Fast Start to LaTeX

Peter S. Ho
peterh@cse.unsw.edu.au
Version 1.14

21 August 2002

1 About this document

Before reading this document, it is important to note that this is not a comprehensive guide to LaTeX. The subset of commands dealt with in this document are ones that are commonly used. As such, what you learn from these pages is only a sample of LaTeX's true potential. In addition, I have glossed over details that's only applicable to experienced LaTeX users. Nonetheless, casual users and beginners should find this document useful as a starting point.

As you become more familiar with using LaTeX, you'll quickly outgrow the usefulness of this document. And if you plan to be more than just a casual user, then I thoroughly recommend that you purchase Leslie Lamport's book on LaTeX [Lam 94], this book is considered by the LaTeX community to be *the* authoritative book. It is not my intention to displace Lamport's book, this document should be seen more as a lure to get more people interested in using LaTeX.

If you decide to buy Lamport's book (especially on the second hand market), then be aware that the first edition of his book is no longer current since it refers to an older implementation of LaTeX (now referred to as LaTeX209). The latest release, LaTeX 2_ε, is its replacement and is now simply referred to as LaTeX (just to be confusing).

For users looking for that little something extra, you may wish to have a look at *The LaTeX Companion* [Goo 94], which will show you how to write macro packages. While, for those more interested in TeX (the engine beneath LaTeX), try reading Donald Knuth's *The* TeX*book* [Knu 90]. **Be Warned** that both the Companion and the TeX book are meant for advanced users. For most users, the material covered in Lamport's book should be more than adequate for majority typesetting jobs.

2 What is LaTeX?

LaTeX is a batched oriented document preparation (typesetting) system that is built on top of Donald Knuth's TeX program. The TeX system produces high-quality typeset documents for both ordinary and mathematical text. TeX provides the user with absolute and precise control over the way a document can be typeset, unfortunately this freedom and flexibility requires an extensive knowledge of TeX's primitive command set, which can be a little less than friendly at times.

To overcome some of these difficulties, LaTeX was developed. LaTeX essentially adds to the functionality of TeX by providing a set of high-level (macro) commands. It was hoped

(by the authors of LaTeX) that by simplifying, and in many instances hiding some of TeX's complex typesetting overheads, it would be easier for users to produce consistent and better looking documents.

To be accurate, LaTeX is your typographic designer and TeX is the typesetter. A good document is one that is easy to read and visually exhibits a logical structure. A good typographic design is an essential ingredient in helping the the author to achieve this goal—this is the function of LaTeX.

Being a batch oriented document preparation system, you cannot see the result of changes made to a document instantly. This is in stark contrast to other desk top publishing products that belong to a WYSIWYG (**W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et) philosophy where *immediate* visual feedback is valued highly. There are advantages and disadvantages to both approaches and it is best left to the individual to decide which is better—use the right tool for the job and the one that works is the one to use.

3 How Do I Process a LaTeX Document?

The first thing you need is a LaTeX document, which is essentially an ASCII file with embedded LaTeX commands (these commands will be explained a little later).

Assume that you have a LaTeX file called `myfile.tex`—by convention, all TeX and LaTeX files have a `.tex` suffix to identify its heritage. To process this file, you simply run the following command:

```
latex myfile.tex
```

On completion, you will find three additional files in the current directory—`myfile.dvi`, `myfile.aux` and `myfile.log`. Leaving aside the `.log` and `.aux` files for the time being, `myfile.dvi` is the file that contains the typeset output of `myfile.tex` (assuming `latex` did not detect any errors). The `.dvi` extension indicates that the typeset document is stored in a "device independent" format, which means it cannot be printed or displayed directly without using a *dvi* converter. `.dvi` files are device independent in the sense that it is not constraint by physical print or display limitations.

A dvi program is essentially a "translator" that translates the information in the `.dvi` file into a form that can be understood by a device. Different devices will require different dvi translators. For example, `xdvi` is a **X** Window translator; and `dvips` is a postscript translator for postscript laser printers.

Assuming you have both `xdvi` and `dvips` programs installed, you can view `myfile.dvi` on a **X** Window system by typing the command:

```
xdvi myfile.dvi
```

While you type the command:

```
dvips myfile.dvi | lp
```

in order to pipe the postscript output from `dvips` to the default laser printer as determined by `lp` (the convention for `lp` may vary from site to site).

4 Basic LaTeX Commands

In this section we look at the basic commands you will need to get started. This includes setting up a minimum LaTeX file, changing type style, document sectioning, quoting, footnoting, making a title page and an abstract. The material covered here is by no means comprehensive, but it should be sufficient for most authors. For a full description refer to Lamport's book [Lam 94].

4.1 The Minimum LaTeX File

LaTeX source files are plain ASCII files. Being just plain ASCII makes it extremely portable across different computing platforms. A simple text based editor is all that is needed to create and edit any LaTeX source. A minimum LaTeX source file is one that contains just the following text:

```
\documentclass{article}
\begin{document}


\end{document}
```

The text to be typeset must be enclosed between the `\begin{document}` and `\end{document}` command. Text found after the `\end{document}` command will be ignored. The section of text found before the `\begin{document}` command is called the *preamble*. The preamble can contain only declarations, which are used to specify or modify the document's style or layout quality.

The preamble begins with the `\documentclass` command. The argument enclosed within { } selects one of the major predefined page layout *class*—in this instance, the `article` class was selected. The standard LaTeX classes available for ordinary documents are:

`article`	Primarily used for short to medium length documents.
`report`	Used for medium to long documents.
`book`	Should only be used for writing books.
`letter`	Useful for writing letters.
`slides`	Use for the preparation of slides

Each document can declare one major class only. For a full description of the class and its options, see Lamport's book [Lam 94].

Besides being able to choose a major class, you may also provide these classes with certain *options* that alter its characteristics. Not every class have options, some may not have any options at all. To specify a document-class option, it must be enclosed within `[]` brackets, for example:

```
\documentclass[a4paper,12pt]{article}
```

Multiple options must be separated by commas, but do not use any spaces within the square brackets and after commas. In this example, the `12pt` option specifies that the document is to be typeset in twelve-point type; the `a4paper` option states that the paper size should be A4 (the default paper size is always US-letter). There are quite a few options available, some of the more commonly used options include:

a4paper Sets the paper size to A4 (width 210mm, height 297mm).

letterpaper Set the paper size to US-letter (width 8.5in, height 14in).

10pt This is the default type size used for your document.

11pt This specifies the eleven-points type size. This overrides the ten-point type normally used, which is ten percent smaller.

12pt This specifies the twelve-point type size, which is twenty percent larger than the default ten-point type.

twocolumn This produces a two column page.

twoside This formats the output to enable text to be printed on both sides of the page.

fleqn Place mathematical equations flush against the left margin.

For a full list of options, refer to Lamport's book [Lam 94].

4.2 Changing Fonts

Changing fonts is something that should be used sparingly. Reading a document that changes font frequently can be very difficult to read. In fact, you will more than likely find it distracting. Limit your font changes to highlighting of foreign words, important words or phrases.

4.2.1 The Family Shape

The default type face used in LaTeX documents come from the Computer Modern family (designed by Donald Knuth). The Computer Modern typeface is a *serif* font very similar to another typeface called *Times*. The characters in this sentence is typeset in Computer Modern up-right Roman, which as you can see is the preferred setting for normal text.

In addition to Computer Modern Roman, there is also a *san serif* family and a *Typewriter* family. Here is a sample:

This is the Roman family, the default.	`\textrm{This is the Roman family, the default.}`
This is the San serif family.	`\textsf{This is the San serif family.}`
`This is the Typewriter family.`	`\texttt{This is the Typewriter family.}`

Notice that even when the family shape is changed, the size of the font remains the same.

