LaTeX for Computer Scientists

Les Kitchen
The University of Melbourne
Department of Computer Science and Software Engineering
http://www.cs.mu.oz.au/~ljk/

15 May 2006

© 2003, 2006, Leslie John Kitchen. You may make and distribute unmodified copies of this document in its entirety for non-commercial purposes. In particular, any copies must include this copyright notice and the CVS Id string on the last page. (These conditions are inspired by those used by the ACM.)

Please be aware that this document, in its present state, is work in progress. It is provided, as is, in the hope that it will be a useful aid in learning LaTeX. It may still contain errors, and is certainly incomplete and in need of some re-organization. See corresponding LaTeX source file for details on distribution conditions for source files. The source files will normally be available via the author's webpage, at the URL given above. Note that the distribution conditions may change: Do not assume that distribution conditions for future versions will be identical to those for the current version.

> *This document is based on a document called* Intermediate LaTeX *prepared for a School of Graduate Studies course at the beginning of 2003. The main changes are addition of some additional material, and a change of emphasis towards a Computer Science readership. Having said that, I point out that this document is now approximately twice the size of the document it was based on.*

Non-trivial changes since last version are marked in the margins: small changes by an arrow pointing to the affected text; extended changes by a down-pointing arrow at the beginning of the change, and an up-pointing arrow at the end.

Contents

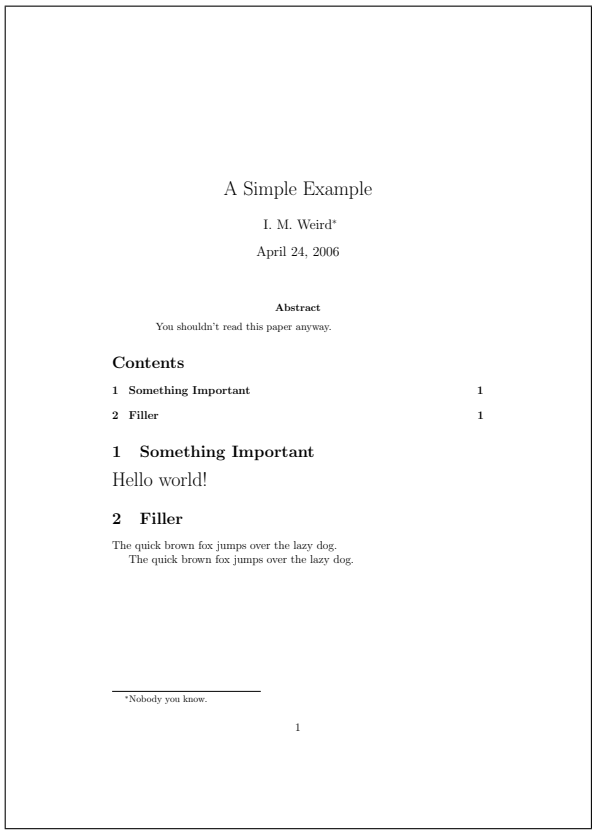
1 What is LaTeX?

1.1 A simple example

```
\documentclass[12pt,a4paper]{article}
% This is a comment
\title{A Simple Example}
\author{I. M. Weird\thanks
    {Nobody you know.}}
\begin{document}
\maketitle
\begin{abstract}
You shouldn't read this paper anyway.
\end{abstract}
\tableofcontents
\section{Something Important}
{\LARGE Hello world!}
\section{Filler}
The quick brown fox
jumps over the lazy dog.

The    quick     brown    fox
jumps
over     the    lazy     dog.
\end{document}
```

produces after processing the printed or displayed output shown in Figure 1.



A Simple Example

I. M. Weird*

April 24, 2006

Abstract

You shouldn't read this paper anyway.

Contents

1 Something Important

Hello world!

2 Filler

The quick brown fox jumps over the lazy dog.
The quick brown fox jumps over the lazy dog.

*Nobody you know.

1

Figure 1: Sample LaTeX output.

1.2 Well, what is it?

LaTeX is in effect a programming language for creating quality typeset documents, designed so that most of the time you use it like a high-level (declarative) markup language.
It thus has some commonality with other markup languages like HTML, but is intrinsically more powerful and more oriented towards quality typesetting.

1.3 History

- Before (electronic) computers, (human) editors would *mark up* a manuscript with instructions to the (human) printers on how to typeset the document.
- Donald Knuth initially developed TeX (along with MetaFont) in the late 1970s.
- TeX is basically a high-quality programmable typesetting engine, combining the text of the

document with embedded markup to specify the formatting automatically.

- MetaFont is a system for designing new fonts.
- Knuth provided a "format" called *plain* TeX as a layer on top of raw TeX.
- Plain TeX provided many convenient features, but was still somewhat difficult and unforgiving.
- Leslie Lamport built LaTeX on top of TeX in the early 1980s to provide better, cleaner support for document production.
- That first released version was LaTeX 2.09.
- Current version is LaTeX 2_ε.
- LaTeX version 3 is in the offing.

1.4 TeX and LaTeX

- LaTeX is built on top of TeX.
- These days, pretty much all usage of TeX is via LaTeX.[1]
- Almost all features of plain TeX are accessible in LaTeX, but are generally used only by wizards, and are mostly hidden from ordinary users.
- From here I'll speak mainly about LaTeX, but remember TeX is always in the background.

1.5 What's good about LaTeX?

All are somewhat inter-related.

- High quality typesetting, especially for mathematics.
- Simple text representation:
  - Create and modify with any text editor.
  - Suitable for version control (CVS, subversion).
  - Scripts can generate LaTeX automatically.
- Cross-platform.
- Stability and compatibility is a major strength.
- Largely "declarative".
- Configurable, extensible and programmable.
- Suitable for collaborative document development.
- Can be used with build tools like `make`.
- Large, active community of users and contributors.
- Many plug-in *packages* available.
- Multilingual.
- Editor support (e.g. `latex-mode` in `emacs`).
- GUI interfaces (TeXshop, WinShell) and WYSIWYG front ends (Lyx).
- Non-proprietary.
- Free!
- Distributed under a GNU-like[2] Free Software licence.

1.6 What's "bad" about LaTeX?

- Less immediate feedback compared with WYSIWYG.[3]
- Lots of braces to match...
  - ...though syntax-aware editors can help a lot.
- Power of programming can also bring the headache of debugging.

[1] One notable exception is `texinfo`, which the GNU project uses to produce "info" documentation from TeX sources.

[2] One feature of the LaTeX licence is that if you modify a standard file, you *must* call it by a different name. This is to avoid compatibility problems where people might think they're using say a standard package, but are in fact getting a mutant.

[3] WYSIWYG stands for "What You See Is What You Get", which however also means "What You See Is *All* You've Got".

- A lot to learn to fully master LaTeX, though you can do most simple document structures fairly intuitively.
- Sometimes default settings are over fussy—though this is configurable (as is almost everything in LaTeX).
- Arguably extra typing initially, though this cost is in the long term greatly outweighed by savings in maintenance and modification.
- Probably not worth the overhead for simple one-off things like letters (unless you set great store by beautiful typesetting even for your correspondence), but then true LaTeX devotees use it even for their shopping lists.

1.7 Summing up

- LaTeX separates logical structuring from visual formatting.
- LaTeX is particularly good for large-scale documents which will have a long lifetime of revisions, and will have to be distributed in various formats and versions.
- While LaTeX can provide groovey typesetting, that's no substitute for good organization and clear expression.
- LaTeX, however, can assist greatly by taking care of a lot of the bookkeeping involved in producing an effectively presented document.

2 Philosophy

2.1 What I can achieve here

LaTeX is way too big to cover in just a couple of hours!

- Show you how some things work.
- Show you how to do some things.
- Give you an idea of what's possible.
- Point you to some references.
- Prepare you to learn and experiment on your own.

I'll concentrate more on *what* you have to do rather than *how* LaTeX does it—though knowing how it works can help your understanding.

2.2 Levels at which you can use LaTeX

1. Straight, ordinary LaTeX.
2. Minor customization of ordinary classes and packages, via options and parameter settings.
3. Defining your own commands, environments, etc.
4. Use of various contributed packages, etc., some in the standard distribution, some you have to download yourself[4].
5. "Getting under the bonnet": redefining LaTeX's standard commands to do what you want; creating your own packages and classes, etc.

I won't touch on 5.

2.3 Learning to use LaTeX

- Learning to use LaTeX is like learning a language: it *is* learning a language:
  - Grammar, vocabulary, idioms,... and practice, practice, practice.
- What it costs you:
  - Some effort in learning and looking things up, especially to start with.
  - A less immediate way of doing things, like making a bowl out of ceramic instead of plasticine: You don't really know how the ceramic bowl will turn out until you've fired it in the kiln.
- What you gain:
  - Much greater expressiveness and control than with WYSIWYG.
  - A record of *how* you did things.

[4] Almost everything available in the TeX world is on the TeXLive CD [10].

  - By looking at other people's LaTeX sources, you can find out how they did things, and learn how to do it yourself.
  - All the technical and aesthetic benefits of LaTeX in typography, especially mathematics.
  - Membership in the global community of LaTeX users.

- LaTeX shines most for large documents that have to undergo revisions and reformatting for different purposes: e.g., chapters of a thesis redone for journal publication.
- A good frame of mind to have is that when you create a LaTeX document, you're not creating just that one-off document, but you're creating (or at least preparing for) all the future versions of that document.

  This can require a little forethought and planning.
- LaTeX is very *linguistic*: you *name* things rather than *point* at them.

3 Basic LaTeX

3.1 Processing

- Starts out with LaTeX source file typically with extension `.tex` (though `.ltx` is sometimes used).
- E.g., `latex foo.tex` (or `latex foo` — `latex` provides a default extension of `.tex` if not given).
- Main product is the corresponding DVI ("device independent") file, in this case `foo.dvi`.
- The DVI file describes where each character glyph and graphic goes on each page.
- The DVI file can be viewed directly using a DVI viewer like `xdvi`.
- More often, the DVI file is converted to PostScript using `dvips`.
- The resulting `.ps` file can be:
  - viewed using a PostScript viewer, like `gv` or GSView (which provide GUIs on top of Ghostscript),
  - printed on most laser printers,
  - "distilled" to PDF using Adobe Acrobat Distiller, or the free `ps2pdf` (which comes with Ghostscript).
- There is also a variant program `pdflatex`, which produces PDF directly from LaTeX source.

3.2 Associated files and re-running

There are a number of files associated with the processing of a `.tex` file. Some of the most important are:

`.aux` The *auxiliary* file, which `latex` uses to keep track of cross-referencing and similar information.

`.log` The *log* file, which contains an even more detailed account of processing than is written to standard output.

`.toc`, `.lof`, `.lot` Used for information respectively for the *Table of Contents*, *List of Figures*, and *List of Tables*.

`.bbl` The bibliography created by `bibtex`, and used by `latex`.

There are a number of other files associated with processing of bibliography, index and glossary. A few things to note:

- `latex` is strictly one-pass: To cope with forward references, like cross-references to later sections, and page numbers in the table of contents, `latex` makes use of information gathered *on the previous* run in the `.aux`, `.toc` files, etc.
- This means that after a significant change to the document, it can take two runs[5] of `latex` to get cross-references, etc., perfectly correct.

[5] Those theoretically inclined will see that this is a kind of fixed-point iteration. It's actually possible to construct pathological LaTeX documents that never converge. I've never known this to happen in practice, but constructing such a document is an interesting brain-teaser.

(On the first run, `latex` was still using cross-reference information generated *before* the change.)

- Symptoms of this are messages from `latex` like "There were undefined references" or "Cross-references may have changed, rerun latex to get cross-references right".
- Some people superstitiously always run `latex` twice, or make use of scripts (usually called something like `makeltx` or `latexmake`), which iterate running `latex` until the cross-references stabilize. See also notes about using `latex` under `make`.
- In practice, this is rarely a problem. You really only need perfectly correct cross-references in the final version. When you're actively developing a document, it usually doesn't matter if the cross-references are a little out of synch.
- Which associated files are actually produced depends on what features you ask for. For example, if you don't have `\tableofcontents`, then no `.toc` file is produced.

3.3 Characters to pages

As the engine underneath LaTeX, TeX works at a number of levels to produce the typeset output. To simplify the real story a bit:

- At the lowest level, TeX deals with *boxes*.
- Boxes are mostly the *glyphs* for printed characters and symbols, but they may be other things like graphics.
- Most of TeX's work is in creating a hierarchical assemblage of boxes from the input to produce an aesthetically typeset result.
- Character boxes get assembled into word boxes.
- Word boxes get assembled into line boxes.
- Line boxes get assembled into paragraph boxes.
- Paragraph boxes get output onto pages.
- In between the boxes goes adjustable space, which TeX can stretch or shrink (within limits) to achieve the best looking output.
- In TeX-speak, for historical reasons, these adjustable spaces are called *glue*, but they're probably better conceptualized as being like invisible springs.
- The main unit of TeX's typesetting is the paragraph. TeX makes its decisions about where to break the text of a paragraph into lines using a quite sophisticated algorithm, which tries to optimize the appearance of the entire paragraph, taking into account a number of typographic factors.
- These typographic factors are assigned various numeric weights. TeX has well-chosen default values. Most of LaTeX's layout effects, like centering and flush right, are achieved not by using a different algorithm, but merely by altering the parameters of TeX's paragraph algorithm.
- Because of this, a change at the end of a paragraph can actually affect the line breaking at the beginning, because of the way it shifts the balance of the optimization.
- This whole-paragraph approach is one of the reasons why TeX's typesetting looks so good, but also why it's difficult to do a proper WYSIWYG version of TeX or LaTeX.
- One consequence is that the typesetting of a paragraph is controlled by the settings in effect at the *end* of the paragraph (when TeX's algorithm kicks in). This affects the use of declarations like `\sloppy`: the scope of the declaration must include the blank line (or `\par`) that ends the paragraph. See [7, p. 94f] for more details.
- TeX's multi-factor approach can be subtle. Sometimes you may see TeX produce something that looks like a no-no, like having a word protrude slightly into the right margin when you wanted fully justified text. Invariably, this is because the alternative line

breaks would have been worse in the overall assessment.

