text{T}_{\text{E}}\text{X}$ - Wikipedia.” URL: <https://ja.wikipedia.org/wiki/XeTeX> (visited on Oct. 26, 2022) (cit. on p. 5).
- [10] “ $\text{X}_{\text{E}}\text{T}_{\text{E}}\text{X}$ - $\text{T}_{\text{E}}\text{X}$ Wiki.” URL: <https://texwiki.texjp.org/?XeTeX> (visited on Oct. 27, 2022) (cit. on p. 5).
- [11] “ $\text{X}_{\text{E}}\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ で日本語する件について.” URL: <http://zrbabbler.sp.land.to/xelatex.html> (visited on Oct. 26, 2022) (cit. on p. 5).
- [12] 八登崇之. “日本人の知らない $\text{T}_{\text{E}}\text{X}$ ” 2010 (cit. on p. 5).
- [13] 幸田露伴. “雲のいろいろ.” 反省雑誌, 1987 (cit. on p. 5).
- [14] “Con $\text{T}_{\text{E}}\text{X}$ t - $\text{T}_{\text{E}}\text{X}$ Wiki.” URL: <https://texwiki.texjp.org/?ConTeXt> (visited on Oct. 26, 2022) (cit. on p. 6).

REFERENCES

- [15] “Biber (LaTeX) - Wikipedia.” URL: [https://en.wikipedia.org/wiki/Biber_\(LaTeX\)](https://en.wikipedia.org/wiki/Biber_(LaTeX)) (visited on Oct. 30, 2022) (cit. on p. 7).
- [16] “TeX Live をホンキで語る - Acetaminophen’s diary.” URL: <https://acetaminophen.hatenablog.com/entry/texadvent2016-20161205#main-7> (visited on Oct. 27, 2022) (cit. on p. 8).
- [17] “標準的なクラスファイルの一覧 - T_EXWiki.” URL: <https://texwiki.texjp.org/?%E3%82%AF%E3%83%A9%E3%82%B9%E3%83%95%E3%82%A1%E3%82%A4%E3%83%AB%E4%B8%80%E8%A6%A7> (cit. on p. 8).
- [18] 奥村晴彦. “ \LaTeX 2_ε 新ドキュメントクラス.” In: (2023). URL: <https://mirror.math.princeton.edu/pub/CTAN/macros/jptex/latex/jsclasses/jsclasses.pdf> (cit. on p. 9).
- [19] “LaTeX の「アレなデフォルト」傾向と対策 - Qiita.” URL: https://qiita.com/zr_tex8r/items/297154ca924749e62471 (cit. on p. 9).
- [20] “日本語組版処理の要件（日本語版）.” URL: <https://www.w3.org/TR/2020/NOTE-jlreq-20200811/> (cit. on p. 9).
- [21] “JIS X 4051 - Wikipedia.” URL: https://ja.wikipedia.org/wiki/JIS_X_4051 (cit. on p. 9).
- [22] “LaTeX コンパイル等 - Yamamoto’s Laboratory.” URL: <http://www.yamamo10.jp/yamamoto/comp/latex/run/run.php> (visited on Oct. 27, 2022) (cit. on p. 9).
- [23] “参考文献の書き方 - 星野光.” URL: <https://www.eng.u-hyogo.ac.jp/faculty/hoshino/pc/latex/bibtex/> (visited on Oct. 27, 2022) (cit. on p. 10).
- [24] “DVI - Wikipedia.” URL: [https://ja.wikipedia.org/wiki/DVI_\(%E3%83%95%E3%82%A1%E3%82%A4%E3%83%AB%E3%83%95%E3%82%A9%E3%83%BC%E3%83%9E%E3%83%83%E3%83%88\)](https://ja.wikipedia.org/wiki/DVI_(%E3%83%95%E3%82%A1%E3%82%A4%E3%83%AB%E3%83%95%E3%82%A9%E3%83%BC%E3%83%9E%E3%83%83%E3%83%88)) (visited on Oct. 27, 2022) (cit. on p. 11).
- [25] “PostScript - Wikipedia.” URL: <https://ja.wikipedia.org/wiki/PostScript> (visited on Oct. 27, 2022) (cit. on p. 11).
- [26] “ \LaTeX の相互参照はいつでも解決（収束）するのか?.” URL: https://qiita.com/kuroky_plus/items/28fc1f4e5610aa5c956e (visited on Oct. 27, 2022) (cit. on p. 12).
- [27] “TeX の実行あれこれ - 雑記帳.” URL: <https://blog.miz-ar.info/2016/12/running-tex/> (visited on Oct. 27, 2022) (cit. on p. 13).
- [28] “「日本語 LaTeX」が多すぎる件について.” URL: <https://www.slideshare.net/zr-tex8r/latex-239371115> (visited on Oct. 30, 2022) (cit. on p. 14).
- [29] “LuaTeX が実はそんなに遅くないかも知れない件 - マクロツイーター.” URL: <https://zrbabbler.hatenablog.com/entry/20170730/1501402087> (visited on Oct. 30, 2022) (cit. on p. 14).
- [30] “pLaTeX が本格的にやばいかもという話 - Acetaminophen’s diary.” URL: <https://acetaminophen.hatenablog.com/entry/2021/06/18/022108> (visited on Oct. 30, 2022) (cit. on p. 14).
- [31] “【詳細】 どうして pLaTeX は LaTeX3 でピンチになるのか - LuaLaTeX Lab.” URL: <https://lualatexlab.blog.fc2.com/blog-entry-44.html> (cit. on p. 14).

