# Chapter 6

# Customising LaTeX

Documents produced with the commands you have learned up to this point will look acceptable to a large audience. While they are not fancy-looking, they obey all the established rules of good typesetting, which will make them easy to read and pleasant to look at.

However, there are situations where LaTeX does not provide a command or environment that matches your needs, or the output produced by some existing command may not meet your requirements.

In this chapter, I will try to give some hints on how to teach LaTeX new tricks and how to make it produce output that looks different from what is provided by default.

## 6.1 New Commands, Environments and Packages

You may have noticed that all the commands I introduce in this book are typeset in a box, and that they show up in the index at the end of the book. Instead of directly using the necessary LaTeX commands to achieve this, I have created a package in which I defined new commands and environments for this purpose. Now I can simply write:

```
\begin{lscommand}
\ci{dum}
\end{lscommand}
```


In this example, I am using both a new environment called `lscommand`, which is responsible for drawing the box around the command, and a new command named `\ci`, which typesets the command name and makes a corresponding entry in the index. Check this out by looking up the

`\dum` command in the index at the back of this book, where you'll find an entry for `\dum`, pointing to every page where I mentioned the `\dum` command.

If I ever decide that I do not like having the commands typeset in a box any more, I can simply change the definition of the `lscommand` environment to create a new look. This is much easier than going through the whole document to hunt down all the places where I have used some generic LaTeX commands to draw a box around some word.

### 6.1.1 New Commands

To add your own commands, use the

`\newcommand{`*name*`}[`*num*`]{`*definition*`}`

command. Basically, the command requires two arguments: the *name* of the command you want to create, and the *definition* of the command. The *num* argument in square brackets is optional and specifies the number of arguments the new command takes (up to 9 are possible). If missing it defaults to 0, i.e. no argument allowed.

The following two examples should help you to get the idea. The first example defines a new command called `\tnss`. This is short for "The Not So Short Introduction to LaTeX $2_\varepsilon$." Such a command could come in handy if you had to write the title of this book over and over again.

```
\newcommand{\tnss}{The not
    so Short Introduction to
    \LaTeXe}
This is ``\tnss'' \ldots{}
``\tnss''
```

This is "The not so Short Introduction to LaTeX $2_\varepsilon$" ... "The not so Short Introduction to LaTeX $2_\varepsilon$"

The next example illustrates how to define a new command that takes two arguments. The `#1` tag gets replaced by the first argument you specify, `#2` with the second argument, and so on.

```
\newcommand{\txsit}[2]
 {This is the \emph{#1}
  #2 Introduction to \LaTeXe}
% in the document body:
\begin{itemize}
\item \txsit{not so}{short}
\item \txsit{very}{long}
\end{itemize}
```

- This is the *not so* short Introduction to LaTeX $2_\varepsilon$
- This is the *very* long Introduction to LaTeX $2_\varepsilon$

LaTeX will not allow you to create a new command that would overwrite an existing one. But there is a special command in case you explicitly want this: `\renewcommand`. It uses the same syntax as the `\newcommand` command.

In certain cases you might also want to use the \providecommand command. It works like \newcommand, but if the command is already defined, LaTeX 2ε will silently ignore it.

There are some points to note about whitespace following LaTeX commands. See page 5 for more information.

### 6.1.2 New Environments

Just as with the \newcommand command, there is a command to create your own environments. The \newenvironment command uses the following syntax:

\newenvironment{*name*}[*num*]{*before*}{*after*}

Again \newenvironment can have an optional argument. The material specified in the *before* argument is processed before the text in the environment gets processed. The material in the *after* argument gets processed when the \end{*name*} command is encountered.

The example below illustrates the usage of the \newenvironment command.

```
\newenvironment{king}
 {\rule{1ex}{1ex}%
      \hspace{\stretch{1}}}
 {\hspace{\stretch{1}}%
      \rule{1ex}{1ex}}

\begin{king}
My humble subjects \ldots
\end{king}
```

■ My humble subjects ... ■

The *num* argument is used the same way as in the \newcommand command. LaTeX makes sure that you do not define an environment that already exists. If you ever want to change an existing command, use the \renewenvironment command. It uses the same syntax as the \newenvironment command.

The commands used in this example will be explained later. For the \rule command see page 119, for \stretch go to page 113, and more information on \hspace can be found on page 113.

