Modern LaTeX

First edition

Matt Kline

If you would like print copies of this book, they can be ordered at cost from
`http://www.lulu.com/shop/matt-kline/modern-latex/paperback/product-23639599.html`

The author apologizes for any typos, formatting mistakes, inaccuracies, and other flubs. He welcomes you to point them out via this book's Git repository at
`https://github.com/mrkline/latex-book`

Questions, comments, concerns, and diatribes can also be emailed to
`matt <at> bitbashing.io`

The author does not have a checking account with the Bank of San Serriffe, but will happily compensate your feedback with a beverage of your choice next time we meet.

First edition (online PDF), typeset May 12, 2018.

To Max, who once told me about a cool program he used to type up his college papers.

Contents

1. Typography and You 1
LaTeX? 2
Another guide? 3

2. Installation 5
Editors 5
Online options 6

3. Hello, LaTeX! 7
Spacing 8
Commands 9
Special characters and line breaks 10
Environments 11
Groups and command scope 12

4. Document Structure 13
Packages and the preamble 13
Titles, sections, and other hierarchy 14
What next? 15

5. Formatting Text 17
Emphasis 17
Meeting the whole (type) family 17
Sizes 18
What next? 20

6. Punctuation 21
Quotation marks 21
Hyphens and dashes 21
Ellipses 22
Spacing 23
What next? 23

7. Layout 24
Justification and alignment 24
Lists 25
Columns 27
Page breaks 27
Footnotes 28
What next? 28

8. Mathematics 29
Examples 29

9. Fonts 31
Changing fonts 31
Selecting font files 32
Scaling 34
OpenType features 34
Ligatures 34
Figures 35
What next? 36

10. Microtypography 37
Character protrusion 37
Font expansion 38
What next? 38

11. Typographie Internationale 39
Unicode 39
The polyglossia package 39
What next? 41

12. When Good Typesetting Goes Bad 42
Fixing overflow 42
Avoiding widows and orphans 42
Handling syntax errors 43

A. A Brief History of LaTeX 44

B. Additional Resources 47
For LaTeX 47
For typography 47

Notes 49

Colophon 51

1. Typography and You

Modern life is a constant battle for your time, fought by endless hordes of ads, apps, clips, emails, sites, shows, and texts. To put ideas into the world amidst this din, your must seize your audience's attention. And when we write, we don't just reach readers with our words. We grab them with *typography*: how those words appear. Good design isn't mere art—it's function. It draws readers in, informs their expectations, and builds a subliminal brand for your work.[1]

> Typography is why these lines evoke memories of awful essays
>
> you wrote in school. Do many books look this way? Why not?

It is why

> **DO IT LATER**

seems oddly reminiscent of a shoe company named after a Greek goddess. With its help,[2] two men landed on a dusty alien plain, carrying a plaque that read:

HERE MEN FROM THE PLANET EARTH
FIRST SET FOOT UPON THE MOON
JULY 1969, A. D.
WE CAME IN PEACE FOR ALL MANKIND

If you care not only about *what* you tell your readers, but also *how* you tell them, you should try LaTeX.* It's a program† for crafting written documents, like essays, papers, and books. And unless you shovel money at Adobe for InDesign or InCopy, you won't find any equals. By carefully handling subtle details, LaTeX can produce beautiful typography with little effort from you, the author. Modern versions can also leverage recent‡ advances in digital typesetting, offering you the same tools used by professional graphic designers and publishers.

---

*Pronounced "lay-tech" or "lah-tech"

†And a markup language, and (sort of) a programming language. More on that later.

‡By recent, I mean "from the mid-1990s", but web browsers and desktop publishing software are only just starting to catch up.

LaTeX?

LaTeX is an alternative to “word processors” like Microsoft Word, Apple Pages, Google Docs, and LibreOffice Writer. These other applications operate on the principle of *What You See Is What You Get* (WYSIWYG), where what’s on screen is the same as what comes out of your printer. LaTeX is different. Here, documents are written as “plain” text files, using *markup* to specify how the final result should look. If you’ve done any web development, this is a similar process—just as HTML and CSS describe the page you want browsers to draw, your markup describes the appearance of your document to LaTeX.

```
\LaTeX{} is an alternative to ``word processors'' like
Microsoft Word, Apple Pages, Google Docs,
and LibreOffice Writer.
These other applications operate on the principle of
\introduce{What You See Is What You Get}
\acronym{(wysiwyg)}, where what's on screen is the same
as what comes out of your printer.
\LaTeX{} is different. Here, documents are written as
``plain'' text files, using \introduce{markup} to specify
how the final result should look.
If you've done any web development, this is a similar
process---just as \acronym{html} and \acronym{css} describe
the page you want browsers to draw, your markup describes
the appearance of your document to \LaTeX.
```

The LaTeX markup for the above paragraph

This might seem alien to you if you’ve never worked with a markup system before. However, it comes with a few advantages:

1. You can handle your work’s content and its presentation separately. At the start of each document, you describe the design you want. LaTeX takes it from there, maintaining consistent formatting across your whole text. Compare this to a WYSIWYG system, where you constantly deal with appearances as you write. If you changed the look of a caption, were you sure to find all the other captions and do the same? If the program formats something in a way you don’t like, is it hard to fix?

2. You can define your own commands, then tweak them to instantly adjust wherever they're used. For example, the `\introduce` and `\acronym` commands seen above are my own creations. The former *italicizes* text, and the latter sets words in SMALL CAPS with a bit of extra L E T T E R S P A C I N G so the characters don't look TOO CROWDED. If I decide tomorrow that I would rather introduce new terms ***with this look***, or that acronyms should look LIKE THIS, I just change the two lines that define those commands. Every spot in this book that uses them is immediately updated.

3. Since the document source is plain text,
    - It can be read and understood with any text editor.
    - Structure is immediately visible and easily replicated.*
    - Content can be generated by scripts and programs.
    - Changes can be tracked with standard version control software.

Another guide?

You might wonder why the world needs another LaTeX guide. After all, it's existed for decades. A quick Amazon search finds nearly a dozen books on the topic. There are plenty of great resources online.

Unfortunately, most of the introductions you will find have two fatal flaws: they are long, and they are old. Their length is anathema to newcomers—if somebody asks about LaTeX, throwing a 200+ page book at them isn't an encouraging start. Their age matters because of how much typesetting has changed since 1986.

When LaTeX was first released that year, none of the publishing technologies we use today existed. Adobe wouldn't debut their Portable Document Format for seven more years. Digital printing was a new field, and desktop publishing was a fledgling curiosity.[3] This shows—badly—in most LaTeX guides. If you look for instructions to change your document's font, you'll get swamped with bespoke nonsense.†

*Compare this to WYSIWYG systems, where it is often unobvious how certain formatting was produced or how to replicate it.

†Take all criticisms of LaTeX's past here with a grain of salt. After all, the fact that all of the technology around it became obsolete—multiple times—is a testament to its staying power.

The good news is that LaTeX has improved by leaps and bounds in recent years. It's time for a guide that doesn't weigh you down with decades of legacy or try (in vain) to be a comprehensive reference. After all, you're a smart, resourceful individual who knows how to use a search engine. This book will:

1. Teach you the fundamentals of LaTeX.
2. Point you to places where you can learn more.
3. Show you how to use modern typesetting technologies and techniques, like OpenType and microtypography.
4. End promptly thereafter.

Let's begin.

2. Installation

You install LaTeX on your computer from a *distribution*. It comes with:

1. LaTeX, the program—the thing that typesets text files into documents.*
2. A common set of LaTeX *packages*. Packages are pieces of code that do all sorts of things, like provide new commands or change a document's style. We'll see lots of them in action throughout this book.
3. Miscellaneous tools, such as editors.