4.2.2 The Shape Attribute

The term "up-right roman" used above describes the family's shape attribute. LaTeX has 4 different shape attributes:

Upright shape. The default.	`\textup{Upright shape.  The default.}`
Italic shape. Used to emphasise text.	`\textit{Italic shape.  Used to emphasise text.}`
Slanted shape. Different to italics.	`\textsl{Slanted shape.  Different to italics.}`
Small caps. Use sparingly.	`\textsc{Small caps.  Use sparingly.}`

4.2.3 The Series Attribute

Besides being able to change the family and shape of a typeface, you can also change its series attribute, *i.e.*:

```
\textmd{Medium series.  Usually . . .}
\textbf{Boldface series.  Often used . . .}
```

Medium series. Usually the default
Boldface series. Often used for headings

4.2.4 Combining Attributes

All the commands that change the family (see section 4.2.1), the shape (see section 4.2.2) and series (see section 4.2.3) may be combined together in a logical fashion to produce a wide variety of type styles.

```
\textbf{\textit{Phew.}}  That was
\textsf{\textbf{too close}} for
comfort.
```

Phew! That was **too close** for comfort.

Remember, too many type changes can be very distracting, try to limit the number of changes. Some type styles may not be available on your system. If LaTeX finds a combination that is not available, it will give you a warning message and substitute a style that it thinks is similar.

4.2.5 Changing Attributes: Command vs. Declarative Format

The attribute commands used so far are parameterised, *i.e.* the text to be modified is given as a parameter to the command being executed. Each of the text-attribute command described above has a corresponding declarative format.

Cmd	Decl	Cmd	Decl	Cmd	Decl
`\textup`	`\upshape`	`\textsc`	`\scshape`	`\textrm`	`\rmfamily`
`\textit`	`\itshape`	`\textmd`	`\mdseries`	`\textsf`	`\sffamily`
`\textsl`	`\slshape`	`\textbf`	`\bfseries`	`\texttt`	`\ttfamily`

The declarative commands may be used just by simply placing it in the text where you want the effect to take place:

```
This bit is in roman, \bfseries but
this bit is in bold, \itshape with
a bit of italics \upshape thrown
in, \mdseries back to roman again.
```

This bit is in roman, **but this bit is in bold, *with a bit of italics* thrown in,** back to roman again.

As you can see, the changes takes place after each declaration. Notice that once you have specified the change, say `\bfseries`, you must explicitly change it back by using `\mdseries` once the effect is no longer desired. This can be tedious, but there is an alternative. The effects of the change can be limited by using { } braces to restrict the scope of the declaration, *i.e.* when TeX encounters the }, it reverts back to the attributes in effect just before the {. The { and } braces are used to mark the begin ({) and end (}) of a new scope. Braces may

be nested, as long as it comes in matching pairs. As expected, it can be used to delimit a mixture of attribute changes:

This is boring roman, **but this is in bold, *with italics* in between**. But that's enough.

```
This is boring roman, {\bfseries but
this is in bold, {\itshape with
italics} in between}.  But that's
enough.
```

4.2.6 Emphasised Text

One form of type shape change not mentioned above is the `\emph` and `\em` command used to emphasise text. Visually, the shape is the same as that assigned to the italics shape. Both these commands are used to emphasise a piece of text. Emphasis should be used sparingly as it is like raising your voice to attract attention.

The `emph` command is useful for emphasising a few words at a time. Here is an example:

Let me *emphasise this*.

```
Let me \emph{emphasise this}.
```

The `\em` declaration on the other hand is more suited to large pieces of text. It works in a manner similar to the declarative commands discussed in section 4.2.5.

Let me *emphasise this.* But that's enough.

```
Let me \em emphasise this. \upshape But
that's enough.
```

As before, you can use { } to delimit the scope of a declaration. In addition, it is also possible to nest emphasis:

You can have *emphasized text* within *emphasized text*.

```
You can have {\em emphasized text
\emph{within} emphasized text}.
```

4.2.7 Type Size

Besides being able to vary the font attributes. The size of the characters in the document may also be varied by using the following commands:

Gnu `\tiny`
Gnu `\scriptsize`
Gnu `\footnotesize`
Gnu `\small`
Gnu `\normalsize`
Gnu `\large`
Gnu `\Large`
Gnu `\LARGE`
Gnu `\huge`
Gnu `\Huge`

The size of the characters altered by these commands are scaled in proportion to the base type size, which by default is 10-points unless explicitly changed in the `\documentclass` command.

Whatever the default size happens to be, it is always reflected by the `\normalsize` declaration. Sizes like `\footnotesize` and `\scriptsize` represents the size of font suitable for footnotes, subscripts and superscripts.

To set a piece of text in the `\small` type size, you can either delimit the scope of the size change by using { }:

This is tiny and normal. `This is {\tiny tiny} and normal.`

or by explicitly changing the size, which is a less desirable way of specifying the size change.

This is tiny and normal. `This is \tiny tiny \normalsize and normal.`

Regardless of the style currently in effect, size changing declarations only effect the roman style. Hence, in order to get large bold letters, you must type `\large\bfseries` and **not** `\bfseries\large`. In addition, size changing commands may not be used in math mode (more about this later) except in certain circumstances, see Lamport [Lam 94] for more information.

4.3 Sectioning

In order make a document more readable, the document should be organised as a hierarchical structure composed of chapters, sections and subsections. A section is started by using the appropriate command followed by the section's title enclosed in { } braces, for example:

2 This is a section `\section{This is a section}`
2.1 This is a subsection `\subsection{This is a subsection}`
2.1.1 This is a subsubsection `\subsubsection{This is a subsubsection}`

In LaTeX, the sectional units are numbered automatically. If a section number is not required, then insert a "`*`" in between the sectional command and the {, *i.e.*

Subsubsection with no numbering `\subsubsection*{Subsubsection ... numbering}`

The following is a full list of sectioning commands:

```
\part          \subsection       \paragraph
\chapter       \subsubsection    \subparagraph
\section
```

The applicability of the sectioning commands is dependent on the class of the document. For instance, the `article` class does not support the `\chapter` command. A subsection must be part of a section, which in the `report` and `book` class, must be part of a chapter. The `\part` command is used mainly to divide long documents into parts. Using the `\part` command in the `article` class does not effect the numbering of smaller sectional units, *i.e.* if the last section in part 1 is section 10, then the first section in part 2 will be section 11.

To add an appendix, you use the `\appendix` command in conjunction with the same sectioning commands. The `\appendix` command itself does not produce any text output, instead it changes the way sectional units are numbered to produce what is appropriate for an appendix.

4.4 Making a Title Page

Every document should have a title page. This usually includes the title of the article, a list of authors and perhaps a publication date. The following example is how you create a simple title page in LaTeX:

How To Get Rich

Alan Bond Christopher Skase

8 August 1987

```
\title{How To Get Rich}
\author{Alan Bond \and Christopher Skase}
\date{8 August 1987}

...
\maketitle
```

The authors are introduced using the `\author` command. The `\and` command is used where there are multiple authors. The `\date` command is optional, omitting it will give you the current date. All three declarations, `\title`, `\author` and `\date` must precede the `\maketitle` command, which in turn must be declared after the `\begin{document}` command.

When LaTeX encounters the `\maketitle` command, it generates a title page on a separate page unless it is an `article`. For `article` documents, titles are printed at the top of the first text page. If this is not desirable, the title can be forced onto a separate title page by adding the option `titlepage` to the `\documentclass` declaration, *e.g.*:

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

Here is a slightly more complicated title page, it contains multiple authors together with their address and special notes:

How To Make Wine

Wanda Cup†
Acme Kitchen Wine Division

Bart Drink‡
Food Technologies

8 August 1987
2nd Edition

⋮

```
\title{How To Make Wine}
\author{Wanda Cup\thanks{Sponsored by
        Acme Kitchens.}\\
        Acme Kitchen Wine Division
    \and
        Bart Drink\thanks{Joint project
        arrangements.}\\
        Food Technologies}
\date{8 August 1987\\2nd Edition}
     ...
\maketitle
```

†Sponsored by Acme Kitchens.
‡Joint project arrangements.