### 6.1.3 Extra Space

When creating a new environment you may easily get bitten by extra spaces creeping in, which can potentially have fatal effects, for example when you want to create a title environment which supresses its own indentation as well as the one on the following paragraph. The \ignorespaces command in the

begin block of the environment will make it ignore any space after executing the begin block. The end block is a bit more tricky as special processing occurs at the end of an environment. With the `\ignorespacesafterend` LaTeX will issue an `\ignorespaces` after the special 'end' processing has occurred.

```
\newenvironment{simple}%
 {\noindent}%
 {\par\noindent}

\begin{simple}
See the space\\to the left.
\end{simple}
Same\\here.
```

See the space
to the left.

Same
here.

```
\newenvironment{correct}%
 {\noindent\ignorespaces}%
 {\par\noindent%
   \ignorespacesafterend}

\begin{correct}
No space\\to the left.
\end{correct}
Same\\here.
```

No space
to the left.

Same
here.

### 6.1.4 Command-line LaTeX

If you work on a Unix-like OS, you might be using Makefiles to build your LaTeX projects. In that connection it might be interesting to produce different versions of the same document by calling LaTeX with command-line parameters. If you add the following structure to your document:

```
\usepackage{ifthen}
\ifthenelse{\equal{\blackandwhite}{true}}{
  % "black and white" mode; do something..
}{
  % "color" mode; do something different..
}
```

Now call LaTeX like this:

```
latex '\newcommand{\blackandwhite}{true}\input{test.tex}'
```

First the command `\blackandwhite` gets defined and then the actual file is read with input. By setting `\blackandwhite` to false the color version of the document would be produced.

### 6.1.5 Your Own Package

If you define a lot of new environments and commands, the preamble of your document will get quite long. In this situation, it is a good idea to create a LaTeX package containing all your command and environment definitions. Use the \usepackage command to make the package available in your document.

```
% Demo Package by Tobias Oetiker
\ProvidesPackage{demopack}
\newcommand{\tnss}{The not so Short Introduction
                   to \LaTeXe}
\newcommand{\txsit}[1]{The \emph{#1} Short
                       Introduction to \LaTeXe}
\newenvironment{king}{\begin{quote}}{\end{quote}}
```

Figure 6.1: Example Package.

Writing a package basically consists of copying the contents of your document preamble into a separate file with a name ending in .sty. There is one special command,

\ProvidesPackage{*package name*}

for use at the very beginning of your package file. \ProvidesPackage tells LaTeX the name of the package and will allow it to issue a sensible error message when you try to include a package twice. Figure 6.1 shows a small example package that contains the commands defined in the examples above.

## 6.2 Fonts and Sizes

### 6.2.1 Font Changing Commands

LaTeX chooses the appropriate font and font size based on the logical structure of the document (sections, footnotes, ...). In some cases, one might like to change fonts and sizes by hand. To do this, use the commands listed in Tables 6.1 and 6.2. The actual size of each font is a design issue and depends on the document class and its options. Table 6.3 shows the absolute point size for these commands as implemented in the standard document classes.

```
{\small The small and
\textbf{bold} Romans ruled}
{\Large all of great big
\textit{Italy}.}
```

The small and **bold** Romans ruled all of great big *Italy*.

One important feature of LaTeX $2_\varepsilon$ is that the font attributes are independent. This means that issuing size or even font changing commands, and still keep bold or slant attributes set earlier.

In *math mode* use the font changing *commands* to temporarily exit *math mode* and enter some normal text. If you want to switch to another font for math typesetting you need another special set of commands; refer to Table 6.4.

In connection with the font size commands, curly braces play a significant role. They are used to build *groups*. Groups limit the scope of most LaTeX commands.

```
He likes {\LARGE large and
{\small small} letters}.
```

He likes large and small letters.

The font size commands also change the line spacing, but only if the paragraph ends within the scope of the font size command. The closing curly brace } should therefore not come too early. Note the position of the

Table 6.1: Fonts.

| | | | |
|---|---|---|---|
| `\textrm{...}` | roman | `\textsf{...}` | sans serif |
| `\texttt{...}` | typewriter | | |
| `\textmd{...}` | medium | `\textbf{...}` | **bold face** |
| `\textup{...}` | upright | `\textit{...}` | *italic* |
| `\textsl{...}` | *slanted* | `\textsc{...}` | Small Caps |
| `\emph{...}` | *emphasized* | `\textnormal{...}` | document font |

Table 6.2: Font Sizes.

| | | | |
|---|---|---|---|
| `\tiny` | tiny font | `\Large` | larger font |
| `\scriptsize` | very small font | `\LARGE` | very large font |
| `\footnotesize` | quite small font | `\huge` | huge |
| `\small` | small font | `\Huge` | largest |
| `\normalsize` | normal font | | |
| `\large` | large font | | |

