CSE300: Technical Writing and Presentation
Lecture 1

Nazmus Saquib

September 12, 2016

Contents

1 Introduction

This document contains the topics covered in the first lecture on LaTeX. A few relevant topics which were left out due to the scarcity of time during class have also been discussed here. This document contains some practice exercises. Try these out for self-evaluation of your understanding. Keep in mind that this document is a work in progress - I might update it from time to time.

Section 2 describes what LaTeX is and why we should use it. Section 3 discusses some basic commands of Unix Terminals. If you only want to learn about LaTeX, you can safely skip this Section. Section 4 describes some installation options. Section 5 describes the basic commands and concepts of LaTeX covered in the class.

We are going to use the *LaTeX Wikibook* as our textbook.

2 LaTeX: What and Why

Without getting too technical, we could say LaTeX is simpy a piece of software to generate formatted output. It is a document preparation system, where generally the desired output is a pdf file. In very simple terms, you could think of it as an alternative to Word Processors. To directly quote *latex-project.org*, "LaTeX is a high-quality typesetting system; it includes features designed for the production of technical and scientific documentation. LaTeX is the de facto standard for the communication and publication of scientific documents."

Let's try to figure out why we would want to use LaTeX.

- As apparent from the previous quote, LaTeX has become the standard of technical writing. Most conferences (at least in CS fields) accept only those documents that are written in LaTeX. Even your thesis book has to be written in LaTeX.
- LaTeX especially excels in outputting mathematical and scientific notations. Although you can output these notations in Word Processors, it is really cumbersome to do so and the formatting is not so eloquent. Moreover, when the document is considerably large, it gets really tough to handle edits in Word Processors. They tend to mess up the format after every edit. Besides this there is a huge latency involved in opening such documents.
- LaTeX provides various ways to logically divide our document and refer to those logical parts.
- It is really easy to modularise our code in LaTeX by dividing up the source code in various files.
- As the output is generally a pdf file, it can be run on any machine. We do not generally have to bother about the type or version of the pdf viewer. In case of Word Processors, more often than not this is a genuine headache.

2.1 Do It Yourself

- Can you think of some other reasons why we would want to use LaTeX? Revisit this question after you have finished reading this document.
- In what scenarios we would prefer Word Processors over LaTeX?

3 A Gentle Introduction to Unix Terminal

Before we begin this Section, keep in mind that using a Unix system (Linux/-Mac) is not an absolute necessity. However, in a couple of courses of this term and the next, you will have no option but to use one. I wanted to talk about Unix a bit more in the class, but due to scarcity of time I could not. So let's go through it now.

Personally, I extensively started using Unix (to be specific - Linux, Ubuntu) from Level-3 Term-2. IMHO (and probably all Unix users will agree) it increased my workflow to a great extent. You will find that once you get used to the terminal (think of it like Windows cmd, only better), you are saving up a lot of time. Without being too drammatic, I can honestly say my greatest regret to-date is not starting using Unix earlier. It might seem a bit daunting at first, the learning curve is steep - but the pain is worth the gain.

Apart from speeding up your work, using the terminal will give you clarity and a better understading of your work. Nowadays a lot of tools abstract away so many steps that people don't even know what they are doing. Perhaps it seems to make our life easy, but problems kick in when you have to do some serious tweaking. As an example, in my final year I've seen some of my friends didn't even know what C compiler they were using. When asked they used to reply with their IDE name. During one particular course we needed to use an external library with our compiler. It was bit of a hassle (not impossible - but a hassle nonetheless) to do so with different IDEs; but a total breeze to do from terminal/command prompt. Some of my friends using Windows failed to work with their IDEs. When they came to me I simply found the location of their compiler, added the location of the binary to system variables, wrote a batch script with compiler commands to use the said library, and showed them to invoke it from cmd. Pretty basic steps, should be a piece of cake for a CS student ... but they seemed to be at total awe at this.