Each major operating system has its own LaTeX distribution:

Mac OS has MacTeX. Grab it from `http://www.tug.org/mactex` and install it per the instructions there.

Windows has MikTeX. Install it from `https://miktex.org/download`. MikTeX offers the unique ability to automatically download additional packages whenever a document uses one that it can't find on your computer.

Linux and BSD use TeX Live. Like most software, this is provided through your OS's package manager. Linux distributions usually contain a `texlive-full` or `texlive-most` package that installs everything you need.†

Editors

Since LaTeX source files are regular text files, you're free to create them with the usual choices, like Vim, Emacs, Sublime, Notepad++, and so on.‡ There are also

*We'll actually install multiple LaTeX programs, but we're getting to that.

†If you'd rather keep the install size down, Linux distributions usually break TeX Live into multiple distro packages. Look for ones with names like `texlive-core`, `texlive-luatex` and `texlive-xetex`. As you work with LaTeX, you may need less-common packages, which usually have names like `texlive-latexextra`, `texlive-science`, and so on. Of course, all of this may vary from one Linux distribution to another.

‡If you've never used any of these, try a few. They're popular with programmers and other folks who shuffle text around screens all day. Just don't use Notepad. Life is too short.

editors designed specifically for LaTeX, which often come with their own built-in PDF viewer. (You can find a fairly comprehensive list on the LaTeX Wikibook, in its installation chapter. See Appendix B.)

Online options

If you'd rather not install LaTeX on your computer, you can try online editors such as ShareLaTeX and Overleaf. This book won't focus on these web-based tools, but almost all of the same basics apply. Of course, you have less control over certain aspects of online versions, like available fonts, the version of LaTeX that's used, and so on.

3. Hello, LaTeX!

Now that you have a LaTeX distribution installed, let's try it out. Open up your editor of choice and save the following as `hello.tex`:

```
\documentclass{article}
% Say hello
\begin{document}
Hello, World!
\end{document}
```

Next, we'll run this file through LaTeX (the program)* to get our document. The installation placed several different versions—or *engines*—on your machine, but throughout this book, we'll always use the newest ones: LuaLaTeX and XeLaTeX.†

If you are using a LaTeX-specific editor, it should contain some method (like a drop-down menu) to select the engine you'd like to use, along with a button to generate your document. Otherwise, run the following from your terminal:‡

```
$ xelatex hello.tex
```

Feel free to use `lualatex` instead—there are a few differences between the two that we'll discuss later, but either is fine for now. With luck, you should see some output that ends in a message like:

```
Output written on hello.pdf (1 page).
Transcript written on hello.log.
```

*Not to be confused with LaTeX the lunchbox, LaTeX the breakfast cereal, or LaTeX the flamethrower. The kids love this stuff!

†See Appendix A for a comparison of all current LaTeX engines.

‡How to work a terminal emulator, make sure the newly-installed LaTeX programs are in your `PATH`, and so on are all outside the scope of this book. As is tradition, the leading dollar sign in this example just denotes a console prompt, and shouldn't actually be typed.

And in your current directory, you should find a newly minted `hello.pdf`. Open it up and you should see a page with this at the top:

> Hello, World!

Congrats, you just created your first document! Let's unpack what we did.

All LaTeX documents begin with a `\documentclass` declaration, which picks a base "style" to use. Many classes are available—and you can even create your own—but common ones include `article`, `report`, `book`, and `beamer`.* For the average document, `article` is probably a good choice. The next line, `% Say hello`, is a *comment*—any text placed after `%` on a line is ignored by the engine, so we use it to leave notes for anybody reading the document's source.† Finally, `\begin{document}` tells LaTeX that what follows is our actual contents, and `\end{document}` states that we are finished.

Let's cover some more basics.

Spacing

LaTeX generally handles inter-word spacing for you, regardless of how many times you mash the space bar or tab key. For example,

```
The number  of   spaces   between words doesn't   matter.
The same is true for space between sentences.

An empty line ends the previous paragraph and
starts the next.
```

yields

> The number of spaces between words doesn't matter. The same is true for space between sentences.
>
> An empty line ends the previous paragraph and starts the next.

Notice that LaTeX automatically follows typographic conventions, such as indenting new paragraphs and leaving more space after a period than it leaves between

*This last one is for slideshows. The name is a German term for a projector.

†Including, perhaps most importantly, a confused version of your future self!

words. One quirk to be aware of is that comments "eat" all of the leading space on the subsequent line, such that

```
This% weird, right?
  is strange.
```

gives

Thisis strange.

Commands

LaTeX provides various commands to issue layout instructions to the engine, and you can define your own as well. Their names always begin with a backslash (`\`), contain only letters, and are case-sensitive.[*] Some commands require parameters, or *arguments*—`\documentclass`, for example, needs to know which class we want. Arguments are enclosed in subsequent pairs of braces, so if some command needed two arguments, we would type:

```
\somecommand{argument1}{argument2}
```

Many commands also take optional arguments, which precede the mandatory ones, are enclosed in square brackets, and are separated by commas. Say you want to inform LaTeX that your document should be printed as double-sided pages[†] in 11 point type.[‡] This is done with optional arguments to `\documentclass`:

```
\documentclass[11pt, twoside]{article}
```

Other commands take no arguments at all—`\LaTeX`, which prints the LaTeX logo, is one example. These commands consume any space that follows them. For example,

[*] `\foo` is different from `\Foo`, for example.

[†] `twoside` introduces commands that only make sense in the context of double-sided printing, such as ones that skip to the start of the next odd page. It also allows you to have different margins for even and odd pages, which is useful for texts like this book.

[‡] We'll discuss font sizes in Chapter 5.

```
\LaTeX is great, but it can be a bit odd sometimes.
```

will give you

> LaTeXis great, but it can be a bit odd sometimes.

You can fix this by adding an empty pair of braces to the command. Of course, the braces aren't needed if there is no space to preserve:

```
Let's learn \LaTeX! \LaTeX{} is a powerful tool,
but a few of its rules are a little weird.
```

gets us

> Let's learn LaTeX! LaTeX is a powerful tool, but a few of its rules are a little weird.

Special characters and line breaks

Some characters have special meanings in LaTeX. We saw above, for example, that `%` starts a comment and `\` starts a command. The full list of special characters is:

```
# $ % ^ & _ { } ~ \
```

Each has a corresponding command to print it in your document. Respectively, they are:

```
\# \$ \% \^{} \& \_ \{ \} \~{} \textbackslash
```

Regardless of what comes after them, you must always add braces to the caret (`^`) and tilde (~). This is a relic from the days when these commands were used to produce *diacritical marks*: once upon a time, users would typeset "jalapeño" with `jalape\~no`. Today, we just type `ñ` into our source file.*

If you're wondering why we print `\` with `\textbackslash` instead of `\\`, it's because the latter is the command to force a line break.

*The ease with which you can do this depends on your keyboard, your editor, and the language settings in your OS. We'll talk much more about non-English languages and Unicode fun in Chapter 11.

```
Give me \\
a brand new line!
```

obeys:

> Give me
> a brand new line!

Use this power judiciously—deciding how to best break paragraphs into lines is one of LaTeX's greatest skills.

Environments

We often format text in LaTeX by placing it in *environments*. These always start with `\begin{name}` and conclude with `\end{name}`, where `name` is that of the desired environment. Take the `quote` environment, which adds additional space on both sides of a block quotation:

```
Donald Knuth once wrote,
\begin{quote}
We should forget about small efficiencies,
say about 97\% of the time:
premature optimization is the root of all evil.
Yet we should not pass up our opportunities in
that critical 3\%.
\end{quote}
```

quotes

> Donald Knuth once wrote,
>
> > We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%.