- Another consequence of TEX's approach is that very little in TEX's typesetting is absolute. For example, the "unbreakable space" `~` does not actually *forbid* a line break. It merely assigns a very high penalty to breaking at that point. In normal situations, TEX's algorithm will avoid breaking there to avoid incurring the penalty in the optimization.
- In doing its job TEX also has to break apart boxes it has already made. For example, it may need to break a word box to hyphenate a word to get a better line break. Or it may need to break a paragraph between two lines to get a good page break.

3.4 Input conventions

- LATEX input is an ordinary text file, which mixes the content of the document with markup to process it.
- The input conventions are reminiscent of the Unix shell (though not the same), in that some characters are "ordinary", while some are "special" and trigger special treatment.
- Most characters (letters, numbers, punctuation) are "ordinary", and represent themselves.
- The following characters are "special":

  `$` Switches TEX into *math mode*.

  `\` Indicates a *command*.

  `{ }` Indicate grouping and enclose the actual arguments of a command.

  `~` Unbreakable interword space.

  `_ ^` Indicate respectively subscripts and superscripts in math mode.

  `#` Used for formal arguments of a command.

  `&` Separates cells in rows of a table.

  `%` Indicates a comment. The comment runs from the `%` to the end of line (a bit like C++ `//` comments).

- The characters `@ [ ]` can be used in ordinary text, but behave specially in certain circumstances.
- The characters `+ - = < >` are mainly used in math mode, and there behave as you'd expect. They can be used in ordinary text, but may not always give what you expect in that in some fonts that character code is assigned to a printed character different from what's on the keyboard. `< >` gives ¡ ¿.
- As with most programming languages, whitespace and newlines are used to delimit tokens. Additional whitespace and newlines beyond this have no effect, though they can be used for source layout.
- One exception is that a one or more consecutive blank lines indicate a paragraph break.[6]
- A consequence of all this is that you can't normally get extra horizontal whitespace on the page by typing extra spaces into your LATEX source. Nor can you get extra vertical whitespace by putting in extra blank lines. These effects have to be achieved by invoking explicit commands.

3.5 Combinations

Some combinations of otherwise ordinary characters are treated specially:

- `‘‘` and `’’` give proper opening and closing (double) quotation marks, aka "inverted commas". (Don't use `"`, the typewriter double quote!)[7]
- Similarly, `‘` and `’` give single quotes. (The latter also gives an apostrophe in ordinary text and a prime in math mode).

[6] A blank line is thus equivalent to the command `\par`, and in some circumstances it's clearer to use the explicit `\par` command.

[7] In `emacs`'s `latex-mode` you *can* type in `"`, because that key is bound to a command that automatically inserts the appropriate LATEX opening or closing quotation marks depending on the context. By way of clarification, `‘` is the ASCII grave accent, aka "backtick", top-left on most keyboards; `’` is the "single-quote" or apostrophe, just right of semicolon on most keyboards. Actually, if you use typewriter double quote, `"`, you get a proper closing double quotation mark.

- In ordinary text, `-` gives an intra-word hyphen, `--` gives an *en dash* used for ranges like "17–42", and `---` gives an *em dash*—used for sentence punctuation.

  In math mode `-` gives a minus sign, as in $-x - y$ (produced by `$-x-y$`), which is different again.[8]

- Less obviously, certain combinations, like `fi` and `fl` give proper typographic *ligatures*: "fi", "fl"—not "fi", "fl".

- Note that you get the ellipsis "…" by the *command* `\ldots` (for "low dots"). Typing three dots `...` in your source gives dots too close together for English typographic conventions, "...". This is a case where something you'd expect to be done by a combination isn't. (There are a number of other dots commands available only in math mode.)

3.6 Commands

- Commands come in two varieties:

  1. `\` followed by a single non-letter character
  2. `\` followed by a sequence of letters

- Examples of the first kind: `\$ \& \% \# \_ \{ \}` produce respectively $ & % # _ { }; while `\,` produces a "thin" space.[9]

- Examples of the second kind: `\S`, `\P` and `\copyright`, which produce respectively the section, paragraph and copyright symbols, §, ¶ and ©; `\maketitle`, which produces the document title, and `\large` which switches to large font size, and `\Large`, which switches to an even larger font size.

- Note that such command names can consist *only* of letters (not even digits), and are case-sensitive.

- Command-name delimiting:

  - The name of the command runs until the first non-letter.
  - Often this will be some punctuation, but may be whitespace.
  - If the command name is terminated by whitespace, then *all* this whitespace is gobbled up.
  - This creates a problem if we want some white space in the output after the result of the command.
  - Various solutions are presented later.

- Arguments:

  - Most commands take one or more arguments.
  - Arguments follow the command name and are enclosed in braces.
  - The outer braces just enclose the argument and are stripped away before the argument is actually passed to the command.
  - Any given command has a fixed arity (number of arguments that it accepts). But see later point.
  - LaTeX checks and generally complains if a command is given insufficient arguments. Additional arguments will not be passed to the command, and in general will end up being treated as following text that just happens to have been put in grouping braces.

- Optional arguments:

  - Quite a few commands also accept *optional* arguments.
  - The optional arguments if supplied are enclosed in square brackets, and usually but not always come before the obligatory arguments.
  - If an optional argument is not supplied, then some default is used.
  - There's the potential problem that some ordinary text with square brackets following the command may be mistaken

[8] If you look closely, you can see that LaTeX puts different space around the minus sign depending on whether it's a unary or binary operator.

[9] The thin space is useful for quotes within quotes, to give a little separation between an outer double quote and an inner single quote, like:

"'Work' is a four-letter word", said John.

for an optional argument. The solution is to enclose that text inside grouping braces, which protects it from being interpreted as an optional argument.

- With rare exceptions, paragraph breaks are not permitted in command arguments. This is a deliberate safety feature, to catch missing closing braces on arguments. It triggers LaTeX's "Runaway argument?" error messsage. The idea is that command arguments should generally be fairly small chunks of material, and if you encounter a paragraph break it's more likely because the argument isn't properly closed.

3.7 Environments

- To provide handling of chunks of material bigger than can reasonably be passed as command arguments, LaTeX also provides *enviroments*.
- Environments are invoked by two matching commands, `\begin` and `\end`.
- They each take as argument the name of the environment (no backslash).
- The text between the `\begin` and the corrresponding `\end` is affected by the formatting of that environment.
- The `\begin` and `\end` must nest properly.
- For some environments, the `\begin` command takes additional arguments to control the formatting of the environment.
- Environments provide local scope for declarations. (That is, the effect of non-global declarations made inside an environment are undone at the end of that environment.)

3.8 After-sentence space

- In typography (at least in English-speaking countries) it's conventional to put a little extra space after punctuation that ends a sentence. (Usually a full stop.)
- But figuring out what is a sentence is way beyond a mere program.
- So LaTeX follows a rule which works correctly almost all the time, and has the advantage of being simple and predictable, so you can easily tell when you need to over-ride LaTeX.
- If the full stop comes after an upper-case letter, then LaTeX assumes it's marking initials (like N.S.W.) and treats the following space as ordinary interword space.
- If the full stop comes after a lower-case letter, then it's treated as marking the end of a sentence, and extra following space is inserted.
- This extra space isn't fixed. Like many spaces in LaTeX it's stretchable (or squashable) as needed to optimize the typesetting, but in general it's larger than an ordinary inter-word space.
- What about when LaTeX gets it wrong?
  - Putting a `\ ` (yes, that's a backslash followed by a space) immediately after the full stop puts an ordinary interword space there, even if the preceding letter was lower case.
    This is useful for abbreviations inside a sentence.
  - Putting a `\@` immediately before the full stop forces it to be treated as end-of-sentence punctuation, making the following space bigger.
- This adjustment may be necessary even when other punctuation, like parens and quotes, intervenes between the full stop and the following space.
- Similarly for other sentence punctuation: question mark, exclamation mark and colon.
- Don't lose too much sleep over this. If you get it wrong, at worst you'll have a little bit too much or too little space after some punctuation, which probably only a dedicated typographer will notice, not your average punter.

3.9 Accents and symbols

- There are a relatively small number of symbols, such as the ¶ and § already mentioned, available in ordinary text. In our Department, `\o`, which produces ø, comes in handy for names like *Søndergaard*.
- A *huge* number of symbols (numbering in the thousands) are available, but mostly only in math mode.

  This is not a serious impediment, since you can always switch temporarily into math mode to get any symbol as you need it. See later.
- There are a number of commands for putting various accents on letters, etc., in ordinary text (and many more in math mode).
- For example: `\‘{e}`, `\’{e}`, `\^{a}`, `\"{o}`, `\c{c}` produce è, é, â, ö, ç respectively.
- To put accents on the letters “i” and “j” you need to use the *dotless* versions, produced by the commands `\i` and `\j`, respectively like ı from `\={\i}`.
- Such commands are adequate for small chunks of non-English text needing accents, but not for producing whole non-English documents. For that see later.

3.10 Declarations, grouping, and scope

- Some commands produce some sort of output, such as the commands to produce special symbols or even tables of contents.
- But many commands just change LaTeX's state: They are *declarations*.
- It'd be a pain if every time you had a declaration you had to later put in another declaration to restore LaTeX to its previous state (which you might not even know).
- Therefore, almost all declarations in LaTeX are local to some *scope*: They have their effect when processed, but only temporarily — their effects are undone and the original state restored on leaving that scope.
- Scope is provided by *environments* and by grouping braces.[10]
- There are, however, a small number of declarations which are *global* and do not obey these local scoping rules. One of these is `\pagestyle`, for instance.

3.11 Appearance of type

- LaTeX provides a number of ways of changing the appearance of the type.
- Used with judgement (and not overdone) such control of appearance can aid understanding, by providing consistent, visual typographic markers.
- Type size:

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

produces

tiny scriptsize footnotesize small normalsize large
Large LARGE huge Huge

Note, sizes are relative to the base size of the document, which normally can be 10pt (default), or 11pt or 12pt, as specified as option to `\documentclass`.

[10] Except, of course for the outer braces that enclose a command argument, which are stripped away before the argument is passed to the command. The outermost braces around the body of a command definition are likewise outermost braces around the arguments of the `\newcommand` command and so are stripped away when the command is defined, and don't provide scoping. This is necessary so that a command can expand, say, to a declaration and still have effect. Analogous remarks apply to the two bodies of an environment definition.

- Type style can be done via declarations:

```
{\rm Roman}
{\it Italic}
{\bf Boldface}
{\sf Sans Serif}
{\sl Slanted}
{\sc Small Caps}
{\tt Teletype or typewriter}
```

produces

Roman *Italic* **Boldface** Sans Serif *Slanted*
Small Caps `Teletype or typewriter`

- Or, usually more conveniently, via commands with an argument.

```
\textrm{Roman}
\textit{Italic}
\textbf{Boldface}
\textsf{Sans Serif}
\textsl{Slanted}
\textsc{Small Caps}
\texttt{Teletype or typewriter}
```

produces

Roman *Italic* **Boldface** Sans Serif *Slanted*
Small Caps `Teletype or typewriter`

- Almost always, the command versions are preferable, since they take care automatically of the so-called *italic correction*:

```
Normal {\it italic text} back.
\\
Normal \textit{italic text} back.
\\
Normal {\it italic text\/} back.
```

produces

Normal *italic text* back.
Normal *italic text* back.
Normal *italic text* back.

It's subtle, but, in the first line, because of the oblique italic font the "t" of "text" leans a bit too close to the "b" of "back". In the second and third lines a *tiny* bit of extra space is added, either automatically by the `\textit` command, or explicitly by the *italic correction* `\/` command.

- However, the declaration forms are more convenient if you're applying the type-style change to a whole chunk of text, like an entire environment.
- Subject to availability of fonts, these type styles can be combined:

```
Ordinary \textit{italic}
or \texttt{typewriter}
\\
\textbf{Bold \textit{italic}
or \texttt{typewriter}}
```

produces

Ordinary *italic* or `typewriter`
Bold *italic* or `typewriter`

For some font families, not all combinations may be available.

- You really should never use such type-changing commands in the body of your document.
- Rather, if some identifiable part of your document needs special typographic treatment, you should define a command (or environment) for it. (See later.) You do the type changing in the definition of the command, then tag the text in your document that needs the special treatment with this command.
- An important exception to this is the `\emph` command. It is really more like semantic markup, indicating that the text is to be *emphasized*, by the normal convention of putting it in italics. It has the corresponding `\em` declaration, which is useful for emphasizing larger blocks of text (like an entire environment).

```
\emph{The convention is that
\emph{emphasized} text within
emphasized text is set in
\emph{roman} style.}
```

produces

The convention is that emphasized *text within emphasized text is set in* roman *style.*

- Never use underlining! Underlining was used to indicate emphasis on limited archaic devices like typewriters. It really has no place in proper typesetting, and (depending on how it's done) can cause numerous problems. If the underline is just below the baseline, it can interfere with descenders[11] and severely affect legibility. If the underline is positioned so as to avoid interfering either with descenders (or with ascenders[12] on the next line), it requires introducing some additional space between the affected lines, which breaks up the visual unity of the paragraph. It's even worse when an underlined word gets hyphenated across two lines, because then more lines are affected.

  In normal text, you can use `\emph`; in mathematics you can use conventions like `\boldmath`, to indicate things like vectors, which are conventionally underlined in typed or handwritten manuscripts.

  Well, I exaggerate by saying never. There are some rare situations in which it might make sense to use underlining. For example, it can provide an additional means of indicating consistent distinctions in mathematics, once you've used up all you can with changes of typeface and accents. And perhaps even in ordinary text you might want underlining for some special indicative purpose, with all its problems. To this end, LaTeX does have some facilities and packages for underlining, both in ordinary text and in mathematics.

[11] A *descender* is that part of letters, like "g" and "q", that goes below the baseline.

[12] An *ascender* is that part of lower-case letters, like "b" and "k", that goes above the height of a lower-case "x".