Table 6.3: Absolute Point Sizes in Standard Classes.

| size | 10pt (default) | 11pt option | 12pt option |
|---|---|---|---|
| `\tiny` | 5pt | 6pt | 6pt |
| `\scriptsize` | 7pt | 8pt | 8pt |
| `\footnotesize` | 8pt | 9pt | 10pt |
| `\small` | 9pt | 10pt | 11pt |
| `\normalsize` | 10pt | 11pt | 12pt |
| `\large` | 12pt | 12pt | 14pt |
| `\Large` | 14pt | 14pt | 17pt |
| `\LARGE` | 17pt | 17pt | 20pt |
| `\huge` | 20pt | 20pt | 25pt |
| `\Huge` | 25pt | 25pt | 25pt |

Table 6.4: Math Fonts.

| | |
|---|---|
| `\mathrm{...}` | Roman Font |
| `\mathbf{...}` | **Boldface Font** |
| `\mathsf{...}` | Sans Serif Font |
| `\mathtt{...}` | `Typewriter Font` |
| `\mathit{...}` | *Italic Font* |
| `\mathcal{...}` | $\mathcal{CALLIGRAPHIC\ FONT}$ |
| `\mathnormal{...}` | *Normal Font* |

\par command in the next two examples. [1]

```
{\Large Don't read this!
 It is not true.
 You can believe me!\par}
```

{Large Don't read this! It is not true. You can believe me!\par}

```
{\Large This is not true either.
But remember I am a liar.}\par
```

{Large This is not true either. But remember I am a liar.}\par

If you want to activate a size changing command for a whole paragraph of text or even more, you might want to use the environment syntax for font changing commands.

```
\begin{Large}
This is not true.
But then again, what is these
days \ldots
\end{Large}
```

\begin{Large}
This is not true.
But then again, what is these
days \ldots
\end{Large}

This will save you from counting lots of curly braces.

### 6.2.2 Danger, Will Robinson, Danger

As noted at the beginning of this chapter, it is dangerous to clutter your document with explicit commands like this, because they work in opposition to the basic idea of LaTeX, which is to separate the logical and visual markup of your document. This means that if you use the same font changing command in several places in order to typeset a special kind of information, you should use \newcommand to define a "logical wrapper command" for the font changing command.

```
\newcommand{\oops}[1]{%
 \textbf{#1}}
Do not \oops{enter} this room,
it's occupied by \oops{machines}
of unknown origin and purpose.
```

\newcommand{\oops}[1]{%
\textbf{#1}}
Do not \oops{enter} this room,
it's occupied by \oops{machines}
of unknown origin and purpose.

This approach has the advantage that you can decide at some later stage that you want to use a visual representation of danger other than \textbf, without having to wade through your document, identifying all the occurrences of \textbf and then figuring out for each one whether it was used for pointing out danger or for some other reason.

[1] \par is equivalent to a blank line

Please note the difference between telling LaTeX to *emphasize* something and telling it to use a different *font*. The `\emph` command is context aware, while the font commands are absolute.

```
\textit{You can also
  \emph{emphasize} text if
  it is set in italics,}
\textsf{in a
  \emph{sans-serif} font,}
\texttt{or in
  \emph{typewriter} style.}
```

*You can also* emphasize *text if it is set in italics,* in a *sans-serif* font, or in *typewriter* style.

### 6.2.3 Advice

To conclude this journey into the land of fonts and font sizes, here is a little word of advice:

> **Remember!** *The* MORE fonts YOU use **in** a `document`, *the* more READABLE and *beautiful it becomes*.

## 6.3 Spacing

### 6.3.1 Line Spacing

If you want to use larger inter-line spacing in a document, change its value by putting the

`\linespread{`*factor*`}`

command into the preamble of your document. Use `\linespread{1.3}` for "one and a half" line spacing, and `\linespread{1.6}` for "double" line spacing. Normally the lines are not spread, so the default line spread factor is 1.

Note that the effect of the `\linespread` command is rather drastic and not appropriate for published work. So if you have a good reason for chang-

ing the line spacing you might want to use the command:

```
\setlength{\baselineskip}{1.5\baselineskip}
```

```
{\setlength{\baselineskip}%
           {1.5\baselineskip}
This paragraph is typeset with
the baseline skip set to 1.5 of
what it was before. Note the par
command at the end of the
paragraph.\par}

This paragraph has a clear
purpose, it shows that after the
curly brace has been closed,
everything is back to normal.
```

This paragraph is typeset with the baseline skip set to 1.5 of what it was before. Note the par command at the end of the paragraph.