Groups and command scope

Some commands change how LaTeX sets the text that follows them. `\itshape`, for example, *italicizes* everything that comes after it. To limit a command's influence to a certain area, we surround it with braces.

```
{\itshape Sometimes we want italics}, but only sometimes.
```

becomes

> *Sometimes we want italics*, but only sometimes.

The braced region is called a *group*, and commands placed inside that group only take effect until it ends. Environments also form implicit groups:

```
\begin{quote}
\itshape If I italicize a quote, the following text will
use upright type again.
\end{quote}
See? Back to normal.
```

typesets

> *If I italicize a quote, the following text will use upright type again.*

See? Back to normal.

Another use of groups is to handle the spacing oddities of zero-argument commands: some prefer `{\LaTeX}` over `\LaTeX{}`.

4. Document Structure

Every LaTeX document is different, but all share a few common elements.

Packages and the preamble

In the last chapter, you built your first document with:

```
\documentclass{article}

\begin{document}
Hello, World!
\end{document}
```

The area between the `\documentclass` command and the start of the `document` environment is called the *preamble*. Here, we perform any setup we need—such as importing packages and defining commands—to control how our document will look.

As mentioned in Chapter 2, *packages* contain code to modify your document in interesting ways. The ones found in your LaTeX distribution come from the Comprehensive TeX Archive Network, or CTAN,* at `https://ctan.org`. Package manuals are also found there, so make it your first stop when learning how to use one.

To import a package, add a `\usepackage` command, using the package's name as its argument. As a simple example, let's write a document with the `metalogo` package, which adds `\LuaLaTeX` and `\XeLaTeX`:

*Curious readers may be wondering what TeX is, and how it differs from LaTeX. The short answer is that TeX is the typesetting system that LaTeX is built on top of—the latter is a framework of commands for the former. (For example, `\documentclass` and friends are provided by LaTeX, but the TeX engine is what's actually laying out your document.) The long answer is at the back of this book under Appendix A. We won't discuss how to use plain TeX here. That's for another book—the TeXbook.

```
\documentclass{article}

\usepackage{metalogo}

\begin{document}
\XeLaTeX{} and \LuaLaTeX{} are neat.
\end{document}
```

should get you a PDF that reads

> XƎLaTeX and LuaLaTeX are neat.

Like other commands, `\usepackage` accepts optional arguments. The `geometry` package, for instance, takes your desired paper size and margins. For US Letter paper with one-inch margins, you type:

```
\usepackage[
    letterpaper,
    left=1in, right=1in, top=1in, bottom=1in
]{geometry}
```

Command arguments can be spaced however you like, so long as there are no empty lines between arguments.

Titles, sections, and other hierarchy

Authors often split their works into sections to help readers navigate them. LaTeX offers seven different commands to break up your documents: `\part`, `\chapter`, `\section`, `\subsection`, `\subsubsection`, `\paragraph`, and `\subparagraph`. Issue the command where you want a section* to start, providing its name as the argument. For example,

```
\documentclass{book}
```

*By this I mean any level, not just `\section`.

```
\begin{document}

\chapter{The Start}
This is a very short chapter in a very short book.

\chapter{The End}
Is the book over yet?

\section{No!}
There's some more we must do before we go.

\section{Yes!}
Goodbye!
\end{document}
```

Some levels are only available in certain document classes—chapters, for example, only appear in books. And don't go too crazy with these commands. Most works only benefit from a few levels of hierarchy.

Sections are automatically numbered—for example, the title of this chapter was produced with `\chapter{Document Structure}`, and LaTeX figured out that it was chapter 4.

What next?

As promised, this book isn't a comprehensive reference, but it *will* point you to places where you can learn more. We'll wrap up most chapters with a list of related topics you could explore next.

Consider learning how to:

- Automatically start your document with its title, the author's name, and the date using `\maketitle`.
- Have LaTeX build a table of contents with `\tableofcontents`.
- Control section numbers with `\setcounter{secnumdepth}` and starred section commands, e.g., `\subsection*{foo}`.

- Create auto-updating cross-references with `\label` and `\ref`.
- Use KOMA Script, a set of document classes and packages that help you customize nearly every aspect of your document, from heading fonts to footnotes.
- Include images with the `graphicx` package.
- Add hyperlinks to your PDF with the `hyperref` package.
- Split large documents into multiple files using `\input`.

5. Formatting Text

Emphasis

Sometimes you need some extra punch to get your point across. The simplest way to emphasize text in LaTeX is with the `\emph` command, which *italicizes* its argument by default:

```
\emph{Oh my!}
```

gives us

> *Oh my!*

There are other tools at our disposal:

```
We can also use \textbf{boldface} or \textsc{small caps}.
```

with

> We can also use **boldface** or SMALL CAPS.

Be judicious in your use of emphasis, especially boldface. It excels at drawing the reader's attention away from everything around it, so too much is distracting.

Meeting the whole (type) family

The styles shown above are just a few of the many available to you. A (mostly) complete list follows:

Command	Alternative	Style
`\textnormal{...}`	`{\normalfont ...}`	the default
`\emph{...}`	`{\em ...}`	*emphasis, typically italics*
`\textrm{...}`	`{\rmfamily ...}`	roman (serif) type
`\textsf{...}`	`{\sffamily ...}`	sans serif type
`\texttt{...}`	`{\ttfamily ...}`	`teletype (monospaced)`
`\textit{...}`	`{\itshape ...}`	*italics*
`\textsl{...}`	`{\slshape ...}`	*slanted, or oblique type*
`\textsc{...}`	`{\scshape ...}`	SMALL CAPITALS
`\textbf{...}`	`{\bfseries ...}`	**boldface**

Prefer the first form (which takes the text to format as an argument) over the second (which affect the group in which they are issued), since the former automatically handle any spacing corrections needed around them.* However, when formatting multiple paragraphs, or when defining the style of other commands,† the second variety is the only option.

Sizes

The default text size is controlled by your document class. It is usually ten points,‡ but this can be adjusted by passing additional arguments to `\documentclass`.◊ To scale text relative to this size, use the following commands:

`\tiny`	Example Text
`\scriptsize`	Example Text
`\footnotesize`	Example Text
`\small`	Example Text
`\normalsize`	Example Text
`\large`	Example Text

*For example, *italic type* amidst upright type should be followed by a slight amount of additional space, called an "italic correction".

†For example, this book's section headers are styled with `\Large\itshape`.

‡The standard digital publishing point, sometimes called the PostScript point, is $1/72$ of an inch. LaTeX, for historical reasons, defines its point (`pt`) as $100/7227$ of an inch and the former as "big points", or `bp`.

◊The standard LaTeX classes accept `10pt`, `11pt`, or `12pt`. KOMA Script classes accept arbitrary sizes with `fontsize=<size>`.

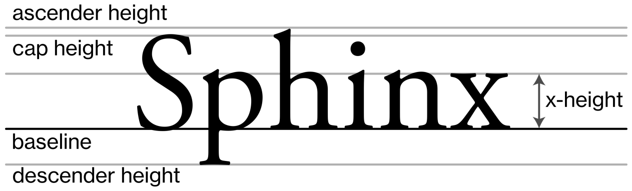
`\Large`	Example Text
`\LARGE`	Example Text
`\huge`	Example Text
`\Huge`	Example Text

Some subtleties are at play here. LaTeX's default type family, Latin Modern, comes in multiple *optical sizes*. Smaller fonts aren't just shrunken versions of their big siblings—they have thicker strokes, exaggerated features, and more generous spacing to improve legibility at their size.

> If you magnify 5 point type and place the result next to 11 point type, the differences are immediately noticeable.