- There are additional type size and style features available in math mode.

3.12 Text justification

- Normally, LaTeX fully justifies the output text, on left and right.
- Other forms of justification can be achieved by the `center`, `flushleft`, and `flushright` environments.

```
\par
How to Dominate Men,
Subjugate Women, and
Stupefy Children
by
Salvador Dali
\par
```

produces

How to Dominate Men, Subjugate Women, and Stupefy Children by Salvador Dali

```
\begin{flushleft}
How to Dominate Men,
Subjugate Women, and
Stupefy Children
by
Salvador Dali
\end{flushleft}
```

produces

How to Dominate Men, Subjugate Women, and Stupefy Children by Salvador Dali

```
\begin{flushright}
How to Dominate Men,
Subjugate Women, and
Stupefy Children
by
Salvador Dali
\end{flushright}
```

produces

How to Dominate Men, Subjugate Women, and Stupefy Children by Salvador Dali

```
\begin{center}
How to Dominate Men,
Subjugate Women, and
Stupefy Children
by
Salvador Dali
\end{center}
```

produces

How to Dominate Men, Subjugate Women, and Stupefy Children by Salvador Dali

The text is a chapter heading from [2].

- There are corresponding declarations, `\centering`, `\raggedright`, and `\raggedleft`. They aren't often used, but come in handy sometimes.

  In fact, I've come to appreciate their handiness more recently. The environment forms, as well as altering TEX's formatting parameters to achieve their effects, also introduce some vertical space above and below the formatted text. In most cases, like a hand-formatted centered heading, this is what you want. The declaration forms just alter the formatting parameters, without introducing the extra vertical space. This can be useful, particularly when you're defining your own formatting environments, when you want the layout effects without the space.

- As usual, you should normally avoid using such explicit justification changes in the body of your document. If some identifiable part of your document needs such special treatment, do it via a user-defined command or environment.

3.13 Line and page breaks

- The `\\` command normally gives a line break (without starting a new paragraph). It's particularly handy in centered text, where usually we want to control the division into lines.

```
\begin{center}
How to Dominate Men, \\
Subjugate Women, and \\
Stupefy Children
\\ by \\
Salvador Dali
\end{center}
```

produces

How to Dominate Men,
Subjugate Women, and
Stupefy Children
by
Salvador Dali

- The `\\` command can take an optional length argument, which specifies how much additional vertical whitespace to insert after the line break.

- There's also a "starred" form `\\*`, which inhibits a page break at the line break. With care, it can be used to keep separate lines together on the same page, although there are perhaps better ways of achieving this.

- Normally, LATEX does a good job of deciding on page breaks, but you can force them if necessary:

  `\newpage` Goes to a new page (new column in two-column mode).

  `\clearpage` Goes to a new page.

  `\cleardoublepage` Like `\clearpage`, but in two-sided printing takes us to an odd-numbered (right-hand) page, inserting a blank left-hand page if necessary.

  Both the "clear" forms also force out any accumulated floats.

- There are also various commands to encourage or discourage line or page breaks at a given point: `\linebreak`, `\nolinebreak`, `\pagebreak`, `\nopagebeak`. They each take an optional argument, a number 0 to 4, to specify the degree of encouragement or discouragement.

3.14 Displays

- Often text needs to be set apart visually, or "displayed"
- LaTeX provides several environments for these purposes:

  `quote` For short quotations.

  `quotation` For longer (multi-paragraph) quotations.

  `verse` For verse and poetry.

- There are also environments for displaying mathematics, which will be described separately.

3.15 Lists

- LaTeX provides a number of environments for lists of items:

  `itemize` For itemized (bulleted) lists.

  `enumerate` For numbered lists.

  `description` For tagged "description" lists.

- Within the list, each item is begun with an `\item` command.
- In a `description` environment, the optional argument to `\item` provides the term to be described.
-

```
\begin{enumerate}
\item First item, which has a
      nested itemized list
      \begin{itemize}
      \item First nested item
      \item Second nested item
      \end{itemize}
\item Second item
\end{enumerate}
```

produces

1. First item, which has a nested itemized list
   - First nested item
   - Second nested item
2. Second item

-

```
Three animals you should
know about are:
\begin{description}
\item[gnat]
     A small animal, found
     in the North Woods,
     that causes no end of
     trouble.
\item[gnu]
     A large animal, found in
     crossword puzzles,
     that causes no end of trouble.
\item[armadillo]
     A medium-sized animal,
     named after a
     medium-sized Texas city.
\end{description}
```

produces

Three animals you should know about are:

gnat A small animal, found in the North Woods, that causes no end of trouble.

gnu A large animal, found in crossword puzzles, that causes no end of trouble.

armadillo A medium-sized animal, named after a medium-sized Texas city.

(This example is take from [7].)

- These environments nest gracefully (up to four deep, at least).
- The "bullets" and number style depend on the level.
- As with most things in LaTeX, the formatting of these environments is highly configurable, though the default serves for almost all purposes.
- There are also some lower-level list primitives for defining your own list structures.
- Many standard LaTeX environments are actually defined in this way as trivial, one-item lists.

3.16 Overall source document structure

- LaTeX documents follow a consistent structure:

```
\documentclass[a4paper]{article}
preamble
\begin{document}
body
\end{document}
```

- In the *preamble* go various declarations to load packages, set various document parameters, and define new commands and environments, etc.
- Nothing in the preamble should generate any text.
- In the *body* goes the actual text of the document, which may be lengthy and spread over several files.

3.17 Document class

- Aside from whitespace and comments, and some uncommon constructs we won't describe here, every LaTeX document has to start with a `\documentclass` declaration.[13]
- The `\documentclass` declaration tells LaTeX what kind of document it is. The standard document classes are:

  `article` An article, such as a conference paper or journal article. Use this when nothing else fits.

  `report` For something longer, like a technical report.

  `book` For a full-scale book.

  `slides` Originally for overhead transparencies. Nowadays probably more useful for producing say PDF "slides" to be displayed on screen. Automatically chooses appropriate big fonts.[14]

  `letter` For letters (like you put in an envelope with a postage stamp).

- Additional classes have been developed by various organizations, like publishers and professional societies, for specific purposes, like to ensure that all papers submitted to a particular conference are formatted in a consistent way.
- The class determines the overall formatting of the document, and may make class-specific commands and environments available.
- Options can be used to modify the standard settings for that document class. Commonly used are:

  `11pt 12pt` To set the document with a base type size of 11 or 12 points, instead of the default 10 points.

  `a4paper` To format for A4 paper, instead of the default U.S. Letter size.

  `oneside twoside` To format respectively for one-sided or two-sided printing.

  In two-sided printing, LaTeX uses different margins for odd-numbered (right-hand) and even-numbered (left-hand) pages. In one-sided printing, all pages are treated as right-hand pages.

  The default for `book` is `twoside`; the default for `article` and `report` is `oneside`.

  `twocolumn` Format for two-column pages.

  `landscape` Format for landscape printing.

- Not all options apply to all classes: for example, you can't have two-column slides. (Well, not via the standard class options.)
- Options like `a4paper`, `twoside` and `landscape` usually affect only LaTeX's formatting of pages. It will sometimes take further downstream intervention to get the

[13] It is actually possible to `\input` the `\documentclass` declaration. This can sometimes be useful for automatically generated documents.

[14] There are a number of packages, like `texpower`, `prosper`, and `foiltex`, designed to support presentations, with some support for animations and transitions in the generated PDF.

actual printer or on-screen display program to respect them.

- For would-be wizards: Classes are defined by `.cls` files, which are just files of LaTeX definitions following special conventions. They are looked up and loaded following the same `TEXINPUTS` path as `\input`. If you're curious, you can look at the standard document classes to see how they're done. Beware, they'll usually contain quite a lot of low-level TeX commands. Try something like `locate article.cls` to find out where they're stored on your system.

3.18 Packages

- LaTeX comes with a *huge* number of standard and contributed "packages", which provide extra facilities.
- As as rule, if you encounter some typographic problem, almost always there's an existing package to do the job.
- Packages are loaded with the `\usepackage` command.
- Its arguments are a list of package names, separated by commas (no spaces), though usually it's cleaner to load just a single package with each call.

```
\usepackage{color,graphicx}
```

  Loads both the `color` and `graphicx` packages.

- `\usepackage` can take an optional argument, which gives a comma-separated list of options which are passed to the packages.

```
\usepackage[french]{babel}
```

  Loads the `babel` package (which provides multilingual support), passing it the `french` option, to set French as the default language.

- For would-be wizards: As with classes, packages are defined by files of LaTeX definitions. For historical reasons, since they were formerly called "styles", they have extension `.sty`. If you're curious, try something like `locate color.sty` to find where the standard packages live on your system, so you can take a look at how they're done. Again, beware, they'll usually contain a lot of low-level TeX.

3.19 Front stuff

- `\maketitle` produces the document's title in the appropriate format for the document class.
- It draws on information from the `\title`, `\author`, and optionally `\date` declarations in the preamble.
- The information for separate authors is separated by `\and`.
- Multiple lines for an author (say to give address) can be obtained by `\\`.
- Inside the arguments of `\title` and `\author` things like acknowledgements and affiliations can be given by the `\thanks` command (whose argument will normally end up as a footnote).
- The abstract is produced by the `abstract` environment.
- The table of contents, list of figures, and list of tables are produced, respectively, by the `\tableofcontents`, `\listoffigures` and `\listoftables` commands.

3.20 Document sectioning

- LaTeX provides a hierarchical organization of sectioning commands—from highest to lowest:

Command	Level
`\chapter`	0
`\section`	1
`\subsection`	2
`\subsubsection`	3
`\paragraph`	4
`\subparagraph`	5

- To maintain the hierarchy, any sectional unit must be contained in the next higher unit.
- However, there is no `\chapter` command defined in the `article` class, so for articles `\section` is the highest-level sectioning command. (This is intended to make it easier to include an article as a chapter of a book.)
- Sectional units are normally numbered automatically: A sectioning command increments the number for that sectioning level, and resets the numbers for lower-level sectional units. "Lower-level" means bigger (deeper) level numbers.
- Sectioning commands take an argument which provides the heading for the sectional unit, the corresponding entry for the table of contents, and a running head for the page (for some page styles).
- They can also take an optional argument which is used for the table of contents and running head for the situations when you want it to be different from the heading.

  A common reason is that for a long section heading, too long to fit in the column width, you might want to introduce a `\\` to put the line break where you want it, but not have the line break in the table of contents or running head.
- There are also "starred" forms, like `\section*`, which are unnumbered, and produce no entry in the table of contents.
- Few documents use the full range: many will use only say `\section` and `\subsection`.
- The table of contents is produced by the `\tableofcontents` command.
- The names of the `\paragraph` and `\subparagraph` commands are perhaps unfortunate. They are different from LaTeX's notion of a typeset paragraph (as indicated by blank line or `\par` command). A `\paragraph` sectional unit may actually consist of several typeset paragraphs.
- The `\appendix` command normally doesn't produce any output, but it changes the numbering of sectional units to be suitable for an appendix. For example, in the `article` class, the `\appendix` command changes the `\section` command to number sections by letters, "A", "B", "C"...
- There is a rarely used, higher-level sectioning command `\part` at level −1. It's anomalous in that a new `\part` *doesn't* reset lower-level counters (`\chapter` or `\section`, depending on the document class).
- There are two user-settable counters `secnumdepth` and `tocdepth`, which control the level at which sectional units are numbered and entered in the table of contents.

3.21 Layout of a LaTeX source file

- Layout (whitespace, indentation, line breaking) in the LaTeX source file is largely ignored.
- However, you should pay attention to layout to enhance readability of the LaTeX source, as you would say for a C program. For example, the layout of the items in an itemized list should approximate the indentation of the final typeset output.
- The comment `%` also magically eats all leading whitespace from the beginning of the next line. This can be very useful to layout and indent the LaTeX by lines without introducing extra spaces into the output.[15]
- Needless to say, a LaTeX source file should include reasonable comments, as they will help the reader.

4 Breaking Up Your Document 1

4.1 `\input`

- Keeping the whole of a large document in a single file can be unwieldy.

[15] A comment `%` can't occur inside a command name.

- `\input{`*file*`}` acts as if the contents of file *file* appeared at this point.
- Analogous to `#include` in C.
- If *file* lacks an extension, `.tex` is provided

  `\input{defns.ltx}` inputs the file `defns.ltx`

  `\input{defns}` and `\input{defns.tex}` both input the file `defns.tex`
- The file is searched for first in the current directory (folder), and then in standard LaTeX system directories.

  (This can be modified using the `TEXINPUTS` environment variable.)

```
export TEXINPUTS=.:$(HOME)/tex:
```

  Two nifty features: The trailing colon means that LaTeX searches the standard LaTeX system directories after those you've specified. If a directory ends with a double slash `//`, then it's searched recursively (that is, into subdirectories as well).
- Inputted files can also contain `\input`, and in turn input further files.
- Main uses
  - Breaking a large document into manageable pieces
  - Re-use of chunks of LaTeX across multiple documents
    - An important table or complicated diagram used across several papers
    - Especially a file containing your own command definitions and other personal customizations
  - Division of labor amongst collaborators: each can work on own part
- See also `\include` in §14

5 Roll-Your-Own

5.1 Commands

- LaTeX provides many commands built-in, but also provides a facility for users to define their own commands, which work almost exactly like those built-in.
- New commands are defined using `\newcommand`, which takes two arguments: The first is the name of the new command (with backslash); the second is the body of the new command's definition — what each instance of the new command is replaced with.[16]
- Something like cross between a `#define` and a function definition in C, because (LaTeX being in effect an interpreted language) there's no distinction between the compilation and run-time phases.
- E.g.

```
\newcommand{\PhoneNo}%
           {+61-3-9234-5678}
My phone number is \PhoneNo.
Note difference between
aa\PhoneNo aa
or
aa{\PhoneNo}aa
and
aa\PhoneNo\ aa
or
aa{\PhoneNo} aa
or
aa\PhoneNo{} aa
```

  produces

> My phone number is +61-3-9234-5678. Note difference between aa+61-3-9234-5678aa or aa+61-3-9234-5678aa and aa+61-3-9234-5678 aa or aa+61-3-9234-5678 aa or aa+61-3-9234-5678 aa

[16] Underneath, TeX uses a lower-level command-definition facility, the most important part of which is the command `\def`. It's sometimes used in package definitions, since it's more efficient than `\newcommand`, because it's at a lower level and does less checking. It's interesting to note, though, that `\def` uses a more general pattern-matching mechanism for passing arguments.

