What is TEX? . . .

. . . by Hans Hagen

Here I reflect on some of the remarks made in the other answers. It's not so much meant as critique, but more as a trigger for further discussion. If you only want to know my answer, you can skip to the last paragraph.

All TEXs are equal . . .

. . . but some are more equal than others.

The answer to this question is not always easy to give. Peter Flom for instance starts his description with "LATEX is . . . " and thereby makes TEX the program equivalent to LATEX the macro-package.

This kind of equivalents are rather common, and many users don't know the difference between PDFTEX (the program) and PDFLATEX (the macro package). This is made even more confusing by the fact that on many systems invoking PDFTEX without explicit macro package mentioned, will load the plain TEX format.

Yet another confusing factor is that TEX is used to describe both a language and its associated interpreter/typesetter. And then there is the TEXbook, which not only describes these two, but also the plain TEX format that ships with the system. So, in the case of TEX we need to distinguish:

language	primitive commands combined with macro definitions
program	interpreter and typesetting engine
package	collection of macros loaded on top of the built in language

If we translate that to commands and files, we end up with:

language	TEX, program specific extensions
program	TEX, PDFTEX, XETEX, ALEPH (OMEGA)
package	plain, AMS-TEX, LATEX, LAMS-TEX, CONTEXT, . . .

LATEX users have several options to invoke TEX:

latex	the (pdf)TEXengine with the LATEX macro package preloaded
pdflatex	idem, but this time the output will be a pdf file
xelatex	the LATEX macro package loaded into the XETEX engine
lambda	the LATEX macro package loaded into the ALEPH or OMEGA

For ConTEXt users life is different. They use a wrapper and thereby use calls like

texexec –pdf somefile.tex the CONTEXT macro package loaded in PDFTEX
texexec –xtx somefile.tex idem but this time loaded in XETEX

For a long time TEX produced DVI output only and one had to postprocess this into a format suitable for a printing engine and for quite a while POSTSCRIPT output was quite popular. Nowadays PDF is the format of choice and PDFTEX can produce this directly. There is no need for a backend like DVIPS (to produce POSTSCRIPT which itself can be converted in PDF) or DVIPDFMX (which converts DVI into PDF).

No matter how you use TEX, you need to keep in mind that when you talk of in terms of what you invoke on the command line, this may not be what others experience. Think of this: by default PDFTEX produces DVI output and unless told explicitly to behave differently, it is just like good old TEX, and in DVI mode still needs backend. Confusing eh?

TEX can produce beautiful documents . . .

. . . but does not give you guarantees.

When people advocate TEX they tend to praise the output of this program as being of high quality and beautiful. In a way this is wishful thinking. There is no doubt that TEX can produce documents that qualify as such, but in practice many documents look just as 'texy' as MSWORD documents look 'wordy' and QUARK output looks 'quarky'. The variations in style (design), font usage and formatting is not that large and a direct result of using the same predefined layout over and over again. For instance, texies make jokes about POWERPOINT presentations (since they can be recognized by the features used) but don't realize that most of their own work stands out in a similar way. They rightfully claim that TEX does a good job on breaking lines into paragraphs but are more tolerant to funny vertical spacing resulting from handcrafted commands that interfere with what the macro package tries to accomplish. Because TEX can do such a good job on justifying text, words sticking into the right margin (overfull boxes) stand out pretty noticeable. I don't want to count the documents posted on the web that demonstrate this 'feature'.

TEX is easy to use . . .

. . . but not everything is easy.

TEX can do clever things with graphics and fonts, but the fact that there are so many questions posted to mailing lists demonstrates that this is less trivial than long time users suggest when they praise TEX to new users. TEX can be an easy system to use, but also a painful experience when one wants to do real clever things. Some things are simply tricky, no matter what system is used.

An important property of TEX usage is that on the average the audience is quite willing to help

newcomers. Nearly always users themselves choose to use TEX. Therefore they are willing to spend time on learning the system.

A strength of TEX and its packages is that one can find resources on usage in bookshop as well as on the web (manuals, faqs, wikis, mailing lists, new groups, etc).

TEX output is always good . . .

. . . it's only you who can mess up things.

In most computer languages, one has to explicitly tell the machinery that some text should be output. Not with TEX. Anything that expands to text will become visible somehow. One can make fun of the fact that those who use word processors may end up with inconsistent spacing, i.e., duplicate spaces in the result. With TEX, you should not be surprised when spacing becomes messed up too due to funny spaces in macros. Be careful of making false claims and dangerous jokes.

