An Introduction to TEX for New Users[1]

Alan Hoenig

John Jay College
17 Bay Avenue
Huntington, NY 11743 USA
(516) 385-0736

ajhjj@cunyvm

Abstract

The purpose of this brief introduction is *not* to present a tutorial into the use of TEX, but rather to introduce the user to the whole notion of what it means to use TEX, how TEX differs from other typesetting systems, and what the advantages are to using TEX.

1 Introducing an Introduction

When researching TEX and its uses, it's easy to feel you've fallen into a slippery pit with sharp, upended spikes at the bottom. What's all this talk of backslashes, macros, LATEX, and bad puns, and what relevance does it have to producing a nicely printed document? And what do you mean, TEX isn't WYSIWYG?

Thus this brief discussion. It's not a tutorial about TEX, for who could create such a thing in only a few pages? I will make mention of certain basic TEX technical matters, but only in passing. My aim is to give my own idiosyncratic view TEX, including an assessment of why you'd want to bother with it. You'll find the TEX pit is not so slippery after all, and those spikes are more like toothpicks. (But there's nothing to be done about the bad puns.)

2 To Begin: What is TEX?

We all of us find ourselves having to communicate information to others in a written fashion. That is, we face the constant need to prepare letters, memos, reports, books, and so on. For high-quality presentations, we need a way to *typeset* this information using the conventions of typesetting that have evolved over the centuries.

We may choose to use a computer to help with this chore. If we do, we need special typesetting software, and TEX is one such software system for performing this typesetting. For certain needs, many feel it is the *best* typesetting system. Before discussing why this should be so, let's remind ourselves how TEX works.

This reminder is important because TEX works differently from other systems that paint type on paper. Many people involved with entering words at keyboards tend to regard word processing and even desktop publishing (DTP) in the same light as typesetting. Such systems revolve around a comforting two-step life cycle:

- Enter the text at the keyboard, observing the screen all the while to see how the final output will appear; and then
- Print the document.

Pay attention to the first item. Most of the time in these programs there is a correspondence between the appearance of the text and stuff you enter at the keyboard and the final printed appearance of the document.

Compare this to the TEX life cycle:

- Enter the text at the keyboard, using a text editor (not TEX—it is not itself a text editor).
- Now run the text file through TEX. With luck, there will be no errors, and we can proceed to the next step. Otherwise, as most TEX users come to know early on in their TEX careers, it's back to step 1.
- A successful run through TEX produces not a document but rather a new file, a so-called *device independent* file. With the aid of a separate program called a *device driver* appropriate to your printer (printing device, hence the term *device driver*), you print the document. Only now does your document appear, right before your eyes.

A lot about TeX can be learned by carefully considering and contrasting these two ways of doing things.

2.1 TeX is More than One Program

While Ventura *Publisher* or Adobe *Pagemaker* are standalone programs, TeX apparently isn't. A careful count indicates we need at least three programs to do TeX.

First of all, there is the text editing program with which we prepare our document file and which is separate from TeX. TeX is pretty tolerant of which such program you can use, but just be aware that TeX itself makes no provision for accepting your text and therefore makes no provision for displaying your text as you type it. (A slight exception: some integrated implementations of TeX do have an editing mode for preparing the manuscript, but strictly speaking, it's not TeX doing the editing, it's an add-on component.)

You may use any editor so long as the resulting file is extended ASCII. In this way, your *source file*—this file that you prepare to feed to TeX—is portable and can be fed to virtually any implementation of TeX working on virtually any computer platform.

On my PC, which is where I do most of my TeXing, I use inexpensive or free editors (they're in the public domain or are shareware). They aren't fancy by any means, but they deliver ASCII text TeX needs. I don't care that there are many fancy things they cannot do, for it's not *they* but TeX that will do the formatting.

2.2 TeX Itself

The program TeX only enters the picture during the second stage of the cycle. TeX requires as input the source file you have just finished. It considers your text in light of the formatting and typesetting commands with which you have peppered your source file, and if all goes well, it delivers as output a `dvi` file. If all does not go well, because you mistyped a TeX command or because your commands are misused, then TeX halts and gives you an error message.

This process of feeding separate source files to TeX, correcting whatever errors may occur, and waiting for a clean `dvi` file reminds many users of the process of programming a computer. After all, creating a working computer program requires creation of a separate program file, which is compiled (again, if all goes well) to produce the final object module.

It's only the object module which "runs" the program. Comparing this with TeX, the source file is like the program file, the `dvi` file is similar to the object module, and the process whereby TeX ingests and analyzes your file is like the compilation process for Pascal or Fortran. For that reason, one often speaks of *compiling* a document with TeX.