-
- [32] “LaTeX Workshop で llmk してみたらアレだった件.” URL: <https://zrbabbler.hatenablog.com/entry/2020/09/23/190840> (visited on Oct. 30, 2022) (cit. on p. 15).
- [33] “Lua - Wikipedia.” URL: <https://ja.wikipedia.org/wiki/Lua> (cit. on p. 16).
- [34] “高速スクリプト言語 [Lua] を始めよう.” URL: <https://ie.u-ryukyu.ac.jp/~e085739/lua.hajime.html> (cit. on p. 16).
- [35] “徹底攻略！ LuaLaTeX で Lua コードを「書く」ためのコツ.” URL: https://qiita.com/zr_tex8r/items/af2905bc93ac2c936a67 (cit. on p. 16).
- [36] “TeX で使うプログラミング言語まとめ.” URL: <https://zenn.dev/k16/articles/e7690b036e534c#luatex%E3%81%AElua> (cit. on pp. 16, 17).
- [37] “Pattern syntax using luacode.” URL: <https://tex.stackexchange.com/questions/499915/pattern-syntax-using-luacode> (cit. on p. 16).
- [38] “思わず Lua で LaTeX してみた.” URL: <http://zrbabbler.sp.land.to/lualatexlua.html> (cit. on p. 16).
- [39] Kent McPherson. “Displaying page layout variables.” In: (2021). URL: <https://ftp.jaist.ac.jp/pub/CTAN/macros/latex/required/tools/layout.pdf> (cit. on p. 17).
- [40] “spacing in LaTeX.” URL: <https://www.sascha-frank.com/spacing.html> (cit. on p. 18).

Appendices

A. はじめてのらてふかんきょうづくり

○ Windows (要検証)

- i. TeX Live のページ (<https://www.tug.org/texlive/acquire-netinstall.html>) から `install-tl-windows.exe` をダウンロード.
- ii. ダウンロードしたファイルを実行. のち, 待ち.

○ Mac

- i. Homebrew をダウンロード.

```
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

- ii. MacTeX をインストール. のち, 待ち.

```
brew install --cask mactex-no-gui
sudo tlmgr update --self --all
sudo tlmgr paper a4
```

- iii. ここで一旦確認. 例えば `main.tex` を作って, コンパイル: `latex main.tex`

```
\documentclass{article}
\begin{document}
  Hello, \LaTeX!
\end{document}

dvipdfmx main.dvi
open main.pdf
```

- iv. Lua \LaTeX も確認. `lualatex main_lua.tex`

```
\documentclass[a4paper,lualatex,ja=standard]{bxjsarticle}
\begin{document}
  Hello, \LaTeX! こんにちは, \LaTeX!
\end{document}
```

- v. `latexmk` を使うための設定ファイル `latexmkrc` を作成. Lua \LaTeX なら以下.

up \LaTeX なら 9 行目の `pdf_mode` を 3 にする.

