Getting Started With LaTeX

Written by Dave Pawlowski, January 9, 2013

1 Introduction

Latex is a document preparation system for high-quality typesetting. It is used by a large part of the scientific community as well as to prepare technical documents. It is not a word processor. The whole point of Latex is to allow the author to focus on their writing, and not on the document style. Latex is great for preparing journal quality articles and has features that can handle sectioning, cross-references, tables, and figures. It makes the typesetting of math formulae straightforward and quick. It automatically generates bibliographies and indices, and seamlessly accommodates the inclusion of graphics.

Learning how to use Latex can be a frustrating process. But, I absolutely promise that once you get through the learning pains, Latex will save you an huge amount of time when preparing your documents. And they will look much more professional than if you used other word processors.

2 Getting Started

Many people make the claim that using Latex is like coding your papers. In a sense, it is. The file that contains your writing also has commands that define the document structure and formatting. In addition, in order to produce the final document, you have to compile the latex document.

A very basic Latex file
Here is an example of a very basic latex document.

```
\documentclass{article}
\begin{document}
I am a very simple Latex document.  There are two lines that
tell latex the type of document, and when the content actually begins.
As you will see, you will include all sorts of information inbetween those
two lines.
```

```
Latex knows to start a new paragraph when you include a single blank line.
Many blank lines in a row will have no additional affect.  Also,
the last thing you have to tell latex is when your content ends, sort of like
an endif statement in a programming language.  This lets the Latex compiler
know when to stop processing the document.
\end{document}
```

You can see the results of this input file here.

The basic contents of a latex input file are simple. You have to tell latex which class of document you are creating, in this case an article. Latex supports many types of documents, each of which have their own basic formatting: report, article, book, and letter. Then you can put a bunch of optional stuff that latex may use later in the document. At some point, you have to tell Latex when the content begins, and at the end, when it ends.

Once you have the basic latex commands, you are ready to try to compile the document. Open up a file called first.tex and create a basic Latex document using the commands that are shown above. Once you've got all the commands and some text, save the file.

In order to compile the document, you use the following command:

```
% latex first.tex
```

The result of this command is several files: first.log, first.dvi, first.aux. The one that we care about is first.dvi. dvi stands for **device independent**, this contains the processed document. However, most people don't have dvi readers, so we use another command to convert to pdf.

```
% dvipdf first.dvi
```

and you will now have a pdf file. You can use the open command to view it:

```
% open first.pdf
```

This will invoke the mac program **preview**.

3 Latex Document Structure

All latex documents have to contain the 3 commands that we used above. Specifically, all documents at their most basic look something like this:

```
\documentclass{article}
preamble
\begin{document}
body
\end{document}
```

The preamble contains information about the document-wide formatting, as well as information about Latex packages that you may use in the body. The body contains the content of the document, as well as small scale text formatting commands.

There are a few Latex syntax guidelines worth remembering:

- Spaces and line breaks aren't important, except that a blank line starts a new paragraph.
- Commands always start with a backslash, \.
- Braces are used for arguments, i.e. document as above.
- Brackets are used for optional arguments, for example:

  ```
  \documentclass[11pt]{article}
  ```

- Commands are case sensitive.

The most common optional arguments to documentclass are:

- **11pt**- uses 11-point font instead of the default size
- **12pt**- uses 12-point font instead of the default size
- **twocolumn**- produces two column output

3.1 Sectioning

Latex has several levels of sectioning that make it easy to structure your document:

```
\section{section name}
   \subsection{subsection name}
      \subsubsection{subsubsection name}
```

The title of each section goes in the braces. Latex will automatically number your sections, and there are options for different number schemes.

3.2 Font Styles

Latex will automatically set the font for you, but you can specify other styles on the fly:

- `{\em text}`- italics
- `{\tt text}`- fixed-width typewriter-like font
- `{\bf text}`- bold font

The use of the curly brackets allows the inclusion of multiple words. If you only wanted to boldface a single word, you don't need the braces, for example.