As mentioned for built-in commands, and it applies as well to user-defined commands, when the command name is terminated by a space, LaTeX eats up *all* the following whitespace, so the following "aa" in the second instance above comes out right up against the expansion of the command. If you really want the space to be there, you need to use one of the contrivances listed here (although the `xspace` package can in most cases make this unnecessary). Note that you couldn't have written `\PhoneNoaa` because LaTeX would have treated that as the name of a different, presumably undefined, command.

- Your own commands can also take (up to nine) arguments:

```
\newcommand{\Shout}[1]{\textbf{#1!}}
I shouted ``\Shout{Keep quiet}''
to the class.
They yelled back
``\Shout{Go away}''.
```

produces

I shouted "**Keep quiet!**" to the class. They yelled back "**Go away!**".

```
\newcommand{\TwoByTwo}[4]%
% Two-by-two matrix in row order
% Works only in math mode!
{\left(\begin{array}{cc}
#1 & #2 \\
#3 & #4
\end{array}\right)%
}
\newcommand{\ct}{\cos\theta}
\newcommand{\st}{\sin\theta}
\[
R_\theta =
\TwoByTwo{\ct}{\st}{-\st}{\ct}
\]
```

produces

$$R_\theta = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

- The expected number of arguments is given as an optional argument after the new command name. This is what LaTeX uses to check that sufficient arguments are provided at each call.
- Command definitions can be put almost anywhere in the document, but it's good practice to collect them all together in the preamble (makes changing easier), and even better to put them in separate files of definitions, which are inputted in the preamble.
- You can define your own commands for convenient abbreviations, e.g., after

```
\newcommand{\bi}{\begin{itemize}}
\newcommand{\ei}{\end{itemize}}
```

```
\bi
\item blah
\item blah blah
\ei
```

produces

- blah
- blah blah

- More importantly, user-defined commands provide flexibility and abstraction:

  If your phone number changes, you only need to change it in one place, in your shared file of definitions.

  This is the software engineering principle of *single point of control*.

  Similarly if you needed to change the format for shouting (say to Huge size) or the format of matrices (say to use square brackets).

  Especially important if you're dealing with large documents that may run to hundreds of pages, with LaTeX source spread over many files.

- Another use of user-defined commands is analogous to a "stub" function in C. Suppose

parts of your document need to be typeset in a particular way, but you don't yet know how to do it. You can invent a command to do that, provide a temporary, stub definition of that command, then push on with developing your document content using that command. The stub definition can be very simple, maybe something like

```
\newcommand{\complicated}[4]%
    {*** #1 * #2 * #3 * #4 ***}
```

Later on, you can replace the stub definition with the real one, and you shouldn't need to make any changes to the body of your document.

- User-defined commands can also have one optional argument (not covered here).
- `\newcommand` will give an error if the command already exists.
- To redefine an existing command, use the analogous `\renewcommand`. This is used particularly for document customization.
- Less often used, but still handy to know about is `\providecommand`, which is kind-of like the opposite of `\renewcommand`. If the command doesn't exist, it gets defined, but if the command already exists, then nothing happens. It can be used to make sure a piece of LaTeX will work, even in an environment where you don't know whether certain commands it uses have been defined or not. The `\providecommand` provides a "default" definition, but one which will be over-ridden by an already existing definition.
- Such commands (however defined) are often referred to as "macros", because they're analogous to macros in other languages and applications like spreadsheets.

5.2 Environments

- Analogously, LaTeX lets you define your own environments, for example:

```
\newenvironment{Warning}[1]%
{\color{#1}
\hrule\smallskip
\begin{center}
\large Warning
\end{center}\em}%
{\smallskip\hrule}
\begin{Warning}{red}
If you go down to the woods
today, you'd better not go
alone.
\end{Warning}
```

produces

Warning

If you go down to the woods today, you'd better not go alone.

- The environment definition has *two* brace-enclosed parts: The first part goes in front of the environment's contents; the second part goes after it.
- Note that environment arguments can be used only in the front part.
- Pretty much everything said above about user-defined commands applies to user-defined environments.
- Environments are more appropriate if you're doing things with large chunks of text, or with the few things that aren't allowed inside command arguments (paragraph breaks and verbatim text).
- There's an analogous `\renewenvironment` for redefining existing environments.

5.3 "Theorems"

- Especially in mathematics, there are identifiable, numbered chunks of material, like theorems, lemmas, corollaries, proofs...
- In other fields, there may be chunks like commentaries or recommendations, exercises, or such.

- LaTeX provides the `\newtheorem` command as a convenient way of defining such "theorem-like" environments.
- For example,

```
\newtheorem{rec}{Recommendation}
\begin{rec}
\label{rec:preamble}
Put all your command, environment,
and theorem definitions in the
preamble.
\end{rec}
\begin{rec}[For mariners]
\label{rec:chrono}
Never go to sea with
two chronometers.
\end{rec}
Recommendations~\ref{rec:preamble}
and~\ref{rec:chrono} are
completely unrelated.
```

produces

Recommendation 1 *Put all your command, environment, and theorem definitions in the preamble.*

Recommendation 2 (For mariners) *Never go to sea with two chronometers.*

Recommendations 1 and 2 are completely unrelated.

- Notice that a `\label` inside a "theorem" picks up its number for cross reference by `\ref`.
- Optional arguments to `\newtheorem` can control numbering: either within some sectional unit, or in the same sequence as another "theorem".
- Reference [4, p. 252ff] describes ways to customize the presentation of "theorems".

6 Customization

6.1 Customization

You can customize your documents in various ways:

- By using various classes and packages, and setting options to them, e.g., the `twocolumn` option supported by almost all document classes.
- By redefining various lengths and commands that LaTeX "advertises" can be modified.
- Do this with respect, though, in that the standard settings were generally designed by typographic experts: Your modifications are likely to be less good.

  For example, LaTeX's standard page layout has rather wide margins. People are often tempted to go for narrower margins, in order to fit more on a page, and save paper. Unfortunately, the resulting long lines, with many words, can be difficult to read. (This, of course, is with one-column layout in mind.)
- Wizards can delve into LaTeX internals and redefine the effect of standard LaTeX constructs. Doing this requires considerable knowledge: Something like the `\section` command does quite a lot of work behind the scenes in terms of layout and formatting, as well as house-keeping like setting table-of-contents entries and page headers, plus being adaptable to different language environments.

7 Cross References

7.1 Labels and references

- You mark a place in your document with `\label`.[17]
- You refer to that place from elsewhere with `\ref` to get the section, subsection, figure, table, item number, etc.
- With `\pageref` to get the page number

[17] While the labels can be pretty arbitrary strings, it's conventional to prefix a label with some indication of what sort of thing it is. So `sec:foo` would be a reference to a section. Similarly for `eqn:foo`, `fig:foo`, `tab:foo`, `itm:foo`, which would, respectively be references to an equation, a figure, a table, and a list item. Beware, though, that in certain packages for German, the colon is an *active* character, and using it in label names can cause problems.

- E.g. with

```
\section{Natural History of Gnus}
\label{sec:gnus}
Here we describe gnus.
```

then elsewhere

```
Information about gnus can be
found in Section~\ref{sec:gnus}
on page~\pageref{sec:gnus}.
```

produces

Information about gnus can be found in Section 3 on page 42.

- `\vref` of the `varioref` package provides more powerful cross-references: includes page reference only if it's to a different page.

  E.g.

```
\usepackage{varioref}
...
See Table~\vref{tab:data}.
```

can produce (among other variants)

See Table 4.2.

See Table 4.2 on the previous page.

See Table 4.2 on the facing page.

See Table 4.2 on page 66.

depending on relative position of the referenced table.

- The `hyperref` package can turn cross references into clickable hyperlinks in PDF (and HTML) output.

7.2 Mechanics of cross references

- `\ref` picks up cross-reference information written to the `.aux` file by `\label` on the previous run.
- So, after a big change, it may take two runs of LaTeX to get cross references exactly right.
- But normally, you don't need to bother, since you only need exactly correct cross references in the final version.
- It may seem that `\label` magically knows what kind of thing it's in (figure, section, subsection, etc.). In fact each of these constructs sets some counter to be the current "ref" (using `\refstepcounter` or equivalent), and `\label` just simple-mindedly picks up the printed representation of this "ref" counter. The cleverness is in the enclosing construct, not in the `\label` command itself.

  One conseqence of this is that if you want to put a cross reference say to a section, then the corresponding `\label` command has to be inside that section (i.e., after the `\section` command), but not inside any lower-level construct, like a subsection or figure.

8 Footnotes

8.1 Footnotes

```
We saw gnus\footnote{A gnu is
an an animal often found in
crossword puzzles.} and gnats.
```

produces

We saw gnus[1] and gnats.

[1] A gnu is an an animal often found in crossword puzzles.

There are number of customizations and packages for changing the placement, numbering, and formatting of footnotes.

9 Floats: Tables and Figures

9.1 Floats

- With normal text, LaTeX inserts page breaks as needed to produce properly filled pages.
- But some things can't reasonably be split across page breaks, like images, diagrams, tables, program listings.
- Keeping such a thing in sequence with the normal text could leave a big ugly white patch if it didn't fit on the current page.
- So typographically we use *floats*, which can "float" through the text to a suitable place where they fit.
- Standard LaTeX provides two kinds of floats, the `figure` and `table` environments.
- The two float environments work in exactly the same way. The only differences are:
  - Convention: figures are used for diagrams and images; tables are used for presentation of tabular data.
  - Figures and tables are numbered and captioned differently, and have their respective `\listoffigures` and `\listoftables`.
- E.g.

```
\begin{figure}
\begin{center}
\resizebox{0.5\linewidth}{!}
  {\includegraphics{daylesford.eps}}
\end{center}
\caption{View of Daylesford Lake.}
\label{fig:daylesford}
\end{figure}
```

  will float the figure, Figure 2, (in this case an EPS image) to a suitable position.

- `table` is completely analogous, except that its content is conventionally a `tabular` environment.

Figure 2: View of Daylesford Lake.

- Both `figure` and `table` accept an optional argument by which you can express your own preference for positioning: `\begin{figure}[htbp!]`.

  But until you know what you're doing, you're far better off accepting LaTeX's standard positioning rules.
- There are variant `figure*` and `table*` forms, which give full-width floats even in two-column layout.

  Normal forms give just single-column width floats.
- You can define new categories of floats, for things like program listings that don't really fit in either of the tables or figures categories.

10 Tabulars and Tables

10.1 Tabulars and tables

- Data laid out in a table can be very important for visual presentation.
- Since the name "table" is already used for the floating table environment, the name "tabular" is used for the environment that creates tabular data.
- A `tabular` environment creates a TeX box (just like a big letter) so it might need additional formatting to be wrapped around it.

- For example,

```
A tabular
\begin{tabular}[b]{l r}
Initials & Age \\
LJK & 52 \\
ACK & 16 \\
\end{tabular}
might be set in-line.
```

produces

	Initials	Age	
	LJK	52	
A tabular	ACK	16	might be set in-line.

- The common column specifiers are: `l r c p{}`, respectively for left-aligned, right-aligned, centered, and paragraph ("wrapped") items. The `p` specifier takes a length argument, which gives the desired width. The `@{}` specifier can be used to override the default outside and inter-column space. See the various examples to get an idea of how these work. A vertical bar `|` makes a vertical line (either between columns, or as part of a frame around the table). The command `\hline` puts a horizontal line between rows of a table (across all columns), while `\cline{`c_1`-`c_2`}` puts the horizontal line across only some columns.

- More usually, a tabular is big enough to need to be set inside a floating table environment. So the LaTeX in Figure 3 produces Table 1, floated to a suitable place.

- There is also `tabular*` environment, which takes an additional width argument and stretches the table to fill that width. (But only if the table contains stretchable inter-column space.)

- There are also additional packages, such as `longtable`, `supertabular`, `colortbl`, and `tabularx`, which provide tables with additional features, like tables which span multiple pages, have colored backgrounds, and proportionally assigned column widths.

```
\begin{table*}
\begin{center}
\begin{tabular}
{| l c p{0.25\linewidth} |}
\hline
Name & Formula & Comment \\
\hline
Sodium Chloride &
NaCl &
Common salt.
Chief solid consistuent of seawater.
\\
Magnesium Sulphate &
Mg${}_2$SO${}_4$ &
Hydrated form is Epsom Salts.
\\
Calcium Fluoride &
CaF${}_2$ &
Occurs as fluorospar.
\\
\hline
\end{tabular}
\end{center}
\caption{Selected salts.}
\label{tab:salts}
\end{table*}
```

Figure 3: LaTeX to produce Table 1.

Name	Formula	Comment
Sodium Chloride	NaCl	Common salt. Chief solid consistuent of seawater.
Magnesium Sulphate	Mg_2SO_4	Hydrated form is Epsom Salts.
Calcium Fluoride	CaF_2	Occurs as fluorospar.

Table 1: Selected salts.

- There is also a `tabbing` environment, which has some limited use.

11 Mathematics

11.1 Math mode

- Mathematics in TEX/LATEX is processed in a special mode, *math mode*.
- Some features, like superscripts, subscripts, and most special symbols are available only in math mode.
- Math comes in two flavors: *inline* and *display*
- Inline math is delimited by:
  `$...$`
  `\(...\)`
  or, more verbosely, by
  `\begin{math}...\end{math}`

  The `$...$` notation is inherited from TEX. It's convenient, but treacherous, since missing a dollar sign can cause a large-scale mess. It is commonly used, and is probably OK for simple formulas.
- Display math is delimited by:
  `\[...\]`
  `\begin{displaymath}...\end{displaymath}`
  And also by a number of other environments which give equation formatting and numbering.
- For example:

```
The inline formula
$\sum_{i=1}^n x_i^2$
can be displayed as
\[
\sum_{i=1}^n x_i^2
\]
Note the difference in
size and formatting.
```

produces

The inline formula $\sum_{i=1}^{n} x_i^2$ can be displayed as

$$\sum_{i=1}^{n} x_i^2$$

Note the difference in size and formatting.