But bear in mind—this is an analogy only! TeX users need have no programming experience, ability, or inclination in order to use TeX with great profit.

2.3 The dvi File

This is a file in which the positions of all elements of your document—letters, figures, punctuation, square root symbols, and so forth—are specified using a very general placement language. There has been no cooperation between printer manufacturers and there are as many ways to tell a printer to advance to the top of the next page, say, as there are different printers.

The author of TeX did not want to get bogged down in these considerations. He felt ill at ease with the concept of anchoring TeX to any one printer or even to any single printer technology, and so he created this general and generic `dvi` language to act as a gateway to all printers. Therefore, yet a third program is needed to translate this general `dvi` language into a form comprehensible to a particular printer. This is the job of the *device driver*, a program intended to do this translation so the printer can paint the characters, lines, and so on onto the actual page.

By virtue of this separation of duties of TeX and of a device driver, TeX becomes relevant across a broad spectrum of printing technologies. Device drivers exist to print your documents on dot matrix printers, on hi-tech laser printers, and on costly phototypesetters. Except for resolution of individual characters, your document is identical across printing hardware. That is, the page breaks, line breaks, position of math characters, and so on will not vary. That makes it possible to use a laser printer as a proofing device. Once you are pleased with the look of your document, you may ship off your source file or your `dvi` file to a service bureau for printing off a single, high-quality copy, which you give to your printer, who makes the plates for the whole kit-and-caboodle of the printing manufacturing process.

2.4 Screen Previewers

There is yet a fourth type of program that is part of the TeX process, called a *screen previewer*. Such a program makes it possible to see on the video terminal what your document will look like. Since screen previewers work much faster than printers and with a lot less bother, it's convenient to have one for your display terminal. Understand, though, that previewers are special cases of device drivers; that is, instead of printing to paper, a screen previewer allow you to "print" a `dvi` file to your computer's monitor.

2.5 The WYSIWYG Issue

Almost everyone knows by now that WYSIWYG stands for "what you see is what you get". With a fancy word processor, a centered chapter title set in some fancy display font really looks that way on your screen. The theory is that you have immediate visual feedback and you can make corrections or revisions right away.

If you refer back to the description of the TEX life cycle, you see that TEX could not possibly act this way. Remember, the process of preparing the document source file and introducing it to TEX are entirely separate. Computer people refer to processes like this as *batch processing* (in contrast to *on-line* WYSIWYG processing). Anyway, since the TEX program lies quiescent at the time you are preparing your document file, it would be impossible for TEX to intercede in the on-screen formatting of your document. To add possible insult to injury, it's a fact that your source file might only *approximately* resemble the look of the final output.

For those of you who die without WYSIWYG programs, let me say that the situation, though bad, could be worse. For some integrated implementations, TEX processes your file so rapidly, passing the resulting `dvi` file to the screen previewer automatically, that it is an "almost WYSIWYG" system.

It seems as if TEX requires perhaps a good deal more work than, say, *Pagemaker*. If this is true, why bother with TEX at all?

3 The Advantages of TEX

I fiercely maintain that TEX is worth the bother, if bother indeed it be. First of all, let's remind ourselves that none of the leading contenders for desktop publishing are particularly painless. There is no royal road to fine typesetting.

Let's look at the WYSIWYG issue first. Is this WYSIWYG deficiency a true deficiency? I and others would argue that it is not. Leslie Lamport, in one of the most spirited defenses of the TEX *Gestalt* I've seen, remarks that the WYSIWYG acronym should be replaced by

WYSI*A*YG

—what you see is *all* you get. For WYSIWYG systems generally require you to achieve the look you want by manually attending to many details you quickly tire of attending to.

F or example, in TEX I was able to create a new command which, when placed in front of a paragraph, is able to select the first letter, enlarge it, box it, create the proper hanging indentation, and to finally drop the capital as you see in this paragraph. In a WYSIWYG system, I might have to stop, position the mouse, and do the same formatting in a somewhat lengthy and tedious procedure. If there are lots of boxed and dropped capitals in the document, there is no painless substitute for this tedium.

There are other things I expect my typesetting program to do. I expect, for example, sections, exercises, equations, and so on to be numbered automatically. Many programs require *you* to perform that chore. I might be able to put up with that, but what happens if I've created a set of 70 or so exercises for a textbook I'm writing, and my editor informs me that I need about 20 more elementary problems at the *beginning* of the exercise set? In this day and age, I don't expect it to be my responsibility to renumber all the exercises by hand. Yet that is what many WYSIWYG systems would demand. TEX, needless to say, does not. It renumbers them for you automatically, as it should.

