LaTeX

by Wikibooks contributors

From **Wikibooks**,
the open-content textbooks collection

Principal authors: Withinfocus · Jtwdog · Orderud · Igjimh · Alejo2083 · Cameronc · Snaxe920 · Basenga

This book is published by Wikibooks contributors.

The current version of this Wikibook may be found at:
http://en.wikibooks.org/wiki/LaTeX

Contents

1 Introduction

What is TeX

TeX (pronounced "Tech") is mainly a low level programming language aimed to typesetting documents. It is very powerful but, since you have to take care of everything, it is difficult to use it for long documents and time-consuming.

What is LaTeX

LaTeX (pronounced either "Lah-tech" or "Lay-tech") is a macro package created by Leslie Lamport based on the TeX typesetting language of Don Knuth. Its purpose is to produce professional looking and correctly typeset documents, in particular those with mathematical formulae. It is currently maintained by the LaTeX3 project. Numerous authors have contributed extensions, called *packages* or *styles*, to LaTeX. A number of these is usually bundled with a TeX/LaTeX software distribution or can be found in the Comprehensive TeX Archive Network (CTAN).

Since LaTeX comprises a group of TeX commands, LaTeX document processing follows a programming perspective. One creates a text file in LaTeX markup, which is then to be read by the LaTeX macro, which produces the final document.

Obviously, this has its disadvantages, compared with a WYSIWYG (What You See Is What You Get) program such as Openoffice.org Writer or Microsoft Word.

- One can't see the final result straight away.
- One needs to know the necessary commands for LaTeX markup.
- It can sometimes be difficult to obtain a certain 'look'.

On the other hand, there are certain advantages to the markup language approach:

- The layout, fonts, tables, etc. is consistent throughout.
- Mathematical formulae can be easily typeset.
- Indexes, footnotes, references, etc., are generated easily.
- It encourages correctly structured documents.
- It is available (and transportable) across many platforms (Windows, Mac, Unix and Linux are all supported).

The LaTeX-like approach can be called WYSIWYM, i.e. What You See Is What You Mean: you can't see how the final version will look like while typing, but you can see only the logical structure of the document, LaTeX will take care of the formatting for you.

The LaTeX document is a plain text file containing the content of the document, with additional markup. When the source file is processed by the macro package, it can produce documents in several formats such as DVI, PDF or PostScript.

Prerequisites

At a minimum, you'll need the following programs to edit LaTeX:

- An editor (You can use a basic text editor like notepad, but a dedicated LaTeX editor will be more useful).
  - On Windows, TeXnicCenter([1]) is a popular free and open source LaTeX editor.
  - On *nix (including Mac OS X) systems, Emacsen and gvim provide special TeX enviroments.
- The LaTeX binaries and style sheets - e.g. MiKTeX [2] for Windows, teTeX [3] for Linux and teTeX for Mac OS X [4].
- A DVI viewer to view and print the final result. Usually, a DVI viewer is included in the editor or is available with the binary distribution.

A distribution of LaTeX, with many packages, add-ins, editors and viewers for Unix, Linux, Mac and Windows can be obtained from the TeX users group at http://www.tug.org/texlive/.

Applications within a distribution

Here are the main programs you expect to find in any (La)TeX distribution:

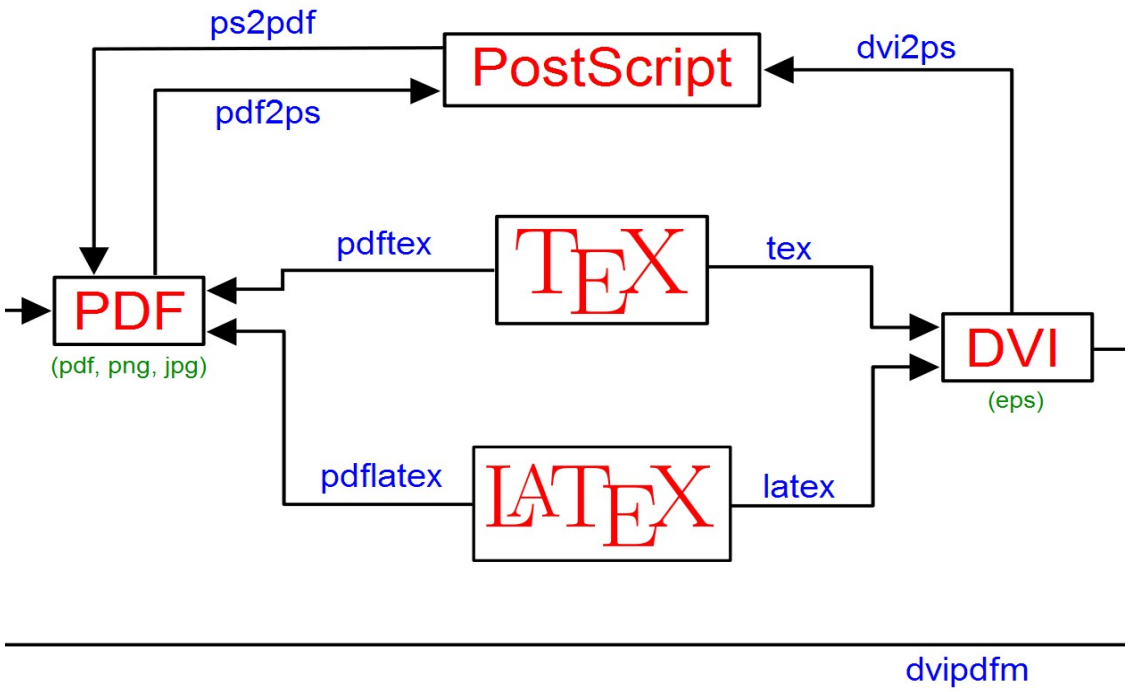
- **tex**: the simplest compiler, it gets a TeX file and creates DVI
- **pdftex**: it gets a TeX file, but creates a PDF file
- **latex**: the most used one: it gets a LaTeX file and creates a DVI
- **pdflatex**: from a LaTeX creates a PDF
- **dvi2ps**: converts the DVI file to PostScript
- **dvipdfm**: converts the DVI file to PDF

When LaTeX was created, the only format it could create was DVI; then the PDF support was added by *pdflatex*, even if several people still don't use it. As it is clear from this short list, PDF files can be created with both *pdflatex* and *dvipdfm*; anyway, the output of *pdflatex* is much better than the other. DVI is an old format, and it does not support hyperlinks for example, while PDF does, so passing through DVI you will bring all the bad points of that format to PDF. Moreover the general output will be better using only *pdflatex*.

Strictly speaking, the document you are writing should be sightly different according to the compiler you are using (*latex* or *pdflatex*), but as we will see later, it is possible to add a sort of abstraction layer so to ignore what you are using, the applications will do everything by themselves.

Note that, since LaTeX is just a collection of macros for TeX, if you compile a plain TeX document with a LaTeX compiler (such as *pdflatex*) it will work, while the opposite is not true: if you try to compile a LaTeX source with a TeX compiler you will get only a lot of errors.

The following diagram shows the relationships between the (La)TeX source code and all the formats you can create from it:



The boxed red text represents the file formats, the blue text on the arrows represents the commands you have to use, the small dark green text under the boxes represents the image formats that are supported. Anytime you pass through an arrow you loose some information, that might decrease the quality of your document. This means that you have to choose the shortest way to reach the final format, that means achieving the best quality. Starting from a LaTeX source, the best way is to use only *latex* for a DVI output or *pdflatex* for a PDF output, converting to PostScript only when it is necessary to do it to print the document.

Most of the programs should be already within your LaTeX distribution, the others come with Ghostscript, that is a free and multi-platform software as well.

2 Absolute Beginners

This tutorial is aimed at getting familiar with the bare bones of Latex. We will begin with creating the actual source Latex file, and then take you through how to feed this through the Latex system to produce quality output, such as postscript or PDF.

The Latex source

The first thing you need to be aware of is that Latex uses a markup language in order to describe document structure and presentation. What Latex does is to convert your source text, combined with the markup, into a high quality document. For the purpose of analogy, web pages work in a similar way: the HTML is used to describe the document, but it is your browser that presents it in its full glory - with different colours, fonts, sizes, etc.

OK, so let us begin by deciding what we will actually get Latex to produce. As I said, we will produce the absolute bare minimum that is needed in order to get some output, and so I think the well known **Hello World!** approach will be suitable here.

1. Open your favourite text-editor. If you use vim or emacs, they also have syntax highlighting that will help to write your files.
2. Reproduce the following text in your editor. This is the Latex source.

```
% hello.tex - Our first Latex example!
\documentclass{article}
\begin{document}
Hello World!
\end{document}
```

1. Save your file as **hello.tex**.

What does it all mean?

`% hello.tex - Our first Latex example!`	The first line is a *comment*. This is because it begins with the percent symbol (%); when Latex sees this, it simply ignores the rest of the line. Comments are useful for humans to annotate parts of the source file. For example, you could put information about the author and the date, or whatever you wish.
`\documentclass{article}`	This line tells Latex to use the article document class. A document class file defines the formatting, which in this case is a generic article format. The handy thing is that if you want change the appearance of your document, substitute article for another class file that exists.
`\begin{document}`	An educated guess would tell you that this command alerts Latex that content of the document is about to commence. Anything above this command is known generally to belong in the *preamble*.
`Hello World!`	This was the only actual line containing real content - the text that we

	wanted displayed on the page.
`\end{document}`	Once again, this is not too difficult to understand. It tells Latex that the document source is complete.

You should also notice that each of the Latex commands begin with a backslash (\). This is Latex's way of knowing that whenever it sees a backslash, to expect some commands. Comments are not classed as a command, since all they tell Latex is to ignore the line. Comments never affect the output of the document.

Note, if you want to use the backslash or percent symbols within your text, you need to actually issue a command to tell Latex to draw the desired symbols, otherwise it will expect a command or a comment! The commands are:

Symbol	Command
%	\%
\	\textbackslash

If you have a string with many of these characters, such as a URI, an easier way of quoting is this: **\verb+A % or \ in here is interpreted as a literal.+** This style of quoting is usually easier to read for URIs.

Generating the document

It is clearly not going to be the most exciting document you have ever seen, but we want to see it nonetheless. I am assuming that you are at a command prompt, already in the directory where *hello.tex* is stored.

1. Type the command: *latex hello* (the .tex extension is not required, although you can include it if you wish)
2. Various bits of info about latex and its progress will be displayed. If all went well, the last two lines displayed in the console will be:

```
Output written on hello.dvi (1 page, 232 bytes).
Transcript written on hello.log.
```

This means that your source file has been processed and the resulting document is called *hello.dvi*, which takes up 1 page and 232 bytes of space.

Note, in this instance, due to the simplicity of the file, you only need to run the Latex command once. However, if you begin to create complex documents, including bibliographies and cross-references, etc, Latex needs to be executed multiple times to resolve the references. But this will be discussed in the future when it comes up.

Viewing the document

Latex has now done its job, so we can view the output. The default format is DVI (device independent), of which viewers exist freely for most platforms. However, the chances are that you

would prefer to have a postscript file or PDF. Fortunately, there exist tools that can convert DVI to PS (and PDF) easily.

Converting to Postscript

Type the command: *dvips hello.dvi -o hello.ps*

dvips is the utility that actually performs the conversion. The first argument is the DVI file to be converted. The *-o* argument says that you want the output to be saved as a file. And the argument immediately after is the name you wish to call it. You could give it any name, but it makes sense to stick with *hello*, as well as giving it an informative *.ps* extension.

Converting to PDF (bad way)

here are two easy routes to get a PDF:

1. Type the command: *dvipdf hello.dvi hello.pdf* (Note that there is no *-o* with this command, because although the utilities look almost identical, they have slightly differing syntax)
2. If you already have a postscript version, then type: *ps2pdf hello.ps hello.pdf*

Now it is simply a matter of using your preferred PS or PDF viewer to see the output. You should see the words *Hello World!* at the top left of the page and the current page number at the bottom, all in a standard times font. These two ways to get the PDF are deprecated and should be used only when there are no other possibilities.

Getting the PDF

Just type:

```
pdflatex hello.tex
```

where *hello.tex* is *exactly* the same file you have previously used. You will get *hello.pdf* and you can view it with any PDF-viewer. Since the document is still very simple, you should not see any difference between the one that was created from the DVI and this one.

Summary

OK, we've created possibly the simplest possible document that Latex will produce (except for a blank page of course!) which is why it is not much to look at. However, now we have seen the basics, and how to actually use the Latex software, we can progress towards the more typical documents that you are likely to produce.

This chapter uses material from Andy Roberts' Getting to grips with Latex with permission from the author.

3 BASICS

The first document

Okay, assuming LaTeX is installed and ready to run, open the editor and type in:

```
\documentclass{article}

\begin{document}
Hello World!
\end{document}
```

Now save it (test.latex), and run it in LaTeX. This should create a DVI file (test.dvi) which can be opened in (you guessed it) a DVI Viewer.

```
latex test.latex
```

Open it and have a look at your first LaTeX document. This will output a document with the words "Hello World!".

Let's examine what has happened:

- Every LaTeX document starts with a **\documentclass{...}** command, which specifies the class of the document. In this case the class is **article**, which means the formatting is set up for a short document, similar to a journal article.
- The **\begin{...}** and **\end{...}** commands specify that anything appearing in between are part of an *environment*. In this case the environment is **document**. Only commands which appear in the **document** environment will appear in the body of the document.
- The space between the **\documentclass{...}** and **\begin{document}** commands is known as the *header*. Commands which appear in here are related to the setup of the document, such as loading add-in packages and setting new commands and environments.

Text

See the subsequent chapter on document structure.

Now we shall show you another example of an environment: a bulleted list.

```
\documentclass{article}

\begin{document}
Hello World!
\begin{itemize}
\item One point
\item deserves another
\end{itemize}
\end{document}
```

We now have the "Hello World" as before, but now below it we have two bulleted points.

Mathematical symbols

See the subsequent chapter on mathematics.

One of the main reasons LaTeX is so powerful is due to the ease with which it handles mathematical symbols, so lets try that now:

```
\documentclass{article}

\begin{document}
Hello World!
\[a^2 + b^2 = c^2 \]
\end{document}
```

This will output a document with $a^2 + b^2 = c^2$.

4 PACKAGES

A great number of additional packages are useful in producing rich LaTeX environments. To use the package "foo," simply include the command

```
\usepackage{foo}
```

in the header of your document. These include

1. Packages improving typing mathematics:
    - amsmath
2. Packages for including new mathematical symbols:
    - amssymb
    - stmaryrd
3. Packages for including commutative diagrams:
    - amscd
    - xypic
4. Packages improving theorem-like environments:
    - theorem
    - ntheorem
    - amsthm
5. Packages for embedding graphics:
    - graphics
    - graphicx
6. Packages providing hyperlnks:
    - hyperref

5 Document Structure

This tutorial progresses significantly from the previous - very simplistic - tutorial. The goal is to produce a fairly basic article, of similar style to what a research paper would resemble. To achieve this efficiently, this tutorial focuses largely on *document structure*.

Latex practically forces you to declare structure within your documents. This is a good thing though. Because once Latex understands how you want your document organised, it will take care of all the tedious business of the layout and presentation for you. The separation of content and layout allows you to concentrate on the job at hand, i.e., communicating your research.

Before I begin to explain how to use various Latex commands, I believe it would be beneficial to see what the end result looks like first, so that we know from the beginning what the effects of our actions will be. Take a look at the [tutorial2/simple.pdf output of this tutorial]. Also, the Latex source is quite long, therefore, I shall only be using extracts throughout the tutorial. The source file is available [tutorial2/simple.tex here] and at the end of this page.

