Getting Started with LaTeX

Chris Spackman

May 26, 2002

Contents

1 Introduction

What is LaTeX?

You might have heard about how cool LaTeX is, but maybe aren't sure what it is. Let's start with what it is not. It is not a word processor. It is not a text editor. It might be best to think of LaTeX as a kind of markup language. It doesn't matter what word processor or text editor you use. While you are writing, you insert markup commands that will be used later when you use `latex`, the program, to create the finished product.

There are a few things we need to understand before we get too deep into talking about LaTeX. (Experts can argue the details of some of the following, but this is intended for lay-people, not experts.) If you are already somewhat computer literate, feel free to skip to section 2, **Before We Start**, which is where we start dealing specifically with LaTeX.

plain text or just **text**. This is a generic standard for computer text. Any computer on any operating system can understand it. The standard contains no layout information at all, so text can only display what you can type in from your keyboard. Upper and lower case are not a problem, but keyboards do not have a key to make text **bold**, or to change fonts. In order to display things like tables, centering and different fonts, word processors add program-specific data into their files. In many proprietary word-processors, this data is not plain text, so their files are not plain text—which makes them very hard for any other program to read. MS Word is an example of this. Markup languages like html, and many open-source word-processors like OpenOffice, AbiWord, and KOffice, use plain text to add the necessary font and layout information. So, their files are not plain text.

To see this in action, open up your favorite word processor and make a document with all sorts of layout and many different fonts. Save it. Now save it again as a text file. (Usually from `File` → `Save as...` and then choose `text` from the drop down menu.) Close the word processor and open it again. Now open the text file you just saved. The entire document contains only one font, right? Everything is the same size, and everything is in the same font. Also, any centering or other layout is either gone or converted to spaces. That is plain text.

text processor : A program that processes text. Literally, it takes text (from a file in the case of LaTeX), does something to that text, and outputs the changes to a new file. LaTeX reads a text file, does pagination, centering, fonts, page numbering, and all the rest, and writes the result to a new file.

finished product : The file you have after running LaTeX on a text file, or the file you want to eventually have. LaTeX usually makes a `.dvi` file, but you can turn those into `.ps` or `.pdf` files very easily. So, for the purposes of this article, the *finished product* could be any of these.

The important thing to understand about the *finished product* is that it contains all the extra information about fonts and layout. It is not a text file that you can edit[1]. If you open it with the appropriate program, you will see how your document looks.

Working with LaTeX

Working with LaTeX is a three step process:

1. Write your report, book, article, or whatever, including LaTeX markup commands.

2. Process your document with the program `latex`. This is usually done from the command line (a dos window, for MS Windows users). You type the name of the program you want to run (`latex` in this case) and the name of the file you want that program to work on. We'll deal with this in a lot more detail in section 10 on page 18.

3. View the finished product. If everything is okay, you are done. If you need to make changes, go back to step one (no you do not have to rewrite your document from scratch—your text file is still there and totally unharmed).

LaTeX vs Word Processors

If LaTeX is a three step process and requires you to learn layout commands, why would anyone prefer it to a regular word processor? There are many reasons. Most word processors are "what you see is what you get" (WYSIWYG) programs that show what you tell them to show, where you tell them to show it. If you want to leave some blank space, you hit the return key a few times. If you want to center

[1] Well, some are text files, but you would never want to try to edit them by hand.

something you highlight it and click on the center button. On the screen, you see the text moved to the center of the page.

WYSIWYG word processors have several limitations. They should really be known as "what you see is usually something like what you might get". Even viewing the same file with the same program but on a different computer can cause font, pagination, or other layout problems. So what you saw when you wrote the document might not be what someone else sees when they read it. Likewise, the output can vary depending on the printer you use.

To be honest, WYSIWYG word processors are often "good enough". Most of the time small differences in pagination are not important. Likewise, few people would care or even notice if one font gets replaced by a similar one.

LaTeX is not WYSIWYG. Because the writing of the document and the layout of the document are separate steps, the layout can be optimized. The layout program knows that it has all the data, so it can look at it all and decide exactly how to best paginate and where to place figures and tables. WYSIWYG word processors cannot do this because they are doing layout while you are typing.