Those of us involved in scholarly publication know that lots of flotsam and jetsam accumulate around any paper—tables of contents, indexes, answers to odd-numbered problems with hints for solution, footnotes, endnotes, and so on. If you set TEX up properly, it's possible that all this and more will be generated automatically every time you run your document through TEX. Not only is all this good stuff taken care of automatically, but it automatically gets revised each time you revise the main document.

4 Logical Document Structure

It's important to me that I create my documents in a form that identifies the parts of the document, rather than how they will look. For example, I would prefer to begin an article something like the brief excerpt shown in Figure 1.

```
\input docmac

\begintitle
   An Introduction to \TeX{}
     for New Users
\endtitle

\beginauthor
   Alan Hoenig
\endauthor

\beginabstract
   This talk...new to \TeX.
\endabstract

\begindocument

\head
   What is \TeX?
\endhead
...
\subhead
   More Details Revealed
\endsubhead
...
\enddocument
```

Figure 1: Logical document structure.

For those who are *really* new to TEX, the word-like things preceded by a backslash are commands that may be recognized by TEX. There are a few other things that need saying about the nature of TEX syntax, but they are not germane to this talk.

Certainly, this is *not* the way I want my document to appear in its final, printed form. But the commands above the actual text identify the function of the text that follows when I prepare the document for input. Then TEX can perform the formatting appropriate for the particular publication.

The important thing to know about TEX commands is they can be strung together to form your own personal typesetting commands. We call these new commands *macros*, short for "macro instruction." Although it's often easy to create simple macros, and to create them on the fly, the process of creating more complicated ones is similar to writing computer programs. Serious debugging may be called for, and this sharpens the comparison between TEX and a high-level programming language we made earlier. Indeed, part of TEX's repertoire includes commands to iterate loops, make decisions, and perform input and output, just like a "real" programming language.

The very first line of this example seems to imply that TEX's first act should be to read in an auxiliary file containing macro definitions for this document. If we've done our jobs well in tagging or marking up our document, and in creating the macro definitions, then it's straightforward to alter the look of my paper without having to revise the paper (except for that first line). I simply instruct TEX to read in a different file with different macro definitions. The tags become typesetting commands.

For example, for the proceedings of a conference to include this introduction, the title part of the paper might look something like

An Intro ... TEX for New Users
Alan Hoenig

but if this paper is not going to be included those proceedings, then I can easily submit it to some other journal where the formatting looks like

An Intro ... TEX for New Users
by
Alan Hoenig

by leaving the document untouched and simply revising the `\begintitle` - `\endtitle` definition in the macro style file. This is the kind of thing that publishers could exploit—while their authors are creating the book according to a generalized markup scheme, style designers can create the definitions of these macros to implement that book's proper style.

We've just seen that when formatting needs change, only the macros change and not our text. A related advantage of macro commands, and TEX's command structure in general, is that when the text does undergo revision, TEX's formatting commands ensure that the proper formatting continues to apply to the revised text. We need not worry further about proper formatting. I defined a `\strangepar` macro so this paragraph is typeset by entering

```
\strangepar We've just seen that ...
```

in my source file. In case this paragraph needs revision, all I do is revise the text, making sure that `\strangepar` precedes the text in the same way, and the same strange formatting will carry through.

Workers early on realized the importance of creating macro files to facilitate the tagging of the logical parts of a document, and people worked hard to create extensive *macro packages* for use with TEX. Another motivation behind the creation of these packages was a hope that these packages might make TEX easier to use. The basic, primitive TEX commands can be combined in so many unusual and flexible ways that a creative macro designer can almost rewrite the standard TEX syntax.

Of the macro packages that have appeared so far, the two most well known are LATEX and AMS-TEX. AMS-TEX specifically designed to simplify the typesetting of mathematical quantities, equations, and displays, and to format the output according to any of various preset style specifications. The author of AMS-TEX has rewritten another set of macros to incorporate the best features of AMS-TEX and (the original) LATEX; this new package is LAMS-TEX.

LATEX helps separate the *structure* of a document from its meaning while at the same time making TEX easier to use. LATEX has been set up to encourage us to create documents with the kind of logical document structure we spoke of earlier. LATEX did make TEX easier to use, but many people feel that certain changes are harder to make within the LATEX model. At the moment, the LATEX macros are being extensively rewritten to eliminate these problems and make them even easier to use.

It's important to remember—whenever you use a macro package, no matter which one, you are still using TEX.

5 TEX's Other Strengths

One real typographic strength of TEX lies in its ability to automatically invoke typographic niceties that other systems only dream about. Let me briefly mention some of them.