This paragraph has a clear purpose, it shows that after the curly brace has been closed, everything is back to normal.

### 6.3.2 Paragraph Formatting

In LaTeX, there are two parameters influencing paragraph layout. By placing a definition like

```
\setlength{\parindent}{0pt}
\setlength{\parskip}{1ex plus 0.5ex minus 0.2ex}
```

in the preamble of the input file, you can change the layout of paragraphs. These two commands increase the space between two paragraphs while setting the paragraph indent to zero.

The `plus` and `minus` parts of the length above tell TeX that it can compress and expand the inter-paragraph skip by the amount specified, if this is necessary to properly fit the paragraphs onto the page.

In continental Europe, paragraphs are often separated by some space and not indented. But beware, this also has its effect on the table of contents. Its lines get spaced more loosely now as well. To avoid this, you might want to move the two commands from the preamble into your document to some place below the command `\tableofcontents` or to not use them at all, because you'll find that most professional books use indenting and not spacing to separate paragraphs.

If you want to indent a paragraph that is not indented, use

```
\indent
```

at the beginning of the paragraph.[2] Obviously, this will only have an effect

[2] To indent the first paragraph after each section head, use the indentfirst package in the 'tools' bundle.

when \parindent is not set to zero.

To create a non-indented paragraph, use

\noindent

as the first command of the paragraph. This might come in handy when you start a document with body text and not with a sectioning command.

### 6.3.3 Horizontal Space

LaTeX determines the spaces between words and sentences automatically. To add horizontal space, use:

\hspace{*length*}

If such a space should be kept even if it falls at the end or the start of a line, use \hspace* instead of \hspace. The *length* in the simplest case is just a number plus a unit. The most important units are listed in Table 6.5.

```
This\hspace{1.5cm}is a space
of 1.5 cm.
```

This is a space of 1.5 cm.

The command

\stretch{*n*}

generates a special rubber space. It stretches until all the remaining space on a line is filled up. If multiple \hspace{\stretch{*n*}} commands are issued on the same line, they occupy all available space in proportion of their respective stretch factors.

```
x\hspace{\stretch{1}}
x\hspace{\stretch{3}}x
```

x x x

When using horizontal space together with text, it may make sense to make the space adjust its size relative to the size of the current font. This can be done by using the text-relative units em and ex:

```
{\Large{}big\hspace{1em}y}\\
{\tiny{}tin\hspace{1em}y}
```

big y
tin y

Table 6.5: TeX Units.

| | | |
|---|---|---|
| `mm` | millimetre ≈ 1/25 inch | ␣ |
| `cm` | centimetre = 10 mm | ⌊——⌋ |
| `in` | inch = 25.4 mm | ⌊————⌋ |
| `pt` | point ≈ 1/72 inch ≈ $\frac{1}{3}$ mm | ‖ |
| `em` | approx width of an 'M' in the current font | ⌊⌋ |
| `ex` | approx height of an 'x' in the current font | ␣ |

### 6.3.4 Vertical Space

The space between paragraphs, sections, subsections, ... is determined automatically by LaTeX. If necessary, additional vertical space *between two paragraphs* can be added with the command:

`\vspace{`*length*`}`

This command should normally be used between two empty lines. If the space should be preserved at the top or at the bottom of a page, use the starred version of the command, `\vspace*`, instead of `\vspace`.

The `\stretch` command, in connection with `\pagebreak`, can be used to typeset text on the last line of a page, or to centre text vertically on a page.

```
Some text \ldots

\vspace{\stretch{1}}
This goes onto the last line of the page.\pagebreak
```

Additional space between two lines of *the same* paragraph or within a table is specified with the

