An introduction to TEX and friends

Gavin Maltby

November 1992

Contents

List of Tables

Chapter 1

Getting acquainted with TEX

TEX is well known to be *the* typesetting package, and a vast cult of TEX lovers has evolved. But to the beginning TEX user, or to someone wondering if they should bother changing to TEX, it is often not clear what all the fuss is about. After all, are not both WordPerfect and Ventura Publisher capable of high quality output? Newcomers have often already seen what TEX is capable of (many books, journals, letters are now prepared with TEX) and so expect to find a tremendously powerful and friendly package. In fact they *do*, but that fact is well hidden in one's initial TEX experiences. In this chapter we describe a little of what makes TEX great, and why other packages cannot even begin to compete. Be warned that a little patience is required—TEX's virtues are rather subtle to begin with. But when the penny drops, you will wonder how you ever put up with anything different.

1.1 The spirit of TEX

In order to really appreciate TEX one needs to get a feel for what I call the "spirit" of TEX. When TEX appears to be making me work overtime to achieve something that I think ought to be perfectly straightforward, consultation with the TEX spirit shows me the error of my ways.

1.1.1 TEX is a typesetter, not a word-processor

TEX was designed with no limiting application in mind. It was intended to be able to prepare practically any document—from a single page all-text letter to a full blown book with huge numbers of formulae, tables, figures etc. The size and the complexity of a TEXable document is limited only by hardware considerations. Furthermore, TEX seeks to achieve all this whilst setting typesetting standards of the highest order for itself. The expertise of generations of professional printers has been captured in TEX, and it has been taught all the tricks of the trade.

Historically, printers prepared a document by placing metal characters in a large tray and arranging and binding them to form a page. This was very precisely done, but the ultimate precision was limited because of the mechanical nature of things and by time considerations. TEX prepares a page in an analogous manner (putting your characters and formulae into "boxes" which are then "glued" together to form the page), but has the advantage of enormous precision because placement calculations are performed by computer. Indeed, TEX's internal unit (the "scaled point") is about one-hundredth of the wavelength of natural light!

"But conventional word processors run on computers , too", you object. Yes, but their fundamental limitation is that they try to "keep up" with you and "typeset" your document as you type. This means that it can only make decisions at a local level (eg, it decides where to break a line just as you type the end of the line). TEX's secret is that it waits until you have typed the *whole* document before it typesets a single thing! This means that TEX can make decisions of a global nature in order to optimise the aesthetic appeal of your document. It has been taught what looks good and what looks bad (having been given a measure of the "badness" of various possibilities) and makes choices for your document that are designed to make it "minimally bad".

But TEX's virtues run much deeper than that, which is just as well because it is possible to get satisfactory, though imperfect, results from some word processors. One of TEX's strongest points is its ability to typeset complicated formulae with ease. Not only does TEX make hundreds of special symbols easily accessible, it will lay them out for you in your formulae. It has been taught all the spacing, size, font, ... conventions that printers have decided look best in typeset formulae. Although, of course, it doesn't understand

any mathematics it knows the grammar of mathematics—it recognises binary relations, binary operators, unary operators, etc. and has been taught how these parts should be set. It is consequently rather difficult to get an equation to look bad in TEX.

Another advantage of compiling a document after it is typed is that cross-referencing can be done. You can label and refer back to chapters, sections, tables etc. by *name* rather than absolute number, and TEX will number and cross-reference these for you. Similarly, it will compile a table of contents, glossary, index and bibliography for you.

Essential to the spirit of TEX is that *it formats the document whilst you just take care of the content*, making for increased productivity. The cross-referencing just mentioned is just part of this. Many more labour-saving mechanisms are provided for through *style files*. These are generic descriptions of classes of documents, teaching TEX just how each class likes to be formatted. This is taught in terms of font preferences, default page sizes, placement of title, author, date, etc. For instance, a `paper` style file could teach TEX that when typesetting a theorem it should embolden the part that states the theorem number and typeset the text of the theorem statement in slanted Roman typeface (as in many journals). The typist simply provides and indication that a theorem is being stated, and then types the text of the theorem *without* bothering to choose any fonts or do any formatting—all that is done by the style file. Style files exist for all manner of document—letters, articles, papers, books, proceedings, review articles, and so on.