Preamble

If you recall from the previous tutorial, the preamble is everything from the start of the Latex source file until the `\begin{document}` command. It normally contains commands that affect the entire document.

```
% simple.tex - A simple article to illustrate document structure.

\documentclass{article}
\usepackage{mathptmx}

\begin{document}
```

The first line is a comment (as denoted by the % sign). The `\documentclass` command takes an argument, which in this case is article, because that's the type of document we want to produce. Other classes that exist are book, report, thesis etc. It is also possible to create your own, as is often done by journal publishers, who simply provide you with their own class file, which tells Latex how to format your content. But we'll be happy with the standard article class for now! `\usepackage` is an important command that tells Latex to utilise some external macros. In this instance, I specified `mathptmx` which means Latex will use the Postscript Times type 1 font instead of the default ComputerModern font. And finally, the `\begin{document}`. This strictly isn't part of the preamble, but I'll put it here anyway, as it implies the end of the preamble by nature of stating that the document is now starting.

Document classes

Every latex document has a certain *document class* in the preample. The class is given by the `\documentclass{...}` command. Valid LaTeX document classes include:

- article

- report
- letter
- book
- slides

Top Matter

At the beginning of most documents will be information about the document itself, such as the title and date, and also information about the authors, such as name, address, email etc. All of this type of information within Latex is collectively referred to as *top matter*. Although never explicitly specified (there is no `\topmatter` command) you are likely to encounter the term within Latex documentation.

An example:

```
\title{How to Structure a \LaTeX{} Document}
\author{Andrew Roberts\\
  School of Computing,\\
  University of Leeds,\\
  Leeds,\\
  United Kingdom,\\
  LS2 1HE\\
  \texttt{andyr@comp.leeds.ac.uk}}
\date{\today}
\maketitle
```

The `\title` command is fairly obvious. Simply put the title you want between the curly braces. `\author` would also seem easy, until you notice that I've crammed in all sorts of other information along with the name. This is merely a common, albeit, ungraceful hack, due to the basic article class. If you are provided with a class file from a publisher, or if you use the AMS article class (`amsart`), then you have a more logical approach to entering author information. In the meantime, you can see how the new line command (`\\`) has been used so that I could produce my address. My email address is at the end, and the `\texttt` commands formats the email address using a mono-spaced font. The `\date` command takes an argument to signify the date the document was written. I've used a built-in command called `\today` which, when processed by Latex, will be replaced with the current date. But you are free to put whatever you want as a date, in no set order. If braces are left empty, then the date is then omitted. Without `\maketitle`, the top matter would not appear in the document, so it is needed to commit your article attributes to paper.

Abstract

As most research papers have an abstract, then there is a predefined commands for telling Latex which part of the content makes up the abstract. This should appear in its logical order, therefore, after the top matter, but before the main sections of the body. This command is available for the document class *report*, but not *book*.

```
\begin{abstract}
Your abstract goes here...
```

```
...
\end{abstract}
```

Sectioning Commands

The commands for inserting sections are fairly intuitive. Of course, certain commands are appropriate to different document classes. For example, a book has chapters but a article doesn't. Here is an edited version of some of the structure commands in use from *simple.tex*.

```
\section{Introduction}
This section's content...

\section{Structure}
This section's content...

\subsection{Top Matter}
This subsection's content...

\subsubsection{Article Information}

This subsubsection's content...
```

As you can see, the commands are fairly intuitive. Notice that you do not need to specify section numbers. Latex will sort that out for you! Also, for sections, you do not need to markup which content belongs to a given block, using `\begin` and `\end` commands, for example.

Command	Level
`\part{`*part*`}`	-1
`\chapter{`*chapter*`}`	0
`\section{`*section*`}`	1
`\subsection{`*subsection*`}`	2
`\subsubsection{`*subsubsection*`}`	3
`\paragraph{`*paragraph*`}`	4
`\subparagraph{`*subparagraph*`}`	5

Numbering of the sections is performed automatically by Latex, so don't bother adding them explicitly, just insert the heading you want between the curly braces. If you don't want section numbers, then add an asterisk (*) after the section command, but before the first curly brace, e.g., `\section*{A Title Without Numbers}`.

The Bibliography

Any good research paper will have a whole list of references. In this example document, I have included one. If you look at the PDF version, then after the first instance of 'Latex' in the introduction, you should notice a numbered reference. And at the end of the document, you can see the full reference.

Fortunately, Latex has a slightly more intelligent approach to managing your references than the average word processor, like MS Word for example, where everything has to be inputted manually (unless you purchase a 3rd party add-on). There are two ways to insert your references into Latex: the first is to store them in an external file and then link them via a command to your current document and use a Bibtex style to define how they appear, or secondly, embed them within the document itself. In this tutorial, I shall quickly cover the latter. Although, the former will be covered in depth in a future tutorial, as it is by far the most efficient and flexible.

There are two stages to setting up your bibliography/references in a document. The first is to set up a bibliography environment, which is where you provide Latex with the details of the references. The second is the actual citation of your references within your document.

The following code was used in creating the bibliography environment for the document in this tutorial. It is located immediately after the last line of the document content, but before the `\end{document}` command.

```
\begin{thebibliography}{9}

   \bibitem{lamport94}
          Leslie Lamport,
          \emph{\LaTeX: A Document Preparation System}.
          Addison Wesley, Massachusetts,
          2nd Edition,
          1994.

\end{thebibliography}
```

OK, so what is going on here? The first thing to notice is the establishment of the environment. `thebibliography` is a keyword that Latex recognises as everything between the begin and end tags as being data for the bibliography. The optional argument which I supplied after the begin statement is telling Latex how wide the item label will be when printed. Note however, that it is not a literal parameter, i.e the number 9 in this case, but a text width. Therefore, I am effectively telling Latex that I will only need reference labels of one character in width, which means no more than nine references in total. If you want more than ten, then input a two-digit number, such as '99' which permits less than 100 references.

Next is the actual reference entry itself. This is prefixed with the `\bibitem{`*`cite_key`*`}` command. The *cite_key* should be a unique identifier for that particular reference, and is often some sort of mnemonic consisting of any sequence of letters, numbers and punctuation symbols (although not a comma). I often use the surname of the first author, followed by the last two digits of the year (hence *lamport94*). If that author has produced more than one reference for a given year, then I add letters after, 'a', 'b', etc. But, you should do whatever works for you. Everything after the key is the reference itself. You need to type it as you want it to be presented. I have put the different parts of the reference, such as author, title, etc., on different lines for readability. These linebreaks are ignored by Latex. I wanted the title to be in italics, so I used the `\emph{}` command to achieve this.

To actually cite a given document is very easy. Goto the point where you want the citation to appear, and use the following: `\cite{`*`cite_key`*`}`, where the *cite_key* is that of the bibitem you wish to cite. When Latex processes the document, the citation will be cross-referenced with the bibitems and replaced with the appropriate number citation. The advantage here, once again, is that

Latex looks after the numbering for you. If it was totally manual, then adding or removing a reference can be a real chore, as you would have to re-number all the citations by hand.

Of course, it may be your preference to use a different referencing system, such as Harvard, instead of the default numerical. This will be covered in the future, in the mean time, why not try to experiment with the Natbib package.

Extras

You will find examples of using lists, as well as a table in the sample document I produced for this tutorial. Tables can be rather tricky, and so I intend to write a tutorial just on this topic in the future. However, please take time to inspect the source code yourself to see, now that you you are beginning to understand the mechanics of the Latex system, if you can also see how these extras where made.

This chapter uses material from Andy Roberts' Getting to grips with Latex with permission from the author.

6 Bibliography Management

For any academic/research writing, incorporating your references into your document is an important task. Fortunately, as Latex was aimed for this sort of work, it has a variety of features that make dealing with your references much simpler. Latex has built in support for citing references. However, a much more powerful and flexible solution is achieved thanks to an auxiliary tool called Bibtex (which comes bundled as standard with Latex.)

Bibtex allows you to store all your references in an external, flat-file database. You can then easily link this database to any Latex document, and cite any reference that is contained within the file. This is often more convenient that embedding them at the end of every document you write. You can have a centralised store of your bibliography, that can be linked to as many documents as you wish (write once, read many!) Of course, you can split your bibliographies over as many files as you wish, so you could have a file of references concerning the *theory of relativity*, and another about *quantum physics*. And if you were writing about the *Grand Unifying Theory* (GUT), which tries to bridge the gap between the inconsistencies of these two theories, then you can easily link both to your current document, as well another file of references about GUT, for example. It's up to you how you store your references, of course.

Bibtex

The previous chapter introduced the idea of embedding references at the end of the document, and then using the `\cite` command to cite them within the text. In this tutorial, I want to do a little better than this method, as it's not as flexible as it could be. Which is why I wish to concentrate on using Bibtex.

A Bibtex database is stored as a *.bib* file. It is a plain text file, and so can be viewed and edited easily. The structure of the file is also quite simple. An example of a Bibtex entry:

```
@article{greenwade93,
    author  = "George D. Greenwade",
    title   = "The {C}omprehensive {T}ex {A}rchive {N}etwork ({CTAN})",
    year    = "1993",
    journal = "TUGBoat",
    volume  = "14",
    number  = "3",
    pages   = "342--351"
}
```

Each entry begins with the declaration of the reference type, in the form of `@type`. Bibtex knows of practically all types you can think of, common ones such as *book*, *article*, and for papers presented at conferences, there is *inproceedings*, etc. In this example, I have referred to an article within a journal.

After the type, you must have a left curly brace `'{'` to signify the beginning of the reference attributes. The first one follows immediately after the brace, which is the citation key. This key must be unique for all entries in your bibliography. It is with this identifier that you will use within your document to cross-reference it to this entry. It is up to you as to how you wish to label each reference, but there is a loose standard in which you use the author's surname, followed by the year of publication. This is the scheme that I use in this tutorial.

Next, it should be clear that what follows are the relevant fields and data for that particular reference. The field names on the left are Bibtex keywords. They are followed by an equals sign (=) where the value for that field is then placed. Bibtex expects you to explicitly label the beginning and end of each value. I personally use quotation marks ("), however, you also have the option of using curly braces ('{', '}'). But as you will soon see, curly braces have other roles, within attributes, so I prefer not to use them for this job as they can get more confusing.

Remember that each attribute must be followed by a comma to delimit one from another. You do not need to add a comma to the last attribute, since the closing brace will tell Bibtex that there are no more attributes for this entry, although you won't get an error if you do.

It can take a while to learn what the reference types are, and what fields each type has available (and which ones are required or optional, etc). So, look at this entry type reference and also this field reference for descriptions of all the fields. It may be worth bookmarking or printing these pages so that they are easily at hand when you need them.

Authors

Bibtex can be quite clever with names of authors. It can accept names in *forename surname* or *surname, forename*. I personally use the former, but remember that the order you input them (or any data within an entry for that matter) is customisable and so you can get Bibtex to manipulate the input and then output it however you like. If you use the *forename surname* method, then you must be careful with a few special names, where there are compound surnames, for example "John von Neumann". In this form, Bibtex assumes that the last word is the surname, and everything before is the forename, plus any middle names. You must therefore manually tell Bibtex to keep the 'von' and 'Newman' together. This is achieved easily using curly braces. So the final result would be "John {von Neumann}". This is easily avoided with the *surname, forename*, since you have a comma to separate the surname from the forename.

Secondly, there is the issue of how to tell Bibtex when a reference has more than one author. This is very simply done by putting the keyword '*and* in between every author. As we can see from another example:

```
@book{goossens93,
    author    = "Michel Goossens and Frank Mittlebach and Alexander Samarin",
    title     = "The Latex Companion",
    year      = "1993",
    publisher = "Addison-Wesley",
    address   = "Reading, Massachusetts"
}
```

This book has three authors, and each is separated as described. Of course, when Bibtex processes and outputs this, there will only be an 'and' between the penultimate and last authors, but within the .bib file, it needs the *and*'s so that it can keep track of the individual authors.

Preserving capital letters

In the event that Bibtex has been set to not preserve all capitalisation within titles, problems can occur, especially if you are referring to proper nouns, or acronyms. To tell Bibtex to keep them, use the

good ol' curly braces around the letter in question, (or letters, if its an acronym) and all will be well! As you can see in the following first entry example above:

Getting current Latex document to use your .bib file

Actually this is not very difficult. At the end of your Latex file (that is, after the content, but before `\end{document}`, you need to place the following commands:

```
\bibliographystyle{plain}
\bibliography{sample}
```

Bibliography styles are files recognised by Bibtex that tell it how to format the information stored in the .bib file when processed for output. And so the first command listed above is declaring which style file to use. The style file in this instance is *plain.bst* (which comes as standard with Bibtex). You do not need to add the .bst extension when using this command, as it is assumed. Despite it's name, the plain style does a pretty good job (look at the output of this tutorial to see what I mean).

The second command is the one that actually specifies the .bib file you wish to use. The one I created for this tutorial was called *sample.bib*, but once again, you don't include the file extension. At the moment, the .bib file is in the same directory as the Latex document too. However, if your .bib file was elsewhere (which makes sense if you intend to maintain a centralised database of references for all your research), you need to specify the path as well, e.g `\bibliography{$HOME/some/where/sample.bib}`.

Now that Latex and Bibtex know where to look for the appropriate files, actually citing the references is fairly trivial. The `\cite{`*`ref_key`*`}` is the command you need, making sure that the *ref_key* corresponds exactly to one of the entries in the .bib file. If you wish to cite more that one reference at the same time, do the following: `\cite{`*`ref_key1`*`, `*`ref_key2`*`, ..., `*`ref_keyN`*`}`.

Why won't Latex generate any output?

The addition of Bibtex adds extra complexity for the processing of the source to the desired output. This is largely hidden to the user, but because of all the complexity of the referencing of citations from your source Latex file to the database entries in another file, you actually need multiple passes to accomplish the task. This means you have to run Latex a number of times, where each pass, it will perform a particular task until it has managed to resolve all the citation references. Here's what you need to type:

1. `latex bib` (doesn't require .tex extension)
2. `bibtex bib` (doesn't require .bib extension)
3. `latex bib`
4. `latex bib`

After the first Latex run, you will see errors such as:

```
LaTeX Warning: Citation `lamport94' on page 1 undefined on input line 21.

...

LaTeX Warning: There were undefined references.
```

The next step is to run bibtex on that same Latex source (and not on the actual .bib file) to then define all the references within that document. You should see output like the following:

```
This is BibTeX, Version 0.99c (Web2C 7.3.1)
The top-level auxiliary file: bib.aux
The style file: plain.bst
Database file #1: sample.bib
```

The third step, which is invoking Latex for the second time will see more errors like "`LaTeX Warning: Label(s) may have changed. Rerun to get cross-references right.`". Don't be alarmed, it's almost complete. As you can guess, all you have to do is follow its instructions, and run Latex for the third time, and the document will be output as expected, without further problems (as a DVI file. See [latextutorial1.html tutorial 1] if you need reminding how to convert them into PS or PDF.)

What about citation styles other than numerical, e.g., Harvard?

Hopefully, you have already looked at the PDF output of this tutorial. You will have noticed that the document ends with an automatically generated 'References' section. Each reference is numbered and each citation corresponds to the numbers. The numeric style of citation is quite common in scientific writing. In other disciplines, the author-year style, e.g., (Roberts, 2003), such as *Harvard* is preferred, and is in fact becoming increasingly common within scientific publications. A discussion about which is best will not occur here. I shall merely provide details for those who wish to have the choice.

Natbib

Natbib is a package written for Latex to do just this job. In fact, it can supersede Latex's own citation commands, as Natbib allows the user to easily switch between Harvard or numeric. In order to demonstrate Natbib, I've modified the original Latex file to take advantage of the additional functionality. Therefore, for the rest of this section, I shall be referring to *bib-har.tex* which can be accessed, along with the PDF of its output, at the bottom of this page.

The first job is to add the following to your preamble in order to get Latex to use the Natbib package:

```
\usepackage{natbib}
```

Also, you need to change the bibliography style file to be used, so edit the appropriate line at the bottom of the file so that it reads: `\bibliographystyle{plainnat}`. Once done, it is basically a matter of altering the existing `\cite` commands to display the type of citation you want.