4.5 The `begin ... end` Environment

So far, you have only seen environments enclosed by { } braces. The use of braces to enclose large environments can be difficult to maintain in a sea of text. LaTeX has another environment construct:

`\begin{`*name*`} ... \end{`*name*`}`

where *name* is the name of the environment. The text in between the enclosing `\begin` and `\end` command will be processed according to *name*. Blank lines before the environment mean that it is a complete paragraph. A blank line after the `\end` command means that the following text is to be a new paragraph. Any blank lines immediately following the `\begin` command and immediately preceding its `\end` command are ignored.

4.6 Quotations

There are two different ways to display quotations: `\quote` and `\quotation`. The `\quote` environment is meant for a short quote or a sequence of short quotes separated by blank lines.

Here are two famous quotes:

> To be or not to be. *William Shakespeare*
>
> This is a recession we had to have. *Paul Keating*

```
Here are some famous quotes:
\begin{quote}
To be ...  to be. \emph{William Shakespeare}

This ... had to have. \emph{Paul Keating}
\end{quote}
```

The `\quotation` environment, on the other hand, is meant for long quotes that have more than one paragraph separated by blank lines. For example:

Here is a quote from Lamport's book:

> Environments for making quotations can be used for other things as well.
>
> Many problems can be solved by novel applications of existing environments.

```
Here is a quote from Lamport's book:
\begin{quotation}
Environments for ... as well.

Many problems ... existing environments.
\end{quotation}
```

4.7 Making an Abstract

An abstract is like a preface, it tells the readers what to expect from the article. Abstracts usually follow on from the `\maketitle` command (see section 4.4 for more details). To make an abstract, you simply use the `abstract` environment:

Abstract

> Investing is easy as long as it is someone else's money. It always pays to get out before you make a loss.

```
\begin{abstract}
Investing is easy as long
as it is someone else's money.
It always pays to get out
before you make a loss.
\end{abstract}
```

The abstract command places the abstract text where the command occurs, which for most authors would be immediately after the `\maketitle` command. If the `titlepage` document class option is specified or if the document is classed as a `report` (see section 4.4), then the abstract will appear on a page by itself.

4.8 Footnotes

Footnotes are useful for qualifying/justifying points made in the main text. In some instances, it may be used to add commentary that may not fit into the general context of the document. Footnotes are (usually) placed at the bottom of the referenced page:

I love LaTeX[1] and so will you.

⋮

```
I love \LaTeX\footnote{This
is a type setting language.}
and so will you.
```

[1] This is a type setting language.

Don't leave a space between the word to be footnoted and the `\footnote` command as that will give you an unwanted space between the word and the footnote marker.

4.9 Verbatim Mode

There are times in a document where you require a piece of text to be printed "as is". The `verbatim` environment is an environment that allows the author to do precisely that:

When you don't want LaTeX to alter the appearance of the text, use the verbatim environment.

```
Notice that I can use the following
characters here without fuss: #%&$_\^~
```

```
When you don't want \LaTeX\ to
alter the appearance of the
text, use the verbatim environment.
\begin{verbatim}
Notice that I can use the following
characters here without fuss: #%&$_\^~
\end{verbatim}
```

Notice that each space typed produces a space in the output, likewise new lines appear where you placed them. Anytime you type inside the `verbatim` environment will not be interpreted by LaTeXexcept for the `\end{verbatim}` character sequence.

The `verbatim` environment begins on a new line. A blank line after the `verbatim` environment starts a new paragraph for the following text.

For a shorter piece of text inside a paragraph, we use the `\verb` command. The piece of text is not enclosed within braces, but by a pair of identical characters:

This `bit of text` has `been verb'ed`.

```
This \verb+bit of text+ has
\verb9been verb'ed9.
```

Noticed that the first verbatim text is enclosed between matching "+" symbols while the second used matching "9" symbols. The matching symbols can be any character as long as it is not a space, a letter, a "*" or a character appearing within the argument.

4.10 Lists

Lists are a useful way of displaying information that stands out. LaTeX has three list making environments: `itemize`, `enumerate` and `description`.

In the `itemize` environment, each item in the list is marked by a bullet, while `enumerate` lists are numbered, for example:

- Each item of the list is begun with a `\item` command.
- You can have as many `\items` as you need.
- Each item will be marked by a bullet marker.
    1. You can even nest lists within lists.
    2. Notice how `enumeration` list uses numbers.

  LaTeX gives you up to 4 levels of nesting. This is usually more than enough.
- Blank lines before an item have no effect.

```
\begin{itemize}
  \item Each ... \verb+\item+ command.
  \item You ... \verb+\items+ as you need.
  \item Each ... a bullet marker.
    \begin{enumerate}
      \item You ... within lists.
      \item Notice ... uses numbers.
    \end{enumerate}
    \LaTeX\ gives ... usually more than
    enough.

  \item Blank lines ... no effect.
\end{itemize}
```

In the `description` environment, the marker for each item is replaced by a marker that you specify, for example:

Cars Essential personalised transport.

Bicycles For the health conscious.

Boats For the rich and famous.

```
\begin{description}
  \item[Cars] Essential personalised transport.
  \item[Bicycles] For the health conscious.
  \item[Boats] For the rich and famous.
\end{description}
```

The item labels are specified as an option to the `\item` command enclosed within [ ] brackets. To include a [or a] within `\item[]`, you will need to enclose them between { } braces, *i.e.*

[**Cars**] Essential personalised transport.

[**Bicycles**] For the health conscious.

[**Boats**] For the rich and famous.

```
\begin{description}
  \item[{[Cars]}] Essential ... transport.
  \item[{[Bicycles]}] For the health conscious.
  \item[{[Boats]}] For the rich and famous.
\end{description}
```

5 Symbols, Spacing, Dashes and those Odd Fiddlely Bits

This section covers a range of miscellaneous things that you may need or need to know some time or other.

5.1 Special Symbols

As with most typesetting languages, there are a number of special symbols that LaTeX reserves for its use, such as:

```
#  $  %  &  ~  _  ^  \  {  }
```

These symbols are used in one way or another as LaTeX commands. Unfortunately, there are times that you may need to include these characters in you document. Seven of the special symbols can be produced by typing a \ in front of the corresponding character:

$ & % # _ { } are easy.

```
\$ \& \% \# \_ \{ \} are easy.
```

The other three special characters ~, ^ and \, can be produced by encapsulating them within verbatim environments (see section 4.9 for more details).

~, ^ and \ are not easy.

```
\verb+~+, \verb+^+ and \verb+\+ are not easy.
```

5.2 Dashes

There is more to dashes than meets the eye. LaTeX can produce 3 sizes of dash. The first is the more common hyphenation, this is accomplished using a single "-":

Multi-media is a hyphenated word.

```
Multi-media is a hyphenated word.
```

Hyphenations (*i.e.* a single "-") should never be used to separate number ranges *e.g.* pages 10–12. You need a medium dash, which is done using two dashes, "--". It is generally recommended that you do not put a space before or after the dashes.

For homework read pages 10–12.

```
For homework read pages 10--12.
```

The last commonly used dash is the punctuation dash. This extra long dash is created using three dashes, "---":

You look tired, go to bed—immediately.

```
You look tired, go to bed---immediately.
```

Don't confuse a dash with a minus, which looks quite different. Here is a single dash, "-"; and here is a minus "−" taken from the math display environment.