Optical sizes were standard in the days of metal type, but many digital typefaces lack them, given how much more work it demands from the type designer.*

But points and optical sizes don't tell the whole story. Each typeface has its own proportions, which make a huge difference in perceived size. (Compare Garamond, Latin Modern, and Helvetica, all at 11 points.) Shown below are some common terms:



Type sits on the *baseline*, rises to its *ascender height*, and drops to its *descender height*. The *cap height* refers to the size of uppercase letters, and the *x-height* refers to the size of lowercase letters.

If the previous commands don't give you a size you need, you can create custom ones with `\fontsize`, which takes both a text size and a distance between baselines. This must be followed with `\selectfont` to take effect. For example, `\fontsize{30pt}{30pt}\selectfont` produces

*If you are fortunate enough to have a typeface with multiple optical sizes, LuaLaTeX and XeLaTeX can make good use of them! See Chapter 9 for more on font selection.

large type with no
additional space
between lines

Note that without additional spacing, or *leading*,[*] descenders from one line nearly collide with ascenders and capitals on the line below. Leading is important—without it, blocks of text become uncomfortable to read, especially at normal body sizes.

Let your type breathe.[†]

What next?

- Learn how to underline text with the `ulem` package.[‡]
- Use KOMA Script to change the size and style of your section headings.
- Learn the difference between italic and oblique type.
- Change the default text style (used by `\textnormal` and `\normalfont`) by redefining `\familydefault`.

[*] This term comes from the days of metal type, when strips of lead or brass were inserted between lines to space them out.[4]

[†] For a discussion on optimal leading, see *Practical Typography*, listed in Appendix B.

[‡] Other typographical tools—like italics, boldface, and small caps—are generally preferable to underlining, but it has its uses.

6. Punctuation

You would much rather encounter a panda that eats shoots and leaves than one that eats, shoots, and leaves.[5] Punctuation is a vital part of writing, and there's more to it than your keyboard suggests.

Quotation marks

LaTeX doesn't automatically convert `"straight"` quotes into correctly-facing "curly" ones:

```
    "This isn't right."
```

will get you

> ”This isn’t right.”

Instead, use `` ` `` for opening quotes and `'` for closing quotes.*

```
    ``It depends on what the meaning of the word `is' is.''
```

quotes a former US president as,

> “It depends on what the meaning of the word ‘is’ is.”

Hyphens and dashes

Though they look similar, hyphens (-), en dashes (–), em dashes (—), and minus signs (−) serve different purposes.

*If your keyboard layout happens to have keys for "curly" quotes (“ ”), feel free to use those instead! Also, don't use `"` for closing double quotes. Not only does ` ``example"` look a bit unbalanced, but `"` is used as a formatting command when typesetting certain languages, like German. (See Chapter 11 for more on international typesetting.)

Hyphens have a few applications:[6]

- They allow a word to be split between the end of one line and the start of the next. LaTeX usually handles this automatically.
- Some compound words use hyphens, like *long-range* and *field-effect*.
- They are used in phrasal adjectives. If I ask for "five dollar bills", do I want five $1 bills, or several $5 bills? It's clearer that I mean the latter when typeset as *five-dollar bills*.

Unsurprisingly, they are produced with the hyphen character (`-`).

En dashes are for ranges such as "pages 4–12", and connected words, like "the US–Canada border". LaTeX places one wherever it sees two adjacent hyphens (`--`).

Em dashes are used to separate clauses of a sentence. Other punctuation—like parenthesis and commas—play a similar role. Em dashes are typeset with three hyphens (`---`).

Minus signs are used for negative quantities and mathematical expressions. They are similar in length to an en dash, but sit at a different height. Minus signs are set with `\textminus`, or with the hyphen character when in a math environment (see Chapter 8).

Ellipses

A set of three dots used to indicate a pause or omission is called an *ellipsis*. It is set with `\dots`.

```
I'm\dots{} not sure.
```

becomes

I'm… not sure.

Ellipses are spaced differently than consecutive periods. Don't use the latter as a poor substitute for the former.

Spacing

As we discovered in Chapter 3, LaTeX inserts additional space between periods and whatever follows them—presumably the start of the next sentence. This isn't always what we want! Consider honorifics like Mr. and Ms., for example. In situations like these, we also need to prevent LaTeX from starting a new line after the period. This calls for a *non-breaking space*, which we set with a tilde.

```
Please call Ms.~Shrdlu.
```

produces proper spacing:

Please call Ms. Shrdlu.

In other occasions, such as when we abbreviate units of measurement,* we want thinner spaces than our usual inter-word ones. For these, we use `\,`:

```
Launch in 2\,h 10\,m.
```

announces

Launch in 2 h 10 m.

What next?

- Add hyphenations for uncommon words using `\hyphenate` or `\-`.†
- Learn other commands for spacing, such as `\:`, `\;`, `\enspace`, and `\quad`.
- Use the `csquotes` package's `\enquote` to simplify nested quotations, e.g., "She exclaimed, 'I can't believe it!'"
- Discover the typographical origins of terms like *en*, *em*, and *quad*.
- Familiarize yourself with the difference between `/` and `\slash`.

*There are also dedicated packages for doing so, like `siunitx`.

†LaTeX usually does a good job of automatically hyphenating words, based on a dictionary of patterns stored for each language. You should rarely need to use these commands.

7. Layout

Justification and alignment

LaTeX justifies text remarkably well. Instead of considering lines individually—as most word processors, web browsers, and e-readers do—it examines all possible line breaks in a given paragraph, then picks whichever ones give the best overall spacing.[7] Combined with its ability to automatically hyphenate words, which permits line breaks in many more places,[8] this approach produces better paragraph layouts than almost any other software.

But sometimes we don't want our text justified. If you would like it to be flush left, place it in a `flushleft` environment or add `\raggedright` to the current group. To center it, place it in a `center` environment or add `\centering` to the current group. And to flush it against the right margin, use a `flushright` environment or `\raggedleft`.

```
\begin{flushleft}
This text is flush left, with a ragged right edge.
Some prefer this layout since the space between words
is more consistent than it is in justified text.
\end{flushleft}

\begin{center}
This text is centered.
\end{center}

\begin{flushright}
And this text is flush right.
\end{flushright}
```

sets

This text is flush left, with a ragged right edge. Some prefer this layout since the space between words is more consistent than it is in justified text.

This text is centered.

And this text is flush right.

Lists

LaTeX provides multiple environments for creating lists: `itemize`, `enumerate`, and `description`. In all three, each item starts with an `\item` command. The first environment, `itemize`, creates bulleted lists. With:

```
\begin{itemize}
\item 5.56 millimeter
\item 9 millimeter
\item 7.62 millimeter
\end{itemize}
```

you get

- 5.56 millimeter
- 9 millimeter
- 7.62 millimeter

`enumerate` numbers its lists:

```
\begin{enumerate}
\item Collect underpants
\item ?
\item Profit
\end{enumerate}
```

produces

1. Collect underpants
2. ?
3. Profit

The `description` environment starts each item with some emphasized *label*, then indents all subsequent lines for that item:

```
\begin{description}
\item[Alan Turing] was a British mathematician who
    laid much of the groundwork for the field
    of computer science.
    He is perhaps most remembered for his model of
    computation, the Turing machine.
\item[Edsger Dijkstra] was a Dutch computer scientist.
    His contributions in many subdomains---such as
    concurrency and graph theory---are still in wide use.
\item[Leslie Lamport] is an American computer scientist.
    He defined the concept of sequential consistency,
    which is used to safely communicate between tasks
    running in parallel.
\end{description}
```

gives us

Alan Turing was a British mathematician who laid much of the groundwork for the field of computer science. He is perhaps most remembered for his model of computation, the Turing machine.

Edsger Dijkstra was a Dutch computer scientist. His contributions in many subdomains—such as concurrency and graph theory—are still in wide use.