3.3 Lists

To create any time of list, you need to enter a list environment. Environments are common ways in Latex to perform formatting on a block of text. This is opposed to inline formatting, where the formatting is applied to a text element. To enter an environment, you enclose the text with in a `\begin{}..\end{}` block, just like you enclose the text of your latex file using the `\begin{document} and \end{document}` commands. In some sense, this is the "document" environment. There are 4 types of lists in Latex:

- Bulleted- to create a bulleted list, you use enclose your list with the commands `\begin{itemize}` and `\end{itemize}`. Each item in your list is prefaced with the `\item` command (no braces).
- Enumerated- to create a numbered list, use `\begin{enumerate}` and `\end{enumerate}`, and again, use `\item`.
- Descriptive- composed of subheadings followed by one or more indented paragraphs. To create a descriptive list, use `\begin{description}` and `\end{description}` and use `\item`.

You can also make nested lists by defining another list environment within a list environment. Latex will handle the nesting and make an alternative bullet or numbering scheme.

3.4 Special Characters

Since certain characters are used in LaTeX commands (e.g., the backslash and curly braces), if you want to actually print these characters in your document, there are special commands that tell Latex to print these characters (not to treat them as part of a command). Here are some of those characters, along with the commands to print them:

```
character command
       \        $\backslash$
       $        \$
       %        \%
       ^        \^
       &        \&
       _        \_
       ~        \~
       #        \
       {        $\{$
       }        $\}$
```

4 Math Mode

Being physicists, one of the most useful aspects of Latex is math mode, which makes it easy to write equations and other formulae neatly and quickly. There are two ways to use math mode, ordinary, or inline, math mode, where the math material to be typeset is done so on the same line as the surrounding text, and display math mode, where the material is separate from the text.

4.1 Different Modes

4.1.1 Inline

To enter inline math mode, you simply use the dollar sign, $, to let Latex know that you are doing so. Typing simple math is relatively straight forward. For example, typing `$a^2+b^2=c^2$` results in: $a^2 + b^2 = c^2$.

4.1.2 Display

You can enter a math environment using `\begin{equation}` and `\end{equation}`. The result of this is material that is under the text preceding it. Latex will take care of numbering the equations for you. For example:

```
\begin{equation}
a^2+b^2=c^2
\end{equation}
```

will produce the following:

$$a^2 + b^2 = c^2 \tag{1}$$

Remember that Latex doesn't care about whitespace, and this holds for math mode as well.

4.2 Basic Math

Arithmetic Operations: The plus (+), minus (-), division (/) symbols have the usual meaning. To denote multiplication explicitly (this is rarely necessary), use `\cdot` (producing a centered dot) or `\times` (producing an "x"). The equal, less than, and greater than symbols on the keyboard work as expected; to get less than or equal, use `\le`; similarly, `\ge` gives greater than or equal.

Square roots are generated with the command `\sqrt{...}`. For example, `$z=\sqrt{x^2+y^2}$`.

Subscripts and superscripts: These are indicated by carets (ˆ) and underscores (_), as in $2n̂$ or a_1. If the sub/superscript contains more than one character, it must be enclosed in curly braces, as in `$2^{x+y}$`.

Fractions: Fractions are typeset with `$\frac{x}{y}$`, where x stands for the numerator and y for the denominator.

Sums and Integrals: The symbols for sums and integrals are `\sum` and `\int`, respectively. These are examples of "large" operators, and their sizes are adjusted by TeX automatically, depending on the context (e.g., inline vs. display math). Note that the symbol generated by `\sum` is very different from the capital sigma Greek symbol, `\Sigma`; the latter should never be used to denote sums. TeX uses a simple, but effective scheme to typeset summation and integration limits: Namely, lower and upper limits are specified as sub- and superscripts to `\sum` and `\int`. For example, `$\sum_{k=1}^n k = \frac{n(n+1)}{2}$`. (Note that the "lower limit" "k=1" here must be enclosed in braces, because it is more than 1 character long.)

Greek Letters: The commands for Greek letters are easy and intuitive: Just type ϵ, δ, ν, ϕ, etc. To get upper case versions of these letters, capitalize the appropriate command; e.g., Δ gives a capital Delta.

