A Super Brief, Yet Super Awesome, LaTeX Cheat Sheet

Richard (Rick) G. Freedman, ~~Wake Forest University~~ University of Massachusetts Amher
Philip (Phil) Thomas, University of Massachusetts Amherst

~~6 February 2011~~ 14 June 2013

Contents

1 A Brief Introduction

LaTeX is a programming language that can be used for writing documents. It is especially useful for the mathematics and sciences fields due to its ease of writing special symbols and equations while also making them look good. Most textbooks are actually written in LaTeX. Due to the programming aspect, writing documents in LaTeX allows a lot more freedom to format how the document should look. However, it is important to learn how these formatting techniques

function in order to know what to expect (did you notice how Rick's school affilitation goes off the page?).

In this cheat sheet, we discuss some of the basics for writing documents in LaTeX. We hope it can help you get started with learning this useful language - LaTeX will make writing homework assignments, publications, posters, and many other documents far easier. For additional practice, the actual .tex file is included for you to see what we did. Feel free to modify it and see how your changes affect this document.

1.1 What You Will Need to Get Started

In order to write documents in LaTeX, you will need to download the standard LaTeX packages and compilers. There are a lot of packages and several compilers, and we barely scratch the surface about a few of them in Sections 6.3 and 6.6. Don't worry about finding all of them scattered across the internet; the essentials can all be found in one place. For Windows users, we suggest downloading MikTex. For Linux (and other *nix) users, we highly recommend TexLive. You can write your code in any text editor as well as any LaTeX IDE. Many of them are out there for free; so play with them and find what you like. We personally use TexMaker.

2 Special Symbols and Their Uses

To write a comment like the one at the top of the .tex file, use the percent sign (%). To place symbols like the percent sign into the compiled paper, use the escape character before the symbol (it is the forward slash (\) found above the ENTER key on the keyboard).

In fact, the escape character is used to begin any command in LaTeX. For example, to start this new paragraph, the command \par was written. However, if you looked at the .tex file itself, you would see that we are writing the escape character for the compiled paper as $ \setminus $. This is because two escapes characters in a row indicates a line break
that forces us onto the next line as just happened here. However, most journals request that the command \newline is used instead as done
here. They look similar in this document, but a journal's style file can modify how the \newline command looks. The two escape character version is unmodifiable.

So the next question is probably "Why the dollar signs ($)?" They place us into a special thing called "Math Mode." We will explain it later, but it is good to know that the dollar sign is also a special symbol. Other special symbols are the curly braces ({ and }) which contain parameters for commands, tilde ($\sim$) which generates a unit of whitespace that cannot be broken between two lines for word wrapping, carrot ($\hat{}$) which is used for superscripting in math mode, and underscore (_) which is used for subscripting in math mode. The ampersand (&) and pound (#) are also special symbols, but their uses vary by the context

of the document. Just remember to escape these characters (or use $ \sim $ for the tilde) when trying to print them in the document; we have done so in the .tex file if you read this paragraph.

3 Formatting Commands

Many commands in LaTeX will format the compiled paper. In order to understand what is formatted, it is usually bounded in some way just as Math Mode applies to anything between two dollar signs. In particular, there are two ways formatting is done:

3.1 In-Line Formatting Commands

These format only a little bit of text at a time. Because they only affect a small amount of text, these commands are a single command that receives the text in curly braces ({ and }) afterwards. Some important examples we will use are below. To show what they do, the actual command is used on the text in curly braces.

- \textbf{**Text to write in bold**}
- \textit{*Text to write in italics*}
- \underline{Text to underline}
- \sout{~~Text to strike out~~} You will need to use the *ulem* package (see Section 6.3)
- \textsc{Text to write in all capital letters without 'shouting'}

Best of all, these can be placed within each other (like nested blocks of code). ***That is why this sentence is written in bold, italics, and underlined!***

3.2 \begin...\end Formatting Commands (Also Called *Environments*)

When a large amount of text is formatted at one time with a command, it is easier to not have to contain it within brackets. This is why some formatting commands have two pairs of commands: \begin{format command} and \end{format command}. Think of them as the left and right curly braces of the in-line formatting commands. Anything between a \begin and \end with the same format command will be formatted. Some examples include:

{center} centers the text

{flushright} moves the text to the right side

{flushleft} moves the text to the left side

```
\{verbatim\} types everything character for character, including commands.
This is great for typing program code since there are so many special symbols
that would have been escaped.  Notice that we should not have escaped the curly
braces around the word 'verbatim' and been more careful with margins.
```

{tiny} shrinks the font to a really small size {huge} enlarges the font to a really large size

Besides formatting commands, \begin and \end are used for special segments of the document as well. These (which are far more useful) include:

- Creating lists. Each entry is indicated by the \item command. There are many types of lists including:
  1. {itemize} Bulleted lists
  2. {enumerate} Numbered lists
  3. {description} Labelled lists where each item starts with an emphasized word provided in brackets like \item[word]. Here's a secret: the brackets after the command \item can be used for the other list environments. Try them out and see how it looks different from the description environment.
- The paper itself is bounded between \begin{document} and \end{document} and THIS IS A REQUIRED ENVIRONMENT
- {tabular} creates a table after providing a little more information. We will briefly explain them in Section 5.
- {displaymath} will also put us in a special Math Mode that gets its own line that can present larger symbols. For example, in-line Math Mode's sum symbol looks like $\sum_{i=0}^{n} f(i)$ while displaymath Math Mode's sum symbol looks less squished

  $$\sum_{i=0}^{n} f(i)$$

  (the shortcut for the displaymath environment is two consecutive dollar signs $$ ... $$ as seen in the .tex file).

The only warning with all these commands is that any \end must have the same command as the most recently unmatched \begin in the file. This will line them up so that they match like the in-line brackets.

4 Math Mode Commands

Anything between dollar signs $ ... $ is written in math mode. It will accept many commands that are otherwise not available and formats the text to look "mathy" (so sentences should not go here). Since it is so straightforward, we will just list some commands used in Math Mode for common symbols:

- any text is *anytext* (Note that spaces are ignored in Math Mode since text is assumed to be a string of variables)
- \leq is $\leq$
- \geq is $\geq$
- \neq is $\neq$
- \(Greek letter) makes the Greek letter. Capitalizing the first letter determines whether or not the Greek letter is capital. For example, \Omega is Ω and \alpha is α. Not all of them are available, though. Some have alternative designs such as \phi (ϕ) and \varphi (φ)
- \in is $\in$
- \subset is $\subset$
- \subseteq is $\subseteq$
- \cup is $\cup$
- \cap is $\cap$
- \vee is $\vee$
- \wedge is $\wedge$
- \neg is $\neg$
- \ldots is $\ldots$
- \cdots is $\cdots$
- \cdot is $\cdot$
- \infty is ∞
- \sum is $\sum$ (use superscript and subscript for indexing)
- \prod is $\prod$ (use superscript and subscript for indexing)
- \int is $\int$ (use superscript and subscript for limits of integration)
- \partial is ∂
- \mid is $\mid$

- \leftarrow is $\leftarrow$ (guess what happens if you replace 'left' with 'up,' 'down,' 'right,' or 'leftright')
- \Leftarrow is $\Leftarrow$ (guess what happens if you replace 'Left' with 'Up,' 'Down,' 'Right,' or 'Leftright')
- \; adds extra white space to better separate symbols

\not preceding another Math Mode command slashes through it to get things like 'not divisible by:' $\not|$ Also, to help keep track of the left and right grouping symbols, \left and \right may precede parentheses, brackets, curly braces (which must be written as \leftbrace and \rightbrace), vertical bars (above the ENTER key), floor functions (written \lfloor and \rfloor), ceiling functions (written \lceil and \rceil), and period for nothing (which is useful in set notation, conditional probability, and piecewise functions). Another benefit of \left and \right is that they adjust their size appropriately in the displaymath environment: compare

$$(a+\frac{b}{c}] \text{ with } \left(a+\frac{b}{c}\right]$$

which also shows that the symbols do not need to match up. This is the purpose of the period when only one symbol is needed on one side to scale in the displaymath environment. The compiler will complain if the commands \left and \right do not match up correctly.