Leslie Lamport is an American computer scientist. He defined the concept of sequential consistency, which is used to safely communicate between tasks running in parallel.

Columns

We often split layouts into several columns, especially when printing on A4 or US Letter paper, since it allows more comfortable line widths at standard 8–12 pt text sizes.* You can either add the `twocolumn` option to your document class, which splits everything in two, or you can use the `multicols` environment from the `multicol` package:

```
One nice feature of \texttt{multicol} is that you can
combine arbitrary layouts.
\begin{multicols}{2}
This example starts with one column,
then sets the following section as two.
The \texttt{multicols} environment splits the text
inside it so that each column is about the same height.
\end{multicols}
```

is split into

> One nice feature of `multicol` is that you can combine arbitrary layouts.
>
> This example starts with one column, then sets the following section as two. The `multicols` environment splits the text inside it so that each column is about the same height.

Page breaks

Some commands, like a book's `\chapter`, insert page breaks. You can add your own with `\clearpage`. If you are using the `twoside` document class option for double-sided printing, you can break to the front of the next page with `\cleardoublepage`.

*You'll see different advice depending on where you look, but as a rule of thumb, design layouts to have between 45 and 80 characters (including spaces) per line. If a line is too long, readers have an uncomfortable time scanning for the start of the next one. If a line is too short, it doesn't have much inter-word spacing to adjust, which can lead to odd gaps or excessive hyphenation.

Footnotes

Footnotes are useful for inserting references, or for remarks that readers might find helpful, but aren't crucial to the main text. The `\footnote` command places a marker at its location in the body text, then sets its argument at the bottom of the current page:

```
I love footnotes!\footnote{Perhaps a bit too much\dots}
```

proclaims

> I love footnotes!*

What next?

- Control paragraph spacing, either with KOMA Script options, or with the `parskip` package.
- Set the page size and margins with the `geometry` package.
- Customize list formatting with the `enumitem` package.
- Create tables with the `tabular` environment.
- Align text with tab stops using the `tabbing` environment.
- Customize footnote symbols and layout with the `footmisc` package and KOMA Script.
- Create horizontal and vertical space with commands such as `\vspace`, `\hspace`, `\vfill`, and `\hfill`.
- Learn what units LaTeX provides for specifying spacing.
  (We've already mentioned a few here, like `pt`, `bp`, and `in`.)

*Perhaps a bit too much…

8. Mathematics

LaTeX excels at typesetting mathematics, both in body text, e.g., $x_n^2 + y_n^2 = r^2$, and as standalone formulas:

$$\sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

The former is typed inside `$...$` or `\(...\)`, and the latter within `\[...\]`. In these math environments, the rules of LaTeX change:

- Most spaces and line breaks are ignored, and spacing decisions are made for you based on typographical conventions for mathematics. `$x+y+z$` and `$x + y + z$` both give you $x + y + z$.
- Empty lines are not allowed—each formula occupies a single "paragraph".
- Letters are automatically italicized, as they are assumed to be variables.

To return to normal "text mode" inside a formula, use the `\text` command. Other formatting commands mentioned in Chapter 5 work as well. From

```
\[ \text{fake formulas} = \textbf{annoyed mathematicians} \]
```

we get

$$\text{fake formulas} = \textbf{annoyed mathematicians}$$

Examples

Typesetting mathematics is arguably the raison d'être of LaTeX,* but the topic is so broad that giving it decent coverage would take up half of this book. Given how wide the field of mathematics is, there are *many* different commands and environments. Here, more so than any other topic, you owe it to yourself to find some real references and learn what LaTeX is capable of. Before moving on, though, let's show some examples of what it can do.

*Well, TeX

1. `x = \frac{-b \pm \sqrt{b^2 - 4 a c}}{2a}`

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

2. `e^{j \theta} = \cos(\theta) + j \sin(\theta)`

$$e^{j\theta} = \cos(\theta) + j\sin(\theta)$$

3.
```
\begin{bmatrix}
x' \\
y'
\end{bmatrix} =
\begin{bmatrix}
\cos \theta &  -\sin\theta \\
\sin \theta & \cos \theta
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix}
```

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

4.
```
\oint_{\partial \Sigma} \mathbf{E} \cdot
\mathrm{d}\boldsymbol{\ell}
= - \frac{\mathrm{d}}{\mathrm{d}t}
    \iint_{\Sigma} \mathbf{B} \cdot \mathrm{d}\mathbf{S}
```

$$\oint_{\partial \Sigma} \mathbf{E} \cdot \mathrm{d}\boldsymbol{\ell} = - \frac{\mathrm{d}}{\mathrm{d}t} \iint_{\Sigma} \mathbf{B} \cdot \mathrm{d}\mathbf{S}$$

9. Fonts

Digital fonts have changed almost entirely in the past thirty years. Originally, LaTeX used METAFONT, a system designed by Donald Knuth specifically for TeX. As time went on, support for PostScript* fonts was added. Today, LuaLaTeX and XeLaTeX support the font formats you're most likely to encounter on your computer: TrueType and OpenType.†

TrueType was developed by Apple and Microsoft in the late 1980s. Most of the fonts that come pre-installed on your system are likely in this format. TrueType files generally end in a `.ttf` extension.

OpenType was first released by Microsoft and Adobe in 1996. Improvements over TrueType include its ability to embed various "features", such as alternative glyphs and spacing options, into a single font file. OpenType files usually end in an `.otf` extension.

Changing fonts

By default, LuaLaTeX and XeLaTeX use Latin Modern, an OpenType rendition of LaTeX's original type family, Computer Modern. While these are high-quality fonts, they're probably not the only ones you ever want to use. For others, we turn to the `fontspec` package:

```
\documentclass{article}

\usepackage{fontspec}
\setmainfont[Ligatures=TeX]{Source Serif Pro}
\setsansfont[Ligatures=TeX]{Source Sans Pro}
```

*One of Adobe's original claims to fame, PostScript is a language for defining and drawing computer graphics, including type. It remains in use today.

†Mac versions of LaTeX also support Apple's AAT, but let's limit this discussion to more ubiquitous formats.

```
\setmonofont{Source Code Pro}

\begin{document}
Hello, Source type family! Neat---no? \\
\sffamily Let's try sans serif! \\
\ttfamily Let's try monospaced!
\end{document}
```

should produce something like*

> Hello, Source type family! Neat—no?
> Let's try sans serif!
> `Let's try monospaced!`

The `Ligatures=TeX` option lets you use the punctuation shortcuts from Chapter 6 (e.g., `--` for en dashes, or ` `` ` and `''` for curly quotes) instead of forcing you to enter the the corresponding characters, which probably aren't on your keyboard. You often don't want these substitutions when setting monospaced type, though. Text that uses it—such as code—is usually meant to be printed verbatim. `"Hello!"` shouldn't turn into “`Hello!`“.

