LaTeX is Radical - Part II

Slightly beyond the basics.

Version 1.1.2

David Goulette

April, 26 2014

Contents

List of Figures

1 Before you begin reading...

Make sure you have downloaded all of the necessary files for compiling this document. If you want to build this .tex file you need the following supplemental files:

- `IntroToLaTeXpart2-version1.0.tex`
- `IntroToLaTeXpart2-version1.0.pdf`
- `alice.jpg`
- `teaparty.jpg`
- `RiemannSphereCircle.pdf`
- `myreferences.bib`

All of these files can be found on my website:

http://www.sjsu.edu/people/david.goulette/courses/latex/

You need to download all of the above files and save them in the same directory. If you want to build the .tex file it needs to have the three picture files in the same directory to work and also the `.bib` file with the bibliography references. I explain the details of how you can add pictures to documents in section 4. And I explain how to make a bibliography in section 5.

2 Indroduction

Welcome to the second part of my introduction to LaTeX! In this document I will introduce a variety of important LaTeX skills that you might not need quite as frequently as those found in part 1, but you will benefit from knowing them. I am separating this document from the "basic introduction" for a few reasons.[1] One reason is that part 1 covered basic topics, skills and concepts that one must know in order to branch out and learn all of the many options LaTeX has to offer. I have attempted to choose material in such a way that after you read my material, you will find that most new things you want to learn will just me a variation on what I have already taught you (up to a point of course... there are some advanced things that LaTeX can do which I will not mention). So I wanted to stick to basic concepts that come up more frequently in part 1. Admittedly my bias for my choice of topics was toward writing mathematical papers, but I tried to emphasize the key LaTeX skills that are needed in general. This document, on the other hand, will cover things that you might need less frequently, but you will need them eventually if you do anything slightly more advanced or professional. Another reason that I wanted to separate this section is that this document will need more than just the .tex file to compile it. That is because I intend to teach you how to add pictures (which requires separate picture files for each picture you want to add to your document) and I also want to teach you how to make a bibliography with BiBTeX (which requires a separate .bib file). Since I want to give you my code so that you can compile it from scratch yourself, you will need to download a few files and have them all in the same folder/directory on your computer when you compile. This is a little more complicated than what I wanted to explain in the basic introduction. Finally, just about every section in this paper will only *introduce* you to the details of a concept, show you some examples, and then direct you to outside sources that are far more complete than I will be here. It is my hope that I will have helped you get past the initial LaTeX learning curve and that you will be able to teach yourself whatever you want to know with free online sources.

As you know, this is a work in progress and you are seeing my initial rough drafts of this material, so here is a list of the things I plan to have in this document (in the order I plan to write them, but not necessarily the order that they will appear in the final version).

1. How to have multiple columns in all or part of a document.
2. How to add pictures/figures to your document, label them and reference them.
3. How to create a basic bibliography using BiBTeX.
4. How to create your *own* LaTeX commands, macros and functions that do whatever YOU want them to do!
5. How to create formatted theorems, corollaries, proofs, definitions, etc. using `amsthm`. Also how to label them and reference them of course.
6. Finally, a section of references for further information and advice that goes beyond what I have explained. This section will possibly have some advanced examples like, tables, tabbing, commutative diagrams (with tikz), variations on the equation and align environments etc. Mostly this section will be intended to make you aware of what LaTeX can do and give you pointers to the resources where you can learn it on your own.

[1] Just in case you have not read the first part, you can get it here:
http://www.sjsu.edu/people/david.goulette/courses/latex/

3 Multiple columns and paper orientation

It is very easy to make your document have two columns globally. When I say "globally" I mean that your *entire* document will have two columns. All you have to do is add an optional argument to your `\documentclass` function at the very beginning of your preamble. Instead of

```
\documentclass{article}
```

do this instead:

```
\documentclass[twocolumn]{article}
```

Optional arguments to any document class go in square brackets before the curly braces. This two column option creates global two-column format. Now, I can't show you an example of this because I don't want two column format in this whole document! So you will have to try it on your own.

I use this often when I am writing quizzes or test for my students and I want to save paper. I want the margins to be small so I can fit as much as I can on the page but I don't want the lines to stretch really wide across the page (which is hard to read). But aside from this, I like the look of two columns because they are very easy to read (newspapers and professional websites usually have narrow columns for this reason; you can read them faster). You will find that many professional journals are published in two column format as well. Just in case you don't like the amount of space that is put between the columns, you can adjust this width with by adding the following line in your preamble:

```
\setlength{\columnsep}{length}
```

The `\setlength` function sets the length of the first argument to be whatever length in you specify in the second argument. So in this case we are setting the length of the column separation. And of course the value for `length` can be whatever you want (as long as it fits inside of the margins!). Also, the length you put in the middle can use any of the units we discussed in part 1. So you could do this for example:

```
\setlength{\columnsep}{1cm}
```

By putting this in your preamble you are setting the column separation to be 1cm globally.

Sometimes, when I want two-column format, I will change the paper orientation to landscape as well. Landscape with two-columns saves paper and is also very readable. You can accomplish this with another optional argument in your document class declaration:

```
\documentclass[twocolumn, landscape]{article}
```

This will, of course, rotate your paper orientation by π radians.

But sometimes I want a bit more flexibility with multiple columns. Occasionally I want multiple columns for just a *portion* of my document but not the whole thing. And sometimes you might want three, or four columns instead of just two. In these cases you should opt for the `multicol` package. (See the preamble to this document.) This package allows you to create easy multiple columns inside of an environment like this:

```
\begin{multicols}{3}
Blah blah, text goes here...
\end{multicols}
```

And this is what you will get. By-the-way, take care to note that the *package* is called `multicol` but the environment syntax has to have `multicols` with an `s` on the end. The above code will create 3 columns because of the `3` in the second argument to the

`multicols` environment declaration. So of course if you want two columns, just put a 2 instead. You can have up to 10 columns using the `multicol` package. Also it is very important that if you decide to use the `multicols` environment that you DO NOT use the `twocolumn` option in your `\documentclass` declaration at the beginning of your document (like I described earlier). If you do them both, they will clash and might give you errors. The `twocolumn` option is a built in LaTeX option. Side note: you will learn that occasionally some packages clash with basic LaTeX functions and sometimes different packages will clash with each other and give you errors. Since packages are independently developed and freely available, this is inevitable. But with some experience (and good internet searching skills) you can figure out what to do and what not to do. (O.k., so I will end this three column format now.)