- A lot of things work differently in math mode. Most noticeable is that, aside from their use to separate tokens, spaces are ignored in math mode. (In ordinary text, a space normally gives an inter-word space — it doesn't in math mode.) LATEX uses its own spacing rules in math mode, which incorporate a lot of knowledge about mathematical typesetting (like different spacing around the same symbol depending on whether it's being used as a unary or binary operator). So `$- x - y z$` and `$-x-yz$` both produce $-x - yz$.

  Best advice is to just write the content of the formula, and leave it to LATEX to determine the spacing. There are special spacing commands to use in math mode, but generally you should use them only on rare occasions to fine-tune the spacing in the formula, when LATEX's rules haven't got the aesthetics quite right.

→ In almost all cases, LaTeX knows more about mathematical typesetting than you do.

11.2 Subscripts and superscripts

- In math mode, subscripts are indicated by `_` and superscripts by `^`
- Grouping is important:

```
Note the difference between
$a^n+k_j-1$ and $a^{n+k}_{j-1}$.
```

produces

Note the difference between $a^n + k_j - 1$ and a_{j-1}^{n+k}.

- Note:

```
${}^4_2$He
```

produces

${}^4_2$He

where we've put the subscript and superscript on "nothing", indicated by the empty pair of grouping braces.

11.3 A menagerie of symbols and operators

- LaTeX supports a huge number of mathematical symbols and operators, far more than can be mentioned here.

```
$\alpha$ $\xi$ $\Xi$ $\theta$ $\phi$
\[
n! = \prod_{k=1}^{n} k
\]
\[
\sin^2\theta + \cos^2\theta = 1
\]
\[
\log \sqrt[n]{x} = \frac{\log x}{n}
\]
Compare $log$ with $\log$
and with $\mbox{log}$. \\
Compare $difference$
with \emph{difference}. \\
Compare $k$ with k.
```

produces

α ξ Ξ θ ϕ

$$n! = \prod_{k=1}^{n} k$$

$$\sin^2\theta + \cos^2\theta = 1$$

$$\log \sqrt[n]{x} = \frac{\log x}{n}$$

Compare log with log and with log.
Compare $difference$ with *difference*.
Compare k with k.

Notice that in math mode, a sequence of letters is treated as the mathematical product of single-letter variables. So `log` in math mode *doesn't* get you the name of the log function, (which is conventionally typeset in roman type). The best way of doing this is to use the command `\log`. Similarly for `\cos`, `\sin`, etc. The `\mbox` command sets its argument in roman type, even inside a mathematical formula, so can be used for "log-like" functions which aren't pre-defined, and also for introducing a few words of text inside a formula, like "if" and "otherwise". (The various AMS LaTeX packages provide slightly better ways of doing this, though.)

- This also means that putting a word inside dollar signs is *not* a shortcut for emphasizing it via italics. The font used and letter spacing

will be different. Conversely, if you refer to a variable from your formula in running text, you need to put it inside dollar signs, so that it will be typeset consistently as a mathematical variable, not as an ordinary letter.

11.4 "Large" delimiters

```
\begin{displaymath}
{\left( \frac{x+1}{x+2} \right)}
^ 2
\end{displaymath}
```

produces

$$\left(\frac{x+1}{x+2}\right)^2$$

Also works with other "bracketing" delimiters, like `(...)`, `[...]`, `\lbrace...\rbrace`, `|...|`, `\|...\|`, etc., plus "." (see how equation 1 in §11.6 below is done).

11.5 Arrays

The `array` environment is analogous to the `tabular` environment, but in math mode. For example,

```
\[
I =
\left(
\begin{array}{cccc}
1      & 0 & \ldots & 0 \\
0      & 1 &        & 0 \\
\vdots &   & \ddots & \vdots \\
0      & 0 & \ldots & 1
\end{array}
\right)
= \left(\delta_{ij}\right)
\]
```

produces

$$I = \left(\begin{array}{cccc} 1 & 0 & \ldots & 0 \\ 0 & 1 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{array} \right) = (\delta_{ij})$$

11.6 Equations

- The `equation` environment displays a formula and gives it an equation number, which can be labelled and referred to.

  For example:

```
\begin{equation}
|x| = \left\lbrace
\begin{array}{r l}
x & \mbox{if }x \ge 0 \\
-x & \mbox{otherwise}
\end{array}
\right.
\label{eqn:abs}
\end{equation}
```

  produces

$$|x| = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{otherwise} \end{cases} \tag{1}$$

  And elsewhere

```
Equation~\ref{eqn:abs} defines
the absolute value.
```

  produces

> Equation 1 defines the absolute value.

- The `eqnarray` environment displays numbered equations aligned in something like a three-column array:

```
\begin{eqnarray}
x & = & at^2 + bt + c \\
y & = & dt^2 \nonumber \\
  &   & \mbox{} + et + f
\end{eqnarray}
```

produces

$$\begin{aligned} x &= at^2 + bt + c && (2)\\ y &= dt^2 \\ &\quad + et + f && (3) \end{aligned}$$

- A subtle point is that the empty `\mbox` is to force LaTeX to treat the leading plus sign on the the line as a binary operator, and space it appropriately.
- There is even more support for math layout and symbols in the AMS LaTeX packages (that's the American Mathematical Society, like the ACM of mathematics).

12 Boxes, Lengths, Space, Counters

12.1 Caveat

- The concepts and commands described here can be very powerful tools for low-level formatting.
- However, you should almost never use them in the body of your document.
- If you have some meaningful piece of your document that needs formatting in a particular way, you should almost always define an appropriate command or environment for that piece.
- Use the low-level formatting commands only in the definition of that command or environment.
- Use the command or environment in the body of your document.
- This, in keeping with the general principle, insulates the body of your document from formatting changes: separating formatting from content.

12.2 Lengths

- Lengths in TeX, which LaTeX rides upon, are rather rich; however, to simplify a little:
- Lengths can be *rigid*, a definite size, for which a unit must be specified (even if the length is zero).
- Some of the units used in TeX are `pt` for printers' points[18], `mm` for millimeters, `cm` for centimeters, and `in` for inches.
- Particularly useful are the printers' measures `em` and `ex`, being respectively the nominal width of an 'M' and the nominal height of an 'x' in the current font.

  These provide lengths that scale up and down naturally with the choice of font size.
- Lengths can be negative.
- LaTeX provides numerous predefined lengths.
  - Some are "read-only"—just for information

    For example, `\linewidth`, the width of the current environment (width of text on a normal page, width of current column in multicolumn layout or `minipage`).
  - Most can be modified, for customization.

    For example

    ```
    \setlength{\parindent}{0pt}
    ```

    sets the paragraph indent to zero.

    For example

    ```
    \setlength{\parindent}{2em}
    ```

    would let the paragraph indent vary with the font.
- You can create your own new lengths, e.g.:

  ```
  \newlength{\figwidth}
  ```

[18] There is also `pp` for PostScript points, which are slightly larger.

- Lengths can be a multiple of an existing length, e.g.:

```
\setlength{\figwidth}{0.5\linewidth}
```

  would set the length `\figwidth` to half the current `\linewidth`.

- Length can be *rubber*, or stretchable.

  There's a stretchable length called `\fill`, and `\stretch`{r}, which has r times the stretchability of `\fill`.

  Stretchable lengths are really more like gas under pressure: They expand to fill available space, sometimes in competition with other stretchable lengths.

12.3 Boxes

- In TEX a *box* is a rectangle of something.
- It might be as simple as a single character, or as complicated as table or a diagram.
- The main work of TEX is to "glue" boxes together into bigger boxes (e.g. letters into words, words into lines, lines into paragraphs), and arrange them properly on the page.
- As we've seen, the `tabular` environment makes a box.
- More simply, `\mbox` makes a box out of its text argument.
- `\fbox` is like `\mbox`, but it puts a frame around its box.
- `\makebox` and `\framebox` are fancier versions, which permit specification of width and alignment.
- The `\raisebox` command can be used to raise (or lower) a box and also lie to TEX about how big it is, that is make TEX treat it as if its size were different from what it really is. This can be useful for fine-tuning layout.
- The `\parbox` command and `minipage` environment set material in a certain width and make a box out of it
- You can also save already typeset material into a named box, and re-use it. Relevant are the `\newsavebox`, `\sbox`, `\savebox`, and `\usebox` commands, and the `lrbox` environment. This has a number of uses, for example:
  - If you have say a complicated logo generated by internal LATEX commands repeated on every page, say as part of the page head, then it will save a lot of processing time if you save the logo into a named box and re-use it, rather than regenerate it every time (as would happen if you did it via a command).
  - Similarly, say if you want to put CVS keyword expansions into the foot of every page. Because of the dollar signs (and other unpredictable active characters that might appear in file names, like underscores), you have to capture the CVS keyword inside say a `\verb`, but you can't pass these around. The solution is to make a saved box of the CVS keyword using the `lrbox` environment, and then use this box wherever needed.[19]

12.4 Space and rules

- `\hspace`{w} produces a horizontal space of width w.

  This space, though, is removed if it's at the beginning or end of a line.

  The `\hspace*` form gives space that is never removed.

- `\vspace`{h} produces a vertical space of height h.

  This space, though, is removed if it's at the beginning or end of a page.

  The `\vspace*` form gives space that is never removed.

- In either case, the length can be negative (brings things closer together, even overlapping), or stretchable (useful for relative placement).

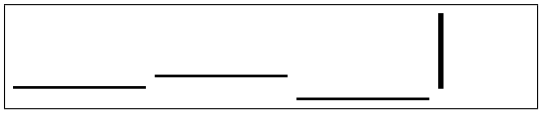
[19] There are packages to facilitate tyepsetting of CVS or subversion keywords.

- There are various predefined useful spaces of convenient sizes, horizontal: `\,` (thin space), `\quad` (1em wide), `\qquad` (2em wide), `\hfill` (stretchable); vertical: `\smallskip`, `\medskip`, `\bigskip`, `\vfill` (stretchable).
- Not quite spaces, but similarly "stretchable" and handy are `\dotfill` and `\hrulefill`.
- A *rule* is a rectangular blob of ink.

  Among other uses, it can make horizontal and vertical lines.

```
\rule{5em}{1pt}
\rule[1ex]{5em}{1pt}
\rule[-1ex]{5em}{1pt}
\rule{2pt}{1.0cm}
```

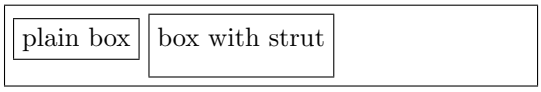
  produces



  A very useful special case is a rule with zero width, called a *strut*. It is invisible, but takes up room vertically, and can be used for positioning, especially in places where explicit vertical spaces can't be used, such as in mathematical formulas.

```
\fbox{plain box}
\fbox{\rule[-2ex]{0pt}{4ex}%
    box with strut}
```

  produces



  Similarly for zero-height rules, though they're probably less useful.

12.5 Counters

- *Counters* are like integer variables: they're what LaTeX uses behind the scenes to number sections, pages, etc.
- You can create your own counters, manipulate them, and use them to number your own entities (like say an exam question environment).
- Relevant commands are: `\newcounter`, `\setcounter`, `\addtocounter`, `\value`, `\stepcounter`, `\refstepcounter`
- Also `\the`*ctr* and `\arabic`, `\roman`, `\Roman`, `\alph`, `\Alph`, `\fnsymbol`
- For example, to have all your chapter numbers (and references) be in upper-case Roman numerals, you can say

```
\renewcommand{\thechapter}%
    {\Roman{chapter}}
```

  in the preamble. Don't get carried away with this, though.

13 Graphics: Images and Diagrams

13.1 The `picture` environment

- Standard LaTeX provides the `picture` environment.
- It's adequate for simple "box and arrow" diagrams and has the advantage that it uses only the facilities of LaTeX and doesn't rely on any external drivers.
- Some visual GUI diagram editors, like `xfig`, can export in the form of LaTeX picture environments.
- But for most uses, it's too limited and cumbersome.
- Various packages, like `eepic`, provide a `picture` environment with extended capabilities. ←

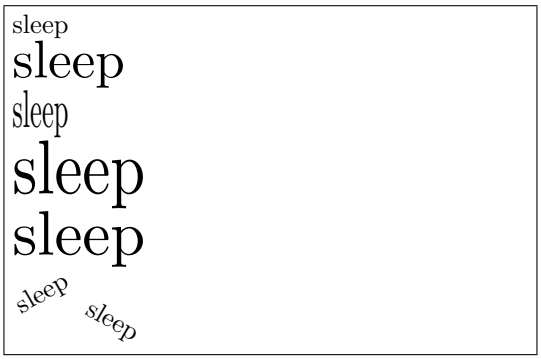
13.2 Graphics operators

- Originally there was the `graphics` package, but now the `graphicx` package is almost always used, since it provides additional features.

- `\scalebox`
- `\resizebox`
- `\rotatebox`
- For example,

```
sleep \\
\scalebox{2}{sleep} \\
\scalebox{1}[2]{sleep} \\
\resizebox{5em}{2em}{sleep} \\
\resizebox{5em}{!}{sleep} \\
\rotatebox{30}{sleep}
\rotatebox{-30}{sleep}
```

produces



- These rely on capabilities beyond DVI, by passing "specials"
- They work with `dvips` and `pdflatex`, but may not work with other drivers.
- The `pstricks` package provides very powerful facilities for creating graphics, but works only with PostScript output.