A really good resource for all things math mode can be found at http://en.wikipedia.org/wiki/Help:Displaying_a_formula.

5 Tables

Tables are produced in Latex using the tabular environment, as in `\begin{tabular}` and `\end{tabular}`. The best way to see how to make a table is to actually see one.

n	n!
1	1
2	2
3	6
4	24
5	120
6	720
7	5040
8	40320
9	362880
10	3628800

This table has produced using the following code:

```
\begin{tabular}{|c|l|}
  \hline
  n & n! \\
  \hline
  1 & 1\\
  2 & 2\\
  3 & 6\\
  4 & 24\\
  5 & 120\\
  6 & 720\\
  7 & 5040\\
  8 & 40320\\
  9 & 362880\\
  10 & 3628800\\
  \hline
\end{tabular}
```

First, we the tabular environment is entered. When beginning the tabular environment, there is a required format specification, the stuff you see in the second set of curly braces. That tells latex how many columns to use, how to justify the text in the columns, and where to put vertical bars (using the `|` key). In this example, we have two columns, the first is center-justified and the second is left justified. In addition, there is a vertical bar on both sides of the table, as well as one separating the two columns. Other options here are:

`l`	specifies a column of left-justified text
`c`	specifies a column of centered text
`r`	specifies a column of right-justified text
`p{width}`	specifies a left-justified column of the given width
`|`	inserts a vertical line between the columns
`@{text}`	inserts the given *t*ext between the columns

Once you've set the table up, its time to add the content. Horizontal bars are added using the `\hline` command. Then, each row in the table is written. The content of each column is separated by the ampersand, &. Since we have only two columns, only one ampersand is used in each row. Since whitespace doesn't matter in latex, you let Latex know to start a new row using the general new line command, two backslashes, \\.

Notice that there is no need to give any information on the dimensions of the table. Latex does all that for you.

You can have text that spans multiple columns very easily. Also, you can include normal Latex typesetting commands:

	Singular		Plural	
	English	**Italian**	English	**Italian**
1st Person	I go	**vado**	we go	**andiamo**
2nd Person	you go	**vai**	you all go	**avete**
3rd Person	he goes	**va**	they go	**vanno**
	she goes	**va**		

```
\begin{tabular}{|l||l|l||l|l|}
\hline
 &\multicolumn{2}{l|}{Singular}&\multicolumn{2}{l|}{Plural}\\
\cline{2-5}
 &English&{\bf Italian}&English&{\bf Italian} \\
\hline\hline
1st Person&I go&\textbf{vado}&we go&\textbf{andiamo}\\
2nd Person&you go&\textbf{vai}&you all go&\textbf{avete}\\
3rd Person&he goes&\textbf{va}&they go&\textbf{vanno}\\
 &she goes&\textbf{va}& & \\
\hline
\end{tabular}
```

Notice how the last line has two blank cells. Also, the `\cline` command has been used,

n	n!
1	1
2	2
3	6
4	24
5	120
6	720
7	5040
8	40320
9	362880
10	3628800

Table 1: A table showing the result of taking the factorial of the numbers 1 – 10.

which “clears the line”.

5.1 Captions

Usually when you include a table in a document, it should be accompanied by its own caption. This is simple in latex, you just add the `\caption` command.

```
\begin{table}
\centering
\begin{tabular}{|c|l|}
  \hline
  n & n! \\
  \hline
  1 & 1\\
  2 & 2\\
  3 & 6\\
  4 & 24\\
  5 & 120\\
  6 & 720\\
  7 & 5040\\
  8 & 40320\\
  9 & 362880\\
  10 & 3628800\\
  \hline
\end{tabular}
\caption{\small A table showing the result of taking the factorial of the numbers 1 -- 1
\end{table}
```

(I like to make my captions have smaller text than the rest of the document, hence the `\small` command.)

The only wrinkle is that I had to define a second environment to handle having material that is technically part of the table, but isn't inside it. That is accomplished using the table environment. The tabular environment is concerned with arranging elements in a tabular grid, while the table environment represents the table more conceptually. This explains why it isn't tabular but table that provides for captioning. Notice that this also allows me to center the entire table, using the `\centering` command.