You might think that, okay, so let's just use the Windows cmd. You could, but there are some problems. (There are workarounds, but still, that's just extra work. Disclaimer - I haven't used any version of Windows after 7, so I don't really know the current condition.) It does not provide some basic functionalities like maximizing cmd, adding new tabs etc. Most Unix terminal tools have a Windows cmd counterpart, but not all commands are covered. Even among the ones covered, in regards to ease of use and availability of instructions, Unix commands are way ahead.

Anyways, that was a desperate attempt at convincing you to use Unix. Now let's get to the real deal. All the following discussions are made with respect

to Ubuntu, Linux. But most of these commands are pretty standard ones, so you should not face any problem. Note that I'm going to cover only a few basic commands required to get you up and running with LaTeX using Linux terminal. Checkout Section 3.7 for some user-friendly Unix guides. If you want to learn more about Unix, these can be some pretty good resources.

3.1 Launching and Exiting the Terminal

There are three ways to launch the terminal:

1. From dash home. This is the icon at the top left (the whole left panel is known as dash). If you click it, you should see a search bar. Just type in terminal, and hit enter.

2. Shortcut in dash. There should be a terminal icon, just click on it. This shortcut might not be present by default.

3. Just type *Ctrl + Alt + T*.

Now that we know how to launch our terminal, let's check out how we can exit out of it. One obvious way is to simply click on the cross button at top left. Another way is through the *exit* command. So just type in 'exit' in your terminal, and hit enter.

3.2 Unix Direcotry Structure

The directory structure of Unix resembles a tree, where the root of the tree is '/' (yup, that's just a forward slash). As you can imagine, all the files and directories can be expressed with respect to this root directory for its absolute path. For example, when you open up a terminal window or the file manager, you are in the *home* folder. The absolute path of this home folder is */home/<username>*. Here *username* in most cases is the user name that you provided during installation. Throughout this document I will use angle brackets (<, >) to denote variable values. In my machine the home directory is */home/saquib2527*. There is a cool way to find out the absolute path of your home folder. Just type in *echo ~* in your terminal and hit enter (that's the *tilde* character after 'echo', just below your `Esc` key).

We can either use absolute paths as discussed above in our commands, or we can use relative paths as discussed in Section 3.4.

3.3 List all Files and Directories (*ls* or *l*)

To view a list of all files and directories in the current directory, we use the *ls* command. Just to gain some confidence that this is actually what's happening, let's perform the following steps:

1. Open up terminal and file manager.

2. Invoke the *ls* command.

3. match the output of the command with the files/directories shown in the file manager. You should see they are the same set of files/directories.

Now that we are pretty confident the command does what it's supposed to do, let's go to the next level. Instead of invoking only 'ls', let's add a directory name at the end of it, say *Desktop*. So now our command is *ls Desktop*. It should list the files/directories in *Desktop*.

As a side note, to speed up our work, here are a few tips:

- Try to use *auto-complete* feature. When you are in *home* folder, you can simply type 'De' and hit `Tab`, the word *Desktop* will be filled in automatically. Here's a question for you - why doesn't it auto-complete when I type 'D' only?

- You can cycle through your commands using `up` and `down` arrow keys in terminal. This way you don't have to type in the full command when you want to invoke any of the previous ones.

- Use *Alt + Tab* to cycle through opened windows. This way you don't have to reach out for the mouse.

- [**advanced**] Try to use a terminal based text editor like *vim*, this will also speed up your work. Bottom line, if you can get rid of the mouse, your work will be faster.

3.4 Change Directory (*cd*)

To **c**hange **d**irectory the command to invoke is *cd <dst>*. Here *dst* is our destination directory. Assuming we are in the *home* folder and there is a folder named *latex* in *Desktop*, we can change our directory to *latex* by invoking *cd Desktop/latex*. Notice that the path we are providing is a *relative* one with respect to the current directory, i.e. we are saying there is a directory named *Desktop* inside the current directory, which again contains the directory *latex*. This is same as invoking the command in my machine this way - *cd /home/saquib2527/Desktop/latex* using absolute path. There are two special directories which comes handy while moving around:

- **.** a single dot represents the *current directory*.