Selecting font files

Typefaces come packaged as multiple files for their various weights and styles—a typical set includes upright, *italics*, **bold**, and ***bold italics***. Given a typeface's name, `fontspec` can usually deduce the names of its files.†

However, many typefaces come in more than two weights. The version of Futura used in this book, for example, comes in light, book, **medium**, **demi**, **bold**, and **extra bold**. Additional styles, such as SMALL CAPITALS, may be stored in separate

*Assuming, of course, that you have Adobe's open-source fonts installed.[9]

†This is one of the places where XƎLaTeX and LuaLaTeX differ in a way that's noticeable to the casual user. The former uses system libraries—such as FontConfig on Linux—to find the font files for the typefaces you request. The latter has its own font loader, based on code from FontForge.[10] The expected name of a font might differ between the two engines—refer to the `fontspec` manual for details. If you have trouble getting LaTeX to find *any* of your fonts, try rebuilding your system's font cache. (Figuring out how to do so is left as a web search for the reader.)

files as well.* We might want to hand-pick weights to achieve a certain look or better match the weights of other fonts in our document.† Continuing to use Futura as an example, say we want to use the "book" weight as our default and "demi" for bold. Assuming the font files are named:

- `Futura-Boo` for our upright book weight
- `Futura-BooObl` for our *oblique book weight*
- `FuturaSC-Boo` for SMALL CAPS, BOOK WEIGHT
- `Futura-Dem` for **upright demi(bold)**
- `Futura-DemObl` for ***oblique demibold***

Our setup might resemble:

```
\usepackage{fontspec}
\setmainfont[
    Ligatures=TeX,
    UprightFont = *-Boo,
    ItalicFont = *-BooObl,
    SmallCapsFont = *SC-Boo,
    BoldFont = *-Dem,
    BoldItalicFont = *-DemObl
]{Futura}
```

Note that instead of typing out `Futura-Boo`, `Futura-BooObl`, and so on, we can use `*` to insert the base name.

*OpenType allows these alternative styles to be placed in the same file(s) as the "main" glyphs for a given weight. If your font supports this, you don't need to do anything—`fontspec` will dutifully switch to them whenever you use `\textsc` or `\scshape`. But for TrueType, and for OpenType fonts that don't take advantage of this feature, you'll have to load a separate file as shown here.

†Compare how the light, book, **and medium weights** of Futura look compared to surrounding type on this page.

Scaling

Creating a cohesive design with multiple fonts is tricky, especially since typefaces might look completely different at the same point size. `fontspec` can help here by scaling fonts to match either the x-height or the cap height of your main font with `Scale=MatchLowercase` or `Scale=MatchUppercase`, respectively.*

OpenType features

As mentioned at the start of the chapter, OpenType fonts provide various features that can be turned on and off. In LaTeX, these are controlled through `fontspec` with optional arguments to `\setmainfont` and friends. They can also be set for the current group with `\addfontfeature`. Let's examine a few common ones.

Ligatures

Many typefaces use *ligatures*, which combine multiple characters into a single glyph.† OpenType groups ligatures into three categories:

Standard ligatures remedy spacing problems a typeface might otherwise have. Consider the lowercase letters f and i: in many serif typefaces, these combine to form the ligature fi, avoiding awkward spacing between f's ascender and i's dot (f i). Other common examples in English writing include ff, ffi, fl, and ffl. Standard ligatures are enabled by default.

Discretionary ligatures, such as ct, are offered by some fonts. They are disabled by default but can be enabled with `Ligatures=Discretionary`.

Historical ligatures are ones which have fallen out of common use, such as those with a *long s* (e.g., ſt). These are also disabled by default but can be enabled with `Ligatures=Historic`.

*One way to sidestep this issue is to have fewer typefaces in your design. Even just one or two, used carefully, can produce amazing results.

†Ligatures fell out of style during the 20th century due to limitations of printing technology and the increased popularity of sans serif typefaces, which often lack them. Today they are making a comeback, thanks in no small part to their support in OpenType.

Multiple options can be grouped together. Say you want discretionary ligatures. In the likely event that you also want `Ligatures=TeX`, you would enable both with `Ligatures={TeX,Discretionary}`. Ligatures can also be disabled with corresponding `*Off` options. If you want to stop using discretionary ligatures for some passage,

```
{\addfontfeature{Ligatures=DiscretionaryOff}...}
```

does the trick.

Some words are arguably typeset better without ligatures—a classic example is shelfful.[11] You can manually prevent ligatures with a zero-width space, e.g., `shelf\hspace{0pt}ful`. You can also use the `selnolig` package to handle most of these cases automatically.

Figures

When setting figures,* you have two choices to make: lining versus oldstyle, and proportional versus tabular. *Lining* figures, sometimes called *titling* figures, have heights similar to capital letters:

A B C D 1 2 3 4

Oldstyle, or *text* figures, share more similarities with lowercase letters:

Sitting cross-legged on the floor... 25 or 6 to 4?

For body text, either choice is fine, but oldstyle figures shouldn't be combined with capital letters. "F-15C" looks odd, as does "V2.3 Release".

The terms *proportional* and *tabular* refer to spacing. Tabular figures are set with a uniform width, such that 1 takes up the same space as 8. As their name suggests, this is great for tables and other scenarios where figures must line up in columns:

Item	Qty.	Price
Gadgets	42	$5.37
Widgets	18	$12.76

**Figure* here refers to what some might call a *numeral* or *digit*—i.e., 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Typographers generally prefer the first term over the other two.

Proportional figures are the opposite—their spacing is, well… *proportional* to the width of each figure. They are usually preferred in body text, where 1837 looks a bit nicer than 1837.

You select figures with the following options:

```
Numbers=  Lining / Uppercase
          OldStyle / Lowercase
          Proportional
          Tabular / Monospaced
```

As with ligature options, these can be combined: proportional lining figures are set with `Numbers={Proportional,Lining}`, and tabular oldstyle figures are set with `Numbers={Tabular,OldStyle}`. Each option also has a corresponding `*Off` variant.*

Finally, some fonts provide *superior and inferior* figures, which are used to set ordinals (1st, 2nd 3rd, …), fractions (25⁄624), and so on. They have the same weight as the rest of the font's characters, offering a more consistent look than shrunken versions of full-sized figures. (Compare the examples above to 1st, 2nd, 3rd, and 25⁄624. Notice how this second set is too light compared to the surrounding type.) Superior figures are typeset with `VerticalPosition=Superior`, and inferiors are set with `VerticalPosition=Inferior`.

What next?

- Learn how `fontspec` can choose optical sizes based on point size, either automatically from ranges embedded in OpenType fonts, or manually using `SizeFeatures`.
- Experiment with letter spacing—or *tracking*—with the `LetterSpace` option. Extra tracking is unnecessary in most cases, but can be useful to make SMALL CAPS a little more READABLE.

*This is especially useful since fonts select figures in different ways. In some, `Numbers=Lining` doesn't work, so you enable oldstyle figures with `Numbers=OldStyle` and return to lining figures (the default) with `Numbers=OldStyleOff`.

10. Microtypography

Microtypography is the craft of improving a document's legibility with small, subliminal tweaks. In other words, it is

> [...]the art of enhancing the appearance and readability of a document while exhibiting a minimum degree of visual obtrusion. It is concerned with what happens between or at the margins of characters, words or lines. Whereas the macro-typographical aspects of a document (i.e., its layout) are clearly visible even to the untrained eye, micro-typographical refinements should ideally not even be recognisable. That is, you may think that a document looks beautiful, but you might not be able to tell exactly why: good micro-typographic practice tries to reduce all potential irritations that might disturb a reader.[12]

In LaTeX, microtypography is controlled with the `microtype` package. Its use is automatic—for the vast majority of documents, you should add

```
\usepackage{microtype}
```

to your preamble and carry on—but let's take a brief look at what the package does.

Character protrusion

By default, LaTeX justifies lines between perfectly straight left and right margins. This is the obvious choice, but falls victim to an annoying optical illusion: lines ending in small glyphs—like periods, commas, or hyphens—seem shorter than lines that don't.* `microtype` compensates by *protruding* these smaller glyphs into the margins.

*Many other optical illusions come up in typography. For example, if a circle, a square, and a triangle of equal heights are placed next to each other, the circle and triangle look smaller than the square. For this reason, round or pointed characters (like O and A) must be made slightly taller than "flat" ones (such as H and T) for all to appear the same height.[13]

Font expansion

In order to to help LaTeX's justification algorithm build paragraphs with more even spacing and fewer hyphenated lines, `microtype` can stretch characters horizontally. You might think that distorting the type this way would be immediately noticeable, but you're reading a book that does so on every page! This effect, called *font expansion*, is applied *very* slightly—by default, character widths are altered by no more than two percent.*

This feature isn't currently available for XeLaTeX. You'll need to use LuaLaTeX if you'd like to take advantage of it.