In addition to style files, there are *macro packages*. A *macro* is just a definition of a new TEX command in terms of existing ones. Don't think small when you think of macros! When typing a document that has a lot of repetition in it, say the same expression is used again and again in different different equations, you can define a macro in your document to abbreviate that expression. But macros can teach TEX how to typeset all sorts of complicated structures, not just parts of an equation. Many macro packages (files that are just collections of definitions) have been written to teach TEX all sorts of applications. There are specialist maths packages ($\mathcal{AMS}$-TEX, $\mathcal{AMS}$-LATEX), general purpose packages (LATEX), packages for setting tree diagrams, Feynmann diagrams, languages like Chinese, Arabic and Ancient Greek, orchestral scores, and many, many more. All these are freely available, a spin-off of the giant TEX cult.

Another facet of the design of TEX allows it to use practically *any* output

device. In fact, TEX doesn't talk to any printers, screens, phototypesetters at all! Instead, when a document is compiled a *device independent* (`.dvi`) is produced—TEX does not compile with any particular output device in mind. Printer drivers are then invoked on this `.dvi` file and, in consultation with the font data for that printer, produce output suitable for the particular device. You can choose an HP Laserjet driver, or an Apple LaserWriter driver, or a dot matrix driver etc. All use the same `.dvi` file as input (and remember the material in there is set to enormous accuracy) and attempt to image that file on the particular device as faithfully as possible. If you are using a top of the line laser printer or phototypesetter, then TEX's massive internal precision will not be wasted. Alternatively, a dot matrix printer will give a coarse approximation of the ideal image that is suitable only for proof-reading. In addition to portability, these `.dvi` files help ensure that there are very few printing surprises when you move from one device to another: how many times has your favourite word-processor made you reformat a document when you wish to change printers?

There are many other motivations one could cite for the superiority of TEX. But it is time that we started to get our hands dirty. One last comment: TEX was not designed to supplant secretaries and professional printers—it was designed to aid them in their work and, in the words of the TEX designer Donald Knuth, allow them to "go forward and create masterpieces of the publishing art".It also allows those who generate the material to be typeset—mathematicians, physicists, computer scientists, etc—to prepare their own documents in a language that is intimately linked to the language we use for writing such material.

The novice reader will still have no idea of what a TEX source file looks like. Indeed, why do we keep referring to it as a *source file*? The fact of the matter is that TEX is essentially a *programming language*. Just as in any compiled language (e.g., Pascal, C) one prepares a source file and submits it to the compiler which attempts to produce an object file (`.dvi` file in the TEX case). To learn TEX is to learn the command syntax of the commands that can be used in the source file.

1.1.2 Typical TEX interfaces

TEX was designed to run on a multitude of computers. It is therefore the case that the documentation for TEX and its "friends" LATEX, $\mathcal{AMS}$-TEX, etc. is

not computer specific. Only command syntax is described—i.e., the content of your source file—but few details of how to get from there to a printout are given. Those details are left to site-specific documents.

The average user loses little in using TEX on, say, a PC rather than on a bigger machine. Indeed, compilation times on the new PCs begin to rival those on a Sun Sparc Station 2 (no slouch). Running on top of DOS can cause memory problems when very large documents are being prepared. That aside, the quality of the document is not affected because of the careful design of TEX—whether you work on a a machine with massive floating point precision or a modest XT the `.dvi` files produced on compilation will be identical; and when those files are submitted to printer equivalent printer drivers (say for an HP LaserJet III attached to a Sun in one case and a PC in the other) the output will be identical because the font information they draw on is identical.

By the nature of TEX most time is spent editing the source document (before submitting it for compilation). No special interface is necessary here, you just use your favourite text editor (perhaps customising it to enhance TEXnical typing. Thus TEX user interfaces are usually small and simple, often even missing. One frequently uses TEX at command line level, just running the editor, compiler etc. as you need them. Sometimes a TEXshell program is present, which runs these for you when you choose various menu options.