You can adjust the width of the separation of the columns in the `multicols` environment in the same way that I mentioned before. So you can set a global column separation in the preamble with `\setlength{\columnsep}{length}` function and `multicol` will use that separation length. If you don't put this in your preamble then it will just use a default margin separation length. For this document, I chose to put this in the preamble:

```
\setlength{\columnsep}{4mm}
```

So the 3 column text you see above has a 4mm separation between the columns. But the great thing is that you can easily change the column separation settings in in the middle of the document by using the same function right before the `multicols` environment declaration. This will override the global setting in your preamble and it will only effect the multicolumn sections that *follow* the command. None of the mulicolumn sections that *precede* the new `\setlength` function will be affected. So you will see that my next multicolumn section below has a wider space than the one above. Another useful thing that I will demonstrate here is how to add an optional center line between the columns. This is accomplished with the following command: `\setlength{\columnseprule}{length}`. And the `length` will be the *thickness* of the line. By default this thickness is set to 0, meaning there is no line between the columns by default. I will show you an example of both of these things right now using some dummy text created by the `lipsum` package.[2] The only thing the `lipsum` package does is create dummy text for demonstration purposes, which is exactly what I am doing here!

```
\setlength{\columnsep}{1cm}
\setlength{\columnseprule}{0.5pt}
\begin{multicols}{2}
\lipsum[1-2]
\end{multicols}
```

This block of code will create two columns of text, but before the multicolumn block stars, I set the column width to be one centimeter (which overrides my 4mm global setting), and will put a thin vertical line that is only a half of a point wide. Oh yeah, and the function `\lipsum[1-2]` will create the first two paragraphs of dummy text from "Lorem ipsum dolor sit amet,...", like this:

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

[2] For more information see:
http://www.ctan.org/pkg/lipsum

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

If you want a new multicolumn environment that does not have a ruled line between the columns, all you have to do is reset the `\columnseprule` length to zero like this:

```
\setlength{\columnseprule}{0pt}
\begin{multicols}{2}
\lipsum[3]
\end{multicols}
```

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Here is one last thing about `multicols` that will be useful. When you are inside of the `multicols` environment you will sometimes want to break to the next column in the same way you do a `\pagebreak`. Fortunately a function that comes with the `multicols` package is `\columnbreak` which does the trick. I will show you the code on page (page 6), since it is a little long, and the result will be on it's own page (page 7), to make it clear.

In the following example I am creating the first page in a basic algebra exam. I want it to be two columns, I want a dividing line between the columns, but I also want to leave enough space for students to work. After each of the first two questions I force some fixed vertical space with `\\[length]` which we covered in part 1 of the LaTeX introduction. But note that after the third question I used `\vfill` and then a `\columnbreak`. This will ensure that the rest of the column is filled with vertical blank space, and then the next question is forced to the top of the next column. After the fourth question I want to leave lots of space for their work, so I use another `\vfill` and a `\pagebreak` (since I want this to fill the rest of the page). Now it is important that you fill the rest of the columns if you want it to stretch to the end of the page to get this result.

```
\setlength{\columnsep}{5mm}
\setlength{\columnseprule}{.5pt}
\begin{multicols}{2}
\noindent
\underline{\textbf{Exam}}\\
\noindent
1) Solve:  $x^2-9 = 0$\\[2in]
\noindent
2) Solve: $(x+5)^2-10=0$\\[3in]
3) Solve:  $x^2+2x-9=0$\\

\vfill
\columnbreak
\noindent
4) Complete the square of the following quadratic in order to put it in vertex form.
Also find the $x$ and $y$ intercepts of the function.  Finally, graph the function and
label the vertex and the intercepts. Show your work!
\[
f(x)=x^2+4x-8
\]

\vfill
\pagebreak
\end{multicols}
```

Exam

1) Solve: $x^2 - 9 = 0$

2) Solve: $(x + 5)^2 - 10 = 0$

3) Solve: $x^2 + 2x - 9 = 0$

4) Complete the square of the following quadratic in order to put it in vertex form. Also find the x and y intercepts of the function. Finally, graph the function and label the vertex and the intercepts. Show your work!

$$f(x) = x^2 + 4x - 8$$

Notice that I did the numbering of each question on the exam manually in the example above. I actually would not do it this way. I would use an the `enumerate` environment and let LaTeX number the problems for me automatically. That way I could rearrange the problems or remove problems without worrying about changing the numbers (I just wanted to keep the example simple.) For more on this topic, see the .tex file for this document and look right before this paragraph where I have an alternate version of the above example that is commented out (so you can't see it in the final document). In that example I use automatic numbering.

There are more things that the `multicol` package does which I am not mentioning, and some technical details that I am avoiding as well. You will need to play with it and also read the documentation here: http://www.ctan.org/pkg/multicol

4 Adding pictures to your document

This is a really big topic and I will only be able to give you some of the basic ways to add pictures. My goal here is to give you a few examples that work for the basic cases to get you started and then point you to references that are more detailed and comprehensive.

4.1 Picture file types

Without going into too many details I want to briefly explain two important types of graphics formats that are common in computer graphics: *bitmaps* and *vector images*. (The word "vector" doesn't mean drawing vector arrows in math, it is a type of image encoding.)

Bitmaps are pictures that use pixels to create the image. Common bitmap file extensions are:

- ▷ `.bmp`
- ▷ `.gif`
- ▷ `.jpeg`
- ▷ `.jpg`
- ▷ `.png`
- ▷ `.pcx`
- ▷ `.tiff`
- ▷ `.psd` (Adobe Photoshop)

←I used `multicol` and an enumerated list here. Also, I altered the bullet point to be a triangle just for fun. Check out the .tex file for details.

The good thing about bitmaps is that they tend to have smaller file sizes (in comparison to vector graphics) and are easier to edit quickly. There are many programs that create these types of files (Microsoft paint, Corel Photo-Paint, Photoshop, Smooth Draw, SketchBook Pro, the list is endless). Bitmaps are great when your picture is very high resolution like a photograph from a digital camera. But if you have a lower resolution bitmap file or if you resize your bitmap file the quality might not be that great. You have likely noticed that when you zoom in on a digital photograph, the quality gets worse and worse. At the extreme, you can see the individual color pixels and you lose the detail.

Vector graphics behave differently and have different strengths and weaknesses. Vector graphic files tend to have a larger file size than bitmaps and they can be harder to edit. But the cool thing about vector images is that you can zoom in on them and they do not lose their sharpness. If you are reading this text in Adobe Reader, try zooming in on one letter in this text. You will notice that the edges of the letter stay sharp. This is because the text in this document is created with vector graphics. This is also partly the reason why .pdf files print with such clean professional quality. So if you want to add pictures to your LaTeX document you should know that low quality bitmap pictures might not look good next to your professional looking vector graphic text.