- TEX's line-breaking scheme is far more successful than other DTP or word processing programs at eliminating obnoxious hyphenations and rivers of space in a paragraph. This is largely because TEX considers the whole paragraph when deciding on line breaks. In extreme examples, the last word of a paragraph can influence the line break of the first line.

- TeX will automatically *kern* adjacent letters properly. A ke is a dollop of white space that is added or subtracted to improve the appearance of a word. Consider:

  unkerned: **W A V E**
  kerned: **WAVE**

- Over the centuries, typesetters have replaced certain pairs of adjacent letters by single letterforms called *ligatures*. If needed (and if the ligature is available in the font), TeX will automatically typeset the ligature. In the standard Roman typesetting, TeX provides these ligatures:

  ﬄ ﬃ ﬀ ﬁ ﬂ

  Compare with the unligatured letters:

  ffl ffi ff fi fl

- TeX is super at doing tables and mathematics.

6 TeX's Siblings

The 10-year effort that resulted in the birth of TeX also produced two other major software systems. By the way, this delivery happened at Stanford University, and the author of all these systems is Donald E. Knuth, to whom we should all render thanks.

The first major software system is METAFONT, the graphic side of TeX. All the letterforms in the Computer Modern family of typefaces were produced by this program. METAFONT would also be perfect for the creation of logos and diagrams for papers. I personally find METAFONT a "neat" program to use, neater in many respects than TeX.

Both TeX and METAFONT are massive Pascal programs, each containing between 20,000 and 30,000 lines of code (depending on how they are pretty-printed). How can any one person thoroughly test and debug such programming monsters? Knuth's answer was the `WEB` system of structured documentation, the second additional system I want to mention.

You create a master `WEB` file which contains lines of code and documentation that have been entered according to the proper `WEB` conventions. This file is then run through two different programs depending on whether you want to work with the documentation or with the program. When the *documentation* is generated, it's in a form which is particularly easy for humans to read and understand. When the *program* is generated, it's in a form particularly easy for machines to understand (but quite difficult for humans to read; this way, you are discouraged from making changes to anything but the master `WEB` file). In practice, `WEB` can be used to generate large-scale, complex computer systems fairly rapidly. But by and large, TeX users don't deal with `WEB`.

Knuth, D.E., 1984. *The TeXbook*. Reading, MA: Addison-Wesley.

Lamport, L., 1986. *LaTeX: A Document Preparation System*. Reading, MA: Addison-Wesley.

Spivak, M.D., 1986. *The Joy of TeX*. Providence: The American Mathematical Society.

Spivak, M.D., 1985. *The PC-TeX Manual*. Mill Valley, CA: Personal TeX, Inc.
Buerger, D.J., 1990. *LaTeX for Engineers and Scientists*. NY: McGraw-Hill.
Electronic manuals.

Doob, Michael, 1990. *Gentle Introduction to TeX*. Avail. from TeX Users Group, Providence, RI.
St. Sauver, J.E., [no date]. *Using TeX on the VAX to Typeset Documents: A Primer.*
Warbrick, Jon, [no date]. *Essential LaTeX.*
Knuth, D.E., 1986. *The MetafontBook*. Reading, MA:

Addison-Wesley.
Knuth, D.E., 1983. *The WEB System of Structured Documentation*. Stanford, CA: Computer Science Department, Stanford University.

Figure 2: A brief TeX bibliography.

7 Learning More about TeX

Assuming I've sparked your interest, let me tell you how you can find out more about TeX.

The TeX canon is *The TeXbook*, written by the author of TeX, Don Knuth. Essentially everything you need to know about TeX is found here, some place or other. Leslie Lamport's *LaTeX: A Document Preparation System* and Mike Spivak's *The Joy of TeX* provide the same service for LaTeX and AMS-TeX.

Beginners constantly demand ever more information about TeX at a lower level; let me mention several works that might be useful in satisfying that demand. First is *PC-TeX Manual* by the Mike Spivak and then there is *LaTeX for Engineers and Scientists* by David J. Buerger.

There are at least three electronic introductions to TeX that you may be interested in. That is, they have been written by caring and generous authors who have placed electronic copies of their manuscripts in the public domain. The first such is Michael Doob's *Gentle Introduction to TeX*; two others are *Using TeX on the VAX to Typeset Documents: A Primer* by Joseph St. Sauver, and *Essential LATEX* by Jon Warbrick. The *TEX Primer* is useful regardless of your computer system, since most of TEX is independent of the computer system.

A continuing source of information on TEX-related material is *TUGboat*, the transactions of the TEX Users Group (P O. Box 9506, Providence, RI 02940 USA; [401] 751–7760).