LaTeX was made by people who know what professional layout is. Word processors require the author to make all the layout choices. Should your margins really be 1 inch, regardless of the font size you are using? More important than the size of the margin is the number of letters per line. The human eye can only take in so many at one time, so lines should generally have no more than about 60 letters per line. In a 10 point font, that might require bigger margins that if you were using a 14 point font. Word processors do not think of this for you. By default, LaTeX does (but you can always change the defaults to suit your needs).

A further plus is that LaTeX's output is exactly the same on every computer and every printer. It is **device independent**.

Of course, LaTeX can take care of footnotes, page-numbering, the cover page, the tables of contents and figures, indexing, images, bibliography, and a whole lot more.[2]

Here are a few more reasons that I prefer LaTeX over word processors:

- Text files—you can use your editor of choice as well as your platform of choice. No lock-in at all.
- LaTeX can compile several separate files into a single document. For long works like books or dissertations this is a huge advantage as there is no need to open the entire work just to make one or two changes. It also facilitates

[2]Some of these features require the use of separate packages. Many are installed along with a typical installation, but you might need to install one or two yourself, depending on your needs.

moving chapters around or rewriting entire chapters without touching the rest of the work. Also, pictures and other graphics are separate from the work until you run it through LaTeX (and then a copy of the graphic is included in the final product). This means smaller, easier to handle, files. It also means you can easily use any program you want to make or change the graphics.

- A variety of programs exist to turn LaTeX files into various other files. You can easily turn one set of LaTeX files into postscript files, PDF files, or HTML.

- Power and flexibility—LaTeX can handle just about any writing task, from a letter to grandma to a textbook of theoretical physics. Many people have written add-on packages that make it simple to do some pretty complex things. LaTeX itself is actually a set of macros designed to make it easier for people to use TeX, the original program and still the engine hidden under the friendly LaTeX exterior.

- Use whatever size font you want when you write your document. I prefer to write in 14 or even 16 point fonts—it is easier on my eyes. Since you save as text, none of the font size info is saved. LaTeX will automagically deal with all the font stuff later. (Think what a pain it would be to write a book in MS Word in 16 point and then have to go change it all to 12 point when you want to print it. `Edit` → `Select All` would only work if you had **no** other font size changes in your document.)

2 Before We Start

There are a few things you need to know about how LaTeX works before we start. Some of this we covered above, but it is worth repeating, just so there is no confusion. As I have already said several times, LaTeX is a program that you run on an already existing text file. You can create that file with whatever writing program you like. While you are writing your document, you include some of the commands we will discuss later. Then, when you are finished writing, you run your text file through `latex`, which results in a new file that is the final product. For many people used to word processors and unaccustomed to the command line, this is a difficult concept to grasp at first.

From now on, I will treat LaTeX (the overall program that includes the markup commands) and `latex`, the program you run to produce the final product, as the same and refer to them collectively as LaTeX. It should be clear from the context which one is meant.

LaTeX treats any number of spaces between words as one space regardless of how many spaces you included. Every space after the first is ignored. Of course if you need to insert extra space it is possible and easy in LaTeX, but the space bar is not the way to do it. Don't worry about the extra space between the end of a sentence and the beginning of the next sentence—LaTeX takes care of that automagically.

If you have used WYSIWYG word processors, you are probably used to hitting the return key several times to add some vertical space between paragraphs or before and after a table. That will not work in LaTeX. Hit the return key one times or ten, the result is still one paragraph break. It is possible to insert vertical space if you want, but in LaTeX, the return key is not how you do it.

In LaTeX, there are four types of hyphens. All are made with the normal hyphen (or minus) key.

- The hyphen. It is used for words broken by a line break (LaTeX, of course, does this for you automagically) and for compound words like ex-wife or e-mail. Typing one hyphen (`-`) will result in a hyphen.

– The *en-dash*. It is used to separate ranges of numbers (see pgs. 234–253). Type two hyphens (`--`) to get an en-dash (it will appear in the final LaTeX'ed file, not in your text editor).

— The *em-dash*. It is used as punctuation—like this. Type three hyphens (`---`) to get an em-dash.