There are many obscure vector graphic file extensions that I have never run into but here are a few that I have:

- ▷ `.ai` (Adobe Illustrator)
- ▷ `.cdr` (CorelDRAW)
- ▷ `.svg`

I use .svg files frequently because that is a an open standard that has is used by many free vector drawing programs. I particularly like working with Inkscape.[3]

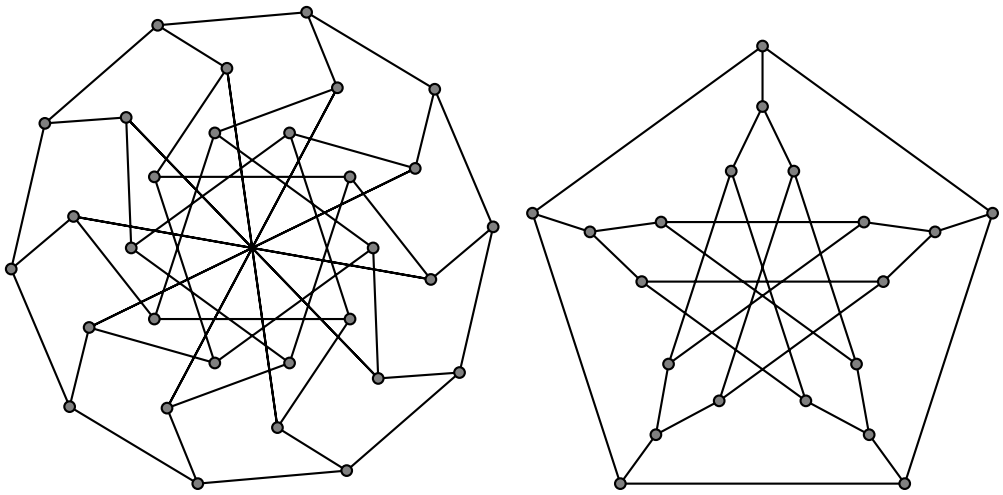
Now, it turns out that .pdf files are a hybrid and can contain both file types, which is a good thing. (Another closely related file type is .eps or "Encapsulated Post Script" which is also a hybrid.) This allows you to add either type of file to your LaTeX document when you use pdflatex as your compile option. But just be aware that low quality bitmap files may look really bad next to your professional quality LaTeX text. But if you are using a digital photo or high resolution scan, then a bitmap file should look fine.

O.k., so now that you know a little bit about image types, you should know that there are two general ways to add images to your document with LaTeX. The first method, which I will discuss briefly in the next section, is a complicated method that I won't say much about but I want you to know it exists. The second method is the easier and most commonly used method that I will explain with a bunch of examples.

4.2 PGF/TikZ

The first method that I will mention for adding a picture (or diagram) to your document, is one that, to be honest, I know very little about. I don't know very much about this because I have never taken the time to learn it (and up to this point I haven't needed it). But I want to mention it just so you know it exists and just in case you are into coding things from scratch yourself. But at the end of the section I want to mention one cool package that I *have* used which is built upon the tools mentioned in this section. (So consider this section optional reading; you can move right on to the next section if you want to get to the every-day/practical/easy way to add pictures.)

The method I speak of here is to directly code your figures yourself using PGF/TikZ. These tools are really powerful and can create some amazing results if you are really good at it. I have added the `tikz` package and I have borrowed an example of some code that creates a pair of graph theory diagrams.[4] Here they are:



If you are interested in the code that generated those cool graphs, check out the .tex file for this document. But again, I only have a vague idea about how this code works (so don't ask me questions about it ☺). If you are the do-it-yourself type, and into computer programming, then `tikz` might be your thing.

But one related package that I *have* used is `tikz-cd`. This package is an easy-to-use package that

[3] Inkscape is a full featured vector drawing program that is great for mathematical drawings. Admittedly, it takes a bit of effort to learn the software but there are some nice resources online for learning the software. The best part is that it is free as in "speech" and "beer." Here is the website: http://www.inkscape.org/en/

[4] If you want to see some great TikZ examples including the one I borrowed here, see:
http://www.texample.net/tikz/examples/
If you would like to learn more see the two manuals here:
http://www.ctan.org/pkg/pgf

is a very simplified version of `tikz`. It is streamlined to do one thing very well: it is great for making commutative diagrams (hence the "cd" in the name). Now if you have never heard of a commutative diagram don't worry about it. But if you are into advanced mathematics these pop up occasionally (in advanced algebra, topology, manifold theory etc.). They are essentially diagrams that map out the relationships between functions. Here is an example that you might see in a smooth manifolds course. Suppose you have the following four functions defined on the following sets:

$$\begin{aligned} q &: U \to \widetilde{U} \\ \varphi &: U \to \mathbb{B}^n \\ \widetilde{\varphi} &: \widetilde{U} \to \mathbb{R}^n \\ \widehat{q} &: \mathbb{B}^n \to \mathbb{R}^n. \end{aligned}$$

Furthermore, suppose these functions commute, meaning: $\widetilde{\varphi} \circ q = \widehat{q} \circ \varphi$. Then it is very nice to depict the relationship between these functions with a commutative diagram like this:

$$\begin{array}{ccc} U & \xrightarrow{q} & \widetilde{U} \\ {\scriptstyle\varphi}\downarrow & & \downarrow{\scriptstyle\widetilde{\varphi}} \\ \mathbb{B}^n & \xrightarrow{\widehat{q}} & \mathbb{R}^n \end{array}$$

The arrows show the direction of the mappings and the overall picture is much clearer.

Here is the code that created that in case you are interested (think of the code as two lines in an align environment but with connected arrows):

```
\begin{center}
\begin{tikzcd}
U \arrow{r}{q} \arrow{d}[swap]{\varphi}
& \widetilde{U} \arrow{d}{\widetilde{\varphi}} \\
\mathbb{B}^n \arrow{r}{\widehat{q}}
& \mathbb{R}^n
\end{tikzcd}
\end{center}
```

If you want information about this package see:

http://www.ctan.org/pkg/tikz-cd

4.3 Adding figures with `graphicx`

Now it is time to learn the more common, every-day method for adding figures to your document. I will keep things simple here with some easy to copy examples. My intent is to explain the topic to someone who is new to this. If you want to jump right to a more detailed presentation on adding graphics, by all means go here:

http://en.wikibooks.org/wiki/LaTeX/Importing_Graphics

But my presentation here will hopefully be sufficient for most basic purposes.

We will need two new packages in this section which I have never mentioned before: `graphicx` and `caption`. The `graphicx` package will help us handle adding the figures and the `caption` package makes adding captions to your figures very easy.