There are also some in-line formatting commands for Math Mode. The more useful ones are:

- ^ {The text to superscript such as 2 in x^2}
- _{The text to subscript such as 0 in x_0}
- \frac{numerator}{denominator} makes a fraction like $\frac{x}{y}$ when given {x}{y}
- \textnormal{any text} will print the text as though Math Mode is not in use

Notice that the superscript and subscript commands DO NOT have an escape character in front of them!

5 Tables

To create a table, you will have an extra argument to give when defining \begin{tabular}{layout}. The *layout* argument will specify how many columns there are, where the text is placed in each column, and if a line separates the columns. A layout example is {|c|c|lr|c|c} which will have 6 columns: a line of separation, a centered column, a line of separation, a centered column, a line of separation, a left column (no line of separation), a right column, a line of separation, a centered column, a line of separation, and a centered column (no line of separation).

When filling in the table, each line is an individual row (columns are separated by ampersands (&)) and they are separated by the \\ line breaks. To place a horizontal line of separtion, use the \hline command. It may be good to look at the table below both in the compiled document and in the .tex file. It will show the correlation between making the table and its code. You will need to start it on a new line (we use \par to do it here).

Hello	there	person	reading	this	!
This column is centered	So is this one	But here it is left	And here it is right	Now we are again centered	As we also are here
Above is	2 columns.	But we did not	separate with	\hline! How cool is	that?!
You can also		have a blank		column that is	empty

We must be careful. As seen above, the table can be made longer than the page. This is when compiling and editing will become a trial and error task. When in the displaymath environment (see Section 3.2), the environment name is changed from *tabular* to *array*, but everything else remains the same.

6 More Stuff

6.1 Sections and Subsections

This whole document has been organized by "chapters" that have their own numbers and titles. They are generated by two particular commands:

\section{title} These are the larger sections. They have a single number and the title. "More Stuff" is a section.

\subsection{title} These are the smaller sections within the section most recently declared in the .tex file. They have two numbers separated by a period (.). "Sections and Subsections" is a subsection of "More Stuff."

6.2 To Make Quoted Text Look Good

LaTeX prints the quote symbols as " and '. These only look good when placed at the end of the text. To get good quotation marks at the beginning of the text, use the backtick ` located below the ESCAPE key on the keyboard. `Thank you very much!'

6.3 Packages

Like in any programming language, other people have made things that can improve our coding lives. Many of these packages already come with a standard LaTeX install (Miktex for Windows and TexLive for Linux), and other ones can be downloaded and placed in the same directory as the .tex file. To use the package in a document, the command \usepackage{packageName} must appear after the \documentclass command. Some packages have special parameters that appear in brackets before the package name. Some common packages to include are:

- \usepackage{amsmath}, \usepackage{amsfonts}, and \usepackage{amssymb} form the set of AMS (American Mathematics Society) packages that the TexMaker IDE always suggests. With these three packages, almost every math symbol is available.
- \usepackage[margin=XX]{geometry} can cheat the margins when you want to save trees or specify particular guidelines. Replace XX with a floating-point number followed by a unit of measurement such as 'in' or 'cm' (no space between the number and unit).
- \usepackage{graphicx} allows graphics to be rendered in the document. It is a necessary evil in LaTeX.
- \usepackage{epsfig} allows PostScript images to be rendered in the document. It is a slightly less evil in LaTeX since the bounding boxes can be computed for you.
- \usepackage[normalem]{ulem} allows some other emphasis in-line formatting commands such as \sout.

6.4 Other Useful Commands

Some commands for formatting look very different from traditional in-line formatting and environments. In particular, they are constants whose values are set such as \parindent=XX for how much to indent paragraphs (with the \par command) and \parskip=XX for how much space to leave between consecutive paragraphs. In both cases, XX is replaced with a floating point and unit as done for the geometry package (see Section 6.3).