- **..** double dots represent the *parent directory*.

Just to make the concept a bit more concrete, the following commands all do the same thing (assuming I'm in *home* folder):

- *cd Desktop/latex*

- *cd ./Desktop/latex* - notice the '**.**' at the beginning of the path. If you omit the dot it would represent a totally different path (why?).
- *cd Desktop/../Desktop/latex*

As a side note, if you invoke *cd* without any parameter, it will take you to the *home* folder.

3.5 Make a New Directory (*mkdir*)

To **m**a**k**e a new **dir**ectory we invoke the *mkdir* command with the name of new directory: *mkdir <dirname>*. As we will see later, when we compile a LaTeX source code, a few extra files get generated. To keep the working directory clean and organized, I always create a new directory and write the source code there. It is not a necessity, but certainly a good practice.

3.6 Do It Yourself

Try to find out the answers to the following questions.

- How can we remove a file/directory?
- How can we copy files/directories?
- How can we rename a file/directory?
- What does the *touch* command do?
- What does the *clear* command do?
- What does *sudo* mean?
- What are manpages?
- What is *apt-get*?
- What are Gnome, Unity, and KDE?
- How can we add tabs to our terminal using keyboard?
- How can we switch among the tabs of terminals using keyboard?

3.7 References

That was a very brief introduction to some Unix commands. For more information you can check out the following resources:

- `https://www.cs.sfu.ca/~ggbaker/reference/unix/unix1.html`
- *Beginning Linux Programming* by Neil Matthew and Richard Stones

- *Advanced UNIX: A Programmer's Guide* by Stephen Prata
- *Unix Power Tools* by Jerry Peek, Shelley Powers, Tim O'Riley, and Mike Loukides
- *Linux Bible* by Christopher Negus and Christine Bresnahan

Just start at the top and gradually make your way to the bottom. Feel free to search the net for *linux beginners* and see if you find anything to your liking.

4 Installation

If you are using Windows, simply search for *miktex*. Download the binaries, install and you are good to go. Apart from the LaTeX compiler, this will also provide you with an IDE.

If you are using Linux, the package you want to install is *texlive* (or *texlive-full*). To install the package simply invoke `sudo apt-get install texlive` (assuming you are using Ubuntu). If this command seems a bit alien, try to find the answers in Section 3.6 first.

I don't have much experience with Mac OS, but if you are using one, check out *MacTeX*.

Apart from these you have some online options, like *ShareLaTeX* and *Overleaf*. But it is highly recommended that you install LaTeX in your local machine.

5 Basic Commands

Okay, so now we can start with the fun part. I'm assuming you've installed LaTeX by now. Let's create a directory. Change your terminal to the newly created directory. Inside this directory create a file named *intro.tex*. Open this up with the default text editor (*gEdit*). This is the file where we are going to write our code. Technically speaking, the extension does not have to be *tex*, but text editors will give us syntax highlighting if we do so. Now that we are done with basic setup, let's take a look at the minimal structure of a LaTeX source file.

5.1 Minimal Source Code

Type in the following code in *intro.tex* file.

```
1 \documentclass{article}
2 %preamble
3 \begin{document}
4 Hello World!
5 \end{document}
```

Let's try to understand what we just did.

Anything that starts with '\' in LaTeX is a command. So in line 1 we are invoking a command, whose name is *documentclass*. Commands can take zero or more arguments, each argument has to be within a pair of curly braces. So the *documentclass* command takes one argument, in this case *article*. The command is quite self-explanatory, it defines the class (type) of our document. Documents can be of various types depending on our requirement. For example, to write a book we use *book* or *memoir* class, to write a document which is not as large as a book but still pretty large we use *report* class, for short documents we use the *article* class, etc. Check out the *LaTeX Wikibook* for details.