Whatever the interface, there are just a few basic steps to preparing a document:

1. Choose a document style to base your document on (e.g., letter, article).

2. Glance through the material you have to type, and decide what definitions might be made to save you a lot of time. Also, decide on the overall structure of the prospective document (e.g., will the largest sectional unit be a chapter or a part?). If you are going to compose as you type, then pause a moment to think ahead and plan the structure of your document. The importance of this step cannot be overstressed, for it makes clear in *your* mind what you want from TEX.

3. Prepare your input file, specifying only the content and the logical structure (parts, sections, theorems,...) thereof and forgetting about formatting details.

4. Submit your input, or source, file to the TEX compiler for compilation of a `.dvi` file.

5. If the compiler finds anything in your source file strongly objectionable, say incorrect command syntax, then return to editing.

6. Run a *previewer* to preview your compiled document on the screen. Resolution is only limited by your screen, and can be very good indeed on some modern monitors.

7. Go back to editing your document until glaring errors have been taken care of.

8. Make a printout of your compiled document, and check for those errors that you failed to notice on the screen.

Performing these steps may be effected through typing at the system prompt (barebones technique) or through choosing menu options in a TEXshell program. The latter will probably provide some conveniences to make your life easier.

If you think this sounds like a lot of work, it is time that you consult with the TEX spirit! Sure your first couple of tries may be hesitant, but before long you'll find that you can take *less* time to prepare a document on TEX than on any other package.

Chapter 2

Getting started with LaTeX

2.1 Why start with LaTeX?

To answer this question we must say a little more about some of the macro packages we mentioned earlier.

The TeX typesetting system was designed by the eminent Stanford computer scientist Donald Knuth, on commission from the American Mathematical Society. It was designed with enormous care, to be ultimately powerful and maximally flexible. The enormous success of Knuth's design is apparent from the vast number of diverse applications TeX has found. In reading the following you must keep one thing clearly in mind: *there is only TeX language, and all the other packages whose names end in the suffix -TeX simply harness it's power via a whole lot of complicated macro definitions.*

TeX proper is a collection of around 300 so called *primitive* typesetting commands. These work at the very lowest level, affording enormous power. But to make this raw power manageable, some macros must be defined to tame raw TeX somewhat. The standard set of macros is called Plain TeX, and consists of about 600 macro definitions. It is clear that these definitions must be made in terms of TeX primitives, or in terms of previously made definitions. Plain TeX, however, is still no place for the timid. A strong working knowledge of TeX is still required to understand the ins and outs of Plain TeX.

In the few years after the initial TeX release (1982), the macro packages $\mathcal{AMS}$-TeX and LaTeX were born. $\mathcal{AMS}$-TeX was written by Michael Spi-

vak, also on commission from the AMS. This package was designed to facilitate the preparation of the numerous books, journals, and review indices that fall under the auspices of the AMS and its affiliates. Married to the macro package was a style file—the AMS preprint style. This was distributed along with the macro package, so that authors submitting to journals could use it in the preparation of their articles. The given style was based on the style used by the *Journal of the American Mathematical Society*, i.e., it conformed to their page sizes and typographical conventions. This meant that people around the world produced papers that were all based on the same style. The clever part is this: when a source file is submitted to a journal *other* than the *Journal of the AMS*, the journal staff simply substitute their style file for the AMS preprint style and the paper will appear completely different *with no other changes to the source code*! To create their style file, a journal just needed to tweak the standard AMS prepint style: for instance, the original preprint style places author addresses at the very end of a paper; If a journal wishes this to appear on the first page then they just modify their in-house version of the style file, and the change will be effected without having to change the file submitted by the author.

LaTeX was written for more general usage. It lacks some of the mathematical finesse inherited by $\mathcal{AMS}$-TeX from the vast experience of the AMS technical staff, but more than makes up for this in its ability to enhance the typesetting of letters, books, poetry, etc. LaTeX also scores high points for its enhanced command syntax.

