What every (LA)TEX newbie should know

Barbara Beeton

Abstract

LaTeX has a reputation for producing excellent results, but at the cost of a steep learning curve. That's true, but by understanding a few basic principles, and learning how to avoid some techniques that may seem obvious but often lead one into the weeds, it's possible to avoid some of that pain.

This presenttion is based on years of looking at good and bad document input and output, answering questions from problem-plagued authors, and trying to write documentation that can be understood on first reading.

Another source of material is the collection of questions and answers provided by the TeX segment of StackExchange. Many newbie questions appear over and over again. Good "duplicate" answers for these have been identified, and links are collected as "Often referenced questions", found at `https://tex.meta.stackexchange.com/q/2419`.

Conventions

In order to avoid overfull lines, error and warning messages shown here will be broken to fit the narrow columns of this article style. Many error messages output by LaTeX will consist of several lines, the first being the message, and the next showing the number of the line on which the error is identified along with the content of that line, up through the error text. A following line, indented so that it, with the numbered line, completes the line as it appears in the input.

Although this presentation will mostly deal with details, please remember that the basic concept of LaTeX is to separate content from structure.

Another applicable concept, one that is often misconstrued in the (LA)TeX community is that of "template". When that term is used here, it means a source file that is an "outline" beginning with `\documentclass` and containing a minimum of basic structural commands into which text and additional definitions can be inserted as appropriate.

Basic structure: Commands, modes and scope

Instructions are communicated to (LA)TeX by means of commands, or "control sequences", which by default begin with a backslash (\). There are two varieties: those which consist of the backslash followed by one character ("control symbol"), and multi-letter commands ("control words") in which only letters (upper- or lowercase) are permitted (no digits or special characters). A control word will be terminated by a space or any other non-letter. But a space after a control symbol will appear as a space in the output.

A user can define new commands, or assign new meanings to existing commands. It's advisable to use `\newcommand` when creating a new definition; this checks to make sure that the command name hasn't been used before, and complains if it has. If it's necessary to redefine a command that already exists, the recommended way is to use `\renewcommand` — but be sure you know what you're doing. For example, redefining `\par` is chancy, as LaTeX uses this "under the covers" for many different formatting adjustments, and it's very easy to mess things up. Single-letter commands are also bad candidates for (re)definition by users, as many of them are predefined as accents or forms of letters not usual in English text; redefining `\i`, for example, can give a nasty surprise if there is the name of a Turkish author in your bibliography. But single-digit commands are not predefined in core LaTeX, so are available for ad hoc use.

TeX, and therefore LaTeX, functions in several distinct modes:

- horizontal — text,
- vertical — beginning of job and between paragraphs,
- math — two varieties: in-text and display.

Starting to input ordinary text is one way to enter horizontal mode. A blank line or explicit `\par` will transition from horizontal to vertical mode. Some operations are limited to a particular mode, or are most effective and predictable within such a mode. For example, it's best to specify `\vspace` and most floats while in vertical mode.

Along with modes, there is the concept of scope, making it possible to localize definitions and operations.

Math mode is one instance of scope; certain characters and operations are valid only within math, and others are invalid there. Within text, math usually begins and ends with `$`, and these must be matched. Display math breaks the flow of text; closing a display returns to text mode unless followed by a blank line or `\par`. More about math later.

Another way of delimiting scope is to wrap it in braces: `{...}`. Within this scope, the meaning of a command may be changed for temporary effect; the definition in effect before the opening brace will be restored as soon as the closing brace is digested. Instead of a brace pair, the commands `\begingroup`...`\endgroup` have the same effect.

In LaTeX, closed environments can be defined, inside which the conditions may be quite different than in the surrounding material. Such environments begin with `\begin{`⟨*env-name*⟩`}` and end `\end{`⟨*env-name*⟩`}`. One example is the `theorem` environment, inside which text is italic. If the environment name at the `\end` doesn't match the one used at `\begin`, an error will be reported:

```
! LaTeX Error: \begin{...} on input line ...
    ended by \end{...}.
```

How to end a paragraph: Not with \\

`\\` does end a line. It is the designated command to end lines in tables, poetry, multi-line math environments, and some other situations. But it does not end a paragraph. A paragraph is ended by a blank line or an explicit `\par`.

Trying to end a paragraph with `\\` can result is some confusing warnings and error messages. For example, `\\` on a line by itself will result in this warning:

```
Underfull \hbox (badness 10000)
    in paragraph at lines ...
```

Furthermore, if the `\\` is preceded by a (typed) space, in addition to the above warning, there may be an extra, unwanted, blank line in the output.

If extra vertical space *is* wanted after a line broken with `\\`, it can be added by inserting an optional dimension, wrapped in brackets: `\\[`⟨*dimen*⟩`]`. If such a bracketed expression is really meant to be typeset, it must be preceded by `\relax`.

Spaces. spurious and otherwise unwanted

A goal of high-quality typesetting is even spacing in text. This is really possible only with ragged-right setting, but even margins are usually preferred, so TEX is designed to optimize spacing in that context.