When you add figures to your document with `graphicx`, you will be adding a preexisting picture file to your document. This picture file needs to be saved in same working directory as the .tex file that you are compiling.[5]

[5] Well... that is not the whole story but I am trying to keep it simple. You can actually add pictures from anywhere on your computer using the `\graphicspath` command. See the section entitled "Graphics storage" here for more on this:
http://en.wikibooks.org/wiki/LaTeX/Importing_Graphics

When you are adding graphics to your LaTeX document with `graphicx`, make sure you are compiling your document with the pdfLaTeX option. Your LaTeX editor should have an option controlling what format it will create after it compiles your document. In TeXworks, for example, there is a pull down menu next to the compile button. I use TeXstudio and pdflatex is the default choice. If you build your file with the pdflatex option, then you can add any of the following picture file types to your document:

- ▷ `.jpg`
- ▷ `.png`
- ▷ `.pdf`
- ▷ `.eps`

An `.eps` file format is similar to a `.pdf` and stands for Encapsulated Postscript (you will likely see references to this in LaTeX documentation). The two file formats on the left are for bitmaps and the two on the right are for vector graphics.[6]

4.3.1 Adding pictures in a fixed position

To add a picture to your document all you just need to have the `graphicx` package added in your preamble and then you can use

```
\includegraphics[options]{filename}
```

In all of the examples in this section, this function will add a picture in precisely in the place where you put this command. The `filename` is the name of the picture file that you want the `graphicx` package to pull into your document. (In section 4.3.2 we will cover how to add pictures is such a way that you will let LaTeX decide the best place to fit the picture.) I will explain the `options` in a minute; you are not required to specify any options. For the `filename`, I recommend you only type the name of the picture file and do not add the extension (like .jpg or .pdf etc.). Just make sure that all of your picture files that you are adding to your document are in your working directory and have different file names. In my current directory I have a file called `alice.jpg` which is a bitmap file of one of the original illustrations of Alice from *Alice in Wonderland* (illustration by John Tenniel in 1865). To add this picture to my document all I need is this:

```
\includegraphics{alice}
```

If I didn't have a file in my current directory with the name "alice," I would get an error. The result of this code is so large that it will end up filling almost all of the next page.

[6] I mentioned earlier that I like to use Inkscape to create vector graphic drawings. By default, Inkscape saves in the `.svg` format, but it has a option of saving as a `.pdf`. So when I do a drawing in Inkscape, I save it as a `.pdf` and then use `graphicx` to add that file to my LaTeX document.


DRINK ME

Since I didn't provide any optional arguments to the `\includegraphics` function, the picture was blown up to fill the page. If you want to make the image smaller you can add the optional input `scale=` inside square brackets. You then set the scale option equal to a scale multiplier. So let's make Alice shrink (don't worry, Alice is quite accustomed to shrinking... she is about to drink the vial of liquid you know):

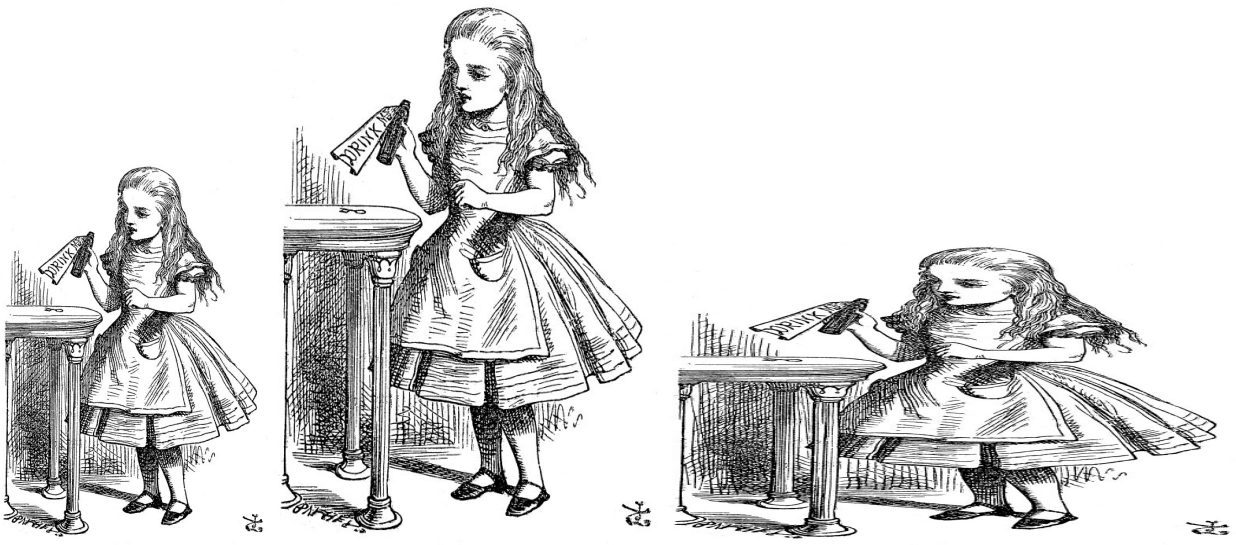
```
\includegraphics[scale=0.25]{alice}
```

I will put the result of this code on it's own line:

This shrunk Alice to a quarter of her original size. You might notice that the picture is indented, that is because I put the picture on it's own line after a `\\` and LaTeX treats this picture just like adding a symbol or text.

You can also specify the options `width` and/or `height` in a similar way to `scale`. You just set them equal to whatever length you want with whatever units you want. If you only specify either `height` or `width` (but not both), then the proportions (aspect ratio) of the image will be preserved. But if you specify both lengths the image will be stretched to exactly those dimensions. Here is an example of all three. The third example has two optional arguments so you need a comma separating the multiple arguments:

```
\noindent
\includegraphics[height=2in]{alice}
\includegraphics[width =2in]{alice}
\includegraphics[height=4in, width=3in]{alice}
```



The first one is 2 inches tall and the width adjusts proportionally. Similarly, the second picture is 2 inches wide and the height adjusts proportionally. But the third picture is forced to the specified dimensions and thus is distorted. Also, the forced dimension caused it to overflow into the margin (so you have to pay attention to your output). Also notice that in the last code example above, I added the three figures in succession without any newline command. These three figures are added on the same line just as if I typed three letters or math symbols in succession. Adding a picture is just like adding any symbol; it just takes up more space. Since the pictures act just like text, I used `\noindent` at the beginning of the line since I wanted the pictures to be flush with the left margin. The bottom of the figures are on the same line. So if you have text in the same line as a picture it will be in the same line like this:

```
\includegraphics[scale=0.1,angle=90]{alice}
$\Leftarrow$ Alice rotated $\Rightarrow$
\includegraphics[angle = -30, scale=0.15]{alice}
```


⇐ Alice rotated ⇒


The example above also shows how you can rotate a picture as well. Positive angles rotate counterclockwise and the units are in degrees.