Line 2 is commented out, we use '%' to comment out texts in LaTeX. Whatever comes after a % in a line is considered to be commented out.

Line 3 and 5 also contain commands. As you can see, these two are a bit different kind of command - together they delineate the beginning and end of an *environment*. We are going to take a good look at *environments* later on. But just for now, keep in mind that the *begin* and *end* commands represent an *environment*. In this case the name of the environment is *document* i.e. the paramemter of the two commands is *document*.

In line 4 (inside the *document* environment) we have a single statement - 'Hello World!'. Our generated pdf file should have this line only. In other words, whatever is inside our *document* environment is our output.

One LaTeX terminology that we should know about is the *preamble*. It is the space between *documentclass* and *\begin{document}*.

5.2 Compilation

Before compiling make sure you have saved your source code. To compile the LaTeX source code we need to invoke *pdflatex <sourcefile>*, in our case *pdflatex intro.tex*. A successful compilation should create three files - *intro.log*, *intro.aux*, and *intro.pdf*. If you see your compiler got stuck, first read the error message. Most of the time the message will contain the line number in the form of *l.x*, here *x* is the line number. Then to exit out of this 'stuck' state press *Ctrl + D*. Try to fix the problem in your source code and compile again. If compilation is successful you should see a bunch of lines, where the last line is "Transcript written on lecture.log".

Let's take a look at the generated files. The pdf file is obviously our generated output. The log file (as is apparent by its name) simply contains the log messages of the compiler. Basically the bunch of texts output during compilation are present in this file, along with a few other information. The interesting file is the aux file (read auxilliary file). The LaTeX compiler happens to be a very simple compiler. In certain scenarios, it needs to perform multiple compilation passes over the source code to generate the final output. If information needs to be conveyed among such successive compilation passes, the compiler uses this file. In general, we would not need to look into either the log file or the aux file.

To open up the generated pdf file we can simply invoke *xdg-open intro.pdf*.

xdg-open is kind of the Swiss-knife of opening stuff - it's going to open a file with it's default program. So if it were a html file instead of a pdf file, it would have been opened in the default browser. In this case, the pdf file will be opened in the default pdf viewer. You should see only a single line in the pdf document - "Hello World!".

5.2.1 Do It Yourself

Make some changes inside the document environment, compile it, and verify the changes.

5.3 Creating a Title

Right now our document is pretty bland, the only thing that we have is a single line saying hello world. Let's add a title to our document. At the beginning of our document environment add the following lines.

```
\title{Introduction to \LaTeX}
\author{Name}
\date{\today}
\maketitle
```

So we can see five new commands:

1. *title* accepts a single argument, the title.
2. *author* accepts a single argument, the author name.
3. *date* accepts a single argument, the date.
4. *today* does not accept any argument, represents today's date.
5. *maketitle* does not accept any argument, must be invoked for the title to be visible.

5.3.1 Do It Yourself

Try to answer the following:

- What happens when we don't supply any argument to *date*?
- What happens when we remove *date*?
- Try to change the order of *title*, *author*, and *date*. Do you see any changes?
- There is in fact a sixth new command - *LaTeX*. Take a look at its output.

5.4 Spaces, Newline, Paragraph, and New Page

LaTeX totally disregards consecutive spaces and converts them to a single space. So if you write:

```
1 This is a short sentence.                This line is not so far.
```

The output will simply be "This is a short sentence. This line is not so far." So obviously the question is what if for some reason we want to output consecutive spaces? It turns out we can simply use ~ (tilde) in place of spaces to make that happen.

You will also notice if you hit *return*, this is also considered as a single space. To actually output a newline we have two options:

- double backslashes \\, and
- the parameterless *newline* command.

To insert a page break we can simply invoke the parameterless *newpage* command.