Citation command	Natbib output
`\citet{goossens93}`	Goossens et al. (1993)
`\citep{goossens93}`	(Goossens et al., 1993)
`\citet*{goossens93}`	Goossens, Mittlebach, and Samarin (1993)
`\citep*{goossens93}`	(Goossens, Mittlebach, and Samarin, 1993)

The main commands simply add a *t* for 'textual' or *p* for 'parenthesised', to the basic `\cite` command. You will also notice how Natbib by default will compress references with three or more authors to the more concise *1st surname et al* version. By adding an asterisk (*), you can override this default and list all authors associated with that citation. There some other less common commands that Natbib supports, such as `\citeyear`, `\citeauthor`, etc., which are fairly obvious.

The final area that I wish to cover about Natbib is customising its citation style. There is a command called `\bibpunct` that can be used to override the defaults and change certain settings. For example, I have put the following in the preamble:

```
\bibpunct{(}{)}{;}{a}{,}{,}
```

The command requires six mandatory parameters.

1. The symbol for the opening bracket.
2. The symbol for the closing bracket.
3. The symbol that appears between multiple citations.
4. This argument takes a letter:
    - *n* - numerical style.
    - *s* - numerical superscript style.
    - *any other letter* - author-year style.
5. The punctuation to appear between the author and the year (in parenthetical case only).
6. The punctuation used between years, in multiple citations when there is a common author. e.g., (Chomsky 1956, 1957). If you want an extra space, then you need `{,~}`.

So as you can see. This package is quite flexible, especially as you can easily switch between different citation styles by changing a single parameter. Do have a look at the Natbib manual, it's a short document and you can learn even more about how to use it.

Customising bibliography appearance

In my mind, one the main advantages of Bibtex, especially for people who write many research papers, is the ability to customise your bibliography to suit the requirements of a given publication. You will notice how different publications tend to have their own style of formatting references, which authors must adhere to if they want their manuscript publishing. In fact, established journals and conference organisers often will have created their own bibliography style (.bst file) for those users of Bibtex, to do all the hard work for you.

It can achieve this because of the nature of the .bib database, where all the information about your references is stored in a structured format, but nothing about style. This is a common theme in Latex in general, where it tries as much as possible to keep content and presentation separate - as it should be!

A bibliography style file (.bst) will tell Latex how to format each attribute, what order to put them in, what punctuation to use in between particular attributes etc. Unfortunately, creating such a style by hand is not a trivial task. Which is why *Makebst* (also known as *custom-bib*) is the tool we need.

Makebst can be used to automatically generate a .bst file based on your needs. It is very simple, and actually asks you a series of questions about your preferences. Once complete, it will then output the appropriate style file for you to use.

It should be installed with the Latex distribution (otherwise, you can download it) and it's very simple to initiate. At the command line, type:

```
latex makebst
```

Latex will find the relevant file and the questioning process will begin. You will have to answer quite a few (although, note that the default answers are pretty sensible), which means it would be impractical to go through an example in this tutorial. However, it is fairly straight-forward. And if you require further guidance, then there is a comprehensive manual available. I'd recommend experimenting with it and seeing what the results are when applied to a Latex document.

If you are using a custom built .bst file, it is important that Latex can find it! So, make sure it's in the same directory as the Latex source file, *unless* you are using one of the standard style files (such as *plain* or *plainnat*, that come bundled with Latex - these will be automatically found in the directories that they are installed. Also, make sure the name of the .bst file you want to use is reflected in the `\bibliographystyle{`*`style`*`}` command (but don't include the .bst extension!).

Summary

Although it can take a little time to get to grips with Bibtex, in the long term, it's an efficient way to handle your references. It's not uncommon to find .bib files on websites that people compile as a list of their own publications, or a survey of relevant works within a given topic, etc. Or in those huge, online bibliography databases, you often find Bibtex versions of publications, so it's a quick cut-and-paste into your own .bib file, and then no more hassle!

Having all you references in one place can be a big advantage. And having then in a structured form, that allows customisable output is another one. There are a variety of free utilities that can load your .bib files, and allow you to view them in a more efficient manner, as well as sort them and check for errors.

This chapter uses material from Andy Roberts' Getting to grips with Latex with permission from the author.

7 TABLES

In academic writing, tables are a common feature, often for summarising results from research. It is therefore a skill that needs mastering in order to produce good quality papers.

However, if there is one area about Latex that I feel is the least intuitive, then I am afraid that this is it. Basic tables are not too taxing, but you will quickly notice that anything more advanced can take a fair bit of construction. So, we will start slowly and build up from there.

The Tabular environment

To begin, we shall first get familiar with some Latex terminology. An *environment* in Latex is a special declaration for formatting specific types of text. For example, previously the *abstract* was introduced, so that the abstract of the paper could be formatted differently that the main body of text. All environments begin and end in the same fashion:

```
\begin{environment-name}
...
...
\end{environment-name}
```

The *tabular* is another such environment, designed for formatting your data into nicely arranged tables. Arguments are required after the environment declaration to describe the alignment of each column. The number of columns does not need to be specified as it is inferred by looking at the number of arguments provided. It is also possible to add vertical lines between the columns here. The following symbols are available to describe the table columns:

l	left-justified column
c	centered column
r	right-justified column
p{*width*}	paragraph column with text vertically aligned at the top
m{*width*}	paragraph column with text vertically aligned in the middle
b{*width*}	paragraph column with text vertically aligned at the bottom
|	vertical line
||	double vertical line

Once in the environment,

&	column separator
\\	start new row (additional space may be specified after \\ using square brakcets, such as \\[6pt])
\hline	horizontal line
\cline{i-j}	partial horizontal line

Note, any white space inserted between these commands is purely down to ones' preferences. I personally add spaces between to make it easier to read.

Very basic table

This example shows how to create a simple table in Latex. It is a three-by-three table, but without any lines.

```
\begin{tabular}{ l c r }
   1 & 2 & 3 \\
   4 & 5 & 6 \\
   7 & 8 & 9 \\
 \end{tabular}
```

1	2	3
4	5	6
7	8	9

Expanding upon that by including some vertical lines:

```
\begin{tabular}{ l | c || r | }
   1 & 2 & 3 \\
   4 & 5 & 6 \\
   7 & 8 & 9 \\
 \end{tabular}
```

1	2	3
4	5	6
7	8	9

To add horizontal lines to the very top and bottom edges of the table:

```
\begin{tabular}{ l | c || r | }
 \hline
   1 & 2 & 3 \\
   4 & 5 & 6 \\
   7 & 8 & 9 \\
 \hline
 \end{tabular}
```

1	2	3
4	5	6
7	8	9

And finally, to add lines between all rows, as well as centring (notice the use of the center environment - of course, the result of this is not obvious from the preview on this web page, so look at the output:

```
\begin{center}
   \begin{tabular}{ l | c || r | }
     \hline
     1 & 2 & 3 \\ \hline
     4 & 5 & 6 \\ \hline
     7 & 8 & 9 \\
     \hline
   \end{tabular}
 \end{center}
```

1	2	3
4	5	6
7	8	9

Text wrapping in tables

Latex's algorithms for formatting tables have a few shortcomings. One is that it will not automatically wrap text in cells, even if it has overrun the width of the page. For columns that you know will contain a certain amount of text, then it is recommended that you use the *p* attribute and specify the desired width of the column (although it may take some trial-and-error to get the result you want).

Before we can proceed, we must introduce the Latex system of using measurements. This is quite flexible, as you can choose from a variety of length units.

- pt - a point is 1/72 inch.
- mm - millimetre.
- cm - centimetre.
- in - inch.
- ex - roughly the height of an 'x' in the current font.
- em - roughly the width of an 'M' (note the uppercase) of the current font.

There are also things known as *command lengths*, which are not fixed values as they depend on the configuration of the current document class and/or preamble. Useful ones include:

- \parindent - the size of the paragraph indent
- \baselineskip - vertical distance between lines.
- \parskip - the extra space between paragraphs.
- \textwidth - the width of a line of text in the local environment (e.g., the lines are commonly narrower in the abstract than in the normal text).
- \textheight - the height of the text on the page.

The examples prepared are quite long because I was illustrating what happens when there is a fair bit of text in table cells.

The Tabular* environment - controlling table width

This is basically a slight extension on the original tabular version, although it requires an extra argument (before the column descriptions) to specify the preferred width of the table.

```
\begin{tabular*}{0.75\textwidth}{ | c | c | c | r | }
  \hline
  label 1 & label 2 & label 3 & label 4 \\
  \hline
  item 1  & item 2  & item 3  & item 4  \\
  \hline
\end{tabular*}
```

However, that doesn't look quite as intended. The columns are still at their natural width (just wide enough to fit their contents whilst the rows are as wide as the table width specified. This looks very ugly. The reason for the mess is that you must also explicitly insert extra column space. Fortunately, Latex has *rubber lengths*, which unlike others, are not fixed, and Latex can dynamically decide how long they should be. So, the solution the current problem is:

```
\begin{tabular*}{0.75\textwidth}{@{\extracolsep{\fill}} | c | c | c | r | }
  \hline
  label 1 & label 2 & label 3 & label 4 \\
  \hline
  item 1  & item 2  & item 3  & item 4  \\
  \hline
\end{tabular*}
```

You will notice the @{...} construct added at the beginning of the column description. More details of this will be coming shortly. Within it is the \extracolsep command, which requires a width. A fixed width could have been used, however, by using a rubber length, such as \fill, the columns are automatically spaced evenly.

@-expressions

It is probably worth addressing the @ specifier now that it has been introduced, even though there aren't very many obvious applications for it.

It typically takes some text as its argument, and when appended to a column, it will automatically insert that text into each cell in that column before the actual data for that cell. It is worth noting that once used, inter-column space between the affected columns is suppressed. To add space, use @{\hspace{*width*}}.

Admittedly, this is not that clear, and so will require a few examples to clarify. Sometimes, it is desirable in scientific tables to have the numbers aligned on the decimal point. This can be achieved by doing the following:

```
\begin{tabular}{r@{.}l}
  3&14159\\
  16&2\\
  123&456\\
\end{tabular}
```

3.14159
16.2
123.456

Its space suppressing qualities actually make it quite useful for manipulating the horizontal spacing between columns. Given a basic table, and varying the column descriptions:

```
\begin{tabular}{|l|l|}
\hline
stuff & stuff \\ \hline
stuff & stuff \\
\hline
\end{tabular}
```

```
{|l|l|}
```

stuff	stuff
stuff	stuff

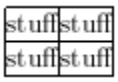
```
{|@{}l|l@{}|}
```

stuff	stuff
stuff	stuff

```
{|@{}l@{}|l@{}|}
```

stuff	stuff
stuff	stuff

```
{|@{}l@{}|@{}l@{}|}
```



Spanning

To complete this tutorial, a quick at how to generate slightly more complex tables. Unsurprisingly, the commands necessary have to be embedded within the table data itself.

Rows spanning multiple columns

The the command for this looks like this: \multicolumn{*num_cols*}{*alignment*}{*contents*}. *num_cols* is the number of subsequent columns to merge; *alignment* is pretty obvious, either l, c, or r. And *contents* is simply the actual data you want to be contained within that cell. A simple example:

```
\begin{tabular}{|l|l|}
\hline
\multicolumn{2}{|c|}{Team sheet} \\
\hline
GK & Paul Robinson \\
LB & Lucus Radebe \\
DC & Michael Duberry \\
DC & Dominic Matteo \\
RB & Didier Domi \\
MC & David Batty \\
MC & Eirik Bakke \\
MC & Jody Morris \\
FW & Jamie McMaster \\
ST & Alan Smith \\
ST & Mark Viduka \\
\hline
\end{tabular}
```

Team sheet	
GK	Paul Robinson
LB	Lucus Radebe
DC	Michael Duberry
DC	Dominic Matteo
RB	Didier Domi
MC	David Batty
MC	Eirik Bakke
MC	Jody Morris
FW	Jamie McMaster
ST	Alan Smith
ST	Mark Viduka

Columns spanning multiple rows

The first thing you need to do is add \usepackage{multirow} to the preamble. This then provides the command needed for spanning rows: \multirow{*num_rows*}{*width*}{*contents*}. The arguments are pretty simple to deduce. With the *width* parameter, you can specify a fixed width if you wish, or, if you want the natural width (i.e., just wide enough to fit the contents of the column) then simply input an asterisk (*). This approach was used for the following example:

```
...
\usepackage{multirow}
...

\begin{tabular}{|l|l|l|}
\hline
\multicolumn{3}{|c|}{Team sheet} \\
\hline
```

```
Goalkeeper & GK & Paul Robinson \\ \hline
\multirow{4}{*}{Defenders} & LB & Lucus Radebe \\
 & DC & Michael Duberry \\
 & DC & Dominic Matteo \\
 & RB & Didier Domi \\ \hline
\multirow{3}{*}{Midfielders} & MC & David Batty \\
 & MC & Eirik Bakke \\
 & MC & Jody Morris \\ \hline
Forward & FW & Jamie McMaster \\ \hline
\multirow{2}{*}{Strikers} & ST & Alan Smith \\
 & ST & Mark Viduka \\
\hline
\end{tabular}
```

Team sheet		
Goalkeeper	GK	Paul Robinson
Defenders	LB	Lucus Radebe
	DC	Michael Duberry
	DC	Dominic Matteo
	RB	Didier Domi
Midfielders	MC	David Batty
	MC	Eirik Bakke
	MC	Jody Morris
Forward	FW	Jamie McMaster
Strikers	ST	Alan Smith
	ST	Mark Viduka

The main thing to note when using \multirow is that for the subsequent rows that are to be spanned, a blank entry for the appropriate cells have to be inserted.

Summary

That's about it for basic tables in my opinion. After you experiment, you do quickly get up to scratch. I must admit, the table syntax in Latex can look rather messy, and so seeing new examples can look confusing. But hopefully, enough has been covered here so that you can create any table you are likely to need for your papers. Unsurprisingly, Latex has plenty more up its sleeve, so expect a follow up tutorial covering more advanced features in the near future.

This chapter uses material from Andy Roberts' Getting to grips with Latex with permission from the author.

8 Importing Graphics

In many respects, importing your images into your document using LaTeX is fairly simple... *once* you have your images in the right format that is! Therefore, I fear for many people the biggest effort will be the process of converting their graphics files.

Image formats for LaTeX

LaTeX compilers can handle several types of images, but you have to take care of which compiler you are using. If you want to create a DVI file, then the only image format you can use is EPS. On the other hand, if you are going to create a PDF, then the input images can be JPEG (to be used for photos), PNG (for graphics) of PDF itself (can contain only pictures, too). In general you might want to be able to create both DVI and PDF, so you should keep an updated version of your pictures in both EPS and one of the formats for PDF. As we will see, the command to insert them is the same, the LaTeX compiler you are running will choose which one to use.

Encapsulated Postscript (EPS)

EPS was defined by Adobe for making it easy for applications to import postscript-based graphics into documents. Because an EPS file declares the size of the image, it makes it easy for systems like LaTeX to arrange the text and the graphics in the best way.

Most decent graphics software has the ability to save images in the EPS format (extension is normally .eps). Obviously, Adobe applications do, since they developed the PS and EPS standards, however there are many other alternatives.

Converting images to EPS (in GNU/Linux)

There are so many utilities available on Linux for converting between graphics formats that you can be overwhelmed. There are plenty that are dedicated to converting a given format into EPS, jpeg2ps, for example. However, I shall cover a better, more generic piece of software, called convert, a part of ImageMagick.

convert supports just about every graphics format that you care to mention, and its beauty is that it will let you convert to and from any format it understands. It is a command-line program, but is very simple to use. You pass two arguments, the first being the filename of your current image, and the second is the filename you wish to give your converted image. Normally, the only difference is in the extension. By default, convert will deduce the input and output formats by the file extensions you supply. For example:

```
convert graph.jpg graph.eps
```

This command takes the JPEG file, graph.jpg, and converts it to an EPS file of the same name. Of course, it would have worked equally well if your original image was a BMP, GIF, PNG, etc.

For a more graphical approach, try The GIMP, which allows you to visually load and save to plenty of image formats. It is so straight-forward that I don't feel it needs a walkthru!