5.3 Quotation Marks

The use of quotation marks is a common occurence in printed text. The normal keyboard representation for a double quote " does not product a very attractive double quote when printed. Attractive double quotes are produced by using the left quote ` (sometimes called a back-tick) and a single right quote ' (also called an apostrophe). As may be expected, using a single quote pair `' produces a single quote, while using 2 pairs of single quotes together ``'' produces a double quote. For example:

This is a 'single' quoted word.

```
This is a `single' quoted word.
```

Here is a "double" quote.

```
Here is a ``double'' quote.
```

5.3.1 Printing the TEX, LATEX and LATEX 2ε Logos

The name of the product, TEX, LATEX and LATEX 2ε are specially kerned logos. Whenever one refers to these produces, the authors of these products expect you to use the specially kerned logos. To comply with the authors' wishes, special commands available:

This is how you write TEX. As you can guess, LATEX and LATEX 2ε are also written in a similar fashion.

```
This is how you write \TeX.  As you
can guess, \LaTeX\ and \LaTeXe\ are
also written in a similar fashion.
```

Notice that there is an extra "\" just after the command `\LaTeX` and `\LaTeXe`. The "\␣" (the ␣ symbol represents a space) forces an ordinary interword space to be inserted after the logo. In general, spaces after a command like `\TeX` are ignored by TEX.

TEX is great, but LATEX is better.

```
\TeX\        is great, but \LaTeX\ is
better.
```

Notice that no matter how many spaces I insert after the `\TeX` command in my source file, it still gets ignored.

5.4 Special Space Commands

As you read earlier (see section 5.3.1), the \␣ command forces an ordinary interword space to be inserted. TEX generally does a very good job of spacing out the text, but it does need the occassional help in working out which periods end sentences. TEX, by default, assumes that a period ends a sentence unless it follows an uppercase letter. Hence, abbreviations within a sentence can fool TEX. To stop a period from being treated as the end of a sentence, use the \␣ command.

The cat, dog etc. loved the food.

```
The cat, dog etc.\ loved the food.
```

The cat, dog etc. loved the food.

```
The cat, dog etc. loved the food.
```

Notice the difference, without the \␣ command TEX not only puts a space after the period it added a little extra in order to start a new sentence.

On the other hand, there are times when a period does end a sentence, but it follows an uppercase letter. In situations like this, you need to explicitly tell TEX that that the period does end the sentence. The command to use is the `\@` command.

Go to section III. But for ...

```
Go to section III\@.  But for ...
```

"Go forth (etc.) and be GOOD."

```
``Go forth (etc.)\ and be GOOD\@.''
```

The `\@` command before the right quote (single or double) or right parenthesis will force the extra space after the quote or parenthesis. The usage of both the \␣ and `\@` apply also to other punctuation characters like ? (question mark), ! (exclamation point) or : (colon).

6 More Useful Commands

In this section, you will find out about a few useful commands that will help you:

- to manage a large LaTeX document;
- change the default line breaks;
- to center and flush text;
- to make tables;
- to build a table of contents (TOC); and to
- to include a bibliography.

6.1 Including Files

If you are about to author a large paper, book or report, it would probably be sensible to break it into smaller bits, perhaps into chapters or parts, and store them in separate smaller files instead having a huge file with everything in it. Regardless of how many separate files there are, you will need a *root* file to specify what is to be "included" in the final document. The root file will be the file LaTeX processes.

There are 2 different ways to specify the inclusion of an external file: `\input` and `\include`. The `\input` command provides the simplest mechanism. By saying:

```
\input{chapter1}
```

in the root file, the file `chapter1.tex` will be inserted right at the current spot in your manuscript when processed by LaTeX. It will appear as if the `\input{chapter1}` command was removed from the root file and replaced by the contents of the file `chapter1.tex`. The `\input` command can be nested, *i.e.* the command `\input` may appear inside `chapter1.tex` and the other input file may in turn contain another `\input` command, and so on.

The other command `\include` works in the same way as `\input` except you have the option of telling LaTeX whether to insert or omit the file named by using the `\includeonly` command. The `include` command can only be used after the `\begin{document}` command (unlike the `\input` command which can be used anywhere). In addition, `\include`'d text always start on a new page, as does the text immediately following the `\include` command.

```
. . .
\includeonly{chapter1, chapter3}
\begin{document}
    \include{chapter1}
    \include{chapter2}
    \include{chapter3}
    \include{chapter4}
\end{document}
```

By using the `\include` mechanism, LaTeX processes the succeeding text as if the file had been inserted thus keeping the numbering of pages, sections, equations, *etc.*, as if the omitted files' text had been included. In the example above, only the files `chapter1.tex` and `chapter3.tex` are included.

The `\includeonly` command itself must be placed in the preamble of the root file. Beware that LaTeX does not read in omitted files and will not be aware of any changes made to omitted files since it was last included.

If the preamble does not contain an `\includeonly` command, then all the `\include` commands will insert the files specified. On the other hand, a `\includeonly{}` command will omit all `\include`'d files.

6.2 Line and Page Breaks

TeX usually does a good job of sorting out line and page breaks, but it occasionally needs a bit of help. Try to avoid fiddling with line and page breaks until you have finished writing the text, chances are it will fix itself.

One of the causes of bad line breaks comes from having words that will not fit on the line within the margins nor will it hyphenate according to the rules TeX has been given. There are three possible solutions:

1. rephrase the sentence or paragraph, this often fixes the problem;
2. give TeX some additional rules on how to hyphenate the offending word;
3. tell TeX not to be so fuzzy about line breaks; or
4. force the offending word onto the next line by ending the line pre-maturely.

6.2.1 Hyphenation

When you get a word that doesn't quite fit on the line, LaTeX usually lets you know by giving you a message like:

```
Overfull \hbox (15.2647pt too wide) in paragraph at lines 155--161
[]\OT1/cmr/m/n/10.95 The text to be type-set must be en-closed be-tween the
commands
```

In this instance, TeX could not find a good place to break the word "command" and it has slipped past the right margin by roughly 15 points. To give TeX more hyphenation rules, you say:

```
co\-m\-mand
```

which indicates to TeX that it would be okay to break the word after *co* or *com*. TeX in all its wisdom will choose the better of the two.

If the word is going to be used quite a bit in your article, you might consider teaching TeX how it should be hyphenated throughout the document. You do that by using the `\hyphenation` command in the preamble:

```
\hyphenation{co-mmand com-mand comm-and}
```

6.2.2 Telling TEX not to be so fuzzy

The command `\sloppy` or the `\begin{sloppypar}\end{sloppypar}` environment can be used to tell TEX not to be so fuzzy about line breaks. You can initiate this in one of 2 ways:

Option 1

```
\sloppy
. . . paragraph of
text . . .
\fuzzy
```

Option 2

```
\begin{sloppypar}
. . . paragraph of
text . . .
\end{sloppypar}
```

With option 1, it is possible to replace the `\fuzzy` command with a blank line to mark the end of the paragraph.

6.2.3 Forcing a Line to Break

The final way to fix a word that won't fit is to break the line and force the word onto the next line. You can do this by using the `\linebreak` command. You normally place the command just before the word you want to force onto the next line.

If you want a line break, but you want to give TEX a bit of a chance to fiddle, then use the `\linebreak` command with an optional number between 0 and 4, *e.g.* `\linebreak[1]`. The arguments, 1, 2, 3 and 4 provides intermediate degrees of insistence, the higher the number the stronger the demand. `\linebreak[0]` is special in that it allows a line break where it would not be permitted normally, *e.g.* within words.

Using `\linebreak` command forces TEX to justify the line that is broken. This may not be what you want, in order to break a line and leave it unjustified, use the `\newline` command:

This line has a line break in it but it won't be justified
when it breaks.

```
This line has a line break in it
but it won't be justified \newline when it
breaks.
```

compared to:

This line has a line break in it but TEX will try to justify it when it breaks.

```
This line has a line break in it
but \TeX\ will try to justify
it \linebreak when it
breaks.
```

6.2.4 Preventing Line-breaks

Sometimes, you want the reverse of not having a line break. You can stop a word from being broken (hyphenated) by enclosing it within an `\mbox`:

The line is going to be rather long and there is bound to be a linebreak that causes a hyphenation, but not this time.

```
The line is going to be
rather long and there is
bound to be a linebreak
that causes a \mbox{hyphenation},
but not this time.
```

An alternative to `\mbox` is the `\nolinebreak` command, which is placed just before the target word. Just like the `\linebreak` command, you can also specify the degree of prohibition using numbers from 0 to 4, with 4 being the strongest; *e.g.* `\nolinebreak[2]`.