If you press *return* twice or more, you will create a new *paragraph*. The difference between newline and paragraph is that newline starts right from the beginning of the line, whereas paragraph has a bit of an indentation.

5.5 Quotation Marks and Special Characters

Generally to output quotation marks we simply type in ' for single quote, and " for double quote. In LaTeX we can also use these for ending quotation marks, but for starting quotation marks we have to use ` (backquote - the key just below *Esc* key) - once for single quote, twice for double quote.

As we've seen before, % is used to comment out anything that comes after it. So how do we output a % sign? We simply precede it with a '\'. Basically we have to write \% - that's all. This is also true for a few other special characters.

If we precede a '\' with a '\' that's just two backslashes. As we've learnt before this represents a newline. To actually output a '\' we have to invoke the parameterless command *textbackslash*.

5.5.1 Do It Yourself

Find out other special characters that can be output by preceding them with a '\'.

5.6 Basic Text Formatting

If we want to output bold text we can invoke the *textbf* command. It accepts a single argument - the text that we want to make bold.

To output italicized text we invoke the *textit* command. It also accepts a single argument - the text that we want to make italicized.

There is in fact another command to make text italicized - the *emph* command. Well, technically, it is used sometimes to make text italicized; sometimes to make it upright. *emph* is kind of a dynamic command - its output varies depending on its surroundings. If *emph* is invoked nested inside of a first level *emph* or a *textit* command, it will make the text upright. In short, it will simply make the text *stand out* compared to its surroundings. If required it will make the text upright, if required it will make the text italic.

5.6.1 Do It Yourself

Play around with *textbf*, *textit*, and *emph*. Make sure you understand the output of *emph*.

5.7 Groups and Group-Level Commands

Groups create *scopes*. To elaborate, we can invoke group level commands which affect the whole group starting from the point of invocation. As an analogy think what happens when we define a local variable in a function. It gets a local scope - it can only be referenced within that function. Similarly, we can create a *group* in LaTeX, and invoke certain commands which only influences that group. We can create a group through curly braces - the starting curly brace delineates the starting of the group, the ending curly brace delineates the ending of the group. All the commands discussed in Section 5.6 have corresponding group level commands. The group level commands of *textbf*, *textit*, and *emph* are *bfseries*, *itshape*, and *em* respectively. Let's see an example.

```
This is normal text.
{
This is also normal text.
\bfseries
But this is bold.
}
This is again normal text.
```

5.7.1 Do It Yourself

Take a look at the code in Section 5.7. Make sure you understand the output. Try out the other group-level commands one after another and see what happens. First try to figure out what the output should be, then take a look at the generated pdf file.

5.8 Lists

There are three types of lists in LaTeX:

1. Ordered lists. The items in ordered lists are numbered. The current list is an ordered list. We use the *enumerate* environment to output ordered list.
2. Unordered lists. These are bullet-point lists. We use the *itemize* environment to output unordered list.
3. Description lists. These are used to define some term. The term is in bold font whereas the definition is in normal font. We use the *description* environment to output description list.

In each of the above cases, items are preceded by the *item* command. The following code gives example of each of the types.

```
\begin{enumerate} %ordered list
   \item CSE300
   \item CSE304
\end{enumerate}

\begin{itemize} %unordered list
   \item CSE300
   \item CSE304
\end{itemize}

\begin{description} %description list
   \item[CSE300] Technical Writing and Presentation
   \item[CSE304] Database Sessional
\end{description}
```

5.9 Logical Partitioning of Document

When our document gets a bit large, it's a good idea to divide it up into some logical partitions. Take this document as an example. Specifically look at the table of contents. It has half a dozen Sections (numbered 1, 2, 3, ...). Some of the Sections have Subsections (numbered 2.1, 3.1, 3.2, ...). Some of the Subsections even have Subsubsections (numbered 5.2.1, 5.3.1, ...). The way to divide up the document is pretty simple:

- Invoke *section* command to start a Section. It takes the name of the Section as the parameter.
- Invoke *subsection* command to start a Subsection. It takes the name of the Subsection as the parameter.
- Invoke *subsubsection* command to start a Subsubection. It takes the name of the Subsubsection as the parameter.