Converting images to EPS (in Windows)

There are lots applications built for Windows that can do the job, however, only a couple are at hand at the moment.

The first choice for me would be Paint Shop Pro. It has always been renowned for its wide support for graphics formats, so this too will be OK for converting files. Currently, the version installed is only v4.12. Although, via Citrix, you can get access the latest version, v8.0. Having said that, v4 is more than adequate at the job of converting to EPS. The steps required are as follows:

1. Load up *PSP* by selecting it from Start/Programs.
2. Select File/Open, or click the open icon.
3. Browse to the location of your image. Select the file and click **Open**.
4. Once image has loaded, select File/Save As.
5. Choose the location to save the image. Click on the drop-down box that specifies the file-type and select 'Encapsulated Postscript'. It should automatically update the filename to have the '.eps' extension. Feel free to change the filename if you wish.
6. By default, PSP will save EPS files in monochrome. If your image in already in black and white, or if don't need to preserve the colour, then do nothing. Otherwise, under 'Options', choose to save in colour, but without a preview.
7. Click **Save**.

It is then important that you then copy your images to a location that LaTeX can read them. For users within the School of Computing, this means your Linux user account.

Corel Draw 9 is also available for Windows users that can achieve similar results. Of course, you are always free to remotely log on to Linux, and use the infinitely flexible convert utility.

Graphicx

As with most things in LaTeX, there is more than one way to skin a cat. And then same is true for importing images into your documents. However, I shall focus on a package called graphicx, which does this particular task quickly, and with little effort and can support all the formats that LaTeX can handle.

Before you any of the following commands will work, you need to put `\usepackage{graphicx}` in the preamble of your document.

The syntax for using graphicx is: `\includegraphics[`*`attr1=val1, attr2=val2, ..., attrn=valn`*`]{`*`imagename`*`}`

As you should hopefully be aware by now, arguments in square brackets are optional, whereas curly braces are compulsory. The argument of the curly braces is the name of the image, write it

without the extension. This way the LaTeX compiler will look for any supported image format in that directory and will take the best one (EPS if the output is DVI, JPEG or PNG if the output is PDF). The variety of possible attributes that can be set is fairly large, and so I shall cover the most useful:

<table>
<tr><td><code>width=xx</code></td><td>Specify the preferred width of the imported image to xx.</td><td rowspan="2">NB. Only specifying either width or height will scale the image whilst maintaining the aspect ratio.</td></tr>
<tr><td><code>height=xx</code></td><td>Specify the preferred height of the imported image to xx.</td></tr>
<tr><td><code>keepaspectratio</code></td><td colspan="2">This can be set to either true or false. When true, it will scale the image according to both height and width, both will not distort the image, so that neither width or height are exceeded.</td></tr>
<tr><td><code>scale=xx</code></td><td colspan="2">Scales the image by the desired scale factor. e.g, 0.5 to reduce by half, or 2 to double.</td></tr>
<tr><td><code>angle=xx</code></td><td colspan="2">This option can rotate the image by xx degrees (anti-clockwise)</td></tr>
<tr><td><code>trim=l b r t</code></td><td colspan="2">This option will crop the imported image by l from the left, b from the bottom, r from the right, and t from the top. Where l, b, r and t are lengths.</td></tr>
<tr><td><code>clip</code></td><td colspan="2">For the <code>trim</code> option to work, you must set <code>clip=true</code>.</td></tr>
</table>

NB. To use more than one option at a time, simply separate with a comma.

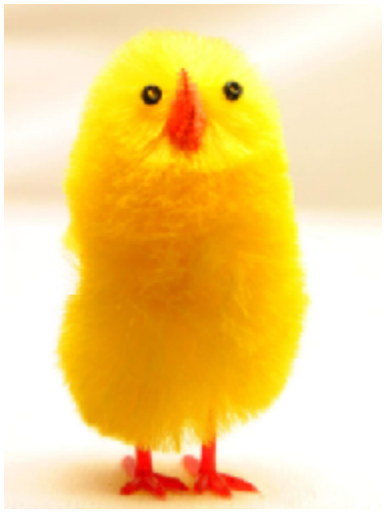
Examples

OK, it's time to see graphics in action. You are, as always, recommended to first look at the example files at the end of the tutorial. Look at the .tex file first, and then at the .pdf output. Here are some examples from said .tex file:

```
\includegraphics{chick.eps}
```

This imports the image, however, it is very large (so I won't display it here!). So, let's scale it down:

```
\includegraphics[scale=0.5]{chick.eps}
```



This has now reduced by half. If you wish to be more specific and give actual lengths of the image dimensions, this is how to go about it:

```
\includegraphics[width=2.5cm]{chick.eps}
```

One can also specify the scale with respect to the width of a line in the local environment (\linewidth), the width of the text on a page (\pagewidth) or the height of the text on a page (\textheight) (pictures not shown):

```
\includegraphics[width=0.5\linewidth]{chick.eps}

\includegraphics[width=0.75\textwidth]{chick.eps}

\includegraphics[height=0.75\textheight]{chick.eps}
```

To rotate (I also scaled the image down too):

```
\includegraphics[scale=0.5, angle=180]{chick.eps}
```

And finally, an example of how to crop an image should you wish to focus in one particular area of interest:

```
\includegraphics[trim = 10mm 80mm 20mm 5mm, clip, width=3cm]{chick.eps}
```

Note the presence of clip, as the trim operation will not work without it.

Summary

That concludes just about all you need to know about how to import your images into a LaTeX document. As I said, the actual conversion is possibly the longest part of the whole process, since the command to include the image is straight-forward.

There is one major topic missing from this tutorial that is relevant, and that is to do with making that image a *figure*. For this, you will want a caption, and maybe cross-reference. However, this was deliberate, as it is not only images that are figures. Therefore, this material is to be covered within its own tutorial of its own

This chapter uses material from Andy Roberts' Getting to grips with Latex with permission from the author.

9 Floats, Figures and Captions

In the previous chapter, the importing of graphics was introduced. However, just having a picture stuck in-between paragraphs does not look professional. For starters, we want a way of adding captions, and to be able to cross-reference. What we need is a way of defining *figures*. It would also be good if Latex could apply similar principles to when it arranges text to look its best, to arranging pictures too. This is where *floats* come into play.

Floats

Floats are containers for things in a document that cannot be broken over a page. LaTeX out of the box recognises "table" and "figure" floats, but you can define new ones of your own (see below). Floats are there to deal with the problem of the object that won't fit on the present page, and to help when you really don't want the object here just now.

So, suppose there is not enough space for your figure: LaTeX will ordinarily save it away for the next time there is enough space, and carry on filling the page with your regular text. As often as not, the figure will appear on the next page, so it's not usually hard to find.

Contrast this to what happens in MS Word, for example. If a image is added but is too large to fit on the current page, it will position it on the next page, but, will leave a large gap, instead of rearranging subsequent text to fill the space. It requires a lot of manual tweaking to rectify, something LaTeX automatically saves you from.

Figures

To create a figure that you want to float, you must use the *figure* environment (tricky, eh?!).

```
   \begin{figure}[placement specifier]
...                figure contents                ...
\end{figure}
```

In the previous section, I was saying how floats are used to allow Latex to handle figures, whilst maintaining the best possible presentation. However, there may be times when you disagree, and a typical example is with its positioning of figures. The *placement specifier* parameter exists as a compromise, and its purpose is to give the author a greater degree of control over where certain floats are placed.

Specifier	Permission
`h`	Place the float *here*, i.e., at the same point it occurs in the source text.
`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.

!	Override internal parameters Latex uses for determining `good' float positions.

What you do with these *placement permissions* is to list which of the options that you wish to make available to Latex. These are simply possibilities, and Latex will decide when typesetting your document which of your supplied specifiers it thinks is best.

Use `\listoffigures` to add a list of the figures in the beginning of the document.

Tables

Although tables have already been covered, it was only the internal syntax that was discussed. The `tabular` environment that was used to construct the tables is not a float by default. Therefore, for tables you wish to float, wrap the `tabular` environment within a `table` environment, like this:

```
\begin{table}
  \begin{tabular}{...}
  ... table data ...
  \end{tabular}
\end{table}
```

You may feel that it is a bit long winded, but such distinctions are necessary, because you may not want all tables to be treated as a float.

Use `\listoftables` to add a list of the tables in the beginning of the document.

Captions

It is always good practice to add a caption to any figure or table. Fortunately, this is very simple in Latex. All you need to do is use the `\caption{`*`text`*`}` command within the float environment. Because of how Latex deals sensibly with logical structure, it will automatically keep track of the numbering of figures, so you do not need to include this within the caption text.

The location of the caption is traditionally underneath the float. However, it is up to you to therefore insert the caption command after the actual contents of the float (but still within the environment). If you place it before, then the caption will appear above the float. Try out the following example to demonstrate this effect.

```
\begin{figure}
  \caption{A picture of a tucan.}
  \centering
  \includegraphics{tucan}
\end{figure}

\begin{figure}
  \centering
  \reflectbox{\includegraphics{tucan}}
  \caption{A picture of the same tucan looking the other way!}
\end{figure}
```

Labels and Cross-referencing

This is a good opportunity to introduce *labels*. Their purpose within Latex is to act as a marker, which can then be referenced to at any point within your document. It is very common to refer to each of your figures within the body of text. However, you don't want to be keeping track of the numbers, so you can use a label instead, and let Latex replace it with the correct figure number.

To add a label, you must embed the following command: `\label{`*marker*`}`.

To then reference a label: `\ref{`*marker*`}`.

Alternatively, to get a page reference: `\pageref{`*marker*`}`.

```
\begin{figure}
  \centering
  \reflectbox{\includegraphics{tucan.eps}}
  \caption{A picture of the same tucan looking the other way!}
  \label{toucan}
\end{figure}

Figure~\ref{toucan} shows a photograph of a toucan.
```

When a label is declared within a float environment, the `\ref` will return the respective fig/table number (although, it must occur **after** the caption). When declared outside, it will give the section number.

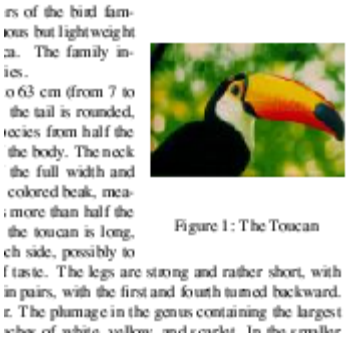
The tilde (~) in the above example is a special symbol within Latex. It represents a non-breaking space. It is useful here because it keeps 'Figure' and what ever number it \ref refers to as a single unit, and won't get split over lines or pages.

Wrapping figures

Although not normally the case in academic writing, an author may prefer that some floats do not break the flow of text, but instead allow text to wrap around it. (Obviously, this effect only looks decent when the figure in question is significantly narrower than the text width.) The package `wrapfig` was developed for this very task.

To use `wrapfig`, you must first add `\usepackage{wrapfig}` to the preamble. This then gives you access to the `\begin{wrapfigure}[`*lineheight*`]{`*alignment*`}{`*width*`}` command. *Alignment* can be either *l* for left, or *r* for right. Lowercase *l* or *r* forces the figure to start precisely where specified (and may cause it to run over page breaks), while capital *L* or *R* allows the figure to float. The *width* is obviously the width of the figure. An example:

```
\begin{wrapfigure}{r}{40mm}
  \centering
  \includegraphics{toucan.eps}
  \caption{The Toucan}
\end{wrapfigure}
```



Another example using the optional parameters:

```
\begin{wrapfigure}[16]{r}[20mm]{4mm}
  \centering
  \includegraphics{toucan.eps}
  \caption{The Toucan}
\end{wrapfigure}
```

An alternative that does much the same thing is the *floatfig*-package. Warning: You might need to download *floatfig* because it's rather a non-standard-package which could cause problems.

Tip for figures with too much white space

It happens that you'll generate figures with too much (or too little) white space on the top or bottom. In such a case, you can simply add space within the float using the \vspace{} command.

```
\begin{wrapfigure}[16]{r}[20mm]{4mm}
  \centering
  \includegraphics{toucan.eps}
  \vspace{-25pt}
  \caption{The Toucan}
\end{wrapfigure}
```

Subfigures

A useful extension is the `subfigure` package. This gives the author the ability to have figures within figures. Subfigures have their own caption, and an optional global caption is also allowed. An obvious application would be to illustrate the before and after effects of a given operation. An example will best illustrate the usage of this package.

```
\begin{figure}[htp]
  \centering
  \subfigure[Original image]{\label{fig:edge-
a}\includegraphics[scale=0.75]{toucan.eps}}
  \subfigure[After Laplace edge detection]{\label{fig:edge-
b}\includegraphics[scale=0.75]{laplace_toucan.eps}} \\
  \subfigure[After Sobel edge detection]{\label{fig:edge-
```

```
c}\includegraphics[scale=0.75]{sobel_toucan.eps}}
  \caption{Various edge detection algorithms}
  \label{fig:edge}
\end{figure}
```

(a) Original image (b) After Laplace edge detection (c) After Sobel edge detection

Figure 1: Various edge detection algorithms

You will notice that the figure environment is set up as usual. For each subfigure, you need to use the `\subfigure[`*sub caption*`]{`*... figure ...*`}` command. If you intend to cross-reference any of the subfigures, see where the label is inserted on the third example. `\caption` will provide the global caption. `subfigure` will arrange the figures side-by-side providing they can fit, otherwise, it will automatically shift figures below. This effect can be added manually, by putting the newline command (\\) before the figure you wish to move to a newline.

NB For subtables, there is a different `\subtable` command specifically for such types, but it works in much the same way.

However, the `subfigure` package is considered deprecated and is superseded by the `subfig` package. The equivalent code would be:

```
\begin{figure}[htp]
  \centering
  \subfloat[Original image]{\label{fig:edge-
a}\includegraphics[scale=0.75]{toucan.eps}}
  \subfloat[After Laplace edge detection]{\label{fig:edge-
b}\includegraphics[scale=0.75]{laplace_toucan.eps}} \\
  \subfloat[After Sobel edge detection]{\label{fig:edge-
c}\includegraphics[scale=0.75]{sobel_toucan.eps}}
  \caption{Various edge detection algorithms}
  \label{fig:edge}
\end{figure}
```

Custom Floats

If tables and figures are not adequate for your needs, then you always have the option to create your own! A good example of such an instance would be a document that gives lots of source code examples of a programming language. One might therefore wish to create a *program* float. The package `float` is your friend for this task. *All commands to set up the new float must be placed in the preamble, and not within the document.*

1. Add `\usepackage{float}` to the preamble of your document
2. Declare your new float using: `\newfloat{type}{placement}{ext}`, where
   - *type* - the new name you wish to call your float, in this instance, 'program'.
   - *placement* - t, b, p, or h (as previously described.
   - *ext* - the file name extension of an auxiliary file for the list of figures (or whatever). Latex writes the captions to this file.
3. The default name that appears at the start of the caption is the type. If you wish to alter this use, `\floatname{type}{floatname}`
4. Changing float style can be issued with `\floatstyle{style}` (Works on all subsequent `\newfloat` commands, therefore, must be inserted before `\newfloat` to be effective.
   - `plain` - the normal style for Latex floats, i.e., nothing!
   - `boxed` - a box is drawn that surrounds the float, and the caption is printed below.
   - `ruled` - the caption appears above the float, with rules immediately above and below. Then the float contents, followed by a final horizontal rule.

An example document using a new *program* float type:

```
\documentclass{article}

\usepackage{float}

\floatstyle{ruled}
\newfloat{program}{thp}{lop}
\floatname{program}{Program}

\begin{document}

\begin{program}
  \begin{verbatim}

class HelloWorldApp {
  public static void main(String[] args) {
    //Display the string
    System.out.println("Hello World!");
  }
}
\end{verbatim}
  \caption{The Hello World! program in Java.}
\end{program}

\end{document}
```

Program 1 The Hello World! program in Java.

```
class HelloWorldApp {
  public static void main(String[] args) {
    //Display the string
    System.out.println("Hello World!");
  }
}
```

The `verbatim` environment is an environment that is already part of Latex. Although not introduced so far, its name is fairly intuitive! Latex will reproduce everything you give it, including new lines, spaces, etc. It is ideal for source code.