− The minus sign. TeX was originally written to do layout for mathematics articles and can do amazing things with equations, matrixes and other stuff many people will never need. To get a 'real' minus sign type $-$. That is two dollar signs with a hyphen between them. In the final copy, it will appear as a minus sign. So, `$-3$` in your text file looks like -3 in the final product. Of course, if you don't care, a regular hyphen might be just as good.

Do not use three periods (aka full-stops) for ellpses. Instead, LaTeX has the `\ldots` command, which takes care of proper spacing between the periods.

Finally, LaTeX distinguishes opening and closing quotes. So do not use the double quote key. Instead, beginning quotes are the anti-apostrophe key (‘). Closing quotes (’) are simply the normal apostrophe. If you want two (“like this”), just hit the appropriate key twice.

(The above points about spacing and quotes are true for your final product, not for the text file containing your work. You can insert all the blank space you want

into your text file. When you run it through LaTeX, the blanks will not appear in the output. This is another advantage, since you can use blank space to make your document easier to read / deal with, without affecting how the final will look.)

Please understand that LaTeX knows what it is doing and will sometimes override your requests—for example by moving figures around or refusing to insert blank space if it just before or after a page break. If it does this, it is usually for a good reason. If you really need that blank space, page break or no, you can still get it. This isn't a bug in LaTeX, it is LaTeX trying to make your document look as professional as possible.

3 Getting Started

Okay, let's get started. First we need to tell LaTeX what kind of a document we are writing, so put this on the first line:

```
\documentclass[a4paper,12pt]{article}
```

`\documentclass`—almost all[3] LaTeX commands begin with a backslash (\) followed by the command ('documentclass' in this case) and possibly some arguments for that command in squiggly brackets ({ and }). The regular brackets [and] are used occasionally for optional arguments. In this case, 'documentclass' defines what sort of document we are writing and here takes 'article' as its main argument. The optional arguments 'a4paper,12pt' tell LaTeX that we want to use a 12 point font instead of the default 10 point size and that we are using a4 paper instead of the default of 8.5 x 11 inch paper.

The basic document classes are 'article' 'report' 'book' and 'letter'. There are also several other, more specialized, document classes. The document class that you chose influences things like the layout of the title page and page numbering and style. (If you want to, you can override any of the defaults.) Not all commands are available in all classes. Articles and letters, for example, do not have chapters or prefaces and the author's address is usually only included in letters.

In general, unless you are working on your dissertation or a novel, article and letter should serve you well. We will practice with the article class here.

Next, we tell the program what to use as the title, the author's name, and the date.

[3] the main exception being font size commands which start with { instead. See section 7.

```
\title{Getting Started with \LaTeX}
\author{Chris Spackman}
\date{\today}
```

The arguments (the stuff inside { and }) can be just about anything, including other commands — as you can see from the use of **LaTeX** inside the title command and **today** inside the date command.

When you process your text file, LaTeX will take this information and place it on the appropriate place on the page, in the appropriate font. No need to fiddle with the layout details unless you do not like the defaults.

Now, hit return once or twice and add:

```
\begin{document}

\maketitle
\tableofcontents
```

The command **begin**{**document**} tells LaTeX that all the preliminary meta-document stuff is finished and we are now starting the actual document.

The area from the **documentclass** to the **begin**{**document**} is called the **preamble**. It is where you include information on extra packages that you might want to use and other meta-information that is not part of the content of the document you are writing.

Just because the title, author, and date information are in the preamble, that doesn't mean that they are automagically included in the final document. The **maketitle** command right after the **begin**{**document**} tells LaTeX to insert the title information.

Finally, we added the **tableofcontents** command just after the title. For most writing tasks, LaTeX will automagically generate a more than adequate table of contents with very little input needed from the author. If you don't want a TOC, then don't include this command.

By the way, **begin**{**environment**} is how you start environments. Environments do various things to the text of your document. There are list environments, center, flushleft and flushright environments, verbatim environments, color environments, and many others. They all start with **begin**{**environment**} and end with **end**{**environment**}. More on environments later.

Now that the preliminaries are taken care of, it is time to get to work writing our document. Since this is an article, the largest division is a section.