```
1 #!/usr/bin/env perl
2 $latex = 'uplatex %0 -synctex=1 -file-line-error -halt-on-error %S';
3 $pdflatex = 'pdflatex %0 -synctex=1 -file-line-error -halt-on-error %S';
4 $lualatex = 'lualatex %0 -synctex=1 -file-line-error -halt-on-error %S';
5 $xelatex = 'xelatex %0 -synctex=1 -file-line-error -halt-on-error %S';
6 $biber = 'biber %0 --bblencoding=utf8 -u -U --output_safechars %B';
7 $bibtex = 'upbibtex %0 %B';
8 $dvipdf = 'dvipdfmx %0 -o %D %S';
9 $pdf_mode = 4; # or 3 (for up $\text{\LaTeX}$ )
```

- vi. もっかい確認. `latexmk main.tex` でコンパイル. `main.tex` には相互参照や文献参照を追加してもいい. `pdf_mode` には注意.
- vii. Visual Studio Code をインストール.
- viii. VSCode で拡張機能 LaTeX Workshop をインストール.
- ix. VSCode でコマンドパレットから `settings.json` を開き, 以下を追加する:

```

1  "latex-workshop.latex.recipes": [
2    {
3      "name": "latexmk",
4      "tools": [
5        "latexmk_tool"
6      ]
7    },
8  ],
9  "latex-workshop.latex.tools": [
10   {
11     "name": "latexmk_tool",
12     "command": "latexmk",
13     "args": [
14       "-outdir=%OUTDIR%",
15       "%DOC%"
16     ],
17   },
18 ]
```

- x. 最後の確認. VSCode で `main.tex` を開き, コマンドパレットから `Build LaTeX project` を実行.

これでとりあえず $\text{up}\mathbb{L}\mathbb{A}\mathbb{T}\mathbb{E}\mathbb{X}$ でも $\text{Lua}\mathbb{L}\mathbb{A}\mathbb{T}\mathbb{E}\mathbb{X}$ でもその他のなにかでもローカル環境で動く! はず (`latexmkrc` は文書に合わせて適宜変える必要あり).

ここから先はすべて個人の好みである. 例えば

- VSCode の設定ファイル `settings.json` の `recipes` と `tools` は好きなように組合せて作れるため,
 - ログファイルを出力しないようにする (高速)
 - $\text{Lua}\mathbb{L}\mathbb{A}\mathbb{T}\mathbb{E}\mathbb{X}$ + `latexmk`
 - $\text{up}\mathbb{L}\mathbb{A}\mathbb{T}\mathbb{E}\mathbb{X}$ + `latexmk`
 - $\text{up}\mathbb{L}\mathbb{A}\mathbb{T}\mathbb{E}\mathbb{X}$ + `upBm\mathbb{T}\mathbb{E}\mathbb{X}` + $\text{up}\mathbb{L}\mathbb{A}\mathbb{T}\mathbb{E}\mathbb{X}\times 2$ + `dvipdfmx`
 とかを用意しておき, 適宜ショートカットで切り替えながら使う.
- `latexmk` でコンパイルが終わるたび, 中間ファイルを全消去する.
- 出力を別フォルダにすることで, 中間ファイルでフォルダがごちゃごちゃするのを抑制する.
- 保存するたびに自動でコンパイルさせる.
- 出力 PDF の表示を VSCode のプレビューで行う. もしくは外部アプリで行う.

などなど色々工夫ができる.

B. なぜ \LaTeX なのか

これまでずっと \LaTeX を使う人に向けて、その説明をしてきた。では、そもそも \LaTeX を使う理由はあるのか。