Summary

That concludes all the fundamentals of floats. You will hopefully see how much easier it is to let Latex do all the hard work and tweak the page layouts in order to get your figures in the best place. As always, the fact that Latex takes care of all caption and reference numbering is a great time saver. Don't forget to have a look at the example .tex files below.

This chapter uses material from Andy Roberts' Getting to grips with Latex with permission from the author.

10 FORMATTING

The term formatting is rather broad, but in this case it needs to be as this section will guide you various text, paragraph and page formatting techniques. Formatting tends to refer to most things to do with appearance, it makes the list of possible topics quite eclectic: text style, font, size; paragraph alignment, interline spacing, indents; special paragraph types; list structures; footnotes, margin notes, etc.

A lot of the formatting techniques are required to differentiate certain elements from the rest of the text. It is often necessary to add emphasis to key words or phrases. A numbered or bulleted list is also commonly used as a clear and concise way of communicating an important issue. Footnotes are useful for providing extra information or clarification without interrupting the main flow of text. So, for these reasons, formatting is very important. However, it is also very easy to abuse, and a document that has been over-done can look and read worse than one with none at all.

Text formatting

Quotes

Latex treats left and right quotes as different entities. For single quotes, ` (on British and American keyboards, this symbol is found on the key adjacent to the number 1) gives a left quote mark, and ' is the right. For double quotes, simply double the symbols, and Latex will interpret them accordingly. (Although, you can use the " for right double quotes if you wish).

```
To `quote' in Latex
```

To ‘quote’ in Latex.

```
To ``quote'' in Latex
```

To “quote” in Latex.

```
To ``quote" in Latex
```

To “quote” in Latex.

The right quote is also used for apostrophe in Latex without trouble.

Dots and Dashes

A sequence of three dots is known as an *ellipsis*, which is commonly used to indicate omitted text. Simply inputting three dots doesn't give the best output because the spacing is not correct. Therefore, you should use `\ldots` to get the right result.

```
Ellipsis...
```

Ellipsis...

Ellipsis ...

```
Ellipsis \ldots
```

Latex has three specific types of dashes (the "hyphen", the "en-dash", and the "em-dash"), each a different size, and each for a different purpose:

Input	Output	Purpose
-	-	inter-word
--	–	page range, 1–10
---	—	punctuation dash — like this

Accents

It does not take long before you need to use an accented character within your document. Personally, I find it tends to be names of researchers whom I wish to cite which contain accents. It's easy to apply, although not all are easy to remember!:

Command	Output	Command	Output
\`{o}	ò	\'{o}	ó
\"{o}	ö	\H{o}	ő
\^{o}	ô	\~{o}	õ
\v{o}	ǒ	\={o}	ō
\b{o}	o̲	\.{o}	ȯ
\d{o}	ọ	\c{o}	o̧
\r{o}	o̊	\t{oo}	o͡o
\i	ı		

Symbols

Latex has *lots* of symbols at its disposal. The majority of them are within the mathematical domain, and later chapters will cover how to get access to them. For the more common text symbols, use the following commands:

Command	Symbol	Command	Symbol
\%	%	\#	#
\$	$	\&	&
\{	{	\}	}
_	_	\S	§
\P	¶	\dag	†
\ddag	‡	\textbackslash	\
\textbar	|	\textless	<
\textgreater	>	\textemdash	—
\textendash	–	\textregistered	®
\texttrademark	™	\textquestiondown	¿
\textexclamdown	¡	\textcircled{a}	ⓐ
^{a}	ª	\copyright	©
\pounds	£		

Of course, these are rather boring, and it just so happens that for some more interesting symbols, then the Postscript ZipfDingbats font is available thanks to the `pifont`. Hopefully, you are beginning to notice now that when you want to use a package, you need to add the declaration to your preamble, in this instance: `\usepackage{pifont}`. Next, the command `\ding{`*number*`}`, will print the specified symbol.

Please note, currently, on the SoC machines, when this package is used, the electronic versions when viewed does not display the symbols correctly. However, when printed, it outputs as intended. Once I work out what the problem is, I shall update this section to instruct on how to rectify.

Emphasising text

In order to add some emphasis to a word or phrase, the simplest way is to use the `\emph{`*text*`}` command. That's it! See, dead easy.

I want to *emphasize* a word.

```
I want to \emph{emphasize} a word.
```

Font styles

I shall really only scratch the surface about this particular field. This section is not about how to get your text in Verdana size 12pt! There are three main font families: roman (such as Times), sans serif (eg Arial) and monospace (eg Courier). You can also specify styles such as italic and bold. The following table lists the commands you will need to access the typical font styles:

Command	Style
`\textnormal{...}`	document font family
`\emph{...}`	*emphasis*
`\textrm{...}`	roman font family
`\textsf{...}`	sans serif font family
`\texttt{...}`	`typewriter font family`
`\textup{...}`	upright shape
`\textit{...}`	*italic shape*
`\textsl{...}`	*slanted shape*
`\textsc{...}`	SMALL CAPITALS
`\textbf{...}`	**bold**
`\textmd{...}`	normal weight and width

You may have noticed the absence of underline. This functionality has to be added with the `ulem` package. Stick `\usepackage{ulem}` in your preamble. By default, this overrides the `\emph` command with the underline rather than the italic style. It is unlikely that you wish this to be the desired effect, so it is better to stop `ulem` taking over `\emph` and simply call the underline command as and when it is needed.

- To disable `ulem`, add `\normalem` straight after the document environment begins.
- To underline, use `\uline{...}`.
- To add a wavy underline, use `\uwave{...}`.
- And for a strike-out `\sout{...}`.

Finally, there is the issue of size. This to is very easy, simply follow the commands on this table:

Command	Output
`\tiny`	sample text
`\scriptsize`	sample text
`\footnotesize`	sample text
`\small`	sample text
`\normalsize`	sample text
`\large`	sample text
`\Large`	sample text
`\LARGE`	sample text
`\huge`	sample text
`\Huge`	sample text

Paragraph Formatting

Altering the paragraph formatting is not often required, especially in academic writing. However, it is useful to know, and applications tend to be for formatting text in floats, or other more exotic documents.

Paragraph Alignment

Paragraphs in Latex are usually fully justified (i.e., flush with both the left and right margins). For whatever reason, should you wish to alter the justification of a paragraph, there are three environments at hand, and also Latex command equivalents.

Alignment	Environment	Command
Left justified	`flushleft`	`\raggedright`
Right justified	`flushright`	`\raggedleft`
Center	`center`	`\centering`

All text between the `\begin` and `\end` of the specified environment will be justified appropriately. The commands listed are for use within other environments. For example, `p` (paragraph) columns in `tabular`.

Paragraph Indents

Paragraph indents are normally fine. The size of the indent is determined by a parameter called `\parindent`. The default length that this constant holds is set by the document class that you use. It is possible to override using the `\setlength` command.

`\setlength{\parindent}{`*`length`*`}` will reset the indent to *length*.

Be careful, however, if you decide to set the indent to zero, then it means you will need a vertical space between paragraphs in order to make them clear. The space between paragraphs is held in `\parskip`, which could be altered in a similar fashion as above. However, this parameter is used elsewhere too, such as in lists, which means you run the risk of making various parts of your document look very untidy (that is, even more untidy than this type of paragraph style!) It may be better to use a document class typeset for this style of indentation, such as `artikel3.cls` (written by a dutch person, translates to article3).

Line Spacing

It's rarely necessary to require anything other than single line spacing. But for those of you who require it, this is how...

1. Add `\usepackage{setspace}` to the document preamble.
2. This then provides the following environments to use within your document:
   - `doublespace` - all lines are double spaced.
   - `onehalfspace` - line spacing set to one-and-half spacing.
   - `singlespace` - normal linespacing.

Special paragraphs

For those of you who have read most/all of the tutorials so far, you will have already come across

some of the following paragraph formats. Although seen before, it makes sense to re-introduce here, for the sake of completeness.

`Verbatim` Environment

This environment was used in an example in the previous tutorial. Everything input between the *begin* and *end* commands are processed as if by a typewriter. All spaces and new lines are reproduced as given, and the text is displayed in an appropriate monospace font. Ideal for typesetting program source code, for example.

```
\begin{verbatim}
The verbatim environment
  simply reproduces every
 character you input,
including all  s p a c e s!
\end{verbatim}
```

```
The verbatim environment
  simply reproduces every
 character you input,
including all  s p a c e s!
```

Note: once in the `verbatim` environment, the only command that will be recognised is `\end{verbatim}`. Any others will be output, verbatim! If this is an issue, then you can use the `alltt` package instead.

```
\begin{alltt}
Verbatim extended with the ability
to use normal commands.  Therefore, it
is possible to \emph{emphasize} words in
this environment, for example.
\end{alltt}
```

```
Verbatim extended with the ability
to use normal commands.  Therefore, it
is possible to emphasize words in
this environment, for example.
```

Remember to add `\usepackage{alltt}` to your preamble to use it though!

`Listing` Environment

This is also an extension of the verbatim environment. The extra functionality it provides is that it can add line numbers along side the text. The command: `\begin{listing}[`*`step`*`]{`*`first line`*`}`. The mandatory *first line* argument is for specifying which line the numbering shall commence. The optional *step* is the step between numbered lines (the default is 1, which means every line will be numbered).

To use this environment, add `\usepackage{moreverb}` to the document preamble.

`Quote` and `Quotation` Environment

There are two environments available for quoting passages within your own documents, with only a subtle difference between them. The `quote` environment is designed for a short quotation, or a series of small quotes, separated by blank lines. `quotation` on the other-hand, is for use with longer quotes, of more than one paragraph. Both are indented on either margin, and you will need to add your own quotation marks if you want them.

`Abstract` Environment

This should be fairly obvious. For academic papers, an abstract is always included at the beginning. It is typically formatted differently from the rest of the text --- indented either side, and a slightly smaller font. At least, that is what the default presentation is like. See [latextutorial2.html tutorial 2], or [tutorial7/paras.tex paras.tex] and [tutorial7/paras.pdf paras.pdf] for examples of using abstract.

`Verse` Environment

This environment is intended for displaying poetry. I can't really see many people using this within a typical document, however, it won't hurt to briefly explain its usage. As always, as an environment, it needs to `\begin` and `\end`. Once in, new stanzas are created with a blank line, and new lines within a stanza are indicated using the newline command, `\\`. If a line takes up more than one line on the page, then all subsequent lines are indented until explicitly separated with \\.

See the [tutorial7/paras.tex paras.tex] and [tutorial7/paras.pdf paras.pdf] for examples of this in action.

List structures

Lists often appear in documents, especially academic, as their purpose is often to present information in a clear and concise fashion. List structures in Latex are simply environments which essentially come in three flavours: `itemize`, `enumerate` and `description`.

All lists follow the basic format:

```
\begin{list_type}

  \item The first item
  \item The second item
  \item The third etc \ldots

\end{list_type}
```

`Itemize`

This environment is for your standard bulleted list of items.

```
\begin{itemize}
  \item The first item
  \item The second item
  \item The third etc \ldots
\end{itemize}
```

- The first item
- The second item
- The third etc ...

Enumerate

The enumerate environment is for ordered lists, where by default, each item is numbered sequentially.

```
\begin{enumerate}
  \item The first item
  \item The second item
  \item The third etc \ldots
\end{enumerate}
```

1. The first item
2. The second item
3. The third etc ...

Description

The description environment is slightly different. You can specify the item label by passing it as an optional argument (although optional, it would look odd if you didn't include it!). Ideal for a series of definitions, such as a glossary.

```
\begin{description}
  \item[First] The first item
  \item[Second] The second item
  \item[Third] The third etc \ldots
\end{description}
```

First The first item

Second The second item

Third The third etc ...

Nested Lists

Latex will happily allow you to insert a list environment into an existing one (up to a depth of four). Simply begin the appropriate environment at the desired point within the current list. Latex will sort out the layout and any numbering for you.

```
\begin{enumerate}
  \item The first item
  \begin{enumerate}
    \item Nested item 1
    \item Nested item 2
  \end{enumerate}
  \item The second item
  \item The third etc \ldots
\end{enumerate}
```

1. The first item
   (a) Nested item 1
   (b) Nested item 2
2. The second item
3. The third etc ...

Customising Lists

Customising many things in Latex is not really within the beginners domain. Whilst not necessarily difficult, per se, because beginners are already overwhelmed with the array of commands and environments, moving on to more advanced topics runs the risk of confusion.

However, since the tutorial is on formatting, then I shall still include a brief guide on customising lists. Feel free to skip!

Customising Enumerated Lists

The thing people want to change most often with Enumerated lists are the counters. Therefore, to go any further, a brief introduction to Latex *counters* is required. Anything that Latex automatically numbers, such as section headers, figures, and itemised lists, there is a counter associated with it that controls the numbering. Each counter also has a default format that dictates how it is displayed whenever Latex needs to print it. Such formats are specified using internal Latex commands:

Command	Example
`\arabic`	1, 2, 3 ...
`\alph`	a, b, c ...
`\Alph`	A, B, C ...
`\roman`	i, ii, iii ...
`\Roman`	I, II, III ...
`\fnsymbol`	Aimed at footnotes (see below), but prints a sequence of symbols.

There are four individual counters that are associated with itemised lists, each one represents the four possible levels of nesting, which are called: `enumi`, `enumii`, `enumiii`, `enumiv`. Each counter entity holds various bits of information about itself. To get to the numbered element, simply use `\the` followed immediately (ie no space) by the name of the counter, eg `\theenumi`. This is often referred to as the *representation* of a counter.

Now, that's most of the technicalities out of the way. To make changes to the formatting of a given level: `\renewcommand{\`*representation*`}{\`*format_command*`{`*counter*`}}`. Admittedly, the generic version is not that clear, so a couple of examples will clarify:

```
%Redefine the first level
\renewcommand{\theenumi}{\Roman{enumi}}
\renewcommand{\labelenumi}{\theenumi}

%Redefine the second level
\renewcommand{\theenumii}{\Alph{enumii}}
\renewcommand{\labelenumii}{\theenumii}
```

The method used above first explicitly changes the format used by the counter. However, the element that controls the label needs to be updated to reflect the change, which is what the second line does. Another way to achieve this result is this:

```
\renewcommand{\labelenumi}{\Roman{enumi}}
```

This simply redefines the appearance of the label, which is fine, providing that you do not intend to cross-reference to a specific item within the list, in which case the reference will be printed in the previous format. This issue does not arise in the first example.

Customising Itemised Lists

Itemised lists are not as complex as they do not need to count. Therefore, to customise, you simply change the labels. The itemize labels are accessed via `\labelitemi`, `\labelitemii`, `\labelitemiii`, `\labelitemiv`, for the four respective levels.

```
\renewcommand{\labelitemi}{\textgreater}
```

The above example would set the labels for the first level to a greater than (>) symbol. Of course, the text symbols available in Latex are not every exciting. Why not use one of the ZapfDingbat symbols, as described in the [#symbols symbols] section. Or use a mathmatical symbol:

```
\renewcommand{\labelitemi}{$\star$}
```

Footnotes

Footnotes are a very useful way of providing extra information to the reader. It is normally non-essential, and so can be placed at the bottom of the page, which means the main body remains concise.

The footnote facility is easy to use. The command you need is: `\footnote{`*`text`*`}`. Do not leave a space between the command the word where you wish the footnote marker to appear, otherwise Latex will process that space and will leave the output not looking as intended.

```
Creating a footnote is easy.\footnote{An example footnote.}
```

Latex will obviously take care of typesetting the footnote at the bottom of the page. Each footnote is numbered sequentially - a process, as you should have guessed by now is automatically done for you.

It is possible to customise the footnote marking. By default, they are numbered sequentially (Arabic). However, without going too much into the mechanics of Latex at this point, it is possible to change using the following command (of which needs to be placed at the beginning of the document, or at least before the first footnote command is issued).