With $\mathcal{AMS}$-TeX and LaTeX being released at around the same time (1984–1985), there were born many $\mathcal{AMS}$-TeX literate but LaTeX illiterate users, and conversely. LaTeX was easier to learn because of its more friendly syntax, and also provided powerful cross-referencing commands that $\mathcal{AMS}$-TeX did not. So the AMS commissioned another project to furnish LaTeX users with the additional power of $\mathcal{AMS}$-TeX while not compromising the LaTeX command syntax or cross-referencing commands. This resulted in the $\mathcal{AMS}$-LaTeX macro package and associated style file for submission to journals.

That is why we will kick off our TeX careers with LaTeX! It is easier to learn and provides many conveniences, and the user who requires additional mathematical typesetting prowess can easily move on to $\mathcal{AMS}$-LaTeX. Much of what we say will be true for TeX itself, but we shall regard LaTeX as the lowest common-denominator. By far the majority of LaTeX and $\mathcal{AMS}$-LaTeX users will never have to learn "raw" TeX, for they will be shielded from direct

exposure by the numerous powerful macro packages. In the rare case that something way out of the ordinary is required, the local TEX guru can be consulted.

2.2 LATEX formats, and we compose

The *free form nature* of the input file is essential to the spirit of TEX. As we type, we do not concern ourselves with linebreaks and pagebreaks so much as the content of what we are typing. In fact, we'll see that TEX will choose nice line breaks even for bizarre looking input. This is just part of the concept of only having to describe the *logical* structure of the document to LATEX, and not worry about nuisance-value formatting details. We inform LATEX of the logical structure of our document by telling it when to begin a new paragraph, subsection, section, chapter, theorem, definition, remark, poem, list etc. When typing a particular element of the logical structure, we need pay little attention to how we lay our source file out.

A consequence of this is that we have to go to a bit of effort to mess things up. Starting a new line, for instance, entails more than just pressing Return because LATEX will just regard the next word you type as exactly that—the next word in the paragraph. You have to specifically ask for a line to be terminated. Things like this may seem to be a bit of a nuisance, but it is a small price to pay for the automatic formatting that necessitated it. Further, such small inconveniences have been localised to rare events. I have, for instance, not once forced a new line up until this point in the present document.

2.3 Document styles

We have explained the concept of a document style during our discussion of the virtues of TEX and the discussion of $\mathcal{AMS}$-TEX. It remains to name a few, and indicate where they would be used. One *always* has to choose a document style when preparing a document with LATEX.

The basic document styles in LATEX are `letter`, `article`, `report`, and `book`. Many more are available, but these few cover the majority of straightforward applications. This is because styles are not rigid—you can impose

your own parameter choices if you want. So one chooses the style that most closely approximates the document you have in mind, and performs some minor tweaks here and there. The `article` style is used for documents that are to have the appearance of a journal or magazine article. The `report` style is usually used for larger documents than the `article` style. These styles really only differ in their choice of default page size, font, placement of title and author, sectional units, etc. and on how they format certain LaTeX constructs. You use the same LaTeX commands in each. Since the examples here will be small, we will choose to use the `article` document style.

There are a number of possible options with each document style. The syntax for choosing a document style follows. Don't worry if this leaves you with no idea of how to choose a document style, for we will soon be seeing some examples. Also, remember that an argument in square brackets is optional, and can omitted altogether (including the brackets).

`\documentstyle [`*options*`]{`*style*`}` where *style* is the main document style (eg `report`) and the optional argument *options* is a list of document style options chosen from the following list:

`11pt` chooses 11-point as the default font size for the document, instead of the default 10-point.

`12pt` chooses 12-point as the default font size.

`twoside` formats output as left and right pages, as in a book.

`twocolumn` produces two-column magazine like output.

`titlepage` applies to the `article` style only, causing the title and abstract to appear on a page each.

In fact there are many, many more document style options but we won't mention any more here.