Also, some documents are better with a tabel of contents and title section (like this one). The command \tableofcontents will generate a table of contents where the command is specified and \maketitle will generate a title section with a given title (use command \title{title} beforehand), authors (use command \author{author(s)} beforehand), and the date of compilation (unless the command \date{date} is specified beforehand).

Footnotes can be made using the command \footnote{Text to appear in footnote}. They are numbered in order and placed at the bottom of the page during compilation. A variation of footnotes that is sometimes used for author information when generating the title section is the command \thanks{Author Information}.

6.5 References and Citations

When revising a document, it can be common to reorder sections, figures, and tables as well as edit the bibliography. Rather than have to change all the references and citations each time, LaTeX keeps track of them for you! When creating a section or caption, include the command \label{labelName} before ending it (as we have done for all sections in the .tex file). Then in the document, feel free to always write Section/Figure/Table~ \ref{labelName} and

the compiler will fill in the appropriate name in place of the \ref command. The tilde (∼) is a special character that makes sure that the number is not separated from the word preceding it. Likewise, the bibtex file's names for each bibliographic entry may be cited in this way in case the reference list changes. Simply use the command \cite{bibitemName}.

6.6 Compiling via Command Line

For Linux users who want to compile LaTeX documents without the IDE (especially if you code in a text editor instead), there are two paths to take depending on the file format of figures (if no images of either format, then both work). If a compiler error occurs, the best thing to do is make the specified change (warning: LaTeX gives bad error messages) in the .tex file, type a capital 'R' in the command line menu, and press ENTER; this terminates the compilation and spits out a lot of text, but avoiding the compiler's debug menu is worth it:

PostScript (.eps and/or .ps) First, compile the .tex file with *latex* in order to get a .dvi file. If you have referenced sections, then a second compilation will be necessary due to the one-pass compiler missing the label of later sections:

```
> latex myfile.tex
> #It doesn't hurt to compile again if there are references
> latex myfile.tex
```

If you have a bibtex file to also compile, then now is the time to synchronize them (since all the other steps convert the .dvi file). When compiling, an auxiliary .aux file is generated that lists which references are cited in each .bib file (since that information is in the .tex file). So synchronize the auxiliary file with the bibtex file:

```
> bibtex myfile.aux
```

Now recompile the document TWICE. The first time generates the bibliography section and a list of citation information (such as 'Author Year'), and the second time fills in the \cite commands with this information obtained from the previous compilation. Note that if the bibliography (not the in-text citations) changes, then synchronization needs to be done again by DELETING THE AUXILIARY FILE and then starting over the entire compilation process.

```
> latex myfile.tex
> latex myfile.tex
```

Second, convert the .dvi file into a .ps file using *dvips* which is 'dvi' followed by 'ps:'

```
> dvips myfile.dvi
```

Lastly, convert the .ps file into a .pdf file using *ps2pdf*:

```
> ps2pdf myfile.ps
```

The .pdf file is now ready, and the PostScript images are properly rendered.

PDF First, compile the .tex file with *pdflatex* in order to get a .pdf file. If you have referenced sections, then a second compilation will be necessary due to the one-pass compiler missing the label of later sections:

```
> pdflatex myfile.tex
> #It doesn't hurt to compile again if there are references
> pdflatex myfile.tex
```

If you have a bibtex file to also compile, then now is the time to synchronize them. When compiling, an auxiliary .aux file is generated that lists which references are cited in each .bib file (since that information is in the .tex file). So synchronize the auxiliary file with the bibtex file:

```
> bibtex myfile.aux
```

Now recompile the document TWICE. The first time generates the bibliography section and a list of citation information (such as 'Author Year'), and the second time fills in the \cite commands with this information obtained from the previous compilation. Note that if the bibliography (not the in-text citations) changes, then synchronization needs to be done again by DELETING THE AUXILIARY FILE and then starting over the entire compilation process.

```
> pdflatex myfile.tex
> pdflatex myfile.tex
```

The .pdf file is now ready with all PDF images included.

6.7 This Is Only the Beginning

LaTeX can do far, far more than these things. However, this should be enough to understand the basics. We can discuss other commands if we need them later, but they will be similar in syntax to the commands discussed above. Enjoy! Happy LaTeX'ing!