`\renewcommand{\thefootnote}{\arabic{footnote}}`	Arabic numerals, e.g., 1, 2, 3...
`\renewcommand{\thefootnote}{\roman{footnote}}`	Roman numerals (lowercase), e.g., i, ii, iii...
`\renewcommand{\thefootnote}{\Roman{footnote}}`	Roman numerals (uppercase), e.g., I, II, III...
`\renewcommand{\thefootnote}{\alph{footnote}}`	Alphabetic (lowercase), e.g., a, b, c...
`\renewcommand{\thefootnote}{\Alph{footnote}}`	Alphabetic (uppercase), e.g., A, B, C...
`\renewcommand{\thefootnote}{\fnsymbol{footnote}}`	A sequence of nine symbols (try it and see!)

Margin notes

I dare say that this is a commonly used function, however, I thought I'd include it anyway, as it is simple to use command. `\marginpar{`*`margin text`*`}` will position the enclosed text on to the outside margin. To swap the default side, issue `\reversemarginpar` and that move margin notes to the opposite side.

Summary

Phew! What a busy tutorial! A lot of material was covered here, mainly because formatting is such a broad topic. Latex is so flexible that we actually only skimmed the surface, as you can have much more control over the presentation of your document if you wish. Having said that, one of the purposes of Latex is to take away the stress of having to deal with the physical presentation yourself, so you need not get too carried away!

This chapter uses material from Andy Roberts' Getting to grips with Latex with permission from the author.

11 Page Layout

Latex and the document class will normally take care of page layout issues for you. For submission to an academic publication, this entire topic will be out of your hands, as the publishers want to control the presentation. However, for your own documents, there are some obvious settings that you may wish to change: margins, page orientation and columns, to name but three. The purpose of this tutorial is to show you how to configure your pages.

Page Dimensions

A page in Latex is defined by a myriad of internal parameters. Each parameter corresponds to the length of an element of the page, for example, `\paperheight` is the physical height of the page. The `layout` package provides the `\layout` macro to dynamically draw the page layout within a document (albeit, scaled down slightly!), also providing the values of many of the dimensions, which can be very insightful. View the [tutorial8/playout.pdf page layout (pdf)] before moving on.

```
\usepackage{layout}
```

It will not have been immediately obvious - because it doesn't really cause any serious problems - is that the default page size, for all of the standard document classes is *US letter*. This is shorter by 3/4 inch, and slightly wider by 1/4 inch, compared to A4 (which is the standard in the UK). As I said, it's not a great problem, and most printers will print the page without a hiccup. However, it is possible to specify alternative sizes.

```
\documentclass[a4paper]{article}
```

The above example illustrates how to pass the optional argument to the `\documentclass`, which will then modify the page dimensions accordingly. The standard document classes that are a part of Latex are built to be fairly generic, which is why you have the flexibility of specifying the page size. Other classes may have different options (or none at all). Normally, 3rd party classes come with some documentation to let you know.

Additionally, there are several packages designed to solve the problem of varying pages sizes, which override any defaults setup by the document class. Examples such as `a4` are rather specialised in preparing only one type of page. One of the most versatile packages for page layout needs is `geometry`. It will feature a number of times throughout this tutorial as it has many tricks up its sleeve! Anyway, to set the page size, add the following to your preamble:

```
\usepackage[a4paper]{geometry}
```

a4paper is just one of the many pre-defined page sizes built-in, other include: `a0paper, a1paper, ..., a6paper, b0paper, b1paper, ..., b6paper, letterpaper, legalpaper, executivepaper.`

Page Orientation

When you talk about changing page orientation, it usually means changing to landscape mode, since portrait is the default. I shall introduce two slightly different styles of changing orientation.

The first is for when you want all of your document to be in landscape from the very beginning. There are various packages available to achieve this, but the one I prefer is the `geometry` package. All you need to do is call the package, with *landscape* as an option:

```
\usepackage[landscape]{geometry}
```

Although, if you intend to use `geometry` to set your paper size, don't add the \usepackage commands twice, simply string all the options together, separating with a comma:

```
\usepackage[a4paper,landscape]{geometry}
```

The second method is for when you are writing a document in portrait, but you have some contents, like a large diagram or table that would be displayed better on a landscape page. However, you still want the consistency of your headers and footers appearing the same place as the other pages.

The `lscape` package is for this very purpose. It supplies a `landscape` environment, and anything inside is basically rotated. No actual page dimensions are changed. This approach is applicable to books or reports than to typical academic publications.

These margins are a bit narrow!

Readers from a word processing background are probably all thinking why there is so much white space surrounding the text. There is a good reason, and it's all down to readability. Have a look in a few books, and pick a few lines at random. Count the number of characters per line. I bet the average is about 66. Studies have shown that it's easier to read text when there are 60-70 characters per line - and it would seem that 66 is the optimal number. Therefore, the page margins are set to ensure that readability remains as good as possible. White space is often left in the inner margin for the assumption that the document will be bound, and so space must be allowed for this.

Regardless of all this, should you wish to manipulate the margin parameters, then once again, `geometry` is the way to go. (However, before I go on, you may like to simply try the `a4wide` package first, which outputs wider lines, with no need to configure margin lengths). You could directly change the relevant parameters with the `\setlength` command, however, the following method is much easier to remember.

Unsurprisingly, there are four possible parameters for page margins that can be set with `geometry`: *top, bottom, left, right*.

`\usepackage[top=`*length*`, bottom=`*length*`, left=`*length*`, right=`*length*`]{geometry}`

Simply substitute your desired length (e.g,. `3cm`) for each parameter you want to change.

Page Styles

Page styles in Latex terms refers not to page dimensions, but to the running headers and footers of a document. You may have noticed that in every example document produced for these tutorials, each page has its respective number printed at the bottom, even though not a single command has been issued to tell Latex to do it! That is because the document class in use has already defined the page style for you. Of course, you don't need to stick with the defaults if you don't want to.

There are two commands at your disposal for changing the page style. `\pagestyle{`*style*`}` will apply the specified style to the current and all subsequent pages. `\thispagestyle{`*style*`}` will only affect the current page. The possible styles are:

empty	Both header and footer are clear
plain	Header is clear, but the footer contains the page number.
headings	Header displays page number and other information which the document class deems important, e.g., section headers.
myheadings	Similar to above, however, is possible to control the information in the header.

An issue to look out for is that the major sectioning commands (`\part`, `\chapter` or `\maketitle`) specify a `\thispagestyle{plain}`. So, if you wish to suppress all styles by inserting a `\pagestyle{empty}` at the beginning of your document, then the style command at each section will override your initial rule, for those pages only. To achieve your intended result, you will have to follow the offending commands with `\thispagestyle{empty}`.

Customising with `fancyhdr`

The `fancyhdr` package is by far the most convenient way of controlling the appearance of your headers and footers. It is very versatile, and I am only going to give you a taste of what you can do. For a more complete guide, the author of the package produced this documentation.

To begin, add the following lines to your preamble:

```
\usepackage{fancyhdr}
\pagestyle{fancy}
```

Both the header and footer are comprised of three elements each according to its horizontal position (left, centre or right). To set their values, the following commands are available:

`\lhead[`*lh-even*`]{`*lh-odd*`}` `\lfoot[`*lf-even*`]{`*lf-odd*`}`

`\chead[`*ch-even*`]{`*ch-odd*`}` `\cfoot[`*cf-even*`]{`*cf-odd*`}`

`\rhead[`*rh-even*`]{`*rh-odd*`}` `\rfoot[`*rf-even*`]{`*rf-odd*`}`

Hopefully, the behaviour of the above commands is fairly intuitive: if it has *head* in it, it affects the head etc, and obviously, *l*, *c* and *r* means left, centre and right respectively. Documents can be either one- or two-sided. Articles are by default one-sided, books are two-sided. Two-sided documents differentiate the left (even) and right (odd) pages, whereas one-sided do not. An example:

```
\fancyhead{}
```

```
\fancyfoot{}

\lhead{Andrew Roberts}
\rhead{\today}
\rfoot{\thepage}
```

It is often necessary to clear any defaults or a previous style definition, and the first two lines of the above example will do this. The commands are an alternative interface to customising the headers/footers that fancyhdr offers, and so by not passing anything to them, it assumes that you want it all blank.

The result of these commands will put my name at the top left, todays date at the top right, and the current page number at the bottom right of the page. Even if the document class was two-sided, because no optional text has been supplied for the even pages, the style will be used for all pages. For two-sided, it's common to mirror the style of opposite pages, you tend to think in terms of *inner* and *outer*. So, the same example as above for two-sided is:

```
\lhead[Andrew Roberts]{}
\rhead[]{Andrew Roberts}
\lhead[]{\today}
\rhead[\today]{}
\lfoot[\thepage]{}
\rfoot[]{\thepage}
```

This is effectively saying my name is top outer, todays date is top inner, and current page number is bottom outer.

NB. If you want to make the ***article class two-sided****, use* **`\documentclass[twoside]{article}`**.

Page n of m

Some people like to put the current page number in context with the whole document. Latex only provides access to the current page number, however, you can use the `lastpage` environment to find the total number of pages, like this:

```
\usepackage{lastpage}
     ...
     \cfoot{\thepage\ of \pageref{LastPage}}
```

Note the capital letters. Also, add a backslash after \thepage to ensure adequate space between the page number and 'of'. And do recall, when using references, you have to run latex an extra time to resolve the cross-references.

Multi-column Pages

It is common to see articles and conference proceedings formatted with two columns of text. However, such publishers will usually provide you with their own document class, which automatically implements this format, without you having to do anything. It is very easy to format your page in this

way. If you are using a standard Latex document class, then you can simply pass the optional argument *twocolumn* to the document class: `\documentclass[twocolumn]{article}` which will give the desired effect.

While this simple addition will do the job 9 out of 10 times, it is widely acknowledged that there are many limitations of this approach, and that the `multicol` package is much more useful for handling multiple columns. It has several advantages:

- Can support up to ten columns.
- Implements a *multicol* environment, therefore, it is possible to mix the number of columns within a document.
- Additionally, the environment can be nested inside other environments, such as `figure`.
- `Multicol` outputs *balanced* columns, whereby the columns on the final page will be of roughly equal length.
- Vertical rules between columns can be customised.
- Column environments can be easily customised locally or globally.

Floats are not fully supported by this environment. It can only cope if you use the starred forms of the float commands (eg `\begin{figure*}`) which makes the float span all columns. This is not hugely problematic, since floats of the same width as a column may be too small, and you would probably want to span them anyway.

To create a typical two-column layout:

```
\begin{multicols}{2}
  lots of text
  \ldots
\end{multicols}
```

The parameter `\columnseprule` holds the width of the vertical rules. By default, the lines are omitted as this parameter is set to a length of 0pt. Do the following before the beginning of the environment:

```
\setlength{\columnseprule}{1pt}
```

This will draw a thin line of 1pt in width. A thick line would not look very pleasing, however, you are free to put in any length of your choosing. Also, to change the horizontal space in between columns (the default is set at 10pt, which is quite narrow) then you need to change the \columnsep parameter:

```
\setlength{\columnsep}{20pt}
```

Manual Page Formatting

There may be instances, especially in very long documents, such as books, that Latex will not get all page breaks looking as good as it could. It may, therefore, be necessary to manually tweak the page formatting. Of course, you should only do this at the very final stage of producing your document, once all the content is complete. Latex offers the following:

`\newline`	Breaks the line at the point of the command.
`\linebreak[number]`	Breaks the line at the point of the command, but also stretches the line to the margin. If the optional number argument is supplied, you convert the command from a demand to a request. The number must be between 0-4, with 4 being the most insistent.
`\newpage`	Ends the current page
`\pagebreak[number]`	Breaks the current page at the point of the command. If the optional number argument is supplied, you convert the command from a demand to a request. The number must be between 0-4, with 4 being the most insistent.
`\nopagebreak[number]`	Stops the page being broken at the point of the command. If the optional number argument is supplied, you convert the command from a demand to a request. The number must be between 0-4, with 4 being the most insistent.
`\clearpage`	Ends the current page and causes any floats encountered in the input, but yet to appear, to be printed.

Summary

This tutorial is relatively short, largely due to the fact that the whole Latex ethos is concentrate on the content, and let Latex (and/or other typographers who have developed suitable document classes) decide on the best presentation. The next step to achieve greater control of page layout is to set about designing your own class. Unfortunately, that is not a straightforward task, and is often best left to the professionals!

This chapter uses material from Andy Roberts' Getting to grips with Latex with permission from the author.

12 MATHEMATICS

One of the greatest motivating forces for Donald Knuth when he began developing the original TeX system was to create something that allowed simple construction of mathematical formulae, whilst looking professional when printed. The fact that he succeeded was most probably why Tex (and later on, Latex) became so popular within the scientific community. Regardless of the history, typesetting mathematics is one of Latex's greatest strengths. However, it is also a large topic due to the existence of so much mathematical notation.

Maths environments

Latex needs to know beforehand that the subsequent text does in fact contain mathematical elements. This is because Latex typesets maths notation differently than normal text. Therefore, special environments have been declared for this purpose. They can be distinguished into two categories depending on how they are presented:

- *text* - text formulae are displayed inline, that is, within the body of text were it is declared. e.g., I can say that $a + a = 2a$ within this sentence.
- *displayed* - displayed formulae are separate from the main text.

As maths require special environments, there are naturally the appropriate environment names you can use in the standard way. Unlike most other environments however, there are some handy shorthands to declaring your formulae. The following table summarises them:

Type	Environment	Latex shorthand	Tex shorthand
Text	`\begin{math}...\end{math}`	`\(...\)`	`$...$`
Displayed	`\begin{displaymath}...\end{displaymath}`	`\[...\]`	`$$...$$`

All of these methods are safe to use except `$$...$$`, which can have different spacing to `\[...\]` and which does not support equation numbers on the left (class option `fleqn`).

Additionally, there is second possible environment for the *displayed* type of formulae: `equation`. The difference between this and `displaymath` is that `equation` also adds sequential equation numbers by the side.

Symbols

Mathematics has lots and lots of symbols! If there is one aspect of maths that is difficult in Latex it is trying to remember how to produce them. There are of course a set of symbols that can be accessed directly from the keyboard:

```
+ - = ! / ( ) [ ] < > | ' :
```

Beyond those listed above, distinct commands must be issued in order to display the desired

symbols. And there are *a lot!* Greek letters, set and relations symbols, arrows, binary operators, etc. Too many to remember, and in fact, they would overwhelm this tutorial if I tried to list them all. Therefore, for a complete reference document, try [tutorial9/symbols.pdf symbols.pdf]. We will of course see some of these symbols used throughout the tutorial.

Fractions

To create a fraction, you must use the `\frac{`*numerator*`}{`*denominator*`}` command. (For those who need their memories refreshed, that's the *top* and *bottom* respectively!) You can also embed fractions within fractions, as shown in the examples below:

`\frac{x+y}{y-z}` $\frac{x+y}{y-z}$

`\frac{\frac{1}{x}+\frac{1}{y}}{y-z}` $\frac{\frac{1}{x}+\frac{1}{y}}{y-z}$

Powers and indices

Powers and indices are mathematically equivalent to superscripts and subscripts in normal text mode. The carat (^) character is used to raise something, and the underscore (_) is for lowering. How to use them is best shown by example:

Power	`x^n`	x^n
	`x^{2n}`	x^{2n}
Index	`n_i`	n_i
	`n_{ij}`	n_{ij}

Note: if more than one character is to be raised (or lowered) then you must group them using the curly braces (`{ }`).

Also, if you need to assign both a power and an index to the same entity, then that is achieved like this: `x^{2i}_{3j}` (or `x_{3j}^{2i}`, order is not significant).

x^{2i}_{3j}

Roots

Typically, for the majority of times, you are after the square root, which is done easily using the following command: `\sqrt{x}`. However, this can be generalised to produce a root of any magnitude:

`\sqrt[n]{arg}` $\sqrt[n]{x}$

Latex will automatically ensure that the size of the root notation adjusts to the size of the contents.