By default, multiple consecutive spaces are interpreted as a single space. Also, a slightly wider space is left at the end of a sentence, making it easy to tell where the sentence ends. (In French typography, or in the presence of `\frenchspacing`, all spaces are treated the same.) When other unequal spacing is observed in a line, something is fishy.

A sentence is presumed to end with a period or similar punctuation. But abbreviations also end with periods, and abbreviations occur frequently in academic documents, and the wider space isn't wanted there. To indicate an ordinary space, insert a backslash after the period, as in `e.g.\ this or that`, or, if the line should not break after the abbreviation, insert an unbreakable space, as in `Dr.~Knuth`.

A similar, but reverse, situation can occur when an uppercase letter is followed by a period. This is assumed to be the initial of a name; it usually is, and an ordinary interword space is set. But sometimes the uppercase letter is at the end of an acronym, and that ends a sentence. In such a case, add `\@` before the period, and it will restore the wider end-of-sentence space.

But sometimes wider spaces appear in text where they are not expected. This is often caused by spaces inadvertently included in definitions. The end-of-line (here called EOL) is interpreted as a space. (Different operating systems define an EOL differently, but that is taken care of by the TEX engine.) A neatly laid-out definition may be the culprit:

```
\newcommand{\abc}{
    \emph{abc def}
  }
```

will output unwanted spaces *abc def* when used. This can be avoided by placing a `%` sign at the ends of the lines that cause the problem:

```
\newcommand{\ABC}{%
    \emph{abc def}%
  }
```

Then using that command *abc def* will not have the unwanted spaces.

It isn't necessary to use the `%` sign after a control word; remember that a space there just ends the command and is then discarded. But there are places where adding a `%` can cause trouble. After defining any numeric value, TEX will keep looking for anything else that can be interpreted as numeric, so if a line ends with `\xyz=123`, no `%` should be added. Or, if setting a dimension, say `\parindent=2pc`, TEX will keep looking for `plus` or `minus`; a better "stopper" is an empty token, `{}`. (If "plus" or "minus" is there and happens to be actual text, a confusing error message will be produced, but that is rare, and beyond the scope of this presentation.)

There are some other, more obscure situations where unwanted spaces can show up. One is when multiple index entries are inserted in the same place in a file. Often, these are placed on separate lines, and the EOL principle takes effect. Since the multiple spaces aren't consecutive, they remain in the output. Add `%` signs judiciously, remembering to keep one intentional space.

I learned just recently of a really obscure and surprising space. It occurs, like this, in the middle of a w or d, and is caused by the application of a small frame around the colored element by the tcolorbox. This must be suppressed explicily, like this:

```
\usepackage{tcolorbox}
\newcommand{\pink}[1]{{\fboxsep=0pt
    \colorbox{red!20}{#1}}}
```

The resulting word is colorized with no unwanted spaces. While this is really beyond the scope of this presentation, it's something that one should be aware of. If it happens, seek expert assistance.

Font changes

Font changes are a time-honored method of communicating shades of meaning or pointing out distinct or particularly important concepts. Many such instances are built into document classes and packages; for example, theorems are set in an *italic* font, section headings in **bold**, and some journals set figure captions in sans serif to distinguish them from the main text.

LaTeX provides two distinct methods for making font changes. Commands of one class take an argument and limit the persistence of the change to the content of that argument; these have the form of `\textbf{...}` for **bold**, `\textit{...}` for *italic*, etc. The other class sets the font style so that it will not change until another explicit change is made, or it is limited by the scope of an environment; some examples are `{\itshape...}`, `{\bfseries...}`, and `{\sffamily...}`. These command names are best looked up in a good user guide.

Several font-changing commands do different things depending on the context. `\emph{...}` will switch to italic if the current text is upright, or to upright if the current text is italic. Within math, `\text{...}` will set a text string in the same style as the surrounding text; thus, within a theorem, `\text{...}` will be set in italic. If this string should always be upright, like function words, `\textup{...}` should be used instead.

Basic TeX defined two-letter names for most font styles. All of these are of the persistent type. They should be avoided with LaTeX, as some of the LaTeX forms provide improvements, such as automatic application of the italic correction, which would otherwise have to be input explicitly.

Math

Math is always a closed environment. If started, it must be ended explicitly and unambiguously. Within text, math begins and ends with `$`; there must therefore be an even number of `$` signs in a document. LaTeX also provides `\(...\)` for in-text math, but most users stick with the `$`. Many different display environments are defined by the packages amsmath and mathtools, and it is worthwhile to learn them by reading the user guides. (mathtools loads amsmath, so it's not necessary to load amsmath separately.)

Within math, all input spaces are meaningless to (LA)TeX; they can be entered in the source file as useful to make it readable to a human. Blank lines, however, are considered errors. This was a decision in the design of TeX to make it easy to detect an unterminated math element, because math should not span a paragraph break. In both in-text math and displays, the error message will be

```
! Missing $ inserted.
```

If a blank line occurs in a multi-line display environment from amsmath, the *first* error message will be

```
! Paragraph ended before ⟨env-name⟩
    was complete.
<to be read again>
```

This will be followed by *many* more error messages, all caused by the first. These will be confusing and misleading. Always fix the problem identified by the first error and ignore the rest; they will disappear once the first error is fixed, here, the blank line is removed.