Now, all of the figures I have added in the examples above have been added inline, but you can also add figures inside of an environment like `center`. In the next example I will also show you how you can set the width to be a multiple of the width of the page so you don't have to choose a specific width.

```
\begin{center}
\includegraphics[width=.45\columnwidth]{alice}
\end{center}
```

This will make the picture centered and it will fill up 35% of the width of the main text column. Instead of `\columnwidth` you can substitute `\linewidth` or `\textwidth` if you like. I like to use `\columnwidth` because it works inside of a multicolumn environment to fit the thinner column. Here is the result of the last code example:

If you want to crop a picture you can do that too. You need to add the argument `clip=true` to the optional arguments and then `trim=` followed by four lengths representing how much you are going to trim from the *left*, *bottom*, *right* and *top* in that order. The following code is an example of a cropped picture inside the `flushright` environment. I will trim 2cm of the left 12cm off the bottom, 3cm off the right and 0cm off the top. Then the result will be rescaled to fit 30% of the column width (it is not exactly obvious how the trimming will look; you sort-of have to play around with crop lengths and keep building the file and see what happens):

```
\begin{flushright}
\includegraphics[width=.3\columnwidth,trim=2cm 12cm 3cm 0cm,clip=true]{alice}
\end{flushright}
```

Here is the result:

You can also reflect a picture by putting it inside a `\reflectbox{}` which flips anything inside of it right to left (let me emphasize that you can flip anything, including text, in a `\reflectbox{}`):

```
\begin{center}
\reflectbox{\includegraphics[width=3.5in]{alice}}
\end{center}
```

So here is Alice after she has gone through the looking glass:

4.3.2 Captions and floats

A "float" is a certain special type of object in LaTeX. Floats are given the name "float" because they will not necessarily be created exactly where you put them in your code. Instead, when you create a float, you allow LaTeX to decide *where* it will best be placed in the document so that it looks good. The two main types of floats that are available in LaTeX are *figures* and *tables*. To create a figure or a table you need to use the `figure` or `table` environment. Up to this point I have not taught you how to create a figure or a table, so I have never explicitly taught you how to make a floating object yet. In this section I will explain how to make a floating figure with the `figure` environment (I will explain tables in a later section). But first I want to emphasize that none of the pictures that we added in the previous two sections were added in the `figure` environment so *none* of them were floats. All of the pictures we added above were added to the document exactly where we declared them, either right in-line, or inside of an environment like `center` or `flushright` (which are not floating environments). But you *can* include a picture inside of a `figure` environment and then the picture will become a float. The downside to adding pictures in the way we did above, is that you might get a bunch of awkward empty space in some places. See for example, the bottom of page 14, where we added a picture of Alice that filled up 45% of the columnwidth. That picture was too big to fit the rest of the page in that situation, so it forced the picture to the top of the following page, which left a lot of unused blank space at the bottom of page 14. When you use the `figure` environment, this will not happen. LaTeX will move the figure around and adjust the text so that there won't be blank space. It's time for an example.

The following is an example using the figure environment. I will show you the code here and then I will explain it below. This example also shows you how to give a figure a caption using the `caption` package and I also label the figure so we can reference it later:

```
\begin{figure}
\centering
\includegraphics[width=0.6\columnwidth]{teaparty}
\caption[A Mad Tea-Party]{From left to right we see Alice, the March Hare, the
Doormouse, and the Hatter.}
\label{fig:teaparty}
\end{figure}
```

Even though I added this code example *before* I typed this paragraph, LaTeX floated the picture to the top of the next page because it fits better there. So it isn't leaving any empty space on this page. You should look at the code in the .tex file to see what I mean. In the example code above, I am adding the picture file `teaparty.jpg` inside of the figure environment in the exact same

Figure 1: From left to right we see Alice, the March Hare, the Doormouse, and the Hatter.

way I explained in the sections above. The only thing that is new is that it is inside of the figure environment. I used the `\centering` function to center what is inside of the figure environment. This is not quite the same as using the `center` environment. I recommend you only use the `\centering` command inside of another environment like I am doing here (or inside of any box that creates a paragraph).[7] When you create a figure you can easily add a caption with the `caption` package which gives you the function `\caption[list entry]{caption text}`. The first, optional argument to this function is in square brackets. This `list entry` is the text that gets added to the list of figures that you see on page 1. The second argument in curly braces is the caption that is added below the actual figure. Make sure to put the `\caption` function right after the `\includegraphics` line. When you add a caption, it creates a figure number which you can label and reference with the same methods we learned in part 1 of the introduction to LaTeX. Only figures that have numbers will be added to the list of figures that you see on page 1. If you want a caption but you don't want the figure to be numbered, you can use the `caption*{caption text}` version of the function instead.

Because we labeled the Mad Tea-Party picture, we can reference it. If we type:

```
In figure \ref{fig:teaparty} we see John Tenniel's Mad Tea-Party illustration.
```

This, of course, is the result:

In figure 1 we see John Tenniel's Mad Tea-Party illustration.

By-the-way, I mentioned the list of figures at the beginning of the document. That is very easy to create. All you have to do is type `\listoffigures` wherever you want it to go, and it will automatically be formatted for you. It works just like `\tableofcontents`.

Now sometimes you might not like the place where LaTeX places your float. If that ever happens you have some options. One is to add one (or more) of the various optional placement commands to the `figure` environment declaration. These go in square brackets like the one you can see here (I have added the `t!` commands):

[7] For more on the differences between the `center` environment and the `centering` command, see:
http://tex.stackexchange.com/questions/23650/when-should-we-use-begincenter-instead-of-centering

Figure 2: Drink me!

```
\begin{figure}[t!]
\centering
\includegraphics[width=0.3\columnwidth]{alice}
\caption{Alice}{Drink me!}
\label{fig:drinkme}
\end{figure}
```

If you just put a `t` in the square brackets, then you are requesting that LaTeX to *try* to put the float at the top of the page. By adding the `!` I am *asking* LaTeX to break the rules and do anything it can to try and force the picture to be at the top of the page (but even this doesn't always work though). If you want to learn more optional placement commands that are available, read this page which has a lot of details:

http://en.wikibooks.org/wiki/LaTeX/Floats,_Figures_and_Captions

There are some drawbacks to using floating figures. If your document has a lot of figures that are very close together with very little text in between, then LaTeX will have difficulty figuring out where to put everything. So if I would have done section 4.3.1 with all floats instead of forcing the pictures, it would not have worked. Also there might be times where you want a sequence of figures one right after the other and you don't want any text between them (like if you are showing still frames of something that is changing over time). This generally will not work well with floating figures. If you want a bunch of picture in a row, especially if you want them to take up a complete page or more, it is best to use fixed position pictures like we had in section 4.3.1.