We add:

```
\section{Introduction}
```

The **section** you no doubt understand—it is simply the command to start a new section. The argument {**Introduction**} gives the section the name 'Introduction'. In the final document, this section will be numbered, with the title 'Introduction'. An entry will also be automatically added for this section in the table of contents.

If you do not want numbered sections, put an asterisk after the command name, like this:

```
\section*{Introduction}
```

Now this section will not be numbered and **will not** be listed in the table of contents. It will still be given the title 'Introduction', just like with the regular section command. But what if you don't want numbered sections, but do want a table of contents? Try this:

```
\section*{Introduction}
\addcontentsline{toc}{section}{Introduction}
```

Actually there are better ways to do this. We will look at them in a future article.

Like with **title**{}, the section name can contain other commands. We could have called this section **section**{**Introduction to** **LaTeX**}. A more likely use would be to change the font for some or all of your section name — to *italics* perhaps.

Now, we type our introduction. To start a new paragraph just leave a blank line between the two paragraphs. Repeat: to start a new paragraph, leave one (or more) blank lines between the paragraphs. That is all you need to do.

4 Subsections, Etc.

You might want more than just sections in your article. Predictably, under sections are `subsections` and under them are `subsubsections`. Paragraph divisions are also possible.

Subsections

Subsections are one step below sections and the command is basically the same:

```
\subsection{Name of the subsection}
```

Like with sections, you can prevent each subsection from being numbered and included in the TOC by using an asterisk:

```
\subsection*{Name of the subsection}
```

Etc.

You can go all the way down to subparagraphs if you like.

```
\subsubsection{Name of the subsection}

\paragraph{Name of the paragraph}

\subparagraph{Name of the subparagraph}
```

The asterisk form is also possible for all of these.

Note that you do not have to use any of these. Even **section** is totally optional. For a short essay, you might not need anything other than a title.

Further, there is no need for you to use **paragraph** unless you want the paragraph distinguished in some way. For normal text, you can get paragraph breaks just by leaving a blank line between the paragraphs in your text file. Nowhere in this article have I used the paragraph command.

5 Reserved Characters

You may have realized that you cannot easily include a \ or the { } in your text. In fact there are ten reserved characters—characters that have special meanings to LaTeX and will cause you all sorts of headaches if you aren't careful with them. The reserved characters are:

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

In order to use these characters in your documents, they must be 'escaped' (marked so that LaTeX knows to treat them as normal characters) by preceding them with a backslash (\).

The extra-special cases are the tilde (~), the circumflex (^), and the backslash itself.

The tilde is used to create an unbreakable space, for those times when you want to make sure that two or more words stay together on the same line. Unfortunately, `\~` is used both to print an ordinary tilde and also to print the tilde over letters. `\~{n}` yields ñ and `\~{}` yields ~, So it is best to include the {}, to avoid putting the tilde over the next letter.

The circumflex is also a royal pain in the butt. It is used in math mode for making superscripts (x^2 for example) and is also used for accenting other letters (`\^{o}` = ô). Another way to print the tilde, and the easiest way to print the circumflex by itself is with the verbatim command:

```
\verb-^-
\verb-~-
```

To get a backslash in your document, you must use **`$\backslash$`**. This is because the double backslash (\\) is used as a line break and thus is not available as an escape for the backslash.

The list of reserved characters above looks like this in the latex file:

```
\# \$ \% \& \_ \{ \} \verb-~- \verb-^- $\backslash$
```

Comments

By the way, the percent sign (%) is used to put comments into your text files. Everythign from a % to the end of the line is ignored by LaTeX and will not appear in the final product. Comments can be helpful if more than one person is working on a document, or with big projects where you might want to include some information, for example, about when you last modified some part of the file.

6 Footnotes

To include a footnote, just type **`footnote`**{text of your footnote} right next to where you want the reference to appear in the final product. LaTeX will automagically number the footnote and place the footnote text at the bottom of the proper page. If you have footnotes, you should run LaTeX at least three times, just to be sure.