Besides having line breaks within a word, you can also get an unwanted line break in an inter-word space, *i.e.* the space in between two words. You can prevent this by using the `~` character. For example:

```
Mrs.~Smith      Diagram~1
```

It would be ugly to have an unsightly line break just after the word "Diagram", by using the `~` character, you are telling TEX to treat the word to the left and right of the `~` as a unit.

6.3 Centering and "Flushing" Text

There are times when you need to center a block of text, a table, or even a figure. This is easily accomplished by using the `center` command (note the way it is spelt). For example:

This block of
text is going
to be centered. Notice how
easy it is.

```
\begin{center}
This block of\\text is going\\   to be
centered.  Notice how\\easy it is.
\end{center}
```

By leaving a blank line after the center environment, LATEX starts a new paragraph.

```
By leaving a blank line after the center
environment, \LaTeX\ starts a new
paragraph.
```

If you don't want a new paragraph after the center environment, then do not leave a blank line after the `\end{center}` command.

To flush the text to the left or right of the page, use the `flushleft` and `flushright` environments respectively, eg:

This is a flushed
left line of text.

But this is a
flushed right line.

```
\begin{flushleft}
This is a flushed\\ left line of text.
\end{flushleft}
\begin{flushright}
But this is a\\ flushed right line.
\end{flushright}
```

The `flushleft` command is seldom used this way with the `\\` command since a `\\` command in normal mode also produces a flush left line of text. However, if you let TEX do the line breaking for you, then the body of the text will set with a ragged-right, eg:

Notice how this block of text is not justified in this environment. The flushleft environment ensures that we don't get justified text.

```
\begin{flushleft}
Notice how this block of text is not
justified in this environment.  The
flushleft environment ensures that
we don't get justified text.
\end{flushleft}
```

Likewise, the ragged behaviour also occurs with the `flushright` environment, but the ragged edge is to the left.

All three environments have a declarative equivalent, *i.e.*:

ENVIRONMENT	DECLARATIVE
flushleft	`\raggedright`
center	`\centering`
flushright	`\raggedleft`

which can be used inside other environments like a `quote`, a `parbox`, a `figure` or a `table` environment. The only difference is that the declarative version does not start a new paragraph. You'll need a blank line or a `\end` command to effect the paragraph unit's format. Here's an example from Lamport's book:

This is a text that comes at the end of the preceding paragraph.

Here is a quote who's lines are
flushed right inside a quote
environment.

```
This is a text that comes at
the end of the preceding
paragraph.
\begin{quote}
  \raggedleft Here is a quote
  who's lines are\\ flushed
  right inside a quote environment.
\end{quote}
```

6.4 Creating a Tables

Tables can be used to display information to the reader in a manner that does not require a lot of words. Well constructed tables can often convey lots of information quickly. To construct text based tables in LaTeX, we use the `tabular` environment. The descriptions given here for the `tabular` environment also applies to the `array` environment, which is the table environment for typesetting maths. The best way to explain the use of the `tabular` environment is to see an example:

Expenses Summary		
1	Taxi	$10.00
2	Dinner	$5.00
3	Stationery	$20.00
	Sub-total	$35.00
4	Hotel	$100.00
5	Phone	$20.00
	Total	$155.00

```
\begin{tabular}{|c|l|r|}
  \hline\hline
  \multicolumn{3}{||c||}{Expenses Summary}\\
  \hline\hline
  1  & Taxi                & \$10.00\\
  2  & Dinner              & \$5.00\\
  3  & Stationery          & \$20.00\\
  \cline{3-3}
     & \textbf{Sub-total} & \$35.00\\
  4  & Hotel               & \$100.00\\
  5  & Phone               & \$20.00\\
  \cline{3-3}
     & \textbf{Total}      & \$155.00\\
  \hline
\end{tabular}
```

The argument, `{|c|l|r|}`, immediately following the `tabular` environment specifies the number of columns and how these columns are to be formatted. The vertical bar, `|`, puts a

vertical line extending the full height of the environment in the specified place. Typing two vertical bars, `||`, instead of one will generate two vertical lines the length of the environment in the specific place.

Each column is denoted using a single letter, which also serves to indicate how the column is to be aligned:

- c Centre the text in the column.
- l Align the text to the left hand side of the column.
- r Align the text to the right hand side of the column.

So if you want a three column table to have centered text, you can specify this by setting the `tabular` environment's argument to `{ccc}`.

Columns in the `tabular` environment are separated by the `&` (ampersand) symbol. The width of each column is adjusted automatically to accommodate the longest word in that column. A `\hline` command after a `\\` or at the beginnning of the environment draws a horizontal line across the full width of the environment. Shorter horizontal lines can be drawn using the `\cline{`x - y`}` command, which simply draws a horizontal line across columns x through y inclusively (columns are number 1 to n, starting with the leftmost column).

Every row in the table must be terminated using a pair of backslashes, `\\`, except for the last row. The only exception is when a `\hline` command follows the last line in the table.

To span an item across multiple columns, you will need to use the `\multicolumn` command, for example:

```
\multicolumn{3}{||c||}{Expenses Summary}
```

The first argument, `{3}`, simply states the number of columns we are going to span. The second argument, `{||c||}`, states how we want the text to be aligned (it can only be one off `l`, `c` or `r`). In this instance, the text is to be centered within the column and we also want two vertical lines to be drawn on either side of the column. The final argument contains the text we want to appear in the multicolumn column. Here is a slightly more complicated table:

Dates		Events
21	March	Assignment Due
30	May	Big date

```
\begin{tabular}{llc}
  \multicolumn{2}{c}{Dates} &
  \multicolumn{1}{c}{Events}\\
  21 & March & Assignment Due\\
  30 & May   & Big date\\
\end{tabular}
```

The `tabular` environment can be used anywhere you want (well almost). TEX essentially treats the `tabular` object as a rather large letter. As such, it can appear in the middle of a paragraph or even a word if you wish, but tables are more useful when put in a center environment; or better still, in a figure or table (see section 6.5).

6.5 Tables and Figures

Figures and tables are objects that cannot be broken across pages. As a consequence, these objects must be "floated" to convenient places in the document. The `figure` and `table`

environments are two LaTeX environments that provide this functionality. Both these environments are identical except for the way it captions the object and the placing of the entry in the "List of tables" and "List of figures" page (see section 6.6 for more information).

The figure goes here!
Figure 9. The caption.

⋮

Here is the bit of text that mentions figure 9 for the first time.

```
Here is the bit of text
that mentions figure\ref{fig:myfig}
for the first time.
\begin{figure}
  \begin{center}
  \leavevmode
  The figure goes here
  \end{center}
  \caption{The caption.}
  \label{fig:myfig}
  \centering
\end{figure}
```

In this instance, the figure floated to the top of the page and the text that referenced the figure is towards the bottom of the page. The `label` and `ref` commands in the example allow for figures to be referenced anywhere in the document (see section 6.7 for more details), but the `label` command must appear after the `caption` command in the body of the `figure` or `table` environment. The `caption` command starts a caption with a *Figure x:* for you, where x is generated automatically by LaTeX. The construct works in the same way for tables, except that a *Table x:* is generated for the caption instead. You can only use the `caption` command in the `figure` and `table` environments.

You can put anything in the body of a figure or table, it will be processed in paragraph mode just like any other text. By default, LaTeX assumes the object will take up the same width as the text.