So forcing the position of a picture can be the best way to go, but LaTeX on it's own will not number any fixed position picture using the methods in section 4.3.1. But fortunately the `caption` package has a solution for this with the alternate function `\captionof`. So if you want to create a figure with a caption but you want to force the positioning this this the method: first, you have to

add the figure inside of some *non-floating* environment like `center` or `flushleft`. Second, after you include the graphic, you use the function:
`\captionof{float type}[list entry]{caption text}`.
Again, the `list entry` is the text that goes in the list of figures and the `caption text` is what goes directly below the figure. The `float type` is needed becaues this caption is not inside of the `figure` environment but LaTeX needs to know which list to put this figure on. So even though this example isn't inside of an actual float, you need to specify the "float type" so that LaTeX knows how to number the caption. The float type should either be "figure" or "table" (unless you use some other custom float type which I won't explain here). Figure 3 below is a mathematical figure that uses this method of forcing a fixed position but with a figure caption. In the example I choose the float type to be `figure` so that the number that is given puts it in the list of figures in the correct order. This is a picture that I drew in Inkscape to demonstrate the concept of projecting a circle from a sphere to a plane via stereographic projection. Here is the code:

```
\begin{center}
\includegraphics[width=.75\columnwidth]{RiemannSphereCircle}
\captionof{figure}[A circle on the Riemann sphere.]{The Riemann sphere
$\mathbb{S}^2$ is shown.  The red rays, etc.}
\label{fig:circle}
\end{center}
```

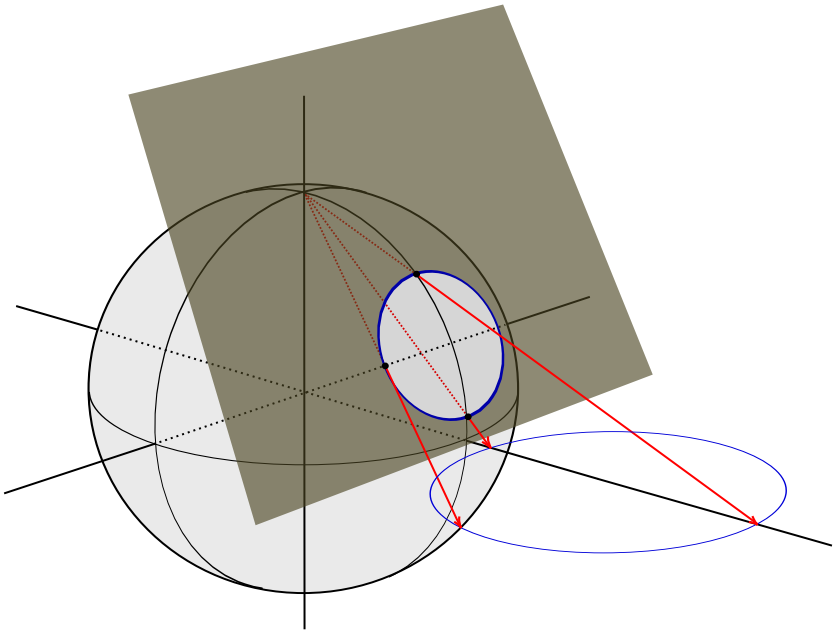


Figure 3: The Riemann sphere $\mathbb{S}^2$ is shown. The red rays, which emanate from the north pole $(0, 0, 1)$, show the process of stereographic projection of points on the sphere to points in the complex plane $\mathbb{C}$. The intersection of a plane with the sphere is a circle *on the sphere* and the projection of this intersection is a standard Euclidean circle *in the plane*.

So this shows you that your figure caption can have math and various script styles as well. Because I created this picture in Inkscape, it is a vector graphics file. I saved the file as a .pdf in Inkscape, which allowed me to easily add it to this document with the `graphicx` package. You should view this document in Adobe reader and zoom in on figure 3. You will find that the lines stay sharp. Then zoom in on the bitmap picture in figure 1 and you will see that the picture degrades as you zoom in.

The last thing I will show you how to do in this section is to add subfigures inside of a figure. But I need to warn you that the methods I use here *might* clash with other packages that you will

want to use. Everything I do here works just fine with the packages that I have chosen for this document though (and most likely these methods will work for you in general). This method uses the `subfigure` environment but some people recommend using the package called `subfig` instead (which I am not using here).[8] O.k., with all of the mild warnings aside here is how it works. You need to add the `subcaption` package to your preamble (which I have done). This package provides you with a new environment `\begin{subfigure}[optional position]{mandatory width}`. This environment allows you to add subfigures inside of a figure. The "optional position" in brackets is not necessary when you use it in the way that I will use it here , and I have never needed it anywhere. But you might someday, so read the documentation on `subcaption`. (See the footnote with the link.) The "mandatory width" is the width of the box that holds the subfigure. To see an example, the result of the following code example can be seen in figure 4. The cool thing about subfigures is that you can give a caption to the subfigures inside of the figure and *also* give a caption to the whole figure. The subfigure captions will create a sub-lettering that will allow you to label and reference the subfigures as well. This is extremely useful when you are comparing multiple figures side-by-side in a mathematical paper. Here is the code for the example and I will explain some of the details below the code block:

```
\begin{figure}[b]
\centering
\begin{subfigure}{0.2\textwidth}
\includegraphics[width=\textwidth]{alice}
\caption{Alice}
\label{subfig:alice}
\end{subfigure}
\quad
\begin{subfigure}{0.35\textwidth}
\includegraphics[width=\textwidth]{teaparty}
\caption{Mad Tea-Party}
\label{subfig:teaparty}
\end{subfigure}
\quad
\begin{subfigure}{0.35\textwidth}
\includegraphics[width=\textwidth]{RiemannSphereCircle}
\caption{The Riemann Sphere}
\label{subfig:sphere}
\end{subfigure}
\caption[A figure with subfigures]{This example shows how to have subfigures
inside of a figure, create captions for all of them, label them, and later
reference them separately. Notice that the code for this caption also creates the
text that you see in the list of figures at the beginning of the document.}
\label{fig:threepics}
\end{figure}
```