The *n* is optional, and without it will output a square root. Also, regardless of the size of root you're after, e.g., *n*=3, you still use the `\sqrt` command.

Brackets

The use of brackets soon becomes important when dealing with anything but the most trivial equations. Without them, formulae can become ambiguous. Also, special types of mathematical structures, such as matrices typically rely on brackets to enclose them.

You may recall that you already have the `( ) [ ]` symbols at your disposal, which should be more than adequate for most peoples' needs. So why the need for a dedicated section? Well, I think that can be shown by example:

`\[ (\frac{x^2}{y^3}) \]` $(\frac{x^2}{y^3})$

`\[ \left(\frac{x^2}{y^3}\right) \]` $\left(\frac{x^2}{y^3}\right)$

The first example shows what would happen if you used the standard bracket characters. As you can see, they would be fine for an equation a simple equation that remained on a single line (e.g., (3 + 2) x (10-3) = 35) but not for equations that have greater vertical size, such as those using fractions. The second example illustrates the Latex way of coping with this problem.

The `\left...` and `\right...` commands provide the means for automatic sizing of brackets. You must enclose the expression that you want in brackets with these commands. The dots after the command should be replaced with one of the characters depending on the style of bracket you want.

There are in fact many more possible symbols that can be used, but are somewhat uncommon. Please check out table 5 from the symbols reference ([tutorial9/symbols.pdf symbols.pdf]) for the rest.

Alternatively, check out the nath package, which provides auto-scaling delimiters.

Matrices

Latex doesn't have a specific matrix command to use. It instead has a slightly more generalised environment called `array`. The `array` environment is basically equivalent to the `table` environment. Arrays are very flexible, and can be used for many purposes, but we shall focus on matrices. You can use the array to arrange and align your data as you want, and then enclose it with appropriate left and right brackets, and this will give you your matrix. For a simple 2x2 matrix:

```
\[ \left[
  \begin{array}{ c c }
     1 & 2 \\
     3 & 4
  \end{array} \right]
\]
```

If you often include a lot of maths in your documents, then you will probably find that you wish to have slightly more control over presentation issues. Some of the topics covered make writing equations more complex - but who said typesetting mathematics was easy?!

Adding text to equations

I doubt it will be every day that you will need to include some text within an equation. However, sometimes it needs to be done. Just sticking the text straight in the maths environment won't give you the results you want. For example:

```
\[
  50 apples \times 100 apples = lots of apples
\]
```

$$50apples \times 100apples = lotsofapples$$

There are two noticeable problems. Firstly, there are no spaces between numbers and text, nor spaces between multiple words. Secondly, the words don't look quite right --- the letters are more spaced out than normal. Both issues are simply artifacts of the maths mode, in that it doesn't expect to see words. Any spaces that you type in maths mode are ignored and Latex spaces elements according to its own rules. It is assumed that any characters represent variable names. To emphasise that each symbol is an individual, they are not positioned as closely together as with normal text.

There are a number of ways that text can be added properly. The typical way is to wrap the text with the `\mbox{...}` command. This command hasn't been introduced before, however, it's job is basically to create a text box just wide enough to contain the supplied text. Text within this box cannot be broken across lines. Let's see what happens when the above equation code is adapted:

$$50\text{apples} \times 100\text{apples} = \text{lots of apples}$$

The text looks better. However, there are no gaps between the numbers and the words. Unfortunately, you are required to explicitly add these. There are many ways to add spaces between maths elements, however, for the sake of simplicity, I find it easier, in this instance at least, just to literally add the space character in the affected `\mbox`(s) itself (just before the text.)

```
\[
  50 \mbox{ apples} \times 100 \mbox{ apples} =
  \mbox{lots of apples}
\]
```

$$50 \text{ apples} \times 100 \text{ apples} = \text{lots of apples}$$

Formatted text

Using the `\mbox` is fine and gets the basic result. Yet, there is an alternative that offers a little more flexibility. You may recall from [latextutorial7.html Tutorial 7 (Formatting)], the introduction of font formatting commands, such as `\textrm`, `\textit`, `\textbf`, etc. These commands format the argument accordingly, e.g., `\textbf{bold text}` gives **bold text**. These commands are equally valid within a maths environment to include text. The added benefit here is that you can have better control over the font formatting, rather than the standard text achieved with `\mbox`.

```
\begin{equation}
  50 \textrm{ apples} \times 100 \textbf{ apples} =
  \textit{lots of apples}
\end{equation}
```

$$50\text{apples} \times 100\mathbf{apples} = \textit{lotsofapples}$$

However, as is the case with Latex, there is more than one way to skin a cat! There are a set of formatting commands very similar to the font formatting ones just used, except they are aimed specifically for text in maths mode. So why bother showing you `\textrm` and co if there are equivalents for maths? Well, that's because they are subtly different. The maths formatting commands are:

Command	Format
`\mathrm{...}`	Roman
`\mathit{...}`	Italic
`\mathbf{...}`	Bold
`\mathsf{...}`	Sans serif
`\mathtt{...}`	Typewriter
`\mathcal{...}`	Calligraphy

The maths formatting commands can be wrapped around the entire equation, and not just on the textual elements: they only format letters, numbers, and uppercase Greek, and the rest of the maths syntax is ignored. So, generally, it is better to use the specific maths commands if required. Note that the calligraphy example gives rather strange output. This is because for letters, it requires upper case characters. The reminding letters are mapped to special symbols.

Changing text size of equations

A simple approach is to utilise the predefined sizes for maths elements:

`\displaystyle`	Size for equations in display mode
`\textstyle`	Size for equations in text mode
`\scriptstyle`	Size for first sub/superscripts
`\scriptscriptstyle`	Size for subsequent sub/superscripts

A classic example to see this in use is typesetting continued fractions. The following code provides an example.

```
\begin{equation}
  x = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + a_4}}}
\end{equation}
```

As you can see, as the fractions continue, they get smaller (although they will not get any smaller as in this example, they have reached the `\srciptscriptstyle` limit. If you wanted to keep the size consistent, you could declare each fraction to use the display style instead, e.g.,

```
\begin{equation}
  x = a_0 + \frac{1}{\displaystyle a_1
          + \frac{1}{\displaystyle a_2
          + \frac{1}{\displaystyle a_3 + a_4}}}
\end{equation}
```

Another approach is to use the `\DeclareMathSizes` command to select your preferred sizes. You can only define sizes for `\displaystyle}`, `\textstyle`, etc. One potential downside is that this command sets the global maths sizes, as it can only be used in the document preamble.

However, it's fairly easy to use: `\DeclareMathSizes{ds}{ts}{ss}{sss}`, where *ds* is the *display size*, *ts* is the *text size*, etc. The values you input are assumed to be point (pt) size.

NB the changes only take place if the value in the first argument matches the current document text size. It is therefore common to see a set of declarations in the preamble, in the event of the main font being changed. E.g.,

```
\DeclareMathSizes{10}{18}{12}{8}   % For size 10 text
\DeclareMathSizes{11}{19}{13}{9}   % For size 11 text
\DeclareMathSizes{12}{20}{14}{10}  % For size 12 text
```

Multi-lined equations

`eqnarray` environment

One way to the various elements neatly aligned is to use a table, and place equations inline. Let's have a go:

```
\begin{tabular}{ r l }
\(10xy^2+15x^2y-5xy\) & \(= 5\left(2xy^2+3x^2y-xy\right)\) \\
 & \(= 5x\left(2y^2+3xy-y\right)\) \\
 & \(= 5xy\left(2y+3x-1\right)\)
 \end{tabular}
```

That doesn't look too bad. Perhaps the extra space between the equation on the left and the equals sign being larger than the gap on the right doesn't look perfect. That could be rectified by adding an extra column and putting the equals sign in the central one:

```
\begin{tabular}{ r c l }
  \(10xy^2+15x^2y-5xy\) & \(=\) & \(5\left(2xy^2+3x^2y-xy\right)\) \\
   & \(=\) & \(5x\left(2y^2+3xy-y\right)\) \\
```

```
   & \(=\) & \(5xy\left(2y+3x-1\right)\)
\end{tabular}
```

Looking better. Another issue is that the vertical space between the rows makes makes the equations on the right-hand side look a little crowded. It would be nice to add a little space --- make the rows in the table a little taller. (You need to add `\usepackage{array}` to your preamble for this to work.)

```
% add some extra height. Needs the array package. See preamble.
\setlength{\extrarowheight}{0.3cm}
```

However, by this stage, we've had to do quite a bit of extra leg work, and there is still one important disadvantage at the end of it. You can't add equation numbers. Because once you are within the `tabular` environment, you can only use the inline type of maths display. For equation numbers, you need to use the display mode `equation` package, but you can't in this instance. This is where `eqnarray` becomes extremely useful.

`eqnarray`, as the name suggests borrows from the `array`} package which is basically a simplified `tabular` environment. The `array` package was introduced in [latextutorial9.html tutorial 9 (Mathematics I)] for producing matrices. Here is how to use `eqnarray` for this example:

```
\begin{eqnarray}
  10xy^2+15x^2y-5xy & = & 5\left(2xy^2+3x^2y-xy\right) \\
   & = & 5x\left(2y^2+3xy-y\right) \\
   & = & 5xy\left(2y+3x-1\right)
\end{eqnarray}
```

As you can see, everything is laid out nicely and as expected. Well, except that each row within the array has been assigned its own equation number. Whilst this feature is useful in some instances, it is not required here --- just the one number will do! To suppress equation numbers for a given row, add a `\nonumber` command just before the end of row command (`\\`).

```
\begin{eqnarray}
  10xy^2+15x^2y-5xy & = & 5\left(2xy^2+3x^2y-xy\right) \nonumber \\
   & = & 5x\left(2y^2+3xy-y\right) \nonumber \\
   & = & 5xy\left(2y+3x-1\right)
\end{eqnarray}
```

If you don't care for equation number at all, then rather than adding `\\nonumber` to every row, use the starred version of the environment, i.e., `\begin{eqnarray*} ... \end{eqnarray*}`.

NB There is a limit of 3 columns in the `eqnarray` environment. If you need anymore flexibility, you are best advised to seek the AMS Maths packages.

AMS Math package

The AMS (American Mathematical Society) mathematics package contains a lot of powerful features. In the case of multiline equations, one major problem with the `eqnarray` and `eqnarray*` environments is that they produce uneven spacing over entire documents. In other words, while

equations are lined with each other within every block, different blocks don't have any relationship to each other. One remedy to this problem is to use the `align` environment.[5] The AMS-math package is used by first declaring, in the preamble,

\usepackage{amsmath}

The `align` environment can then be used as follows:

```
\begin{align}
  10xy^2+15x^2y-5xy & = & 5\left(2xy^2+3x^2y-xy\right) \nonumber \\
   & = & 5x\left(2y^2+3xy-y\right) \nonumber \\
   & = & 5xy\left(2y+3x-1\right)
\end{align}
```

which would, in this case, produce individually numbered equations.

Breaking up long equations

Let us first see an example of a long equation. (Note, when you view the example document for this topic, you will see that this equation is in fact wider than the text width. It's not as obvious on this webpage!)

Latex doesn't break long equations to make them fit within the margins as it does with normal text. It is therefore up to you to format the equation appropriately (if they overrun the margin.) This typically requires some creative use of an `eqnarray` to get elements shifted to a new line and to align nicely. E.g.,

```
\begin{eqnarray*}
  \left(1+x\right)^n & = & 1 + nx + \frac{n\left(n-1\right)}{2!}x^2 \\
  & & + \frac{n\left(n-1\right)\left(n-2\right)}{3!}x^3 \\
  & & + \frac{n\left(n-1\right)\left(n-2\right)\left(n-3\right)}{4!}x^4 \\
  & & + \ldots
\end{eqnarray*}
```

It may just be that I'm more sensitive to these kind of things, but you may notice that from the 2nd line onwards, the space between the initial plus sign and the subsequent fraction is (slightly) smaller than normal. (Observe the first line, for example.) This is due to the fact that Latex deals with the + and - signs in two possible ways. The most common is as a binary operator. When two maths elements appear either side of the sign, it is assumed to be a binary operator, and as such, allocates some space either side of the sign. The alternative way is a sign designation. This is when you state whether a mathematical quantity is either positive or negative. This is common for the latter, as in maths, such elements are assumed to be positive unless a - is prefixed to it. In this instance, you want the sign to appear close to the appropriate element to show their association. It is this interpretation that Latex has opted for in the above example.

To add the correct amount of space, you can add an *invisible* character using `{}`, as illustrated here:

```
\begin{eqnarray*}
  \left(1+x\right)^n & = & 1 + nx + \frac{n\left(n-1\right)}{2!}x^2 \\
  & & {} + \frac{n\left(n-1\right)\left(n-2\right)}{3!}x^3 \\
  & & {} + \frac{n\left(n-1\right)\left(n-2\right)\left(n-3\right)}{4!}x^4 \\
  & & {} + \ldots
\end{eqnarray*}
```

Alternatively, you could avoid this issue altogether by leaving the + at the end of the previous line rather at the beginning of the current line.

There is another convention of writing long equations that Latex supports. This is the way I see long equations typeset in books and articles, and admittedly is my preferred way of displaying them. Sticking with the `eqnarray` approach, using the `\lefteqn{...}` command around the content before the = sign gives the following result:

```
\begin{eqnarray*}
  \lefteqn{\left(1+x\right)^n = } \\
  & & 1 + nx + \frac{n\left(n-1\right)}{2!}x^2 + \\
  & & \frac{n\left(n-1\right)\left(n-2\right)}{3!}x^3 + \\
  & & \frac{n\left(n-1\right)\left(n-2\right)\left(n-3\right)}{4!}x^4 + \\
  & & \ldots
\end{eqnarray*}
```

Notice that the first line of the `eqnarray` contains the `\lefteqn` only. And within this command, there are no column separators (`&`). The reason this command displays things as it does is because the `\lefteqn` prints the argument, however, tells Latex that the width is zero. This results in the first column being empty, with the exception of the inter-column space, which is what gives the subsequent lines their indentation.

Controlling horizontal spacing

Latex is obviously pretty good at typesetting maths --- it was one of the chief aims of the core Tex system that Latex extends. However, it can't always be relied upon to accurately interpret formulae in the way you did. It has to make certain assumptions when there are ambiguous expressions. The result tends to be slightly incorrect horizontal spacing. In these events, the output is still satisfactory, yet, any perfectionists will no doubt wish to *fine-tune* their formulae to ensure spacing is correct. These are generally very subtle adjustments.

There are other occasions where Latex has done its job correctly, but you just want to add some space, maybe to add a comment of some kind. For example, in the following equation, it is preferable to ensure there is a decent amount of space between the maths and the text.

```
\[f(n) = \left\{
\begin{array}{l l}
  n/2 & \quad \mbox{if $n$ is even}\\
  -(n+1)/2 & \quad \mbox{if $n$ is odd}\\ \end{array} \right. \]
```

Latex has defined two commands that can be used anywhere in documents (not just maths) to insert some horizontal space. They are `\quad` and `\qquad`

A `\quad` is a space equal to the current font size. So, if you are using an 11pt font, then the space provided by `\quad` will also be 11pt (horizontally, of course.) The `\qquad` gives twice that amount. As you can see from the code from the above example, `\quad`s were used to add some separation between the maths and the text.

OK, so back to the fine tuning as mentioned at the beginning of the document. A good example would be displaying the simple equation for the indefinite integral of *y* with respect to *x*:

$$\int y \, \mathrm{d}x$$

If you were to try this, you may write:

```
\[ \int y \mathrm{d}x \]
```

$$\int y \mathrm{d}x$$

However, this doesn't give the correct result. Latex doesn't respect the whitespace left in the code to signify that the *y* and the d*x* are independent entities. Instead, it lumps them altogether. A `\quad` would clearly be overkill is this situation --- what is needed are some small spaces to be utilised in this type of instance, and that's what Latex provides:

Command	Description	Size
`\,`	small space	3/18 of a quad
`\:`	medium space	4/18 of a quad
`\;`	large space	5/18 of a quad
`\!`	negative space	-3/18 of a quad

NB you can use more than one command in a sequence to achieve a greater space if necessary.