13.3 Including graphics

- Use `\includegraphics`.
- Many graphics and image formats are supported: for example, PDF, EPS (Encapsulated Postscript), PNG, JPEG, GIF.
- EPS requires using a PostScript driver, like `dvips`.
- PDF requires the use of `pdflatex`.
- See Figure 2 and the LaTeX for it on page 24.
- The graphic can be resized and rotated via the graphics commands mentioned above, or via special keyword arguments to the `\includegraphics` command provided by `graphicx`.
- If you don't specify a file extension in `\includegraphics`, then LaTeX searches through a (configurable) list of possible extensions.

  This means that under some conditions, you don't need to commit to the particular graphics format in your document: For Figure 2, I could have said just `\includegraphics{daylesford}`, and LaTeX would have picked up what was available (in this case EPS).
- Generally, in order process the graphic properly LaTeX needs access to its "bounding box" (which describes how big it is). Formats like EPS and (encapsulated) PDF store this information in the file in a form that LaTeX can access. But for other formats you may have to take additional steps to tell LaTeX about the bounding box.

14 Breaking Up Your Document 2:

14.1 `\include`

- Running `latex` on a large document can be very time-consuming, especially when you're developing a document and need to run `latex` often to debug your document.
- In rare cases, a document can be so big that it exceeds your computer's memory capacity to process it all in one go.
- You could break up your document into pieces, and `\input` these pieces from a top-level document, then comment-out `\input`s for all but the pieces you're actively working on. But this is cumbersome, and messes up cross references.

- `\include` provides this capability, but in a much more convenient way.
- `\include` acts like `\input`, but for each file it reads in, it maintains its cross-referencing information in its own corresponding individual `.aux` file.
- In the preamble, you can put an `\includeonly` command, giving it a comma-separated list of included files.

  Only those files are processed, but they use (mostly correct) cross-reference and page-number information from the `.aux` files from previous runs.
- For example, on the simple top-level document:

```
\documentclass{article}
\includeonly{experiments,results}
\begin{document}
\include{front}
\include{introduction}
\include{experiments}
\include{results}
\include{conclusions}
\end{document}
```

  `latex` would process and produce output from only the files `experiments.tex` and `results.tex`.
- One restriction is that the output of an `\included` file must always start a new page.

  Since `\included` files almost always are top-level pieces, like chapters, which start on a new page anyway, this is hardly ever a problem in practice.

  If the pieces don't naturally start on a new page, like sections of an article, then you can `\include` them during development, and turn them into `\input` for the final runs.

15 Bibliography

15.1 DIY: `thebibliography`

- LaTeX provides a `thebibliography` environment for formatting bibliographies
- You can use it directly, but it's only really viable for the simplest cases

15.2 Using BibTeX

- Much better is to have your bibliography automatically generated and formatted.
- Where you want your bibliography, you put `\bibliography{`*`list of biblio databases`*`}`.

  The argument is a comma-separated list of special databases (`.bib` files) to use for bibliographic information.

  Many bibliographic databases are already available in BibTeX format. They may be a shared resource of a research group. The online database CiteSeer provides BibTeX entries.

  Usually, though, you'll need to create at least some BibTeX entries of your own.
- You also need to specify `\bibliographystyle{`*`style`*`}` where *style* specifies the style you want for your bibliography.

  LaTeX's standard bibliography styles are `plain`, `unsrt`, `alpha`, and `abbrv`.

  Plus there are numerous contributed styles, including those required for various journals.

  Writing new bibliography styles can be done, but it's a job for wizards. More feasible is to make small modifications of existing styles.

  The LaTeX book [7] says that the `\bibliographystyle` declaration can go anywhere *after* `\begin{document}`, but I've had it work as expected when put in the preamble.

  For example,

```
\bibliographystyle{plain}
\bibliography{ljk-latex}
```

  appears towards the end of this document.
- In your document you cite a paper using the `\cite` command giving a unique identifier for each paper:

```
Stuff about bibliography can be
found in~%
\cite[\S4.3 \& Appendix~B]%
{latex-adps},
and in~\cite[Ch.\ 13]{latex-comp}.

The most important {\LaTeX} book
is~\cite{latex-adps}.  Very useful
is~\cite{latex-comp}, while
more specialized books are~%
\cite{latex-gracomp,latex-webcomp}.

A huge amount of information is
available on~\cite{texliveCD},
including~%
\cite{lshort,UKtexfaq,texlive-guide}.
```

produces

> Stuff about bibliography can be found in [7, §4.3 & Appendix B], and in [4, Ch. 13].
> The most important LaTeX book is [7]. Very useful is [4], while more specialized books are [6, 5].
> A huge amount of information is available on [10], including [8, 3, 9].

where the references are to papers in the bibliography.

- `\nocite` causes a reference to be listed in the bibliography without putting a citation in the text.

  The form `\nocite{*}` pulls in *all* the papers from the listed databases. It's useful for preparing stand-alone bibliographies, though perhaps the `biblist` package is better for this purpose.

- Extended bibliographic facilities are provided by a number of packages, including `natbib`, `cite`, `citesort`, `overcite`, `chicago`, `chapterbib`, `bibunits`. See [4].

15.3 Running `bibtex`

- Getting your bibliography and citations completely correct *may* take up to four steps:

  1. Run `latex` once on your document.

     `latex` will complain about undefined citations and references, but this is normal.

     On this pass `latex` records the identifiers of the cited papers in the `.aux` file.

  2. Run `bibtex`. It reads the `.aux` file, searches the databases for the cited papers, and writes the formatted bibliography to the corresponding `.bbl` file.

  3. Run `latex` again. It will pick up the bibliography automatically and put it at the indicated place, but it may still complain about undefined references.

  4. Run `latex` a third time, to finally resolve any references to the bibliography.

- Note that it's probably cleanest if the command argument is the base of the `.tex` file, like `latex foo` and `bibtex foo` for `foo.tex`—`latex` looks at the corresponding `.tex` file, and `bibtex` looks at the corresponding `.aux` file.

- Of course, you only need to go through this full process if you're starting from scratch, and you want a final version.

- During development you normally only need to run `bibtex` when you've added new citations (or removed old ones).

- If you just run `latex` once, at worst your citation numbers may be temporarily out of synch. This is usually acceptable during development.

16 Collaborative Document Development

16.1 Version control—CVS

- LaTeX has two important characteristics:

  1. Sources are ordinary text files

2. A large document can be broken across a number of files, using `\input`, `\include`, and `\includegraphics`.

- These make LaTeX documents particularly suited for being put under a version-control system like CVS or subversion, with all the benefits that brings, such as:
  - Team development: Shared repository with individual checked-out workspaces, possibly on remote machines.
  - Version control and branching.
  - Tracking and documentation of rationale for changes via log messages.
  - Diffs for focussing on changes.
  - ...
- Just a few things to note:
  - Because CVS diffs are line based, it's not a good idea to make unnecessary changes to the line structure of your source, since they make the real diffs hard to locate.

    For example, if you change a few words in a paragraph in the source file, it's not a good idea to re-fill that paragraph in your text editor (say by `M-Q` in `emacs`).

    Some text editors, on Windoze in particular, store text paragraphs as single long lines, and just wrap them to fit for the display. If you change a single character of that paragraph, the whole paragraph will show up as a diff line, again making it hard to see the real differences.
  - Because `$` is an active character in LaTeX, it takes some contrivance to get CVS keyword substitutions, like `$Id$` to work. Sure, you can easily put them inside a comment, but getting the CVS stuff to print in the document is trickier. It can be done—see how it's done in the source of this document to get some
→ ideas, or use one of the packages providing this facility.
  - Remember, if you're checking-in other document source files, like images, which are binary, not text, then you'll need to specify `-kb` on the `cvs add` command.

16.2 Build tools—`make`

- Because all the LaTeX-related processors, like `latex`, `bibtex`, `dvips`, etc., are shell-invokable programs, it's very convenient to use a build tool, like `make` to generate your final documents.
- With GNU `make`, you can write makefile rules like:[20]

```
%.dvi : %.tex ; latex $<
%.ps : %.dvi ; dvips -t a4 -o $@ $<
%.pdf : %.ps ; ps2pdf $< $@
```

  Or, if you prefer to go direct to PDF:

```
%.pdf : %.tex ; pdflatex $<
```

- If your top-level LaTeX document includes other files (whether LaTeX or graphics), then you can specify them as additional dependencies of the corresponding DVI file (or PDF file if you go straight to PDF).
- It is possible to generate such dependencies automatically. There are a few Perl scripts around that purport to do this. I've even written one myself. But they aren't entirely satisfactory. Problems:
  - In C, all `#includes` are visible in the source file, while in LaTeX an `\input` may result from the expansion of a user-defined command, which may involve some arbitrary computation. So you can't completely find out what's input except by almost fully simulating `latex`.
  - Having automatically generated `.tex` files (such as is done by `transfig`) can interact badly with `latex`'s search along its search path given by the environment

[20] In most cases, it's better also to provide the `-Ppdf` option to `dvips`, so `dvips` chooses font representations that convert better into PDF.

variable `TEXINPUTS`. Suppose you `\input` a file that doesn't exist yet but is supposed to be created as needed in the current directory. The path search for dependencies might find another file by the same name elsewhere on the search path, and take that as satisfying the dependency. And `make` will therefore never trigger creation of that needed `.tex` file.

One better solution to this is to grovel through the `.log` file, and parse out the messages that `latex` writes there when it reads a file. An even better solution (though still not perfect) would be to redefine LaTeX's various input commands so that they write the dependencies to a file in proper makefile format. Anyone feel like writing such a package?

A less strenuous solution would be to define your own variant versions of the commands, say `\Input`, `\Include`, and `\Includegraphics`, which call the underlying LaTeX commands, but also record what file has been read. Then always use your variant versions in your document. Still not a perfect solution...

- One aspect of LaTeX which `make` doesn't handle well is that LaTeX actually entails some circular dependencies, which fundamentally violates `make`'s mode of operation, which assumes (reasonably enough for most compilers) that the dependency graph is acyclic. Most obviously, `latex` both needs and produces the `.aux` files. Furthermore, `latex` needs the `.bbl` file produced by `bibtex`, but `bibtex` needs the `.aux` file produced by `latex`.

  The pragmatic solution is to express just the "mainstream" dependencies in the makefile, like of the DVI file on the top-level and inputted LaTeX and graphics files, and be prepared occasionally to intervene by hand: ↓ Use something like `make -W foo.tex foo.dvi` to get `make` to act is if `foo.tex` had been changed, thus forcing a re-run to resolve ↑ cross-references. Or remove the DVI file. Or, less elegant, but simpler, just type and delete a character in your source file, then save it and remake (you'll probably be editing it anyway). Likewise, have a target to force running of `bibtex` as needed.

- Another issue is that, even if `latex` encounters errors, it will still usually create a DVI file (if only a partial one), "successfully" from `make`'s point of view.

  If the target you're aiming to create is a say a PostScript file using `dvips`, then even after you've modified the LaTeX sources to fix the errors, `make` will still think that the PostScript file is up to date with respect to the current DVI file. Again, the pragmatic solution is to force rebuilding.

- As well as using `make` to control running `latex`, you can also use it to control other transformations of files that contribute to your document, for example:

  – Automated conversion of diagrams produced by `xfig` into EPS using `transfig`. Much more convenient than always needing to invoke the "export" function in the GUI.

  – Automated processing of images, say for contrast enhancement for better presentation using programs from the pbmplus/netpbm or ImageMagick packages. You may need to do different kinds of enhancement for different purposes for all your images.

16.3 Automatic generation of LaTeX

- It's reasonably straightforward to write scripts (especially Perl scripts) or programs to generate LaTeX, either complete stand-alone documents, or fragments that can be inputted into other LaTeX documents.

- This has numerous uses. For example,

  – If you have a lot of numeric data, you can write a script to wrap the appropriate LaTeX around the data to format it nicely into a table (using the `tabular` environment). Be warned, this often involves a lot of backslashes, since

in both C and Perl you need to backslash-escape a backslash in order to get it printed in the output.

– Suppose you have certain constants (numeric or string) that must appear consistently in both program source code and in documentation. ("The maximum length of a address field is 128 characters.") Rather than type them in twice, with the risk of inconsistencies, you can create a simple configuration file which defines these constants, and then write scripts to generate consistently both C header files of `#define`s, and LaTeX input files of `\newcommand` definitions. For example, from something like:

```
ADDRMAX 128
```

you'd generate both

```
#define ADDRMAX 128
```

for C, and

```
\newcommand{\ADDRMAX}{128}
```

for LaTeX.

- Generation of LaTeX from XML is mentioned elsewhere.
- There are a number of packages that support conditional text in LaTeX, conceptually similar to `#ifdef` in C, without getting into full programmability of LaTeX (for which see below). The `optional` package is probably the best of these.
- If you need to produce LaTeX documents in a number of different formats, rather than continually editing, you can put all your content into files to be inputted, and automatically generate various top-level LaTeX files, each of which say defines various different settings in the preamble, and then inputs the content in the body.
- Actually the opposite: For the *Image Understanding Environment*, it was necessary, from a common description to generate both documentation and boilerplate C++ code to support some extensions to C++. LaTeX/TeX was an obvious choice for the first task (generating documentation). But then the implementors realised they could also use LaTeX/TeX for the second task: code generation. (This uses some lower-level features by which TeX can open and read and write files.) This is the only case I know of of an (admittedly simple) compiler written in TeX!

17 Miscellaneous Topics

17.1 Index and glossary

- For a large work, like a book, a good index is essential.
- LaTeX provides support for preparing an index:
  – Put `\usepackage{makeidx}` and `\makeindex` in the preamble
  – Attach `\index` with index entry for each place you want indexed, e.g.

```
A gnat\index{gnat} with
gnarled wings gnashed ...
```

  (From [7, p. 75].)
  – Put `\printindex` where you want the index
  – You also need to run the `makeindex` program to process the raw output from the `\index` commands into a formatted index (somewhat analogous to `bibtex`).
- There are analogous facilities for preparation of a glossary
- For more information, see [7, §4.5] and [4, Ch. 12].

17.2 Verbatim text

- Sometimes you want LaTeX *not* to interpret all its special characters.
- Especially if you want to give computer program listings, or LaTeX examples.