Notice that I added the optional `[b]` to the `figure` environment declaration. This is a request to LaTeX to put the figure at the bottom of the page. I actually don't think it is a good idea in this case to force the bottom position; I would prabably not have the optional positioning at all. I just threw it in to show you that you *can do this* if you want, and I wanted to show you the results. Also note that each of the three pictures in this example are added *inside* of their own `subfigure` environment and all of these are nested inside of the `figure` environment. So the entire block of pictures is one floating figure object, but the three pictures will be side-by-side in the same line (once LaTeX decides it's final position). To put a bit of space between the figures I put a `\quad` space

[8] There are conflicting opinions about what is correct here. I think if you use the `subcaption` package you will be just fine. If this is important to you, then you can find out more about these issues at the following sites:
http://en.wikibooks.org/wiki/LaTeX/Floats,_Figures_and_Captions
http://www.ctan.org/pkg/subcaption
http://www.ctan.org/pkg/subfig
http://tex.stackexchange.com/questions/144782/subfigure-and-subfig-packages-deprecated

between each picture. Also note that the width of the first subfigure is 20% of the `\textwidth` and the second two take up 35%. I did this so that the proportions look good next to each other since the first picture is taller than it is wide. But also note that I made the width of the pictures equal to the `\textwidth`. This will ensure that the picture fills up the entire width of the subfigure box. It is important where I placed the captions and labels as well. The caption for the subfigures come right after the lines that include the picture file, and the caption for the entire figure comes right after the last subfigure environment is ended. Then the labels for each of the four captions come right after the corresponding caption. This order is important because you need to make sure you are tying the label to the caption which is creating the numbering for the label, and you want to make sure the caption is placed under the right picture. You will get errors if you change the order. And sometimes I have found that if you have blank lines of code in the environment it will cause problems so avoid having blank lines in your code.

Now that we have created this cool figure and labeled everything we can reference these labels in a variety of ways using `\ref`, `\eqref` (which we covered in part 1 of the introduction) and the new option we have never seen before: `\subref`. We get `\subref` from the `subcaption` package. Here is an example that combines all the different ways you might want to format your references to the figure and subfigures (take note of where I manually added parentheses and where I didn't):

```
Figure \ref{fig:threepics} shows a figure with subfigures.  Figure
\ref{subfig:alice} and \eqref{subfig:teaparty} are bitmaps but figure
\ref{fig:threepics}(\subref{subfig:sphere}) is a vector drawing.  It seems that
\subref{subfig:alice} and \subref{subfig:teaparty} in figure \eqref{fig:threepics}
go together but figure (\subref{subfig:sphere}) doesn't fit in.
```

Here is the result of this code:

Figure 4 shows a figure with subfigures. Figure 4a and (4b) are bitmaps but figure 4(c) is a vector drawing. It seems that a and b in figure (4) go together but figure (c) doesn't fit in.

5 How to make a bibliography with BiBTEX

Another really awesome thing that LaTeX can do for you is create a bibliography for you with a minimal amount of work on your part. You won't have to spend hours fiddling with commas, periods, abbreviations, etc. You won't even have to worry much about the differences between MLA, APA, or Chicago styles (or the many different styles available). You will simply enter in all of the information about your citation into a list of information, then tell LaTeX what style you want, and it will format it all for you. In fact you can do even better than that! If you use Google Books or Google Scholar correctly then you can download all of this info in the correct format instantly, copy paste it to your file and you are done!

(a) Alice (b) Mad Tea-Party (c) The Riemann Sphere

Figure 4: This example shows how to have subfigures inside of a figure, create captions for all of them, label them, and later reference them separately. Notice that the code for this caption also creates the `list entry` text that you see in the list of figures at the beginning of the document.

So, in this section I will show you the basics on how to make a bibliography and how the code needs to be organized, but I won't go over all of the many possible types of citations you might have. Once you know the coding part, I will direct you to online resources that have the details and examples you need.

I should at least mention the various methods for making bibliographies that I *am not* going to explain, just so you know they exist. Then you could go learn them on your own if you want to. It turns out that there are a few different bibliography methods that each have their strengths and weaknesses. One option is to make the citations yourself using the environment called `thebibliography` and the function `\bibitem{}`. This is a direct do-it-yourself approach that is fine if you just have one or two citations. If you want to know more about this see this site:

http://www.math.uiuc.edu/~hildebr/tex/bibliographies.html

or read the section 1 of this wikibook entry:

http://en.wikibooks.org/wiki/LaTeX/Bibliography_Management

Another option is to use the package called `natbib` which used to be popular and many people still use it, but it is no longer supported. I have read that many journals require `natbib` if you want to get a paper published. If you wish to know more about this see section 2.4 here:
http://en.wikibooks.org/wiki/LaTeX/Bibliography_Management

or read the documentation here:
http://www.ctan.org/pkg/natbib

The bibliography method that I will explain here uses BiBTEX, which you can think of as a supplement to LATEX that came bundled with your distribution. You don't need to add any packages to get it to work. It runs on the side of LATEX. BiBTEX, is widely used and has become quite popular. You should know that there is a newer updated replacement of BiBTEX, called BiBLATEX, which may eventually completely replace BiBTEX. I don't know much about it but here is the documentation:

http://www.ctan.org/pkg/biblatex

And there is even one more option that I had never heard of until recently called `biber`. You can find a great discussion comparing the benefits and drawbacks of all four of these different bibliography management systems here:

http://tex.stackexchange.com/questions/25701/bibtex-vs-biber-and-biblatex-vs-natbib

But since BiBTEX is one of the most widely used, I will focus in it only. Once you know how it works you could learn the others easily.

When you use BiBTEX, all of your citation information is stored in a separate file that you need to save with the extension `.bib` instead of `.tex`. Once you create this `.bib` file, you have to save it in the same directory as your `.tex` file. You can give your `.bib` file any name you want. For this demonstration lets call our `.bib` file `myreferences.bib`. If you want to compile this document, you will have to download the `.bib` file that goes with this document.