When it comes to placing the object on the page, LaTeX will try to place the figure or table either above the text at the top of the page, below the text at the bottom of the page, or on a separate page with nothing but figures and/or tables. You can influence this behaviour by specifying a preferred location—*loc*—in the environment, *i.e.*:

`\begin{figure}[`*loc*`]` *body* `\end{figure}`
`\begin{table}[`*loc*`]` *body* `\end{table}`

The *loc* argument above contains a sequence of one to four letters, which represents:

`h`	*Here*	place the object at the position in the text where the environment appears.
`t`	*Top*	place the object at the top of the text page.
`b`	*Bottom*	place the object at the bottom of the text page.
`p`	*Page*	place the object on a separate page with other tables and figures (no text allowed on the page)

If the *loc* argument is missing, LaTeX assumes the default which is `tbp`, *i.e.*: try the top of the page, followed by the bottom, followed by on a separate page if it doesn't fit anywhere. If you put a `!` in the *loc* argument, this tells LaTeX to try harder. The rules that govern the

placement of figures and tables are straight forward (the rules are not covered here, please refer to Lamport's book), but it can produce some interesting results.

On occassion, a figure or table might be placed on a page that doesn't suit. When this occurs, you can either change the *loc* arguments or explicitly surppress the figure or table from appearing on that page, *i.e.*:

`\suppressfloats[`*loc*`]`

where *loc* is:

- `t` No more figures or tables at the top of the current page.
- `b` No more figures or tables at the bottom of the current page.

6.6 Table of Contents, List of Tables and List of Figures

To produce a table of contents, you use the `\tableofcontents` command. When LaTeX encounters this command, it generates a `.toc` file (with the name of the file you're processing as the prefix), which contains the information needed to generate a table of contents. When you've LaTeX'ed the file once, you'll need to do it again in order to read the most recent `.toc` file in to be processed. It is important to note that generating a table of contents is a two parse process.

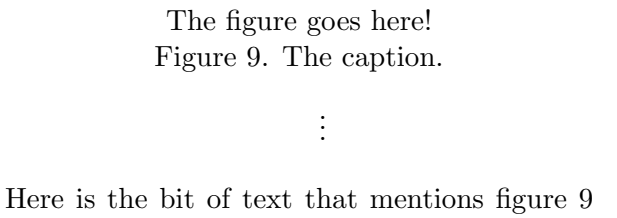
To generate a list of figure (`\listoffigures`) or a list of tables (`\listoftables`), the process is the same, *i.e.*: you must run it through LaTeX twice. The external files produced in this process will have a `.lof` and `.lot` suffix respectively.

Note that where you place these commands in your source file, will be where the list appears in the final document. The same goes for the table of content command.

6.7 Labels and References

With large documents, it is common to have cross references to objects in other parts of the document, *e.g.*: "See table 5 for a summary". Instead of "hard coding" the reference, *i.e.*: the number 5, you can create a symbolic link to the object with the `\label` and `\ref` commands. Hard coded links can become out of date the instant you change or re-arrange portions of a file.

The following example is an one I've used before to to introduce the figure and table environments.



Here is the bit of text that mentions figure 9 for the first time.

```
Here is the bit of text
that mentions figure\ref{fig:myfig}
for the first time.
\begin{figure}
  \begin{center}
  \leavevmode
  The figure goes here
  \end{center}
  \caption{The caption.}
  \label{fig:myfig}
\end{figure}
```

Here the label `fig:myfig` is assigned a number that was generated by the `caption` command. By using the `\ref` command, we get the number that `fig:myfig` represents. In a `\label{`x`}` command, the value that x takes on is context sensitive. If it was placed just after a `section` command, then x will take on the section's number.

If for some reason you wanted to state the page number of the referenced object instead, you can do that by using the `\pageref{`x`}` command in place of the `\ref` command, where x is the label.

As with all one parse compilers, it is necessary to run the document through LaTeX twice in order to get the references right.

6.8 Making a Bibliography

There are two ways to create a bibliography: one involves the use of BibTeX; and the other involves the use of a `thebibliography` environment. Information about BibTeX can be found in Lamport's book and will not be discussed here.

The `thebibliography` environment is much like the `enumerate` environment except that items are marked with a `bibitem` command. Generally placed at the end of a document, this is what a `thebibliography` structure looks like.

For a good read go and buy THE book [Lam 94] in the shop.

⋮

References

⋮

[**Lam 94**] L. Lamport, *A Documentation Preparation System*, Addison-Wesley, Reading, Massachusetts, second edition, 1994.

```
For a good read go and buy THE
book~\cite{lam2e-94} ...

\begin{thebibliography} {abcdefg}
\bibitem[Lam 94]{lam2e-94} L. Lamport, ...
  \emph{A Documentation Preparation ...
...

\end{thebibliography}
```

In this version of `thebibliography` environment, the citation labels (*e.g.*: Lam 94) are user defined. We tell LaTeX we wish to use our own citation labels by adding an optional `[...]` argument to `bibitem`. The `{abcdefg}` argument in the `thebibliography` environment is used to indicate to LaTeX how wide the widest label is, *i.e.*: the argument must be as wide as all the citation labels you intend to define.

If you do not wish to define your own citiation labels, the default is to assign numbers to the citation, *i.e.*:

For a good read go and buy THE book [69] in the shop.

⋮

References

⋮

[**69**] L. Lamport, *A Documentation Preparation System*, Addison-Wesley, Reading, Massachusetts, second edition, 1994.

```
For a good read go and buy THE
book~\cite{lam2e-94} ...

\begin{thebibliography}{99}
\bibitem{lam2e-94} L. Lamport, ...
  \emph{A Documentation Preparation ...
...

\end{thebibliography}
```

As with cross references (see section 6.7), you'll need to run LaTeX at least a couple of times over the source file just to make sure the references are all resolved.

7 All About Math Mode

There are a number of ways to get into a math mode:

```
\(...\)
\[...\]
$...$
```

All three methods are fully-fledged math environments, delimiting the scope of declarations contained within them. For those wishing to use the `\begin...\end` structure, here's how you do it:

`\begin{math}...\end{math}`	
`\begin{displaymath}...\end{displaymath}`	Equivalent to `\[...\]`
`\begin{equation}...\end{equation}`	Gives you numbered equations.

In the following sub-section, I'll give a very brief account of LaTeX's math mode. Please refer to Lamport's book for a full account—there's really too much for a brief introduction document like this.

7.1 Subscript and Superscript

Subscripts and superscript only apply to math mode and are denoted with the `_` and the `^` respectively. Here are some example for its use:

x^{2z}	`x^{2z}`	x^{y^3}	`x^{y^3}`	x_1^y	`$x^{y}_{1}$`
x_{2z}	`x_{2z}`	x^{y_1}	`x^{y_{1}}`	x_1^y	`$x_{1}^{y}$`

7.2 Miscellaneous Functions

Fractions

To make fractions you use the "/" symbol—most suitable for use in running text. However to build bigger fraction, you'll need to use the `frac` command:

```
\[\frac{x*y}{1 + x + \frac{1}{x}}\]
```

$$\frac{x * y}{1 + x + \frac{1}{x}}$$

Roots

Square root uses the `\sqrt` command and excepts a single argument:

What is the square root of $\sqrt{x+2}$ and $\sqrt[n]{3}$.

```
What is the square root
of \( \sqrt{x+2} \) and
\( \sqrt[n]{3} \).
```

Ellipsis

The two commonly used ellipsis are `\ldots` and `\cdots`. For example:

A low ellipsis: $x_1, \ldots, x_n$.
A centered ellipsis: $a_1 + \cdots + a_n$.

```
A low ellipsis: $x_{1}, \ldots, x_{n}$.

A centered ellipsis: $a_{1} + \cdots + a_{n}$.
```

7.3 Math Symbols

Here is a table of Greek letter available in math mode:

α	\alpha	θ	\theta	o	o	τ	\tau
β	\beta	ϑ	\vartheta	π	\pi	υ	\upsilon
γ	\gamma	ι	\iota	ϖ	\varpi	ϕ	\phi
δ	delta	κ	\kappa	ρ	\rho	φ	\varphi
ϵ	epsilon	λ	\lambda	ϱ	\varrho	χ	\chi
ε	varepsilon	μ	\mu	σ	\sigma	ψ	\psi
ζ	zeta	ν	\nu	ς	\varsigma	ω	\omega
η	eta	ξ	\xi				

The uppercase version:

Γ	\Gamma	Λ	\Lambda	Σ	\Sigma	Ψ	\Psi
Δ	\Delta	Ξ	\Xi	Υ	\Upsilon	Ω	\Omega
Θ	\Theta	Π	\Pi	Φ	\Phi		

7.4 Math Symbols

Binary Operation Symbols

$\pm$	\pm	$\cap$	\cap	$\diamond$	\diamond	$\oplus$	\oplus
$\mp$	\mp	$\cup$	\cup	$\triangle$	\bigtriangleup	$\ominus$	\ominus
$\times$	\times	$\uplus$	\uplus	$\bigtriangledown$	\bigtriangledown	$\otimes$	\otimes
$\div$	\div	$\sqcap$	\sqcap	$\triangleleft$	\triangleleft	$\oslash$	\oslash
$*$	\ast	$\sqcup$	\sqcup	$\triangleright$	\triangleright	$\odot$	\odot
$\star$	\star	$\vee$	\vee	$\lhd$	\lhd	$\bigcirc$	\bigcirc
$\circ$	\circ	$\wedge$	\wedge	$\rhd$	\rhd	$\dagger$	\dagger
$\bullet$	\bullet	$\setminus$	\setminus	$\unlhd$	\unlhd	$\ddagger$	\ddagger
$\cdot$	\cdot	$\wr$	\wr	$\unrhd$	\unrhd	$\amalg$	\amalg

Relation Symbols

$\leq$	\leq	$\geq$	\geq	$\equiv$	\equiv	$\models$	\models
$\prec$	\prec	$\succ$	\succ	$\sim$	\sim	$\perp$	\perp
$\preceq$	\preceq	$\succeq$	\succeq	$\simeq$	\simeq	$\mid$	\mid
$\ll$	\ll	$\gg$	\gg	$\asymp$	\asymp	$\parallel$	\parallel
$\subset$	\subset	$\supset$	\supset	$\approx$	\approx	$\bowtie$	\bowtie
$\subseteq$	\subseteq	$\supseteq$	\supseteq	$\cong$	\cong	$\Join$	\Join
$\sqsubset$	\sqsubset	$\sqsupset$	\sqsupset	$\neq$	\neq	$\smile$	\smile
$\sqsubseteq$	\sqsubseteq	$\sqsupseteq$	\sqsupseteq	$\doteq$	\doteq	$\frown$	\frown
$\in$	\in	$\ni$	\ni	$\notin$	\notin	$\propto$	\propto
$\vdash$	\vdash	$\dashv$	\dashv				

Arrow Symbols

$\leftarrow$	\leftarrow	$\longleftarrow$	\longleftarrow	$\uparrow$	\uparrow
$\Leftarrow$	\Leftarrow	$\Longleftarrow$	\Longleftarrow	$\Uparrow$	\Uparrow
$\rightarrow$	\rightarrow	$\longrightarrow$	\longrightarrow	$\downarrow$	\downarrow
$\Rightarrow$	\Rightarrow	$\Longrightarrow$	\Longrightarrow	$\Downarrow$	\Downarrow
$\leftrightarrow$	\leftrightarrow	$\longleftrightarrow$	\longleftrightarrow	$\updownarrow$	\updownarrow
$\Leftrightarrow$	\Leftrightarrow	$\Longleftrightarrow$	\Longleftrightarrow	$\Updownarrow$	\Updownarrow
$\mapsto$	\mapsto	$\longmapsto$	\longmapsto	$\nearrow$	\nearrow
$\hookleftarrow$	\hookleftarrow	$\hookrightarrow$	\hookrightarrow	$\searrow$	\searrow
$\leftharpoonup$	\leftharpoonup	$\rightharpoonup$	\rightharpoonup	$\swarrow$	\swarrow
$\leftharpoondown$	\leftharpoondown	$\rightharpoondown$	\rightharpoondown	$\nwarrow$	\nwarrow
$\rightleftharpoons$	\rightleftharpoons	$\leadsto$	\leadsto		

Miscellaneous Symbols

$\aleph$	\aleph	$\prime$	\prime	$\forall$	\forall	∞	\infty
$\hbar$	\hbar	$\emptyset$	\emptyset	$\exists$	\exists	$\Box$	\Box
$\imath$	\imath	∇	\nabla	$\neg$	\neg	$\Diamond$	\Diamond
$\jmath$	\jmath	$\surd$	\surd	$\flat$	\flat	$\triangle$	\triangle
ℓ	\ell	$\top$	\top	$\natural$	\natural	$\clubsuit$	\clubsuit
$\wp$	\wp	$\bot$	\bot	$\sharp$	\sharp	$\diamondsuit$	\diamondsuit
$\Re$	\Re	$\|$	|	$\backslash$	\backslash	$\heartsuit$	\heartsuit
$\Im$	\Im	$\angle$	\angle	∂	\partial	$\spadesuit$	\spadesuit
$\mho$	\mho						

Beware that some of the symbols mentioned above require a package called `latexsym`. To add this package simply put to following line in the preamble:

```
\usepackage{latexsym}
```

The symbols that need this package are:

\lhd	\unrhd	\Join	\Box
\rhd	\sqsubset	\leadsto	\Diamond
\unlhd	\sqsupset	\mho	

7.5 Other Math Functions

Below is a list of other math functions that may be of use. These functions are treated as a single word and typeset using roman type.

```
\arccos   \cos    \csc    \exp    \ker     \limsup   \min   \sinh
\arcsin   \cosh   \deg    \gcd    \lg      \ln       \Pr    \sup
\arctan   \cot    \det    \hom    \lim     \log      \sec   \tan
\arg      \coth   \dim    \inf    \liminf  \max      \sin   \tanh
```

7.6 Arrays

The `array` environment is similar to the `tabular` environment, except that the `array` environment is used by maths. The syntax for both environments are similar. Here's an example from Lamport's book:

$$\begin{array}{clcr} a+b+c & uv & x-y & 27 \\ a+b & u+v & z & 134 \\ a & 3u+vw & xyz & 2,978 \end{array}$$

```
\(\begin{array}{clcr}
    a+b+c & uv     & x-y & 27\\
    a+b   & u+v    & z   & 134\\
    a     & 3u+vw  & xyz & 2,978
  \end{array} \)
```

For an explanation of `{clrc}` refer to section 6.4 for more details. Each formula in the environment is treated as if it had its own `math` environment.

There are more complex ways to stack the arrays with complex alignments. Again this is beyond the scope of this document. If you are interested in learning more about these functions, please refer to the Lamport's book.

8 Local Packages

The number of packages available for LaTeX is enormous and it would be impossible to list them or even discuss them all here. Luckily, most of them come with documentation that tells you how to use the package. Some even include information about the inner workings of the package (which is useful for learning purposes).

I'll briefly discuss the use of two packages, which we use specifically for laying out software engineering specification documents.

8.1 Including Encapsulated Postscript (EPS) Files

More often than not, there is a need to include some form of graphics in our documents. To do this, we use a package called `graphicx`. Based on the `graphics` package, `graphicx` adds several features to the way graphic files are handled. To utilise this package, you will need to add the:

```
\usepackage{graphicx}
```

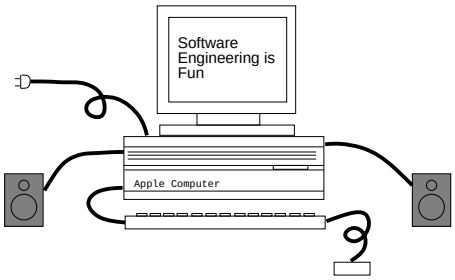
command to the preamble of your document. (*i.e.* just after the `\documentclass` command).