We don't have to explicitly state the end of the partitions, they simply continue until another partition is observed.

On a relevant note, it is really easy to generate table of contents - simply invoke the parameterless command *tableofcontents*. This is a command that required two compilation passes.

5.9.1 Do It Yourself

Write a document with a bunch of Sections, Subsections, and Subsubsections. Try to guess the numbers attached to them before taking a look at the generated pdf file.

5.10 Cross-Referencing

Notice how throughout this document I've been referring to various Sections with their numbers? As it turns out we don't have to hard code the numbers, LaTeX has a cool feature through which we can do it. It's generally known as *cross-referencing*, where two commands are involved - *label* and *ref*. It's a two-step process:

1. First we give the Section (it can also be Subsection, Subsubsection etc.) a *name*, i.e. we *label* it by using the *label* command. This *name* can be anything, but generally it is started with *sec:* for a Section.
2. Then we *refer* to that Section by the given *label* using the the *ref* command.

Here's an example of cross-referencing:

```
We talk about cross-referencing in Section \ref{sec:cr}.

\section{Cross Referencing}\label{sec:cr}
Cross-referencing is pretty awesome!
```

On a relevant note, cross-referencing requires two compilations passes to properly output the document.

5.10.1 Do It Yourself

Find out the problem with hard coding Section numbers while referring to them.

5.11 Using Packages

Just as we can include header files in C/C++, import packages in Java, import modules in Python etc. - we can *use packages* to utilize already defined commands. The place where we do that is the *preamble*. We have to invoke the command *usepackage* which accepts a single argument - the name of the desired package. For example, if we want to output colored text we need the

color package. so in preamble we have to invoke *usepackage{color}*. This will give us access to the *textcolor* command, which takes two arguments - the color and the text. Assuming we want to output 'CSE300' in red, we have to invoke *textcolor{red}{CSE300}*.

5.12 Custom Command

It is really easy to create custom commands in LaTeX. The motivation of using commands in LaTeX is similar to that of using functions in programming languages: *reusability* and *abstraction*. Suppose we have a bunch of steps that we have to perform again and again. Instead of copy pasting them repeatedly, we can simply encapsulate them in a command; and invoke that command when required. This has the added benefit that if we feel like making any changes to our steps, we can simply change the command definition instead of going through the whole code. Moreover, custom commands create an asbtraction to the end users - they don't have to know what's happening inside of the command (well, unless they are thinking of changing that command's functionality). All that the users need to know is the signature of the command.

Now that we've established why we need custom commands, let's take a look at how to create one. We create a custom command with the help of a built-in command named *newcommand*. This command takes at least two parameters - the name of the custom command and its definition. If our custom command takes any argument, we have to supply an optional argument denoting the number of arguments it takes. This optional argument is passed within square brackets, between the name and the definition. Within our definition we can refer to the arguments using *#i*, where *i* denotes the order of the argument i.e. 1 for first argument, 2 for second argument, etc. If our command does not take any argument, we leave out the square brackets. Generally custom commands are defined right at the beginning of the document environment.

Let's create two custom functions. I'm leaving it up to you to find out what they do.

```
1 \newcommand{\hello}{Hello World!}
2 \newcommand{\greetings}[1]{Hello #1!}
```

On a relevant note, can you guess where I used a custom command in this document?

6 Conclusion

Make sure you install LaTeX in your machine if you haven't done so already. Try out the commands and practice problems discussed here. I would strongly suggest that you use a Unix system and exploit the terminal extensively. In case you have any queries, feel free to contact me at *saquib2527@gmail.com*.

Time for one last problem - generate this document using LaTeX ☺.