よく \LaTeX を使うと数式が綺麗と言われる。一昔前までは実際そうだったのであろう。しかし、実際「美しさ」ではもはやほとんど差がないことが多い。というのも、以前 X、もと Twitter で「 \LaTeX で作った数式」「Word で作った数式」「Word で \LaTeX に似るように作った数式」の 3 つを見分けるポストがあり、私はまんまと外した（名誉のために言っておくと、この文を書いているときに再挑戦したらきちんと正解した）。このポストの「これは \LaTeX 」のアンケートの結果も「Word で \LaTeX に似るように作った数式」が半数以上の票で 1 位だった。つまり、似せようと思えば、ほとんどの人が気にならないくらいには Word でも綺麗に数式を作れるのである。

では、なぜ \LaTeX を使うのか。

私は結局「美しさと一貫性」だと思っている。数式に限らず、文書全体の綺麗さが \LaTeX にはある。例えば、2 段組みはきちんと半分ずつか。3 段組みは正確に 3 分割か。横並びの 2 つの図はきちんと対称に並んでいるか。横幅が違う 2 つの図を同じ縮小率で並べられるか。キャプションとの距離感は適切か。相互参照は正確に更新されているか。箇条書きのスタイルは全ページで同じか。フォントはどうか。スライドのスタイルは統一されているか。余白は同じか。前ページと同じブロックの位置はズレていないか。これらはすべて、Word でも気をつけていれば実現できることであろう。しかしたくさんの資料を作らなくてはならなかったり、長大な文を書く時にこれらをすべて気にすることはできるだろうか。こういった部分がデフォルトである程度綺麗な \LaTeX を用いる大きな魅力だと感じる。

デフォルトで綺麗というのは、特に複数人での作業のときそのメリットが現れるだろう。一人で凝るのであれば、自分が気をつければいい。しかし、複数人で一つの文書を作成するとき、統一感を持って美しく成り立たせるためには相当な労力が必要であろう。さらに、予稿集のように各人が提出したものを一つにまとめる場合、確実に統一性を持たせるには一体どれだけのルールが必要だろうか。ページサイズ、余白、日英のフォントとサイズを本文・タイトル・著者名・見出しなどすべて指定し、同じだけ行間の指定も必要である。雛形を配布したとして、図表の位置や見出し・キャプションと本文との距離感はどうだろうか。これらの注意をすべての人が完璧に守れるだろうか。こういった複数人での作業において、 \LaTeX の真価が発揮される。デフォルトで十分綺麗がゆえ、もしくはスタイルファイルを配ってしまえば（詳しい人以外は）いじろうにもいじることはできないため、統一感を持たせることができる。統一感のおかげで読み手の注意を無駄に損なわないことは無視できない効果があるだろう。

もう一つ \LaTeX を用いるメリットを挙げるなら、それは文書の構造を意識できることである。（頑張れば）`\section` などを用いずにまったく同じ見た目の文書は作成できる。しかしそうではなく、`\section` などのコマンドや`\begin{itemize}`のような環境が用意されている、というのはユーザに「ここで節を分けるんだ」「ここで並列にものを並べるのがいい」と意識しながら書くことを促す。書き手が文書の構造をしっかり理解していることは、「文書の中身の一貫性」にもつながると思う。結局 \LaTeX の良さは美しさと一貫性なのである。

文章が綺麗で一貫性があることは、シンプルに読みやすさにつながるだろう。それだけで他の人が読みたいと思う、かはわからないけど、少なくとも「見た目で読む気が失せる」ということはない。文書が美しいことは、人にものを読ませる最低ラインで、だから私は \LaTeX を使うし、使ってほしい。

List of Figures

Fig. 2.1. p \TeX engine (left) vs up \TeX engine (right).^[13]

Fig. 2.2. Example of Con \TeX t (left: source tex file, right: pdf file)

Fig. 3.1. The example of not converging cross-referencing. Left: former page (page 999) of first compilation results. Center: That of even numbered compilations. Right: that of odd numbered compilations.

Fig. 5.1. Example of Lua coding^[36]

List of Tables

Table 5.1. Fonts used in this document.

Table 5.2. List of length units available in \LaTeX .

Table 5.3. List of common length macros.

Table 5.4. List of space control macros.