What next?

As always, see the package manual for ways to tweak these features. `microtype` is capable of a few other tricks, but several only work on older LaTeX engines.† Those we care about—such as letterspacing—can be handled with `fontspec` or other packages.

*Of course, you can use package options to change this limit, or disable the feature entirely.
†i.e., pdfTeX

11. Typographie Internationale

Surprisingly, languages besides English exist. You may want to use them.

Unicode

Digitizing human language is a complicated topic that has evolved significantly since LaTeX's inception. Today, computers usually represent text with Unicode. Briefly,

- A Unicode text file is made from a series of *code points*, each of which can represent a character to be drawn, an accent or other diacritical mark to combine with an adjacent character, or some non-printing character, such as an instruction to print subsequent text right-to-left.
- One or more of these code points combines to represent a *grapheme cluster* or *glyph*, the shapes within fonts that we informally call "characters".

Привéт   नमस्ते

How many characters do you see? How many code points are they built from?

- Modern font formats contain encoding tables which map code points to the glyphs the file contains.

LuaLaTeX and XeLaTeX use these tools to build documents from Unicode input files.* Make sure that the fonts you select contain the glyphs you need—many only support Latin languages.

The polyglossia package

When your document contains languages besides English, consider using the `polyglossia` package. It will automatically:

*LuaLaTeX accepts UTF-8 files, while XeLaTeX is a bit more flexible and also accepts UTF-16 and UTF-32.

- Load language-specific hyphenations and other typographical conventions.
- Switch between user-specified fonts for each language.
- Translate document labels, like "chapter", "section", and so on.
- Format dates according to language-specific conventions.
- Format numbers in languages that have their own numbering system.
- Use the `bidi` package for documents with languages written right to left.
- Set the script and language tags of OpenType fonts that have them.

To use `polyglossia`, specify the main language of your document, along with any other languages you use. Some languages also take regional dialects as an optional argument:

```
\usepackage{polyglossia}
\setdefaultlanguage[variant=american]{english}
\setotherlanguage{french}
```

`polyglossia` will define an environment for each language. These automatically apply that language's conventions to the text within. French, for example, places extra space around punctuation, so

```
Dexter cried,
\begin{french}
«Omelette du fromage!»
\end{french}
```

gives

Dexter cried, « Omelette du* fromage ! »

*Yes, it's *omelette au fromage*. Direct all complaints to Cartoon Network.

What next?

- See the `polyglossia` manual for language-specific commands.
- Look into the `babel` package as an alternative to `polyglossia`.*
- Try typesetting Japanese or Chinese with the `xeCJK` or `luatex-ja` packages.

*`polyglossia` has better support for OpenType font features via `fontspec`. However, it is newer and has a few known bugs. `babel` is a fine substitute if you run into trouble.

12. When Good Typesetting Goes Bad

With luck, you're off to a good start with LaTeX. But as with any complicated tool, you'll eventually run into trouble. Here are some common problems and what you can try to fix them.

Fixing overflow

When LaTeX can't fit a line into a paragraph with good spacing, it gives up, overflowing the line into the margin. You can sometimes remedy this with some "emergency stretch". When you add `\emergencystretch=<width>` to the document's preamble, LaTeX will try to set troublesome paragraphs a second time, stretching or shrinking the space in each line by up to the provided width.* If that still doesn't help, try tweaking the wording of guilty paragraphs. This can be frustrating, but the alternative is for LaTeX to create spacing that is too loose—where   words have   large   gaps   between   them—or too tight, wherewordsareawkwardly crammedtogether.

Avoiding widows and orphans

Typesetters try to avoid *widow* and *orphan* (also called *club*) lines, which appear separated from the rest of their paragraph by a page boundary. LaTeX tries to avoid these, but its page-splitting algorithm is much more primitive than its paragraph-splitting one.† You can make LaTeX try harder to avoid orphans and widows with:

*LaTeX has pretty sane default limits to how much it stretches and shrinks spacing in a paragraph. You probably don't want to make `<width>` larger than an em or two.

†This is because 1980s computers didn't have enough RAM to do so. Seriously—Knuth wrote at the time, "The computer doesn't have enough high-speed memory capacity to remember the contents of several pages, so TeX simply chooses each page break as best it can, by a process of 'local' rather than 'global' optimization."[14]

```
\widowpenalty=<penalty>
\clubpenalty=<penalty>
```

`<penalty>` is a value between 0 and 10000. When these values are maximized, LaTeX is never allowed to leave orphans or widows, at *any* cost.* This may cause odd layouts to be chosen, so be sure to review your pages if you choose large penalties.

Handling syntax errors

If you confuse LaTeX—say, by issuing commands that don't exist, or forgetting to end an environment—it will print an error message,† then display an interactive prompt starting with `?`. Here you can enter instructions for how to proceed. Once upon a time, when computers were thousands of times slower and LaTeX took that much longer to re-run, this was more useful. Today, we probably just want to quit, then try again once we've fixed our document. To exit the prompt, type `X`, then press Enter. Better yet, you can tell LaTeX to give up as soon as it finds trouble by running your engine with the `-halt-on-error` flag:

```
$ lualatex -halt-on-error myDocument.tex
```

*When considering a given layout, LaTeX assigns penalties, or "badness", to anything that arguably makes a document look worse. It chooses whichever layout it can find with the least badness.

†Usually this contains a succinct summary of the problem and the number of the line(s) it occurred on. Occasionally, LaTeX gets *really* confused and emits something so cryptic it gives C++ template errors a run for their money. As you continue to use LaTeX, you'll start to get a feel for what sorts of mistakes cause these rare, but enigmatic messages.

A. A Brief History of LaTeX

Donald Knuth is celebrated among programmers as the man who coined the term *analysis of algorithms* and pioneered many computer science fundamentals we use today. Knuth is perhaps most famous for his ongoing magnum opus, *The Art of Computer Programming*.

When the first volume of TAOCP was released in 1968, it was printed the same way most books had been since the turn of the century: with *hot metal* type. Letters were cast from molten lead, then arranged into lines. These lines were clamped together to form pages, which were inked and pressed against paper.

By March of 1977, Knuth was ready for a second run of TAOCP, volume 2, but he was horrified when he received the proofs. Hot metal typesetting was expensive, complicated, and time-consuming, so publishers had replaced it with phototypesetting, which works by projecting images of characters onto film. The new technology, while much cheaper and faster, didn't provide the quality Knuth expected.[15]

The average author would have resigned themselves to this change and moved on, but Knuth took great pride in his books' appearances, especially for their mathematics. Around this time, he also discovered the growing field of digital typesetting, where glyphs are built from tiny dots, packed together at over 1,000 per inch. Inspired, Knuth set off on one of the greatest yak shaves* of all time. For years, he paused work on his books to create his own typesetting system. When the dust settled in 1978, he had the first version of TeX.†

It's hard to appreciate how much of a revolution TeX was, especially looking back from a time where anybody with a copy of Word can be their own desktop publisher. Adobe's PDF wouldn't exist for another decade, so Knuth and his graduate students devised their own device-independent format, DVI. Scalable fonts were uncommon, so he created METAFONT to rasterize glyphs into dots on

*Programmers call seemingly unrelated work needed to solve their main problem "yak shaving". The phrase is thought to originate from an episode of *The Ren & Stimpy Show*.[16]

†The name "TeX" comes from the Greek τέχνη, meaning *art* or *craft*.[17]

the page. Perhaps most importantly, Knuth and his students designed algorithms to automatically hyphenate and justify text into beautifully-typeset paragraphs.*

LaTeX, short for Lamport TeX, was later developed by Leslie Lamport as a set of commands for common document layouts. It was introduced in 1986 with his guide, *LaTeX: A Document Preparation System*. Other typesetting systems based on TeX also exist, the other most popular today being ConTeXt.