So, to rectify the current problem:

```
\int y\, \mathrm{d}x
```

$$\int y \, \mathrm{d}x$$

```
\int y\: \mathrm{d}x
```

$$\int y \, \mathrm{d}x$$

```
\int y\; \mathrm{d}x
```

$$\int y \, \mathrm{d}x$$

The negative space may seem like an odd thing to use, however, it wouldn't be there if it didn't have *some* use!

The matrix-like expression for representing binomial coefficients is too padded. There is too much space between the brackets and the actual contents within. This can easily be corrected by adding a few negative spaces after the left bracket and before the right bracket.

```
\[\left(\!\!\!
  \begin{array}{c}
    n \\
    r
  \end{array}
  \!\!\!\right) = {^n}C_r = \frac{n!}{r!(n-r)!}
\]
```

Introduction to AMS maths

This section is still under development.

Some very useful LaTeX commands are available using the package named "amsmath". It can be loaded on most LaTeX installations by including this command in the preamble of your document (that is, before the \begin{document} command):

```
\usepackage{amsmath}
```

One very useful command made available through this package permits the alignment of, for example, equals signs (=) on successive lines of an equation. Here is an example:

```
\begin{align}
    x &= a + (b + a) \\
    &= 2a + b.
\end{align}
```

The ampersand symbol (&) is used to specify the points in each line that are supposed to be aligned vertically, but they do not appear in the result, which looks like this:

$$x = a + (b + a) \tag{1}$$
$$= 2a + b. \tag{2}$$

The equation numbers are provided by default. They can be omitted altogether by replacing the word "`align`" with the word "`align*`" at the beginning and end of the aligned equation specification:

```
\begin{align*}
    x &= a + (b + a) \\
    &= 2a + b.
\end{align*}
```

The result looks like this:

$$x = a + (b + a)$$
$$= 2a + b.$$

Alternatively, they can be suppressed on any particular line by adding the expression `\notag`, as shown here:

```
\begin{align}
    x &= a + (b + a) \notag \\
    &= 2a + b.
\end{align}
```

The result looks like this:

$$x = a + (b + a)$$
$$= 2a + b. \qquad (1)$$

Annotations for each line can be added by separating them from the equations using two ampersands (&&):

```
\begin{align}
    x &= a + (b + a) && \text{Axiom C} \\
    &= 2a + b && \text{Axioms A \& F}.
\end{align}
```

Summary

As you can begin to see, typesetting maths can be tricky at times. However, because Latex provides so much control, you can get professional quality mathematics typesetting for relatively little effort (once you've had a bit of practise, of course!). It would be possible to keep going and going with maths topics because it seems potentially limitless. However, with these two tutorials, you should be able to get along sufficiently.

This chapter uses material from Andy Roberts' Getting to grips with Latex with permission from the author.

13 Advanced Topics

Here are some topics that are not really necessary to write a proper document, but could help you making your life easier and giving you some details to modify.

Splitting the document into multiple files

As the typing goes on, your LaTeX file could get big and confusing, so it could be a good idea to split it into several other documents. For example, if you are writing a book, you could consider writing every chapter in its own file. LaTeX makes this very easy thanks to the command:

```
\input{filename.tex}
```

While processing the document, the compiler will simply read the content of *filename.tex* and put it within the document instead of the *input* command. This way you can put all the formatting options in your "root" document and then *input* other files containing only text and very basic commands such as *section* etc. The code will be much cleaner and more readable.

Creating your own package

As known you can easily include any package by the command:

```
\usepackage{packagename}
```

but what if you want to create your own?? It is very simple:

1. create a simple text file called *mypack.sty* and open it with any text editor
2. at the beginning of the text document just write

```
\ProvidesPackage{mypack}
```

note: it has to have the same name of the file without the extension

1. write whatever you want in it using all the LaTeX commands you know. Normally you should define new commands or import other packages.
2. import your new package with the known command

```
\usepackage{mypack}
```

the file *mypack.sty* and the LaTeX source you are compiling must be in the same directory. It will be like that all you have written within your package were within the document itself.

Using different paths

When referring to an external file in the LaTeX document source, if you just write a filename the

compiler will look for it in the same directory of the source. In general you can refer any document on your hardisk, using both relative and absolute paths. In general, you might want to make your source portable (to another computer or to a different location of you hardisk), so always use relative paths. A relative path is always defined in terms of the current directory where you are running the compiler, i.e. the directory where the source you are compiling is. LaTeX uses the standard *nix notation: with a simple dot **.** you refer to the current directory, and by two dots **..** you refer to the previous directory, that is the upper one in the file system three. By a slash / you can separate names of different directories. So by **./** you refer to the current directory, by **../** you refer to the previous directory, by **../../** you refer to two upper directories in the filesystem three. Writing

```
\input{./filename.tex}
```

will have *exactly* the same effect of writing

```
\input{filename.tex}
```

but if you want to put all your files in a different directory called *myfiles*, you can refer to that file by

```
\input{./myfiles/filename.tex}
```

so you can put your files however and wherever you want: you can refer them this way. Do not leave empty spaces in the filenames: they could cause an ambiguous behavior, use underscores _ instead.

It should be noted that LaTeX uses slash / even when on a Windows platform that normally uses backslash \.

14 Tips and Tricks

id est

If you simply use the form

i.e.

LaTeX will treat the periods as end of sentence periods. The correct syntax is

i.e.\

which tells LaTeX not to consider the period as an end of sentence period. This syntax results in a shorter space.

15 Useful Measurement Macros

A list of macros and their values

Units

First, we introduce the LaTeX measurement units. You can choose from a variety of units.

- pt - a point is 1/72 inch.
- mm - millimetre.
- cm - centimetre.
- in - inch.
- ex - roughly the height of an 'x' in the current font.
- em - roughly the width of an 'M' (note the uppercase) of the current font.

Length 'macros'

Some length commands are;

\baselineskip
: The normal vertical distance between lines in a paragraph

\baselinestretch
: Multiplies \baselineskip

\columnsep
: The distance between columns

\evensidemargin
: The margin for 'even' pages (think of a printed booklet)

\linewidth
: The width of a line in the local environment

\oddsidemargin
: The margin for 'odd' pages (think of a printed booklet)

\pagewidth
: The width of the page

\pageheight
: The height of the page

\parindent
: The normal paragraph indentation

\parskip
: The extra vertical space between paragraphs

\tabcolsep
: The default separation between columns in a tabular environment

\textheight
: The height of text on the page

\textwidth
: The width of the text on the page

\topmargin

The size of the top margin!

\unitlength

Units of lenght in Picture Environment

16 Useful Size Commands

The following type size commands, in order of increasing font size, are supported by LaTeX.

- \tiny
- \scriptsize
- \footnotesize
- \small
- \normalsize (default)
- \large
- \Large (capital "l")
- \LARGE (all caps)
- \huge
- \Huge (capital "h")

The default for \normalsize is 10-point, but it may differ for some Document Styles or their options. The actual size produced by these commands also depends on the Document Style and, in some styles, more than one of these size commands may produce the same actual size.

The new size takes effect immediately after the size command; if an entire paragraph or unit is set in a certain size, the size command should include the blank line or the \end{} which delimites the unit. (See Declarations.)

These commands cannot be used in math mode. However, part of a formula may be set in a different size by using an \mbox command containing the size command.

17 Glossary

List of commands (brackets "[]" are optional arguments and braces "{}" are required arguments) :

\space
: force ordinary space

\@
: following period ends sentence

\\[*][extra-space]
: new line

\,
: thin space, math mode

\;
: thick space, math mode

\:
: medium space, math mode

\!
: negative thin space, math mode

\-
: hyphenation; tabbing

\=
: set tab, see tabbing

\>
: tab, see tabbing

\<
: back tab, see tabbing

\+
: see tabbing

\'
: accent or tabbing

\`
: accent or tabbing

\|
: double vertical lines, math mode

\(
: start math environment

\)
: end math environment

\[
: begin displaymath environment

\]
: end displaymath environment

\addcontentsline{file}{sec_unit}{entry}
: adds an entry to the specified list or table

\addtocontents{file}{text}
: adds text (or formatting commands) directly to the file that generates the specified list or table

\addtocounter{counter}{value}

increments the counter

\address{Return address}

\addtolength{len-cmd}{len}

increments a length command, see Lengths

\addvspace

adds a vertical space of a specified height

\alph

causes the current value of a specified counter to be printed in alphabetic characters

\appendix

changes the way sectional units are numbered so that information after the command is considered part of the appendix

\arabic

causes the current value of a specified counter to be printed in Arabic numbers

\author

declares the author(s)

\backslash

prints a backslash

\baselineskip

a length command (see Lengths), which specifies the minimum space between the bottom of two successive lines in a paragraph

\baselinestretch

scales the value of \baselineskip

\bf

Boldface typeface

\bibitem

generates a labeled entry for the bibliography

\bigskipamount

\bigskip

equivalent to \vspace{\bigskipamount}

\boldmath

bold font in math mode

\cal

Calligraphic style in math mode

\caption

generate caption for figures and tables

\cdots

Centered dots

\centering

Used to center align LaTeX environments

\circle

\cite

Used to make citations from the provided bibliography

\cleardoublepage

\clearpage

\cline

\closing

\dashbox

\date

\ddots

\documentclass[options]{style}

Used to begin a latex document
\dotfill
\em
\ensuremath (LaTeX2e)
\fbox
\flushbottom
\fnsymbol
\footnote
\footnotemark
\footnotesize
\footnotetext
\frac
\frame
\framebox
\frenchspacing
\hfill
\hline
adds a line in a tabular environment. See \cline
\hrulefill
\hspace
\huge
\Huge (capital "h")
\hyphenation
\include
\includeonly
\indent
\input
Used to read in LaTex files
\it
Ilatilcs
\item
\kill
\label
\large
\Large (capital "l")
\LARGE (all caps)
\ldots
\left
\lefteqn
\line
\linebreak
\linethickness
\linewidth
\listoffigures
\listoftables
\location
\makebox
\maketitle
\markboth \markright
\mathcal

\mathop
\mbox
\medskip
\multicolumn
\multiput
\newcommand
\newcounter
\newenvironment
\newfont
\newlength
\newline
\newpage
\newsavebox
\newtheorem
\nocite
\noindent
\nolinebreak
\normalsize (default)
\nopagebreak
\not
\onecolumn
\opening
\oval
\overbrace
\overline
\pagebreak
\pagenumbering
\pageref
\pagestyle
\par
    starts a new paragraph
\parbox
\parindent
\parskip
\protect
\providecommand (LaTeX2e)
\put
\raggedbottom
\raggedleft
\raggedright
\raisebox
\ref
\renewcommand
\right
\rm
\roman
\rule
\savebox
\sbox
\sc

\scriptsize
\setcounter
\setlength
\settowidth
\sf
\shortstack
\signature
\sl
\small
\smallskip
\sqrt
\stackrel
\tableofcontents
\telephone
\textwidth
\textheight
\thanks
\thispagestyle
\tiny
\title
\today
\tt
\twocolumn
\typeout
\typein
\underbrace
\underline
\unitlength
\usebox
\usecounter
\value
\vbox{text}
: Encloses a paragraph's text to prevent it from running over a page break
\vdots
\vector
\verb
\vfill
\vline
\vphantom
\vspace

This page uses material from Dr. Sheldon Green's Hypertext Help with LaTeX.

18 History & Document Notes

Wikibook History

This book was created on 2004-01-31 and was developed on the Wikibooks project by the contributors listed in the next section. The latest version may be found at http://en.wikibooks.org/wiki/LaTeX.

PDF Information & History

This PDF was created on 2006-07-19 based on the 2006-07-18 version of the LaTeX Wikibook. A transparent copy of this document is available at http://en.wikibooks.org/wiki/LaTeX/Printable_version. An OpenDocument Text version of this PDF document and the template from which it was created is available upon request at http://en.wikibooks.org/wiki/User_talk:Hagindaz. A color textbook-format version of this document is available at http://en.wikibooks.org/wiki/Image:LaTeX.pdf.

Document Information

- Pages: 90
- Paragraphs: 1771
- Words: 26234
- Characters: 157095

19 Authors, Bibliography, & Image Credits

Principal Authors

- Withinfocus
- Jtwdog
- Orderud
- Igjimh
- Alejo2083
- Cameronc
- Snaxe920
- Basenga

This book uses material from Andy Roberts' Getting to grips with Latex with permission from the author.

This book uses material from Dr. Sheldon Green's Hypertext Help with LaTeX.

All Authors, number of edits

Title Page: Anonymous,10 · Withinfocus,8 · 3mta3,7 · Igjimh,6 · Elwikipedista,5 · Jtwdog,4 · Derbeth,2 · Kernigh,2 · Robert Horning,2 · Alejo2083,2 · Goodgerster,1 · Dmb,1 · Orderud,1

Introduction: Anonymous,21 · 3mta3,3 · Orderud,2 · Alejo2083,2 · Latexing,2 · Sgenier,1 · Igjimh,1 · Ævar Arnfjörð Bjarmason,1 · Withinfocus,1 · Elwikipedista,1 · Goodgerster,1 · Kernigh,1

Absolute Beginners: Anonymous,6 · Jtwdog,3 · Bilbo1507,2 · Dilaudid,1 · Alejo2083,1 · Krischik,1 · Jguk,1 · Withinfocus,1 · Tpr,1 · Snaxe920,1 · Igjimh,1

Basics: Anonymous,6 · Withinfocus,6 · 3mta3,2

Packages: Jtwdog,2 · Withinfocus,2 · Anonymous,1 · Orderud,1

Document Structure: Anonymous,13 · Jtwdog,5 · Snaxe920,4 · Orderud,3 · Igjimh,2
Dilaudid,1 · Withinfocus,1 · Nkour,1 · Bibliography Management: Anonymous,12 · Jtwdog,3 · Igjimh,2 · Orderud,1 · Withinfocus,1

Tables: Anonymous,9 · Jtwdog,4 · Igjimh,1 · Orderud,1 · Dilaudid,1 · Withinfocus,1

Importing Graphics: Anonymous,8 · Jtwdog,2 · NavarroJ,1 · Igjimh,1 · Orderud,1 · Dilaudid,1 · Alejo2083,1 · Withinfocus,1

Floats, Figures, and Captions: Anonymous,13 · Orderud,3 · Jtwdog,2 · Hansfn,2 · Robin,2 · Igjimh,2 · Dilaudid,1 · Alejo2083,1 · Basenga,1 · Withinfocus,1 · HenrikMidtiby,1

Page Layout: Anonymous,13 · Jtwdog,2 · Igjimh,1 · Orderud,1 · Withinfocus,1

Mathematics: Anonymous,25 · Juliusross,5 · Jtwdog,3 · Cameronc,3 · Orderud,1 · Dilaudid,1 · Hagindaz,1 · Basenga,1 · Withinfocus,1 · Robin,1 · Igjimh,1

Advanced Topics: Anonymous,3 · Alejo2083,1 · Jguk,1 · Withinfocus,1

Tips and Tricks: Anonymous,2 · Basenga,2 · Jguk,1 · Withinfocus,1

Links: Orderud,4 · Dmb,1 · Alejo2083,1 · Withinfocus,1

Useful Measurement Macros: Anonymous,11 · Withinfocus,2 · Igjimh,1 · Hokiehead,1

Useful Size Commands: Anonymous,5 · Withinfocus,2 · Igjimh,1

Glossary: Igjimh,11 · Anonymous,5 · Robert Horning,4 · Orderud,3 · Withinfocus,2 · Dmb,2 · IrfanAli,1

Image Credits

Introduction

- LaTeX diagram by Alessio Damato (GFDL)

Mathematics

- AlignedEquationStarred, AlignedEquationNotag, and AlignedEquation1 by Cameronc (GFDL)

All other images have been created by Jtwdog and released under GFDL.

20 GNU Free Documentation License

Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding

them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the

Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents

or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the

Free Software Foundation.