When I enter the table environment, Latex treats the table differently than it did when I only used the tabular environment. Doing this causes Latex to treat the table as a float object, i.e. it isn't placed at the spot in the document that corresponds to where it is defined in the text. Floats exist to deal with the problem of an object that won't fit on the present page. The are not part of the normal stream of text, but separate entities, and are position in a part of the page to themselves (top, middle, bottom, left, right, etc.

All floats take optional positioning arguments when defining the float: `\begin{table}[placement specifier]`. The placement specifier can be one or more of the following:

Specifier	**Position**
h	Place the float here, i.e., approximately at the same point it occurs in the source text (however, not exactly at the spot)
t	Position at the top of the page
b	Position at the bottom of the page
p	Put on a special page for floats only
!	Force the position. Override internal parameters Latex uses for determining good float positions.

The same positioning arguments are used for figures as well, as you will see later.

Latex tries to put the table (or figure) where you want it, but it does have some guidelines that it also tries to obey. For one, Latex really wants the floats to be either at the top or bottom of the page. This means that the float is never sandwiched by the text. Latex will really struggle with positioning if you have several floats very close together in the text. In some cases, they may overlap or run into one another, so it becomes sort an art to space things properly.

By the way, I created that table using the following, in case you were wondering.

```
begin{tabular}{|l|p{5in}|}
\hline
{\bf Specifier}&{\bf Position}\\
```

n	n!
1	1
2	2
3	6
4	24
5	120
6	720
7	5040
8	40320
9	362880
10	3628800

Table 2: A table showing the result of taking the factorial of the numbers 1 – 10.

```
\hline
h&Place the float here, i.e., approximately
at the same point it occurs in the source text
(however, not exactly at the spot)\\
\hline
t&Position at the top of the page\\
\hline
b&Position at the bottom of the page\\
\hline
p&Put on a special page for floats only\\
\hline
!&Force the position.  Override internal parameters Latex uses for determining
good float positions.\\
\hline
\end{tabular}
```

6 Labels and Cross-referencing

One of the advantages to using Latex is that it handles labeling, numbering, and cross-referencing for you. The way that this works is you attach a label to some part of your document, then you reference the labeled object in the text. For example, I can add a label to our table:

```
\begin{table}
\centering
\begin{tabular}{|c|l|}
  \hline
  n & n! \\
  \hline
  1 & 1\\
  2 & 2\\
  3 & 6\\
  4 & 24\\
  5 & 120\\
  6 & 720\\
  7 & 5040\\
  8 & 40320\\
  9 & 362880\\
  10 & 3628800\\
  \hline
\end{tabular}
\caption{\small A table showing the result of taking the factorial of the numbers 1 -- 1
\label{factorial}
\end{table}
```

Once I have a label, I can reference Table **??** using the `\ref{factorial}` which is what I did. The argument of the label just has to be the same as the argument for the \ref. Note that the `\ref{factorial}` command only puts the number in the text, not the word "Table". I have to do that.

Note that this is Table **??**. Latex only counts floats, not the tables that were defined without the *table* environment.

When referencing floats, the label should always come after the caption, if there is one. Otherwise the numbering will be wrong if you reference the object!!!!

6.1 Recompiling

When you add a label and reference to your latex document and compile the document to create your pdf, Latex won't know enough to reference the object that you have labeled properly. Instead, it will print a *?* where ever you have tried to reference something. In order to get the correct reference, you need to recompile the code again. Latex then uses the information that it gained from the first compilation (and stored in the .log or .aux files) to correctly handle the referencing in your pdf file.

So, the processing of your document should always compile it twice:

```
% latex file.tex
% latex file.tex
% dvipdf file.dvi
```

7 Figures

The use of figures is much the same as tables from Latex's point of view. The only difference is that Latex alone isn't capable of handling and interpreting graphics files without the use of additional packages. I've mentioned two of the most widely used packages already, **epsfig** and **graphics**. You tell Latex that you want to use these packages by including them in the Latex document preamble like so: `\usepackage{epsfig}` and `\usepackage{graphics}`. The epsfig package handles all postscript files and graphics packages handles pretty much anything else, i.e., pdf, jpg, bmp, png, etc. When I create documents, I try to only use postscripts, and thus I only use epsfig. There are a variety of unix tools available that will convert jpgs, pngs, etc to postscript files, (Imagemagick is one of the most common unix utilities) and if all else fails, you can always do so using Photoshop.