- You can "escape" these characters, but it gets tedious.
- The `verbatim` environment typsets a chunk of raw text in typewriter font, obeying lines, and treating all characters literally.
- The `\verb` command does this for short pieces of text.
- For example,

```
Ordinary {\LaTeX}
on a new line?
\begin{verbatim}
Ordinary {\LaTeX}
on a new line?
\end{verbatim}
\verb+#$_^\{}+
```

produces

Ordinary LaTeX on a new line?

```
Ordinary {\LaTeX}
on a new line?

#$_^\{}
```

- Because these change the way LaTeX reads in text, they're restricted: can't be passed in arguments to commands.
- There is also the `alltt` environment, which allows some limited LaTeX processing.
- *Not* to be used for visual formatting!

  For that use say the `tabular` or `tabbing` environments.

→ - The `moreverb` and `fancyverb` packages provide additional verbatim support, including input from a file and line numbering.

17.3 Multilingual LaTeX

- LaTeX can support other languages in a number of ways:
  - Allowing language specific customization, e.g. `\chapter` produces "Chapitre" in French
  - Allow other characters and accents
    - Via special markup, like `\"{o}` for ö (only viable for small usage)
    - Via special transliteration schemes (as used for some Indian languages). May need auxiliary programs.
    - By accepting other character sets and encodings (e.g. Chinese, Unicode). May require a special version of the `latex` program.
- See, in particular, the `babel` package.
- This is an area in flux.

17.4 LaTeX and the Web

- There are several ways of putting LaTeX documents on the Web:
  - As PDF, either produced directly using the variant `pdflatex`, or via PostScript using `dvips`[21] and `pdf2ps` (free as part of `ghostscript`) or Acrobat Distiller.
  - By conversion to HTML using LaTeX2HTML or `tex4ht` (or to XML).
  - Via special browser plugins that support (a variant of) LaTeX, like IBM's Techexplorer.
  - All have their pros and cons...
  - All have ways of adding clickable hypertext links to your document, both internal and external (to the Web).
- It's possible to go the other way: from XML to generate LaTeX automatically for nice typesetting of information extracted say from a database.

17.5 PostScript fonts

- By default, LaTeX uses the Computer Modern family of fonts designed by Donald Knuth originally for TeX.
- These give ordinary LaTeX documents a subtlely distinctive look.

[21] As mentioned earlier, best with the `-Ppdf` option.

- LaTeX can make use of many other kinds of fonts, but setting it up is technically demanding.
- However, there are some packages which give you pretty simple access to a number of PostScript font families (provided they're available on your system).

  This document could be set in the standard PostScript Times, Helvetica, and Courier fonts by

```
\usepackage{times}
```

- Before you do this too freely, keep in mind that most of the many mathematical symbols are usually available only in one of the Computer Modern fonts. So, even if you choose a different font family for the main text of your document, any mathematics will still use Computer Modern. This can cause some subtle stylistic clashes.
- Because of the work involved in creating a font, many fonts are proprietary and must be licensed. Still, there are a large number of freely available fonts.

17.6 Page style and headings

- Standard LaTeX supports a number of page styles: `plain`, `empty`, `headings`, and `myheadings` (the last used in conjunction with `\markright` and `\markboth` commands).
- They are set globally with the `\pagestyle{`*`style`*`}` command.
- The `\thispagestyle{`*`style`*`}` sets the page style temporarily for just the current page.
- The page numbering can be set via the `\pagenumbering{`*`num_style`*`}`, where *num_style* can be `arabic`, `roman`, `Roman`, `alph`, or `Alph`. It also resets the page counter to 1.
- There is also a `titlepage` environment, which produces an "empty" style page. This is for doing title pages more complicated than can be done with `\maketitle`.
- Other packages, like `pagestyle` and `fancyheadings` provide more facilities, like three-part titles for the head and foot of each page.

17.7 Color

- `\usepackage{color}`
- Some predefined color names, like `red`, `green`, `blue`...
- You can define your own named colors, in RBG, CMYK, or grayscale.
- Use them for text and boxes.
- For example, the LaTeX input shown in Figure 4 produces the following output:

Some red text and some blue text.
Text in SpringGreen
Some text in a pale blue box.
White on blue in a gray frame
Predefined colors: black white red green blue yellow cyan magenta

- Can also set the background color of the page using `\pagecolor{`*`color`*`}`.

17.8 Hyphenation

- If typesetting could only break lines between words, then it would be very hard to get nice fully justified paragraphs.
- So sometimes a word is broken and hyphenated at a suitable point.
- LaTeX's automatic hyphenation algorithm generally works very well, but sometimes it needs help.
- Locally, you can insert the discretionary hyphen command `\-` into a word that's giving trouble. This inhibits any other hyphenation, so you'll generally need to put

```
\definecolor{paleyellow}%
   {rgb}{1.0,1.0,0.7}
\definecolor{paleblue}%
   {cmyk}{0.1,0.1,0.0,0.0}
\definecolor{midgray}{gray}{0.5}
Some \textcolor{red}{red text}
and some {\color{blue} blue text}.

\noindent
\textcolor[named]{SpringGreen}%
   {Text in SpringGreen}

\noindent
\colorbox{paleblue}%
   {Some text in a pale blue box.}

{
\setlength{\fboxrule}{3pt}
\setlength{\fboxsep}{5pt}
\noindent
\fcolorbox{midgray}{blue}%
{\color{white}
 White on blue in a gray frame}
}

{\large
\noindent
Predefined colors:
\textcolor{black}{black}
\colorbox{black}{%
\textcolor{white}{white}}
\textcolor{red}{red}
\textcolor{green}{green}
\textcolor{blue}{blue}
\textcolor{yellow}{yellow}
\textcolor{cyan}{cyan}
\textcolor{magenta}{magenta}
}
```

Figure 4: LaTeX to produce sample color output.

multiple \-s into the word, at all reasonable hyphenation points.[22]

- Globally, you can use the `\hyphenation` command to declare the allowed hyphenation points for a word whenever it's used. The argument is a space-separated list of words, each with hyphens at the allowed hyphenation points.

17.9 Fragile versus robust commands

- Things in LaTeX (including command arguments) are subject to multiple steps of processing.
- Some arguments are special—because of the way they're handled, they're subject to a double dose of processing.

  An example is the heading argument to a sectioning command. It's processed once by the sectioning command, but it's also stored and gets reprocessed over again when inserted into the table of contents.

  Such an argument is called a *moving argument*.
- Most LaTeX commands stand up fine to this double processing—they are called *robust*.
- However, some commands, because of the way they're implemented, don't work when subject to this double processing—they are called *fragile*.
- So, you can't use a fragile command directly in a moving argument.
- This means that when reading about a command you may need to pay attention to whether it's robust or fragile.
- If you do need to use fragile commands in a moving argument, you can "protect" them by putting `\protect` immediately in front of every fragile command (including those that are in arguments to other commands).

[22] Remember that while the word may now need hyphenation only at one point, future changes elsewhere, like changing the wording of the paragraph, or changing margins, may require different hyphenation.

- Most robust commands are unaffected by `\protect`, so if you're unsure about whether a command is fragile or robust, there's usually no harm in putting in the `\protect` anyway.
- But more usually, you don't bother putting in `\protect` until you need to in response to error messages from LaTeX. (Though, coming from deep inside LaTeX the resulting error messages are often obscure.)

17.10 Some additional useful *free* programs

`xdvi` X-based DVI viewer.

Ghostscript (`gs`) PostScript/PDF interpreter, converter and viewer. The programs `gv` and `GSView` are GUI wrappers around `gs`. The Ghostscript package comes with many Postscript-based conversion programs, PostScript to various image formats, and to and from PDF.

`dvips` Converts DVI to PostScript.[23]

pbmplus/netpbm A collection of image conversion and processing utilities. ImageMagick is similar, arguably better.

`xfig` **and** `dia` GUI programs for drawing 2D diagrams. Can export to various formats supported by LaTeX.

`transfig` `fig2dev` Converts `xfig`'s `.fig` files into various formats supported by LaTeX. Actually, `fig2dev` is the work-horse, callable as a command-line utility (say in makefiles) to do the conversions, while `transfig` is a utility for creating makefiles to do the conversions.

AbiWord and `gnumeric` These are respectively a word-processor (like MS Word) and a spreadsheet (like MS Excel). They can read the corresponding MS formats, and also export LaTeX. Unfortunately, by then all the semantic markup has been lost (one of the advantages of LaTeX). Still, they have some use if you've been given a document in some MS format and need to make some use of it in the LaTeX world.

There are many others...

17.11 LaTeX debugging

Making sense of logs, warnings and error messages, etc.
Sorry, this section still has to be written.
In the meantime, refer to [7, Chapter 8].

17.12 More advanced programming features

- While it's designed to look most of the time like a declarative markup language, LaTeX (because it's built on top of TeX) is in fact a fully fledged programming language.
- The `ifthen` package provides control structures like if-then-else and loops.
- The `calc` package provides facilities for making calculations with counters and lengths much easier—it lets you write expressions.
- The truly intrepid can delve into the arcane underlying programming facilities of TeX.

17.13 Tips'n'tricks

This section is far from complete.

> *The main thing is not just to learn the particular list of tricks given here, but to develop a sense of what's possible.*

1. When creating a LaTeX document for the first time, it's a good idea to run `latex` fairly frequently, so that your search for the

[23] Here is as good a place as any to point out that the LaTeX system does not keep complete information about all its fonts. This would take up too much space. In practice, the full font information is generated as needed (by MetaFont). It is cached, however, and re-used on subsequent runs. So if you're the first person in a while to use a particular font (and size) on your machine, then you'll see (and have to wait for) MetaFont in action generating the full font descriptions. It turns out (obvious if you think about it) that LaTeX doesn't need to know all the information about the font, just sufficient to be able to decide where on the page each character can fit. So, it's only when you run later programs, like `dvips` and `xdvi`, which need this full information (like the actual letter shape), that you might see MetaFont running.

inevitable errors can be focussed on the new stuff you've added since the last error-free run. For this, `\includeonly` comes in very handy, since only that part of the document you're actively working on needs to be fully processed.

How often you need to run `latex` depends on how complicated is what you're typing. If it's just simple textual material, you might go on for pages without rerunning `latex`. For complicated mathematics you might rerun `latex` after adding only a few symbols, to build up the formula bit by bit.

2. For something really complicated, you might even create a "mini-document", which contains the preamble of your actual document, and whose document body contains only the difficult piece of LaTeX you're working on. You can re-run `latex` on this mini-document *very* quickly, and speed up your development cycle. Then you can cut and paste the working stuff into your real document—or better, keep your experimental LaTeX in a separate file and use `\input` to read it into both your mini and real documents.

3. If you put a `\label` just before the `\end{document}`, then the pageref for this label will give the number of pages in the document. It can then be used, say in conjunction with the `fancyheadings` package, to give page numbers like "Page m of n". To get this to work perfectly, you have to attach the `\label` to the last non-space thing, otherwise there's a small chance that LaTeX might put a page break immediately after the non-space thing, before it fully processes the label, so the label might end up on an extraneous apparently all-blank page of its own, making the page count one higher than it should be.

   In fact, this is a special case of a more general issue. If you want to give a pageref to some construct, say a sectioning command, then you need to put the `\label` *immediately* after that construct (without any intervening space), to ensure that no page break can come between the construct and the processing of the label. The only exception to this is the captions of floating environments like figures and tables: Since no page break can come inside the float, the label can come anywhere after the caption (even after intervening whitespace).

   In a similar vein, keep in mind that the references for a figure or table are set by the `\caption`. So, if you mistakenly put the `\label` *before* the caption then it will probably pick up the reference to the surrounding section at time of processing.

4. Handy is `\ensuremath`, which sets its argument in math mode regardless of whether you're already in math mode or not. Its main purpose is in defining commands which can be used in or out of math mode. For example, if you want to use some math-mode feature like a special symbol in ordinary text:

```
\newcommand{\love}%
     {$\heartsuit$}
```

   Used in ordinary text, the first dollar sign puts you in math mode, making the `\heartsuit` symbol ♡ available, and the second dollar sign takes you to back to ordinary text mode (so called "LR", "left to right", mode). But if you happened unsuspectingly to use this command in math mode, weird things would happen: the first dollar sign would take you *out* of math mode, and you'd get an error, because the `\heartsuit` symbol is available only in math mode. Defining the command instead as

```
\newcommand{\love}%
     {\ensuremath{\heartsuit}}
```

   solves the problem: it would work as intended in either context.

5. The `tabular` environment is far more useful than you might think: don't think of it as being restricted to conventional tables.

   For example, suppose you want a particular row/column layout of some images. You

make a `figure` environment so the whole thing can float. The contents of the `figure` environment is a `tabular` environment, which gives you the row/column layout inside the top-level figure. The actual cells of the `tabular` `\includegraphics` the images. There's nothing to stop a `figure` from having multiple captions, so each sub-image can have its own separate caption, though sometimes you need to play tricks with `minipage` environments to make sure LaTeX is in the correct mode when processing the captions. For this sort of thing, the `subfigure` package may be helpful. See [4].

6. The main obvious use of the `\multicolumn` command in a `tabular` is to make a cell that spans multiple columns. But it has many other uses, for example, you can use a one-column `\multicolumn` to make a centered heading over an otherwise `p{}` column.

7. Usually the cleanest way to get a more complicated tabular arrangement is to use nested tabulars—that is, tabulars whose cells are in turn tabulars.[24] In cases like this it's generally better if the internal tabulars have `@{}` at either end of their column specification to suppress the space that's normally put around a free-standing tabular, since the inner tabular will already be getting the ordinary intercolumn space from the outer tabular.