Development continues today, both in the form of user-provided packages for TeX and LaTeX, and as improvements to the TeX typesetting program itself. There are four versions, or *engines*:

TeX is the original system by Donald Knuth. Knuth stopped adding features after version 3.0 in March 1990, and all subsequent releases have contained only bug fixes. With each release, the version number asymptotically approaches π by adding an additional digit. The most recent version, 3.14159265, came out in January 2014.

pdfTeX is an extension of TeX that provides direct PDF output (instead of TeX's DVI), native support for PostScript and TrueType fonts, and micro-typographic features discussed in Chapter 10. It was originally developed by Hàn Thế Thành as part of his PhD thesis for Masaryk University in Brno, Czech Republic.[19]

XeTeX is a further extension of TeX that adds native support for Unicode and OpenType. It was originally developed by Jonathan Kew in the early 2000s, and gained full cross-platform support in 2007.[20]

LuaTeX is similar to XeTeX in its native Unicode and modern font support. It also embeds the Lua scripting language into the engine, exposing an interface for package and document authors. It first appeared in 2007 and is developed by a core team of Hans Hagen, Hartmut Henkel, Taco Hoekwater, and Luigi Scarso.[21]

Building TeX today is an… interesting endeavor. When it was written in the late 1970s, there were no large, well-documented, open-source projects for students to study, so Knuth set out to make TeX into one. As part of this effort, TeX was written in a style he calls *literate programming*: opposite most programs—where

*These same algorithms went on to influence the ones Adobe uses in its software today.[18]

documentation is interspersed throughout the code—Knuth wrote TEX as a book, with the code inserted between paragraphs. This mix of English and code is called `WEB`.*

Unsurprisingly, most modern systems don't have good tooling for the late 1970s dialect of Pascal that TEX was written in, so present-day distributions use another program, `web2c`, to convert its `WEB` source into C code. pdfTEX and XƎTEX are built by combining the result with other C and C++ sources. Instead of taking this complicated approach, the LuaTEX authors hand-translated Knuth's Pascal into C. They have used the resulting code since 2009.[22]

*Knuth also released a pair of companion programs named `TANGLE` and `WEAVE`. The former extracts the book—as TEX, of course—and the latter produces TEX's Pascal source code.

B. Additional Resources

For LaTeX

As promised at the start, this book is incomplete. To keep it short, major LaTeX features—like figures, captions, tables, graphics, and bibliographies—haven't been discussed. Use some of these resources to fill in the gaps:

The LaTeX Wikibook, at `https://en.wikibooks.org/wiki/LaTeX`

The Not So Short Introduction to LaTeX,
available at `https://www.ctan.org/tex-archive/info/lshort/english/`

The ShareLaTeX knowledge base, at `https://www.sharelatex.com/learn`

The TeX Stack Exchange, at `https://tex.stackexchange.com/`

For typography

We've spent most of our time here focusing *what* you can do with LaTeX, and little on *how* you should use it to create well-designed documents. Read on:

Practical Typography, by Matthew Butterick.
Available (for free!) at `https://practicaltypography.com`

Stop Stealing Sheep & Find Out How Type Works, by Erik Spiekermann

Thinking With Type, by Ellen Lupton

Shaping Text, by Jan Middendorp

The Elements of Typographic Style, by Robert Bringhurst

Detail in Typography, by Jost Hochuli

Notes

1. Erik Spiekermann, "Type is Visible Language" (presented at Beyond Tellerrand, Düsseldorf, Germany, May 19–21, 2014), `https://www.youtube.com/watch?v=ggQpDu63kk0`
2. Douglas Thomas, "Over the Moon for Futura", *Never Use Futura* (New York, 2017)
3. *Graphic Means: A History of Graphic Design Production*, directed by Briar Levit (2017)
4. Jan Middendorp, *Shaping Text* (Amsterdam, 2014), 71
5. Lynne Truss, *Eats, Shoots & Leaves* (New York, 2003)
6. Matthew Butterick, "Hyphens and dashes", *Practical Typography*, `https://practicaltypography.com/hyphens-and-dashes.html`
7. Donald E. Knuth and Michael F. Plass, *Breaking Paragraphs Into Lines* (Stanford, 1981)
8. Franklin Mark Liang, *Word Hy-phen-a-tion by Com-put-er* (Stanford, 1983), `http://www.tug.org/docs/liang/`
9. Adobe's open-source typefaces are freely available at `https://github.com/adobe-fonts`
10. *LuaTEX Reference* (Version 1.0.4, February 2017), 10
11. Knuth, *The TEXbook*, (Addison-Wesley, 1986), 19
12. R Schlicht, *The microtype package* (v2.7a, January 14, 2018), 4
13. Jost Hochuli, *Detail in typography* (Éditions B42, 2015), 18–19
14. Knuth, *The TEXbook*, 110

15. Knuth, *Digital Typography* (Stanford, 1999), 3–5

16. “yak shaving”, *The Jargon File*, www.catb.org/~esr/jargon/html/Y/yak-shaving.html

17. Knuth, *The TEXbook*, 1

18. Several sources (http://www.tug.org/whatis.html, https://tug.org/interviews/thanh.html, http://www.typophile.com/node/34620) mention TEX’s influence on the *hz*-program by Peter Karow and Hermann Zapf, thanks to via Knuth’s collaborations with Zapf. *hz* was later acquired by Adobe and used when creating InDesign’s paragraph formatting systems.

19. Hàn Thế Thành, *Micro-typographic extensions to the TEX typesetting system* (Masaryk University Brno, October 2000)

20. Jonathan Kew, “XƎTEX Live”, *TUGboat* 29, no. 1 (2007)

21. http://www.luatex.org

22. Taco Hoekwater, *LuaTEX says goodbye to Pascal* (MAPS 39, EuroTEX 2009), https://www.tug.org/TUGboat/tb30-3/tb96hoekwater-pascal.pdf

Colophon

This guide was typeset with LuaLaTeX in Robert Slimbach's Garamond Premier. His revival is based on roman type by 16th century French punchcutter Claude Garamond. Italics are inspired by the work of Garamond's contemporary Robert Granjon.

Monospaced items are set in `Drive Mono`, designed by Elliott Amblard and Jérémie Hornus at Black Foundry.

Captions use Neue Haas Grotesk, a Helvetica restoration by Christian Schwartz. Other digitizations of the ubiquitous Swiss typeface are based on fonts made for Linotype and phototypesetting machines, resulting in digital versions with all the compromises and kludges from those past two generations of printing technology. Schwartz based his work on Helvetica's original drawings, producing a design faithful to the original cold metal type.

URW Futura makes a few guest appearances. Designed by Paul Renner and first released in 1927, Futura has found itself almost everywhere, from advertising and political campaigns to the moon. Douglas Thomas's recent history of the typeface, *Never Use Futura*, is a fantastic read.

Various bits of non-Latin text are set in Noto, a type family by Google that covers *every* language in the Unicode standard.

Finally, Latin Modern—the OpenType version of Knuth's Computer Modern used throughout the book—and TeX Gyre Termes—the free alternative to Times Roman seen on page 1—are the work of Grupa Użytkowników Systemu TeX, the Polish TeX Users' Group. An overview of these projects can be found at the following locations:
`http://www.gust.org.pl/projects/e-foundry/latin-modern`
`http://www.gust.org.pl/projects/e-foundry/tex-gyre`.