If the appearance of a blank line is wanted for readaility, begin it with a `%` sign. It's also bad form to leave a blank line before display math; a display is usually a continuation of the preceding paragraph, and avoiding a paragraph break also avoids a possible page break before the display.

As in other environments, the `\end` name must exactly match the name specified at `\begin`. A "shorthand" for a single-line, unnumbered display is `\[...\]`. The environments designed for multi-line displays should not be used for a single-line display.

Although LaTeX provided `eqnarray` as a display environment, don't use it. If the display is numbered and the equation is long, the equation can be overprinted by the equation number.

Tables, figures, and other floats

The number of floats, their positions on a page, and the spacing around and between them is defined by the document class. So if something doesn't work as you expect (hope for?), any potential helper will insist on learning what document class is being used.

Input for a float must appear in the source file while there is still enough space on the output page to fit it in. In particular, on two-column pages, a `\begin{figure*}` or `\begin{table*}` must occur in the source *before* anything else is set on the page. The basic float handling does not allow full-width floats to be placed anywhere but at the top of a page; some packages extend this capability, but those won't be discussed here.

Here are the defaults for the basic `article` class.

- Total number of floats allowed on a page with text: 3.

- Number of floats allowed at top of page: 2. Percentage of page allowed for top-of-page floats: 70%/
- Number of floats allowed at bottom of page: 1. Percentage of page allowed for bottom-of-page floats: 30%/
- Minimum height of page required for text: 20%.
- Minimum height of float requiring a page by itself: 50%.

The reference height is `\textheight`. That is, the height of page headers and footers is excluded.

If an insertion is small, must be placed precisely and fits in that location, don't use a float. `\includegraphics` or one of several available table structures should be used directly, often wrapped in `\begin{center} ... \end{center}`/ (Within a float, use `\centering` instead.)

The wrapfig package supports cut-in inserts at the sides of a page or column. Refer to the documentation for details.

By tradition, captions are applied at the top of tables and the bottom of figures. If an insertion is not a float, the usual `\caption` can't be used. Instead, `\usepackage{caption}` and the command `\captionof`.

The document class and preamble

When embarking on a new document, the first thing is to choose the document class. If the goal is publication in a particular journal, check the publisher's instructions to see what is required. Many, but not all, popular journal classes are available from CTAN. If the project is a thesis or dissertation, find out the special requirements, and if your institution provides a tailored class, obtain a copy. Try to determine whether it is actively maintained, and if there is local support. Read the documentation. It is the responsibility of the document class to define the essential structure of the intended document. If the document you are preparing differs in essential ways from what is supported by the document class, the time to get help is *now*.

There will be features not natively supported by the document class; for example, the choice of how to prepare a bibliography may be left to the author. This is why packages have been created.

Most packages are loaded in the preamble; the one exception is `\RequirePackage`, which may be specified before `\documentclass`, and is the place where options should be loaded. Some authors create a preamble that is suitable for one document, then use the same preamble for their next document, adding more packages as they go. And some unwitting newbies "adopt" such second-hand "templates" without understanding how they were created. *Don't do it!*

Start with a suitable document class and add features (packages, options, and definitions) as they become necessary. Organize the loading of packages into logical groups (all fonts together, for example), and be careful not to load a package more than once; if options are needed, any loaded with a non-first `\usepackage` will be ignored. Some packages automatically load other packages; for example, mathtools loads amsmath and amssymb loads amsfonts. And, very important, pay attention to the order of package loading: hyperref must be loaded (almost) last; the few packages that must come after hyperref are all well documented. Read the documentation.

Processing the job

Once the file is created, it's time to produce output. There are several engines to choose from: pdfLaTeX, XeLaTeX, and lusLaTeX. These can be run interactively from the command line, or initiated from an editor. Assuming there are no errors, how many times the file must be processed depends on what features it contains.

(La)TeX is "one-way". If any cross-references or `\cite`s are present, this information is written out to an `.aux` file; information for a table of contents is written to a `toc` file, and other tables are also possible. The bibliography must be processed by a separate program (and its log checked for errors) with the reformatted bib data written ont to yet another file. Then LaTeX must be run (at least) twice more — once to read in the `.aux` and other secondary files and include the bibliography and resolved cross-references, and the second time to resolve the correct page numbers (which will change when the TOC and similar bits are added at the beginning).

All this assumes that there are no errors. Errors will be recorded in the log file. Learn where the log file is located, and make a habit of referring to it. Warnings, such as those for missing characters, will also be recorded there, but not shown online:

```
Missing character: There is no ⟨char⟩
    in font ⟨font⟩!
```

In the log, errors may appear with closely grouped line numbers. If so, and the first is one that interrupts the orderly processing of a scoped environment, following errors may be spurious. So fix the first error and try processing before trying to understand the others; often, they may just go away.

Good luck. With practice comes understanding.