2.4 Preparing a non-mathematical document

We assume that you have read the local guides to TeX at your site and have decided which system environment you want to work in. There you have been shown how to perform the steps required to produce a printed document from a LaTeX source file.

2.4.1 Sentences and paragraphs

Let's create our very first LaTeX document, which will consist of just a few paragraphs.

As mentioned above, paragraph input is free-form. You type the words and separate them by spaces so that LaTeX can distinguish between words. For these purposes, pressing Return is equivalent to inserting a space—it does not indicate the end of a line, but the end of a word. You tell LaTeX that a sentence has ended by typing a period followed by a space. LaTeX ignores extra spaces; typing three or three thousand will get you no more space between the words that these spaces separate than typing just one space. Finally, you tell LaTeX that a paragraph has ended by leaving one or more blank lines. In summary: LaTeX concerns itself only with the logical concepts end-of-word, end-of-sentence, and end-of-paragraph. Sounds complicated? An example should clear things up:

```
\documentstyle{article}
\begin{document}
Words within a sentence are ended by spaces.  One space
between words  is     equivalent      to any number. We are only
interested in separating      one    word      from        the
next, not in formatting       the space between them.
For these purposes, pressing Return
at the end of a line
and starting a new word on the next line
just serves to separate
words, not to cut a line short.
The end of a sentence is indicated by a period
followed by one or more spaces.

The end of a paragraph is indicated by leaving a blank line.
All this
means that we can type without too much regard for layout, and
the typesetter will sort things out for us.
\end{document}
```

produces the result

> Words within a sentence are ended by spaces. One space between words is equivalent to any number. We are only interested in separating one word from the next, not in formatting the space between them. For these purposes, pressing Return at the end of a line and starting a new word on the next line just serves to separate words, not to cut a line short. The end of a sentence is indicated by a period followed by one or more spaces.
>
> The end of a paragraph is indicated by leaving a blank line. All this means that we can type without too much regard for layout, and the typesetter will sort things out for us.

Perhaps you would like to try running LaTeX on the above input. Consult your local guide for details.

Note that we have learned more than just how LaTeX recognises words, sentences and paragraphs. We've also seen how to specify our choice of document style and how to tell LaTeX where our document begins and ends. Any material that is to be printed must lie somewhere between the declaration of `\begin{document}` and that of `\end{document}`. Definitions that are to apply to the entire document can be made before the declaration of the document beginning. The specification of document style must precede all other material.

In future examples we won't explicitly display the commands that select document style and delimit the start and end of the document. But if you wish to try any of the examples, don't forget to include those commands. The `article` document style will do for most of our examples. Of course, the preceding example looks not at all like an article because it is so short and because we specified no title or author information.

Most of what you need to know to type regular text is contained in the example above. When you consider that by far the majority of any document consists of straight text, it is obvious that LaTeX makes this fabulously straightforward. LaTeX will do all the routine work of formatting, and we simply get on with the business of composing.

LaTeX does more than simply choose pleasing line breaks and provide natural spacing when setting a paragraph. Remember we said that TeX has inherited the knowledge of generations of professional printers—well part of that knowledge includes being on the look-out for *ligatures*. These are combinations of letters within words which should be typeset as a single special symbol because they will "clash" with each if this is not done. Have a look at these words

flight, flagstaff, chaff, fixation

and compare them with these

flight, flagstaff, chaff, fixation

See the difference? In the first set I let LaTeX run as it usually does. In the second I overruled it somewhat, and stopped it from creating ligatures. Notice how the 'fl', 'ff', and 'fi' combinations are different in the two sets—in the former they form a single symbol (a ligature) and in the latter they are comprised of two disjoint symbols. There are other combinations that yields ligatures, but we don't have to bother remembering any of them because LaTeX will take care of these, too.

Notice, too, that LaTeX has been taught how to hyphenate the majority of words. It will hyphenate a word if it feels that the overall quality of the paragraph will be improved. For long words it has been taught several potential hyphenation positions.