So, here we'll just focus on the use of epsfig. Again, when you use a figure, Latex wants to create a float object. This is accomplished using the figure environment:

```
\begin{figure}[htp]
  \centering
  \epsfig{file=flare.eps,width=8cm}
  \caption{\small A solar flare!}
  \label{flare}
\end{figure}
```

Will produce Figure **??** (Figure `\ref{flare}`)

7.1 epsfig options

The `\epsfig` command has several optional arguments.

```
\epsfig{figure=, height=, width=, rheight=, rwidth=,
          bbllx=, bblly=, bburx=, bbury=,
          clip=, angle=, silent=}
```

Height and width are the height and width of the documents. If you specify one, but not the other, the document will maintain its proportions. rheight and rwidth specify

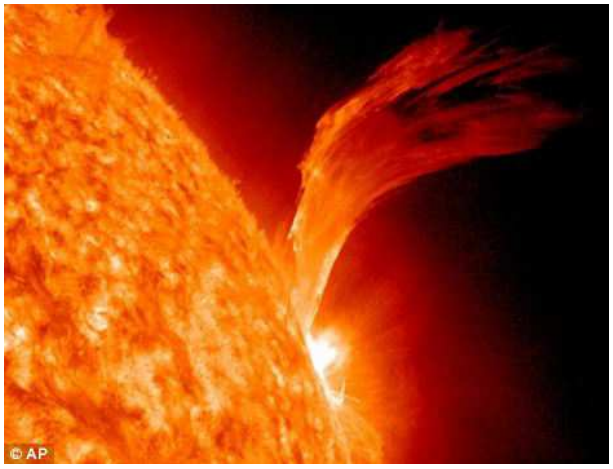

Figure 1: A solar flare!

the reserved height and width of the figure. This is how big the box that encloses the figure will be.

The *clip* option indicates whether or not the figure should be clipped at its bounding box. Clipping prevents lines in the figure from extending beyond the bounding box.

The *angle* Argument allows you to specify that the figure should be rotated. Always specify the angle of rotation in degrees. The figure is always rotated counter-clockwise.

Silent turns of messages printed by Latex as the figure is processed. I've never used this, but maybe I should, as Latex prints a lot of messages.

One of the most necessary stems from the difference between and eps and ps file. When you create a postscript using most programs these days, you usually create and encapsulated postscript (eps). The most glaring difference is that an eps contains a bounding box that surrounds the content of the document, as opposed to a file without a bounding box that doesn't tell Latex, or any other program, where the content is on the page.

If your file does not have a bounding box, then you can specify the part of the page that you want plot using the optional bounding box arguments for epsfig:

```
\epsfig{file=filename.ps,bbllx=lowerleft x coordinate,
bblly=lowerleft y coordinate,bburx=upperright x coordinate,
bbury=upperright y coordinate}
```

The easiest way to get these coordinates is to open up the postscript using **gv**. When you put your mouse on the figure, the coordinates of the cursor are written on the left side of the page under the "variable size" heading. The first number is the x-coordinate and the 2nd, the y-coordinate.

If you open an eps using **gv**, the image will take up the entire gv window, and there will be no "white" part of the page. If you open it and there is blank material, then there is no information about the bounding box in your ps file.

8 Bibliography and Citations

Another main reason that people choose to use Latex is that adding a properly formatted bibliography and citing references is very straightforward. A piece of software called BibTex is the standard tool for creating a bibliography in Latex. Bibtex generally comes as part of any Latex distribution.

In order to create a biobliography, you need to tell Latex what bibliography style to use. This is done using the `\bibliographystyle{stylename}`command, where `stylename` represents some style. Have a look at http://www.cs.stir.ac.uk/~kjt/software/latex/showbst.html to see some of the available styles (many of these are available on the web, while some, like plain, are built in).