7 Playing with Fonts

Although LaTeX takes care of font sizes for section titles and footnotes automatically, you can also change fonts manually whenever you like. For normal text (a paper for school or a letter to grandma, for example), there isn't much reason to change the default font face. Take a look at any novels you might have—how many different fonts are in them? Maybe three or four total? One problem with word processors like MS Word is that people get used to changing fonts just because they can, regardless of whether they should.

Changing the Font Face

Changing a font face for a few words easily accomplished with the **textXX**{**some text**} command, where XX is the two letter abbreviation of the kind of font you want.

- **textbf**{**text**} = **bold face**
- **textit**{**text**} = *italic*
- **textrm**{**text**} = serif
- **textsc**{**text**} = SMALL CAPITALS
- **textsf**{**text**} = sans serif
- **textsl**{**text**} = *slanted text*
- **texttt**{**text**} = `typewriter text`

Additionally, **textsuperscript**{**text**} can be used to [raise] text.

This list is not exhaustive.

Changing the Size of Fonts

The following commands can be used to make fonts bigger or smaller. The exact size is relative to the font size declared in the preamble. Huge is bigger if you choose 12 point for your base font than if you go with the default 10 point. The base font is set as one of the options to the **documentclass** declaration. In this article, we are using 12 point as the base size.

- {**tiny ABCD**} = ABCD
- {**scriptsize ABCD**} = ABCD
- {**footnotesize ABCD**} = ABCD
- {**small ABCD**} = ABCD
- {**normalsize ABCD**} = ABCD
- {**large ABCD**} = ABCD
- {**Large ABCD**} = ABCD
- {**LARGE ABCD**} = ABCD
- {**huge ABCD**} = ABCD
- {**Huge ABCD**} = ABCD

Note that the font size commands look like this:
{**size text**}
unlike most other commands that look like this:
commandname{**text**}.

Emphasis

There is also a useful command for emphasizing words. This is not quite the same as changing the font, but close. The emph command looks like this:

```
\emph{text you want emphasized}
```

One advantage of using the emph command is that it worries about the proper font to use. So an emph command inside of another emph command results in a different font for the doubly emphasized text. It looks like this:

Here is some normal text that I am *now going to* emphasize, *but the word* emphasize *itself is double*-emphasized.

8 Environments

We looked very briefly at environments before, but now let's get a lot more detailed. Remember that environments start with:

```
\begin{environment}
```

and end with:

```
\end{environment}
```

everything in between is affected by that environment. For example, the following code (the \\ starts a new line in the same paragraph):

```
\begin{center}
This text is centered.\\
As is this text.
\end{center}

\begin{flushright}
But this text is right justified.\\
And so is this text.
\end{flushright}
```

results in:

This text is centered.
As is this text.

But this text is right justified.
And so is this text.

Here is a list of environments. Usage is:
begin{**environment**}
...
end{**environment**}

- flushleft
- flushright
- center
- verbatim
- itemize
- enumerate
- description
- quote
- quotation
- verse

Verbatim is explained below and lists (the itemize, enumerate, and description environments) in the next section.

Starting an environment and then forgetting to end it is a very common mistake.

Commands $\Longleftrightarrow$ Environments

Many commands can also be used as environments and some environments can be used as commands. An 'itemize' command would not make much sense, but the verbatim environment is very useful as a command when you only need it for a few words.

The Verbatim Environment

The verbatim environment starts like any other, with a begin{environmentname} command. Everything between the opening and closing of the verbatim environment is printed as is in a typewriter style font and with no LaTeX processing at all. This means you need to put in line breaks and spacing on your own. LaTeX will happily print right off the page if you let it.

Here is the above paragraph, but in verbatim:

```
The verbatim environment starts like any other, with a
begin\{environmentname\} command. Everything between
the opening and closing of the verbatim environment is
printed as is in a typewriter style font and with no
\LaTeX{} processing at all. This means you need to put
in line breaks and spacing on your own. \LaTeX{} will
happily print right off the page if you let it.
```

The Verbatim Command

The verbatim command (actually it is 'verb') is a little unusual in that it doesn't use { or }. Instead, you choose a character to be used as the opening and closing braces. The command looks like this:

```
\verb+this text is verbatim+
```

In this example, I used the plus sign (+) to start and end the content of the verb command. Letters of the alphabet, the asterisk (*), and spaces are the only characters you **cannot** use to open or close the verb command. Of course, you need to use the same letter to open and close a verb command.