   The output shown in Figure 5 is produced by the LaTeX shown in Figure 6 (which also demonstrates next point). Note that the layout is constructed from a three-column one-row outer tabular. Each cell is in turn constructed by a nested tabular: First a fairly simple table of numbers with a trick for aligning the decimal points, then a one-column, three-row layout of images, last a one-cell tabular containing some rotated text. (You can see that for a one-row tabular, like the outer tabular here, the choice of

[24] You can play dirty tricks in tabulars with things like struts and negative vertical space to control positioning—like to make something contained in one cell actually print in the cell above it. But this is very messy and fragile, and not recommended unless you have no other choice.

```
\begin{figure*}
\newcommand{\Img}[1]%
  {\resizebox{#1}{!}%
    {\includegraphics%
      {daylesford.eps}}}
\begin{center}
\begin{tabular}{rlr}
\begin{tabular}{|l r@{$\cdot$}l|}
\hline
\multicolumn{3}{|c|}%
    {\bf Approximations} \\
$\pi$ & 3&14159254\ldots \\
$e$  & 2&71828\ldots \\
$e^\pi$ & 23&14069263\ldots \\
\hline
\end{tabular}
&
\begin{tabular}{@{}c@{}}
\Img{8em} \\ \Img{4em} \\ \Img{2em}
\end{tabular}
&
\begin{tabular}{@{}c@{}}
\rotatebox{90}{This extraneous
text is provided by a rotated box.}
\end{tabular}
\end{tabular}
\end{center}
\caption{Using nested
    \texttt{tabular}s for layout.}
\label{fig:tabular}
\end{figure*}
```

Figure 6: LaTeX to produce Figure 5.

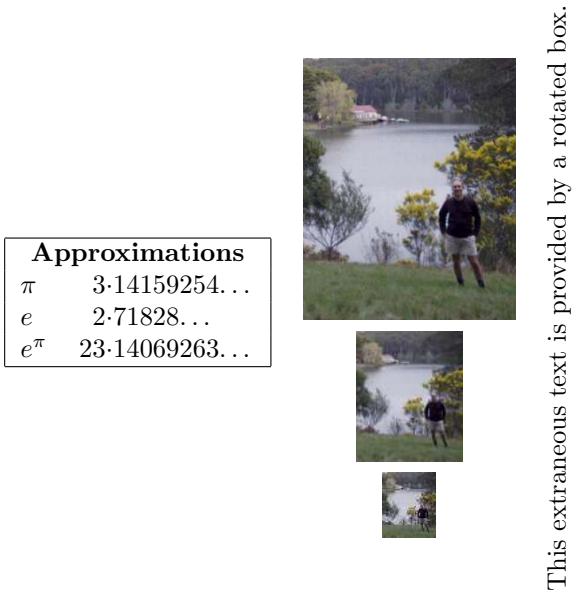


Figure 5: Using nested `tabular`s for layout.

column alignments, like `l`, `c`, `r`, doesn't really matter, since each cell will completely fill its column width.)

This strategy is summarized in Les's LaTeX Proverb:

> Usually, the solution to problems with tabulars is: *more* tabulars.

8. One way of getting a column of numbers aligned on their decimal points is to use a column specifier like `r@{$\cdot$}l`: the integer parts end up right justified in one column, the decimal fractions left-justified in the next, and the usual inter-column space is replaced by the decimal point. See Figure 5. There are, of course, other ways...

9. Knuth decreed that the origin for a TeX page be 1 inch down and 1 inch in from the top left corner of the page. (This was to fit with the common American practice of 1 inch margins all around.) This offset can cause some confusion. Setting the length `\topmargin` to 0pt gives you a 1 inch top margin. To get a smaller top margin, you have to make `\topmargin` a suitable negative length. Similarly for `\oddsidemargin` and `\evensidemargin`. There are some packages to simplify the setting of margins and other page-layout parameters.

10. TeX measures the page from the top left (with the 1 inch offsets). PostScript's coordinate system is from the bottom left corner. So to convert properly from DVI to PostScript, a program like `dvips` needs to know the actual height of the page. (Think about it in terms of the coordinate transformation.) In many (not all) installations, the default setup for `dvips` is for U.S. Letter size paper. If you print the resulting PostScript output on A4 paper, it will be wrongly offset on the page. The way to fix this is to set the paper size explicitly using the `-t a4` option to `dvips`, or in your own `.dvipsrc` configuration file. (Read the `dvips` man pages.)

11. Postscript printing in landscape format will require a `-t landscape` special option to `dvips`.

12. The packages `picinpar`, `floatfig`, and `wrapfig` provide different ways of getting

figures which the text "wraps" around. Again, see [4].

13. The `url` package provides some commands to facilitate typesetting of URLs, similar long file pathnames, and email addresses. See [3]. More simply, the TEX command `\slash` produces a slash after which TEX can put a line break if it needs to. Useful for constructs like "and/or". The `\linebreak[0]` construct can also come in handy in similar situations, to give LATEX a hint as to where it might break something that's too long to fit on a line.

14. Standard LATEX provides for only one-column or two-column printing. The `multicol` package permits more flexible and more general multi-column printing (though this should not be taken to extremes). See [4].

15. Occasionally, the associated files, like `.aux` can get messed up, creating mysterious errors on the next run. One way in which this can happen is when you're switching between `latex` and `pdflatex`. Even though they're both LATEX, the two versions write slightly different information to the `.aux` files, which can confuse the other. The solution to this is just to remove the `.aux` files and start afresh.

16. Anything after `\end{document}` in the source file is ignored. It can provide a useful place for parking stuff that you don't want to be processed, but want to keep around for possible future use.

17. That environment arguments can be used only in the first "front" part of an environment is a consequence of how environments are implemented. It ends up being more a minor annoyance than a real defect. If you need variable content in the second "tail" part, you can always store it somehow in the first part (perhaps in a named box), then retrieve in the second part.

18. Plug for `xdvi` and `gv`: Both `xdvi` and `gv` are set up to work well in development. When their window is exposed/deiconified, they check whether the file they're viewing has been updated, and if so, reload it, as far as possible at the same position in the file. This means that when you're developing a document, you surely *don't* start up a new `xdvi` or `gv` for each edit cycle (which would typically load afresh at page one); instead, you keep the same `xdvi` or `gv` process running, and let it reload and redisplay after each `latex` run the part of the document you're currently working on. This isn't WYSIWYG, but can give you reasonably tight feedback.

    Note that `xdvi` can display embedded PostScript (EPS) figures in a DVI file. It does this by calling `gs` as what in current parlance would be called a "plug-in".

17.14 What's missing?

- Probably quite a lot...
- ... but less than there used to be.

17.15 Learning and getting more

- The file `sample2e.tex` in the standard LATEX distribution illustrates many features of LATEX.
- If you don't want to spend money on books, the "Not So Short Introduction to LATEX 2_ε" [8] and the "TEX FAQ" [3] on the TEXLive CD [10] are very good starting points. This document (or future versions) should be available via `http://www.cs.mu.oz.au/~ljk/latex.html`.
- There is also another LATEX introduction accessible via the MU-CSA website. (I don't have the URL handy.)
- The LATEX "Bible" is Leslie Lamport's *LATEX: A Document Preparation System*[7].

    It covers "standard" LATEX and is probably an indispensable reference for regular LATEX users.
- The TEXlive CD [10] and CTAN [1] (Comprehensive TEX Archive Network) are sources for a vast quantity of LATEX distributions, packages, and associated software and documentation.

- *The LaTeX Companion* [4] is a *very* helpful reference for those who want to do more sophisticated things with LaTeX.

  It describes many of the additional packages that extend the powers of LaTeX, and shows how to modify LaTeX internals, for example how to change the formatting of section headings.

- Reference [6] treats more advanced graphics, including such things as typesetting musical scores and chemical structural formulas.

- Reference [5] shows how to put LaTeX documents on the Web.

17.16 TeX's limitations

- TeX was originally developed nearly 30 years ago. That it's held up so well is a tribute to Donald Knuth's conception.

- However, TeX and thus LaTeX do have some intrinsic limitations.

- For example, because of the way they're internally encoded, TeX can use at most 256 different counter variables, and at most 256 different dimension variables. Very few documents, even the size of books, bump into these limits. But very complicated documents that use many packages, [4] is an example, can run into trouble, and require special techniques.

- Similarly, TeX was set up to use 8-bit characters (leading edge at the time, when many computers still used 6-bit characters). Handling bigger character sets, like Unicode, requires special versions of TeX.

17.17 TeXmacs and Lilypond

- TeXmacs provides similar functionality to LaTeX through a scriptable WYSIWYG interface modelled on `emacs`.

- TeXmacs is different from LaTeX: it has its own representation that overcomes the historical limitations inherited from TeX.

- However, it corresponds closely with LaTeX in that any TeXmacs document can be exported as LaTeX, and any reasonable LaTeX document can be imported into TeXmacs with only little work (mainly to provide TeXmacs definitions of non-standard commands).

- TeXmacs also uses TeX's font machinery to produce high-quality typesetting, particularly of mathematics.

- TeXmacs can also be run in batch mode, and has an open external plain-text representation,[25] and so has much the same advantages as LaTeX in this regard.

- A nifty feature of TeXmacs is the ability to run an interactive terminal session inside a document. This is particularly useful for interacting with a symbolic algebra system, like `maxima`, with the results of the session incorporated into the document.

- The GNU musical typesetting engine, Lilypond, is a front-end for LaTeX.

- It's worth mentioning that both TeXmacs and Lilypond use Scheme (Guile) as their internal scripting language.

References

[1] Comprehensive TeX archive network. Central website is `http://www.ctan.org`; Australian mirror is `http://mirror.aarnet.edu.au/pub/CTAN/`.

[2] Salvador Dali. *The Unspeakable Confessions of Salvador Dali, as told to André Parinaud.* W.H. Allen, London, 1976. Translated from the French by Harold J. Salemson.

[3] *The UK TeX FAQ: Your 244 Questions Answered*, 2002. In file `FAQ/english/newfaq.pdf` on [10].

[25] TeXmacs actually has *three* different text representations: TeXmacs's own representation, which is like XML but much more compact; full XML; and as Scheme list data, which is very convenient for processing TeXmacs documents in Scheme.

[4] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Boston, 1993.

[5] Michel Goossens and Sebastian Rahtz. *The LaTeX Web Companion: Integrating TeX, HTML and XML*. Addison-Wesley, Reading, Massachusetts, 1999.

[6] Michel Goossens, Sebastian Rahtz, and Frank Mittelbach. *The LaTeX Graphics Companion: Illustrating Documents with TeX and PostScript*. Addison-Wesley, Boston, 1997.

[7] Leslie Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, 2nd edition, 1994. The LaTeX "Bible".

[8] Tobias Oetiker, Hubert Partl, Irene Hyna, and Elisabeth Schegl. *The Not So Short Introduction to LaTeX 2ε: Or LaTeX 2ε in 95 Minutes*, 2001. Version 3.20, in file `texmf/doc/guides/lshort-english/lshort.pdf` on [10].

[9] Sebastian Rahtz. *The TeX Live Guide*, 7th edition, 2002. In directory `texmf/doc/tldoc/english/File` on [10] in PDF and HTML.

[10] TeXLive CD. Available from various sources.

A Behind The Scenes: How It's Done

A.1 Listing of `ljk-latex.bib`

```
@string{aw = "Addison-Wesley"}

@book{latex-adps,
   title = {\LaTeX: A Document Preparation
      System},
   author = {Leslie Lamport},
   publisher = aw,
   address = {Reading, Massachusetts},
   year = 1994,
   edition = {2nd},
   note = {The {\LaTeX} ``Bible''},
   isbn = {0-201-52983-1}
}

@book{latex-comp,
   title = {The {\LaTeX} Companion},
   author = {Michel Goossens
      and Frank Mittelbach
      and Alexander Samarin},
   publisher = aw,
   address = "Boston",
   year = 1993,
   isbn = {0-201-54199-8}
}

@book{latex-gracomp,
   title = {The {\LaTeX} Graphics Companion:
      Illustrating Documents with
      {\TeX} and PostScript},
   author = {Michel Goossens
      and Sebastian Rahtz
      and Frank Mittelbach},
   publisher = aw,
   address = "Boston",
   year = 1997,
   isbn = {0-201-85469-4}
}

@book{latex-webcomp,
   title = {The {\LaTeX} Web Companion:
      Integrating {\TeX}, HTML and XML},
   author = {Michel Goossens
      and Sebastian Rahtz},
   publisher = aw,
   address = {Reading, Massachusetts},
   year = 1999,
   isbn = {0-201-43311-7}
}

@manual{UKtexfaq,
   title = {The UK {\TeX} FAQ:
      Your 244~Questions Answered},
   editor = {Robin Fairbairns},
   key = {Fairbairns, Robin},
   year = 2002, mon = may,
   note = {In file
      \path{FAQ/english/newfaq.pdf}
      on~\cite{texliveCD}}
}

@manual{lshort,
   title = {The Not So Short Introduction
      to {\LaTeXe}:
      Or {\LaTeXe} in 95 Minutes},
   author = {Tobias Oetiker and Hubert Partl
      and Irene Hyna and Elisabeth Schegl},
   year = 2001, mon = aug,
   note = {Version~3.20, in file
\path{texmf/doc/guides/lshort-english/lshort.pdf}
      on~\cite{texliveCD}}
}

@manual{texlive-guide,
   title = {The {\TeX} Live Guide},
   edition = {7th},
   author = {Sebastian Rahtz},
   year = 2002, mon = may,
   note = {In directory
      \path{texmf/doc/tldoc/english/File}
```

```
      on~\cite{texliveCD} in PDF and HTML}
}

@misc{texliveCD,
   title = {{\TeX}{L}ive {CD}},
   key = {TeX Live CD},
   note = {Available from various sources.}
}

@book{dali,
   title = {The Unspeakable Confessions of
      {Salvador} {Dali}, as told to
      {Andr\'{e}} {Parinaud}},
   author = {Salvador Dali},
   publisher = {W.H. Allen},
   address = {London},
   year = 1976,
   note = {Translated from the French
      by Harold J.~Salemson.},
   isbn = {0-491-01955-6}
}

@misc{CTAN,
   title = {Comprehensive {\TeX}
      Archive Network},
   key = {CTAN},
   note = {Central website is
      \url{http://www.ctan.org};
      Australian mirror is
      \url{http://mirror.aarnet.edu.au/pub/CTAN/}}
}
```