In addition, most research journals and scientific books have their own style that you can download and use. In this example, I'll use plain.

Once you set the style, you want to tell Latex to create the bibliography:

```
\bibliography{mybib}
```

where *mybib* refers to a file called *mybib.bib* that contains a database of references. The database contains information about the name of the reference, the author(s), the name of the journal, book, etc, the year that it was published, and other information that depends on the type of reference, or entry. This command should be the last command before your `\end{document}` command, or else your bibliography might end up in the middle of your document!

In order to create a reference in Bibtex, you need to define an entry. There are a number of possibilities for this (http://nwalsh.com/tex/texhelp/bibtx-7.html), but the most common are article, inproceedings, and book. Once an entry is defined, Bibtex requires several fields be filled (like author) and will return an error if they are not. In addition, it is possible to include optional fields (like page numbers, or DOI), and even fields that are ignored, for your own reference. A sample entry looks like this:

```
@ARTICLE{pawlowski_params,
   author = {{Pawlowski}, D.~J. and {Ridley}, A.~J.},
    title = "{Quantifying the effect of thermospheric parameterization in a global model
```

```
  journal = {Journal of Atmospheric and Solar-Terrestrial Physics},
 keywords = {Thermosphere, Global climate model, Parameterization},
     year = 2009,
    month = dec,
   volume = 71,
    pages = {2017-2026},
      doi = {10.1016/j.jastp.2009.09.007},
   adsurl = {http://adsabs.harvard.edu/abs/2009JASTP..71.2017P},
  adsnote = {Provided by the SAO/NASA Astrophysics Data System}
}
```

An entry is defined using the symbol, and a name is given for the reference. On subsequent lines, all of the required and optional information is given. In the above example, *adsurl* and *adsnote* are ignored by bibtex. Many journal databases will provide Bibtex entries for a given reference, so keep an eye out for those, as it will save you the effort of adding them yourself.

The best part of Bibtex is that once you create the entry, you can use it for any document that you create. For example, I have a bibliography database called wholebib.bib that I use for every paper or proposal that I write. It was over 1000 references in it. The database that I use for literature is the NASA Astrophysics Data System (ADS). This database provides bibtex entries for every reference in the database. I never have to enter any information about my references.

Once you have created your bib file, and added the two lines above (`\bibliographystyle` and `\bibliography`), you are ready to add some citations in your file.

A citation is added to your latex document using the `\cite{ref}` command, where *ref* is the name that you gave the reference in your bib file.. For example, the following command, `\cite{pawlowski_params}` produces [**?**], and will add the corresponding reference to the bibliography at the end of the document (see the end of **this** document).

Finally, in order to get the citations and references to display properly, you have to add an extra step to your compilation. Assuming the latex document that I created is temp.tex, you should do the following to get your pdf:

```
% latex temp.tex
% bibtex temp
% latex temp.tex
% latex temp.tex
% dvipdf temp.dvi
```

That is not a mistake, you have to use the latex command 3 times. Once so bibtex knows which file contains the bibliography database, then times 2 and 3 as before. You only need to do perform these extra steps if you have added citations since the last time that you compiled.

9 Defining your own commands and environments

The ability to define your own commands and environments is an extremely powerful aspect of Latex that can really be a time saver. For example, in writing the tutorials for this class, I often like to highlight any code that I type, and preface it by a special symbol, i.e.:

% **ls /Documents**

In order to do this, I have defined the command, `\code`. There are two ways that you can do this: (1) define the command in the preamble of your document and (2) define the command in a separate *style* file, and include and includepackage call to it in the preamble of the current file. In all of my Latex documents, I have a line in the preamble: `\usepackage{pawlowski.sty}` that references the file, pawlowski.sty that contains all of my user defined commands. Feel free to download the file here if you would like to see it.

In order to define a command in either of the two above methods, you use the `def` or `\newcommand` command. One of the commands that I have defined is `\tbs`: `\def\tbs{$\backslash$}` which produces a backslash. In this case, `\tbs` is simply a short cut.