This package is useful for including encapsulated postscript (EPS) files. Please note that EPS files are different to full postscript (PS) files. In order to use an EPS file in LaTeX, it must have a *bounding box* declaration. You can check by looking at the first 10 lines of an EPS file, it should contain a line that looks like this:

```
%%BoundingBox: 74 393 513 540
```

but numbers listed will vary.

In the following example, the `examp` file is being included inside a `figure` environment that is centred and labelled with an appropriate `caption` to indicate its meaning.




```
\begin{figure}[htbp]
  \begin{center}
  \includegraphics[width=6cm]{examp}
  \end{center}
  \caption{Example computer.}
  \label{fig:examp.eps}
\end{figure}
```

Figure 8: Example computer.

EPS pictures may be resized as required be adding the dimensions you want the picture resized to. For example:

```
\includegraphics[width=3cm, height=4cm]{pic}
```

This resizes the picture in the file `pic` to a width of 3 cm and a height of 4 cm.

```
\includegraphics[width=3cm]{pic}
```

This resizes the picture to a width of 3 cm. The height of the picture will be resized appropriately to keep the picture's aspect ratio. Likewise, you can resize the height and let `graphicx` take care of the width by replacing the `width` option with a `height` option in the example above.

```
\includegraphics[width=.75\textwidth]{pic}
```

This example resizes the picture relative to the size of the width of the text on the page. This form of relative resizing can save you a lot of work later should you decide to change the width of text that appears on a page.

When you resize a picture, you can specify the dimensions either in absolute (*e.g.*: 3 cm) or relative terms (*e.g.*: `.5\textwidth`, scale to half the width of the text on the page). In absolute scaling, you can utilise the following units:

`cm` centimetres

`mm` millimetres

`in` inches

`em` 1 em is about the width of the letter M in the current font

`ex` 1 ex is about the height of the letter x in the current font

`pc` picas (1pc = 12pt)

`pt` points (1in = 72.27pt)

When you simply don't care about the dimensions of the picture, other than to make it bigger or smaller, you can utilise the `scale` option. For example the following command:

```
\includegraphics[scale=.5]{pic}
```

simply scales the picture down to half its original size while maintaining the picture's aspect ratio.

When specifying a graphics file to include, you do not have to specify the file's dot extension, `graphicx` expects the file to have either a `.eps` or `.ps` file extension. By omitting the filename extension, you leave open the opportunity for other programs like `pdflatex` to convert your LaTeX file into PDF (see section 9.2 for more information).

9 Useful Utilities

9.1 Converting Graphic Files

One of the most commonly asked question is, "how do I convert graphic or image files to Encapsulated Postscript". Under Linux (even Windows), there are a number of conversion utilities available, but I'm only going to document one that I think is the easiest to use under Linux. For those using Windows, you'll have to find your own.

9.1.1 Using `xv`

`xv` written by John Bradley is program that allows you to view and crop images saved in different file formats (*e.g.*: GIF, JPEG, TIFF). Speak to your System Administrator if you can't find the `xv` program. To view a file, you run the command:

```
xv image.gif
```

When the image appears, you right click the image to open its control panel (see figure 1). To convert from one file format to another, you simply click the **Save** button and select the Postscript format from the drop down box at the top of the "Save As" pane, type in a filename and click **Okay**. The next pane that appears simply allows you to make some adjustments to Postscript output, leave the settings as they are and click **Okay**. The file that `xv` creates is actually an EPS file, you can verify this by looking for the "Bounding Box" statement in the first 10 lines of the converted file.

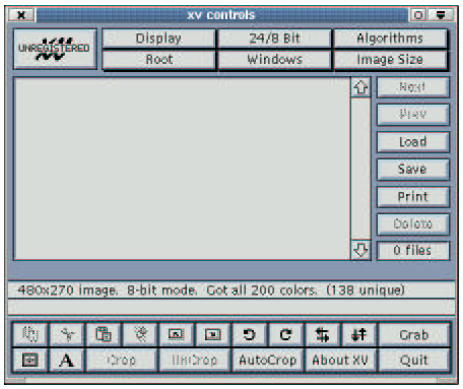


Figure 1: xv control panel

9.1.2 Converting EPS to PDF

When you convert your LaTeX file into PDF (see section 9.2 for details), EPS graphics declared using the `\includegraphic` command (see section 8.1 for details) will not be handled correctly unless the graphic to be included is a PDF.

To convert an EPS graphic to a PDF, you use the `epstopdf` command, for example:

```
epstopdf image.eps
```

where `image.eps` is the EPS file to be converted. When the program terminates, you should find a PDF version of the file with the same filename but with a `.pdf` filename extension in the same directory. If you do not have `epstopdf` installed on your system, see your System Administrator.

9.2 Converting LaTeX Documents Into PDFs

Most up to date LaTeX installations should have `pdflatex` installed; if not, then see your System Administrator. To convert, you simply run the command with the name of the LaTeX file you want converted, for example:

```
pdflatex myfile.tex
```

When the program finishes, you should find a PDF file with the same filename in your current directory but with a `.pdf` filename extension instead.

If you have any `\includegraphics` commands in your LaTeX document, then you must ensure that the graphic to be included is also available as a PDF, otherwise it will not be visible in the final PDF file (see section 9.1.2 for information on how to convert EPS graphics to PDF). If you used the generic `\includegraphics` command in your LaTeX file (see section 8.1) in which you did no specify the suffix of the graphic's filename, then the `graphicx` package will do the right thing by including the PDF version of the graphic when `pdflatex` is used and the EPS version when `latex` is used.

9.3 Converting LaTeX Documents Into HTML

To convert LaTeX documents into HTML files, there is a program called `latex2html` that will do the trick. So the following example,

```
latex2html myfile.tex
```

converts `myfile.tex` into a directory of HTML files called `myfile`. The program's default action is to always create a directory with the same name as the file being converted. If you look inside that directory, you should see an `index.html` file, which is where you should point your browser. The conversion is not always faithful to the source, so some things may not be formatted the way you expected in HTML and you may find it necessary to modify the HTML a little. On most Linux systems, you can usually find out more about this program by using the **man** command, *e.g.*:

```
man latex2html
```

9.4 Spell Checking Your LaTeX Documents

With all the special LaTeX commands embedded in your document, it is still possible to spell check your document (so no excuses for not doing so). With most Linux installations, you should be able to use a program called `ispell`. `ispell` is aware of LaTeX and will skip over all the formatting commands. As it parses your file, it interacts with you, flagging misspelt words and offering possible corrections for you to choose from.

If you are a `emacs` or `xemacs` user, `ispell` is also available from within the editor. To activate it, you have to press `ALT-X`; type the command `ispell-buffer`; and then press the `Enter` key. The whole process is once again interactive so you don't have to leave the editor just to spell check the document.

10 Revision

1.13 Initial release for public viewing.

1.14 Update the section on "Local Packages" and added a new section "Useful Utilities".

References

[Goo 94] Frank Goossens, Frank Mittelback and Alexander Samarin, *The LaTeX Companion*, Addison-Wesley, Reading, Massachusetts, 1994.

[Knu 90] Donald Knuth, *The* TeX*book*, Addison-Wesley, Massachusetts, 1990.

[Lam 86] Leslie Lamport, *A Documentation Preparation System* LaTeX *User's Guide and Reference Manual*, Addison-Wesley, Massachusetts, 1986.

[Lam 94] Leslie Lamport, *A Documentation Preparation System* LaTeX *User's Guide and Reference Manual*, Addison-Wesley, Reading, Massachusetts, second edition, 1994.