9 Lists

Making lists is easy. After starting one of the list environments described below, just use the `\item` command to start a new list item. LaTeX will take care of the formating details. Here is a sample of a short list:

```
\begin{itemize}
\item This is item one.

\item This is item two.

\item Etcetera, etcetera. Blah, Blah.
\end{itemize}
```

and it looks like this:

- This is item one.
- This is item two.
- Etcetera, etcetera. Blah, Blah.

You don't need the spacing between the item commands—I prefer the extra space to make big lists more readable (remember that LaTeX ignores extra blank space, so the extra space has no effect at all on how the list looks in the final LaTeX'ed version). And of course the text of an item can be as long as you like and contain almost any other command, including other lists. The bullets for nested lists are taken care of automagically by LaTeX.

The Itemize Environment

The example above showed the itemize list environment. Bullets change automatically if you nest (put one inside of another) itemize environments.

The Enumerate Environment

This list environment gives you numbered list items. It looks like this:

1. This is the first item of the top level enumerate environment.
2. This is the second. We will now start a new enumerate list.
    (a) “Four score and seven years ago…”
    (b) “I have a dream…”
3. Now we are back to level 1. Let’s end this list.

The above looks like this in the text file:

```
\begin{enumerate}
\item This is the first item of the top level enumerate environment.
\item This is the second. We will now start a new enumerate list.
  \begin{enumerate}
    \item ‘‘Four score and seven years ago\ldots’’
    \item ‘‘I have a dream\ldots’’
  \end{enumerate}
\item Now we are back to level 1. Let’s end this list.
\end{enumerate}
```

Notice the indenting—it makes it easier to distinguish which item is in which list, but doesn’t cause any problems in the final product, because LaTeX ignores the extra blank space.

The Description Environment

This list environment is used when you want to describe or define a term. It automagically makes the target word bold and takes care of the indenting of the rest of the paragraph. The list on page 2 that explains **plain text** and the other terms is a description list. The only thing you need to do differently is tell LaTeX which word or words you are defining. Put those in brackets right after the `\item`, like this:

```
\begin{description}

\item [\LaTeX] is a text formatting program that \ldots

\end{description}
```

That comes out looking like this:

LaTeX is a text formatting program that . . .

10 Finishing and Processing Your Article

You now know enough to write a simple article, assuming you don't want to get too fancy—just some text divided into sections and a nice table of contents. So it is time to learn how to finish your document and run it through LaTeX.

Finishing Up

Now we end the document environment (that we started near the top with the **begin**{**document**} command):

```
\end{document}
```

You now have a text file containing your document and the markup telling LaTeX what to do to it.

Processing with LaTeX

First, save your article as whatever you like, but it must end in `.tex`. I will use `latex-article.tex` in the following. Now, if you don't already have one open, start a terminal (a dos prompt). Type in:

```
latex latex-article
```

and hit return. The name of your file must end in `.tex`, but when running LaTeX, you don't need to actually type the `.tex`. LaTeX is smart enough to know what you mean. (Of course, it doesn't hurt if you include the `.tex`.)

You should see a lot of informational messages flying past. Don't worry about them unless you get an error message. When LaTeX is finished, you will have several new files in your directory, all with the same name as your tex file but with different extensions. In this example, I now have the following files:

```
latex-article.aux  latex-article.log  latex-article.toc
latex-article.dvi  latex-article.tex
```

The `.aux` file contains information that LaTeX uses to construct things like the table of contents and section numbering.

LaTeX reads the `.toc` file to make the table of contents in the final document.

The `.log` file contains informational messages and warnings that LaTeX issued while processing the file.