The `\code` command is a little more complicated, since it takes arguments (the text to be highlighted):

```
\newcommand{\code}[1]{\colorbox{bgblue}{\makebox[.95\textwidth][l]{\% {\bf#1}}}}
```

Here, the command `\code` is defined and can take 1 argument (the [1]). The argument is referenced using the #1 at the end of the command definition. Everything else in that command definition is already defined in Latex (colorbox, makebox,textwidth).

In some cases, you may want to redefine a command. You may want to slightly alter one of your own, or you simply may not like one of Latex's. This is done using the `\renewcommand` command (If you try to create a command that already exists using `\newcommand`, you will get an error). `\renewcommand` works in much the same way as `\newcommand`.

10 Other useful things

10.1 Whitespace

Latex tries to handle the whitespace for you, but sometimes you may want to make a few of your own adjustments.

To produce (horizontal) blank space within a paragraph, use `\hspace`, followed by the length of the blank space enclosed within braces. The length of the skip should be expressed in a unit recognized by LaTeX. These recognized units are given in the following table:

pt	point	(1 in = 72.27 pt)
pc	pica	(1 pc = 12 pt)
in	inch	(1 in = 25.4 mm)
bp	big point	(1 in = 72 bp)
cm	centimeter	(1 cm = 10 mm)
mm	millimeter	
dd	didot point	(1157 dd = 1238 pt)
cc	cicero	(1 cc = 12 dd)
sp	scaled point	(65536 sp = 1 pt)
em	em	Roughly the size of an M in the current font size
en	en	Roughly the size of an n in the current font size

This line contains 5 ems   of space.

To produce (vertical) blank space between paragraphs, use `\vspace`, followed by the length of the blank space enclosed within braces.

Note that negative lengths are allowed, and will cause the text to adjust in the opposite direction as a positive length.

You can also set the paragraph indent lenght using `\parindent{len}` command, and the space between paragraphs using `\parskip{len}` command.

10.2 URLs and files

Latex is capable of handling uniform resource locators and will create clickable links that most pdf viewers can handle. To do this, you need to include the hyperref package in your header. This will make the `\url` and `\href` packages available. The difference between the two is that url only takes one argument, `\url{sitename}` where has href takes two, `\href{sitename}{text description}`. You can link to a website two ways: `http://en.wikibooks.org/wiki/LaTeX/Hyperlinks` or by clicking here. The first was created using url, the second, href.

Both of these commands can be used to link local files, as well as email:

```
\url{run:/path/to/my/file.ext}
\href{run:/path/to/my/file.ext}{text displayed}
```

will produce a clickable link to a local file (not all pdf viewers support this). `\href{mailto:dpawlows@emich.edu}` will produce the following link: .

In order to get colored links, you need to set the colors using the `\hypersetup` command:

```
\hypersetup{colorlinks=true,anchorcolor=black,citecolor=black,filecolor=blue,
 linkcolor=black,menucolor=black,urlcolor=blue}
```

If you use any URLs in your document, it is a good idea to include the `\sloppy` command right after the `\begin{document}` command. Otherwise URLs may not wrap properly in the text.

10.3 Colors

Latex is happy to make use of colors. Use

```
\usepackage[usenames,dvipsnames]{color}
```

in the preamble . This gives you access to the `\textcolor` command and the 68 colors that dvipdf recognizes. `\textcolor{blue}{text}` produces some red text.

10.4 Page Layout

A page in Latex is defined by several internal parameters. Each parameter corresponds to the length of an element of the page, for example, `\paperheight` is the physical height of the page. You can set margin, text height, text width, and a number of other page settings in the preamble. See `http://commons.wikimedia.org/wiki/File:Latex_layout.svg` for a breakdown of the parameters, but a good place to start is to set up the document like so:

```
\setlength{\oddsidemargin}{0 in}
\setlength{\evensidemargin}{0 in}
\setlength{\topmargin}{0 in}
\setlength{\textwidth}{6.5 in}
\setlength{\textheight}{9.0 in}
```

10.5 Line spacing

It is easy to change the linespacing in your document. Use the *setspace* package and then a spacing command `\doublespacing` where you want the document to be double spaced.