A `.bib` file is fairly simple. You do not need a preamble of any kind and there are no environments or anything like that. A `.bib` file is just a formatted data base of information. That is it. Here is an example of an entry that I added to `myreferences.bib`. This is a typical book citation. This happens to be a book that I have at my desk as I am typing this. Here is what I added to the `myreferences.bib` file:

```
@book{LeeTopMan,
  title={Introduction to Topological Manifolds},
  author={John M. Lee},
  series={Graduate Texts in Mathematics},
```

```
  year={2010},
  publisher={Springer}
}
```

The beginning of a citation always begins with the `@` character followed by the type of citation. This is a book so that is what I put (there are many options other than book, which I explain below). Then this is followed by a open curly brace, which is followed by the citation key. I chose to make the key `LeeTopMan`. The key is just the code you will use to cite this source in your text (I will show an example of how to do this below). The key has no effect on the citation in your bibliography. The other entries that follow after the key are fairly self-explanatory. You just need to follow the formatting. Make sure to use the equal signs as I did above and the commas at the end are important. Also make sure to have the final closed curly brace at the very end to finish the citation. If you only wanted this one citation in your bibliography, then what you see above is *all you need* in your entire `.bib` file. That's it! Just save it in your working directory and you are done. Then, in your `.tex` file, wherever you want to put your bibliography (usually at the end of your document, right before the `end{document}` line), you need the following two lines of code:

```
\bibliographystyle{style}
\bibliography{filename}
```

In the place of `style`, you need to put the bibliography style that you want to use, and in the place of `filename` you put the name of your `.bib` file that you have saved in your working directory. Here is what I have at the end of this document:

```
\bibliographystyle{plain}
\bibliography{myreferences}
```

The `plain` style is a common simple bibliography style that comes with LaTeX, and, of course, `myreferences` is the name of my `.bib` file (note that you *do not* put the file extension, just the name of the file).

So now I can use the key that I made to cite this source anywhere in my document. So if I quote something from the book, or just want to mention it, I can use the `\cite{keylist}` function. Here is an example:

```
A great source for information about topological manifolds is \cite{LeeTopMan}.
```

This will be the result:
A great source for information about topological manifolds is [3].

Just to show you one more example, here is another entry that adds a journal article which you will find in `myreferences.bib`:

```
@article{ghrist2008barcodes,
  title={Barcodes: the persistent topology of data},
  author={Ghrist, Robert},
  journal={Bulletin of the American Mathematical Society},
  volume={45},
  number={1},
  pages={61--75},
  year={2008}
}
```

Now that we have two entries in our references list, I can show you how to cite two things at once. Sometimes you want to refer the reader to two sources at the same time like this:

```
If you are really into topology, you should read \cite{LeeTopMan,ghrist2008barcodes}.
```

I referenced two citations by typing their both keys separated by a comma. This is the result:

If you are really into topology, you should read [3, 1].

I have had an example of a book and an article, but here is a list of other common entry types that you can have in BiBTEX:

```
@article                    @mastersthesis
@book                       @misc
@booklet                    @phdthesis
@conference                 @proceedings
@inbook                     @techreport
@incollection               @unpublished
@manual
```

Here is a great website that has examples of all of these and it also shows you what it will look like if you choose different bibliography styles as well:
https://verbosus.com/bibtex-style-examples.html

In my examples above I only had a few fields like `author`, `title`, `publisher`, etc. Here is a list of other fields that you can add to your citation:

```
address          howpublished     pages
annote           institution      publisher
author           journal          school
booktitle        key              series
chapter          month            title
crossref         note             type
edition          number           volume
editor           organization     year
```

Here is a good website that explains what each one of these are and whether and when you should use them:
http://www.fb10.uni-bremen.de/anglistik/langpro/bibliographies/jacobsen-bibtex.html

There are just a few more things that you need to know about BiBTEX that are essential. First, you need to know that compiling your references takes a few steps, especially if you are using a more basic LATEXeditor like TEXworks or TEXshop. The first time you compile your document that has references in a `.bib` file, you need to compile the document with the following four steps:

1. Compile your document using pdfLATEX in the normal way you would any document. But it will not completely compile, and it will *seem* like there were errors of some kind. This is normal.

2. Next you have to run BiBTEX. This is usually an alternate compile option that is available in your TEX editor. For example, it is one of the build options in the pull-down menu next to the compile button in TEXworks. When this step is finished ignore any errors or warning messages you might see.

3. Next, compile your code in pdfLATEX again and wait until it finishes, but again, it will show errors and it won't completely compile. But this is normal.

4. Compile your code in pdfLATEX one last time and if everything was coded correctly, you are now done. You should be able to view your document.

This sequence is required because BiBTEX runs separately from LATEX and it has to cross reference across different files. After you complete this sequence one time, you will be able to make changes to your `.tex` file and compile in the normal way. You don't have to repeat this sequence every time you make changes. It is only needed during the first compilation. If you make changes to your `.bib` file you might have to redo this sequence. If you use a more sophisticated LATEX editor like TEXstudio,

then all you have to do is compile once, in the usual way, and TEXstudio will recognize that you are compiling a new `.bib` file and it does the above four-step sequence for you automatically.

By default, the only references that will appear in your table of contents are ones that you explicitly cite in your paper. If you want to have something in your bibliography that you don't explicitly cite you can use the `\nocite{keylist}` function. And, of course, in the place of "`keylist`" you type the citation keys you want to appear separated by a comma. Any citation keys you put in this function will be added to your bibliography. So any source in your `.bib` file that you do not cite in your paper will not be included. As an example, I added `\nocite{complexfunctions}` to my code right before the bibliography declaration and this added the book by Jones and Singerman to my bibliography even though it isn't cited explicitly. If you want *all* entries in your `.bib` file to be in your bibliography, then use an asterisk in the place of your keylist: `\nocite{*}`. The asterisk causes everything to be added. The nice thing about this is that you can collect a whole bunch of citations as you do some research and just keep collecting them into on `.bib` file, and then you can decide which ones you will include at the very end. So you could reuse that same `.bib` file in the future as well (in case you write multiple papers in the same topic area).

Finally, the type of file that contains the code for the bibliography *style* is a `.bst` file. On my PC with MikTEX, these files are saved in the following directory:

```
C:\Program Files (x86)\MiKTeX 2.9\bibtex\bst
```

But this may be different on your operating system and TEX distribution. If you can find where the `.bst` files are saved on your computer, then you can find what bibliography styles are available to you. Here are the different styles that I have on my computer with the standard MikTEX installation:

- abbrv
- acm
- alpha
- ieeetr
- plain
- siam
- unsrt
- amsplain
- amsalpha

The last two are variations of the `plain` and `alpha` styles that have been created by the American Mathematical Society. If you want to get more bibliography styles that you don't have, you need to get the `.bst` file for that style. And you would need to save the file where LATEX can find it. So the easiest way is to save the `.bst` file you want to use in your current working directory. Here is a website from Reed College that has an MLA and APA style if need those:

http://www.reed.edu/cis/help/latex/bibtexstyles.html

References

[1] Robert Ghrist. Barcodes: the persistent topology of data. *Bulletin of the American Mathematical Society*, 45(1):61–75, 2008.

[2] Gareth A. Jones and David Singerman. *Complex Functions: An algebraic and geometric viewpoint*. Cambridge University Press, 1987.

[3] John M. Lee. *Introduction to Topological Manifolds*. Graduate Texts in Mathematics. Springer, 2010.