To see how your document came out, use xdvi to open the `.dvi` (DeVice Independent) file.

```
xdvi latex-article.dvi
```

WARNING: If you have a footnotes, endnotes, or a table of contents, you need to run LaTeX at least one more time. The first time through, LaTeX builds the `.aux` and `.toc` files. The second time, it will insert the information from these files into your document. A third time might be necessary if the table of contents or something else has caused footnotes to move or some other insult to pagination. In this case, a third time will set everything right again. Unless you are writing a huge book or something, running LaTeX shouldn't take more than a few seconds. So it never hurts to run it three or four times, just to be sure. If you delete any of the `.aux` or `.toc` files, the next time you run LaTeX will be the first time again (that is, if there are no `.aux` are `.toc` files, LaTeXhas to start making them from scratch again). So it is usually best not to delete those files until you are done.

Dealing with Errors

Sometimes LaTeXwill stop in the middle of processing a document. It might look a bit like this:

```
! LaTeX Error: Something's wrong--perhaps a missing \item.

See the LaTeX manual or LaTeX Companion for explanation.
Type  H <return>  for immediate help.
 ...

l.753 \begin{verbatim}

?
```

This tells us that there is a problem somewhere around line 753, and that it might have something to do with a missing \item command. Sometimes you can input a proper command at the prompt (the ? is LaTeX waiting for you to tell it what you meant), but I usually find it easier to quit LaTeX right there (hit Ctrl-c) and go fix the problem in the text file. Also, you can sometimes just hit the return key to have LaTeX skip this error and continue. Not a great idea, but you can do it.

In my limited experience, most errors on simple documents are caused either by forgetting to close an environment or by misspelling a command (resulting in LaTeX complaining about an unknown command).

DVI → PS & PDF

You will prolly want to convert your `.dvi` file into either a postscript (PS) file or a portable document format (PDF) file. Easy. First make a postscript file from the dvi file:

```
dvips -o latex-article.ps latex-article.dvi
```

then, if you want a PDF file, do:

```
ps2pdf latex-article.ps latex-article.pdf
```

Simple and the results not only look great, but anyone on almost any operating system can view and print your finished document as well as your source text file (if you make it available).

pdflatex

There is an even easier way to go from your `.tex` file to a `.pdf` file—a program called `pdflatex`. For reasons we don't need to get into here, there are a few minor inconveniences between latex and pdflatex, mostly involving including images. But if you have no images you have nothing to fear. (And even if you do have images, it is a very simple matter to work around the inconveniences.)

`pdflatex` will convert your `.tex` file straight to a `.pdf` file. So you don't have to worry about all that dvips and ps2pdf stuff.

```
pdflatex latex-article.tex
```

latex2html

If you want to convert your document to html, `latex2html` is the perfect place to start. `latex2html` is a perl script, so it is a little slow and requires that you have perl installed. On the plus side, it works well and if you are running Linux, it should already be installed and ready to go.

txt2tex

Another cool program is `txt2tex` by Kalvis M. Jansons. It takes a text file and adds LaTeX markup, turning it into a LaTeX file that is ready to be run through either LaTeX or `pdflatex`. I tested it by feeding it some literature (*A Tale of Two Cities* and some Mark Twain) from Project Gutenburg. `txt2tex` did a great job—they all went through `pdflatex` with no problems with the markup.

11 Conclusion

That is it, you now know enough to start using LaTeX to create your documents. Not so hard, right? But realize that we have not even looked at a fraction of the capabilities of LaTeX—it may not be perfectly suited to every writing task, but it can certainly handle just about anything that involves the written word. You can insert images, create tables and figures, use just about any writing system in the world (including Asian languages), fiddle with every aspect of the layout, include hyperlinks, and a whole lot more. So enjoy.

Next time we will look at some slightly more advanced topics, including: using packages to extend LaTeX, tables and figures (including graphics), and how to make an index and a bibliography.

12 Hints

Don't forget the **maketitle** and the **tableofcontents** (if you want one) just after the **begin**{**document**}.

Don't forget to close environments. Starting a list and then forgetting the **end**{**itemize**} (or whatever list it was) is one of my most frequent mistakes.

13 Sample LaTeX Document

Here is an article template, so you have an idea how one might look.

```
\documentclass[a4paper,12pt]{article}

\title{}
\author{}
\date{}

\begin{document}

\maketitle
\tableofcontents

\section{}

\begin{itemize}
\item

\item

\end{itemize}


\subsection{}


\end{document}
```

14 About This Document

Copyright Notice