LaTeX also goes to a lot of trouble to try to choose pleasing page breaks. It avoids "widows", which are single lines of a paragraph occurring by themselves at either the bottom of a page (where it would have to be the first line of a paragraph) or at the top of a page (where it would have to be the last). It also "vertically justifies" your page so that all pages have exactly the same height, no matter what appears on them. As testimony to the success of the pagebreaking algorithm, I have (to this point) not once chosen a page break in this document.

2.4.2 Punctuation

Typists have a convention whereby a single space is left after a mid-sentence comma, and two spaces are left after a sentence-ending period. How do we enforce this if LaTeX treats a string of spaces just like a single one? The answer, unsurprisingly, is that we *don't*.

```
To have a comma followed by the appropriate space,  we simply
type a comma follows by at least one space. To end a sentence
we type a period with at least one following space.     No space will
be inserted if we type a comma or period followed straight away
by something other than a space, because there are times when
we won't require any space, i.e., we do what comes narurally.
```

will produce

> To have a comma followed by the appropriate space, we simply type a comma follows by at least one space. To end a sentence we type a period with at least one following space. No space will be inserted if we type a comma or period followed straight away by something other than a space, because there are times when we won't require any space, i.e., we do what comes naturally.

LaTeX will produce suitable space after commas, periods, semi-colons and colons, exclamation marks, question marks etc. if they are followed by a space. In stretching a line to justify to the right margin, it also knows that space after a punctuation character should be more "stretchable" than normal inter-word space and that space after a sentence-ending period should be stretched more than space after a mid-sentence comma. TeX knows the nature of punctuation if you stick to the simple rules outlined here. As we've already said, those rules tell LaTeX how to distinguish consecutive words, sentences, phrases, etc.

Actually, there is more to ending sentences than mentioned above. Since LaTeX cannot speak English, it works on the assumption that *a period followed by a space ends a sentence unless the period follows a capital letter.* This works most of the time, but can fail. To get a normal inter-word space after a period that doesn't end a sentence, follow the period by a *control space*—a `\␣` (a `\` character followed by a space or return). Very rarely, you will have to force a sentence to end after a period that follows a capital letter (remember that LaTeX assumes this doesn't end a sentence). This is done by preceding the period with a `\@` command (you can guess from the odd syntax that this is rarely needed).

It's time we saw some examples of this. After all, this is our first experience of *control symbols* (don't worry, there are many more to come).

```
We must be careful not to confuse intra-sentence periods
with periods that end a sentence, i.e.\ we must remember
that our task is to describe the sentence structure.  Periods
that the typesetter requires a little help with typically result
from abbreviations, as in etc.\ and  others.  We have to work
somewhat harder to break a sentence after a capital letter,
but that shouldn't bother us to much if we keep up our intake
of vitamin E\@.  All this goes for other sentence-ending
punctuation characters, so I could have said vitamin E\@!
Fortunately, these are rare occurrences.
```

results in

> We must be careful not to confuse intra-sentence periods with periods that end a sentence, i.e. we must remember that our task is to describe the sentence structure. Periods that the typesetter requires a little help with typically result from abbreviations, as in etc. and others. We have to work somewhat harder to break a sentence after a capital letter, but that shouldn't bother us to much if we keep up our intake of vitamin E. All this goes for other sentence-ending punctuation characters, so I could have said vitamin E! Fortunately, these are rare occurrences.

Quotation marks is another area where LaTeX will do some work for us. Keyboards have the characters ‘, ‘, and " but we want to to have access to each of ‘, ’, “, and ”. So we proceed like this:

```
‘\LaTeX’ is no conventional word-processor, and
to to get quotes, like ‘‘this’’, we type repeated
‘ and ’ characters.  Note that modern
convention is that ‘‘punctuation comes after
the closing quote character’’.
```

which gives just what we want

> ‘LaTeX’ is no conventional word-processor, and to to get quotes, like “this”, we type repeated ‘ and ’ characters. Note that modern convention is that “punctuation comes after the closing quote character”.

Very rarely, you have three quote characters together. Merely typing those three quote characters one-after-the-other is ambiguous—how should they be grouped? You tell LaTeX how you want them grouped by inserting a very small space called `\,`.