In his answer David mentions the visual separation of paragraphs as a characteristic of TEX. He also explains the difference between changing fonts in TEX and for instance MSWORD. In discussions about the the differences between word processors and TEX, one may argue that in a word processor one never knows where exactly a change of fonts takes place: is the space preceding a bold word bold as well or not. But in a way TEX's ways of dealing with font changes or changes in attributes is not less confusing than e.g. MSWORD's.

Say that we want to narrow a paragraph of text.

```
\def\StartNarrow{\bgroup\leftskip1em\rightskip1em\relax}
\def\StopNarrow {\egroup}
\StartNarrow some lines of text \StopNarrow
```

In such cases grouping is used to make sure that we limit the scope of the feature change. However, in this case, you will not get an narrowed paragraph, unless you provide an explicit paragraph end.

```
\StartNarrow some lines of text \par \StopNarrow
```

The solution is to change the definition to:

```
\def\StartNarrow{\bgroup\leftskip1em\rightskip1em\relax}
\def\StopNarrow {\par\egroup}
```

There are many spacing related features that work this way and the effects are not always clear source code. What is true for one document style may be false for another. It all depends on how your TEX is set up and how well macro writers coordinate their work.

TEX is stable and does not change . . .
. . . but do we really want that to be true?

Don Knuth's wishful thinking that TEX the program would be extended for whatever intended purpose has not been fulfilled. In good old TEX there are two examples of extansions: specials and writes. Specials provide a way to control the backend and are used to achieve special effects like color or to insert additional material like graphics. Without specials, we would have been in big trouble and still manually have to cut and paste copies of graphics. Writes are a neccessity for tables of contents, cross references and other features that demand a feedback loop into a successive run. Normally their usage is hidden by macro packages. By providing these examples of extending TEX Don actually made TEX much more future proof.

There are some non-Knuthian extensions, but not that many. For instance, nobody bothered to write a subsystem for typesetting chemistry as companion to the math typesetting subsystem. It has to be done in macros. So far nobody came up with real robust extensions for numbering lines, parallel output streams and other goodies for the humanities and linguistics. Again, one has to revert to macro writing, in this case of a particularly nasty kind. You may call yourself lucky that publishers are not that demanding.

Nevertheless, one may expect extensions and currently the most prominent ones are ε-TEX, which provided some extra programming features, PDFTEX, which kick-started TEX into the 21th century by providing marginal kerning and optical scaling aka *hz* (Hermann Zapf) optimization as well as a full featured PDF backend, and XETEX which boosted TEX towards unicode and opentype fonts. It's only by efforts like this that TEX is still alive and kicking.

Of course, macro packages play their role as well, and as long as we can find people who feel challenged by beating a language and feature set that does not really match today's programming techniques, we're safe. The competition is not doing much better, simply because the problems that we're facing haven't changed much.

Christina mentions that one of the nice things about TEX is that it's virtually bug free. But, Christina, I have to disappoint you: if your 23 year old document depends on `\leaders` behaving like they did at that time, you may have to find yourself an old copy. Among the most recent bug fixes was a fix to this mechanism and it may make a difference, although the changes are small. (Actually, it does make a difference for the TEXbook.) But . . . in general your claim is true, unless of course you forgot to save your old pattern files, along with an old copy of your macro package. And, are you sure that the metrics or appearance of the fonts that you use didn't change? Btw, there are other examples of stable programs: computer language compilers and interpreters, and this is exactly what TEX is.

What is TEX . . .

. . . and why do I like it?

TEX is a system which permits you to create your own typesetting environment. In its 25^+ year existence various environments evolved, for instance LATEX and CONTEXT so users can start right away. Both can be used comfortably in editing programs and previewing your document is no problem. You can extend their functionality or decide to stick to what is offered. You are in control. Okay, some aspects are difficult to deal with, but that is a direct result of the rich functionality.

If you stick to the paradigm of the particular environment you are using, i.e., keep your document code clean, your documents are stable over a long period of time. You edit in a structured way, you define your layout in an abstract manner, and can produce the final result on any platform you want. You can distribute your source code and let others work with it, thanks to the availability of distributions, support mailing lists, a friendly community, user groups, books and manuals. If you want to use their full power, it will take a while to learn such systems, but for most users TEX is a tool that they will use their whole life. Don Knuth gave us the ability to "go out and create beautiful documents" but you need to pay some attention to get it done. He also formulated the important boundary condition that in 100 years from now, the documents should still be valid input for a TEX processor. Such life long validation gives a comfortable feeling.

Beware: in the process, you can get hooked. And: you need to keep an open mind for the shortcomings, myths and somewhat strange solutions that come with TEX.