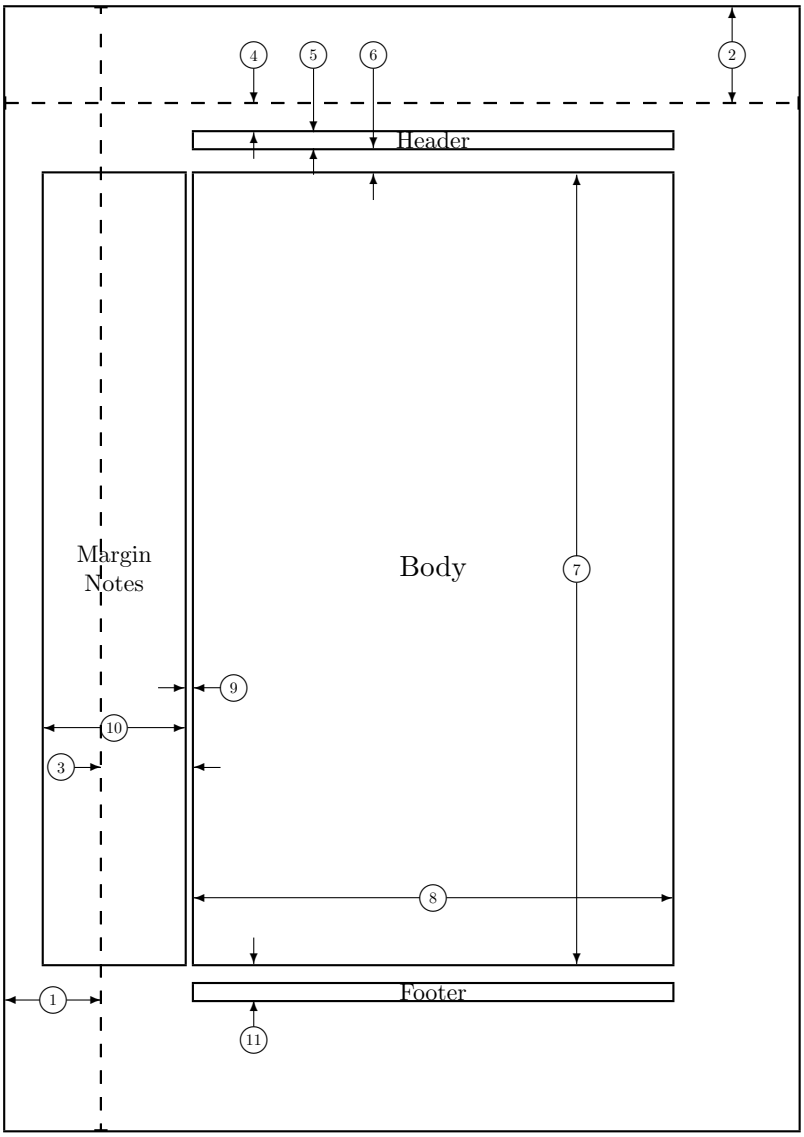
`\\[`*length*`]`

command.

With `\bigskip` and `\smallskip` you can skip a predefined amount of vertical space without having to worry about exact numbers.

## 6.4 Page Layout

LaTeX 2$_\varepsilon$ allows you to specify the paper size in the `\documentclass` command. It then automatically picks the right text margins, but sometimes




| | | | |
|---|---|---|---|
| 1 | one inch + \hoffset | 2 | one inch + \voffset |
| 3 | \oddsidemargin = 22pt<br>or \evensidemargin | 4 | \topmargin = 22pt |
| 5 | \headheight = 12pt | 6 | \headsep = 19pt |
| 7 | \textheight = 595pt | 8 | \textwidth = 360pt |
| 9 | \marginparsep = 7pt | 10 | \marginparwidth = 106pt |
| 11 | \footskip = 27pt | | \marginparpush = 5pt (not shown) |
| | \hoffset = 0pt | | \voffset = 0pt |
| | \paperwidth = 597pt | | \paperheight = 845pt |

Figure 6.2: Layout parameters for this book. Try the layouts package to print the layout of your own document.

you may not be happy with the predefined values. Naturally, you can change them. Figure 6.2 shows all the parameters that can be changed. The figure was produced with the layout package from the tools bundle.[3]

**WAIT!** ...before you launch into a "Let's make that narrow page a bit wider" frenzy, take a few seconds to think. As with most things in LaTeX, there is a good reason for the page layout to be as it is.

Sure, compared to your off-the-shelf MS Word page, it looks awfully narrow. But take a look at your favourite book[4] and count the number of characters on a standard text line. You will find that there are no more than about 66 characters on each line. Now do the same on your LaTeX page. You will find that there are also about 66 characters per line. Experience shows that the reading gets difficult as soon as there are more characters on a single line. This is because it is difficult for the eyes to move from the end of one line to the start of the next one. This is also why newspapers are typeset in multiple columns.

So if you increase the width of your body text, keep in mind that you are making life difficult for the readers of your paper. But enough of the cautioning, I promised to tell you how you do it ...

LaTeX provides two commands to change these parameters. They are usually used in the document preamble.

The first command assigns a fixed value to any of the parameters:

`\setlength{`*parameter*`}{`*length*`}`

The second command adds a length to any of the parameters:

`\addtolength{`*parameter*`}{`*length*`}`

This second command is actually more useful than the `\setlength` command, because it works relative to the existing settings. To add one centimetre to the overall text width, I put the following commands into the document preamble:

```
\addtolength{\hoffset}{-0.5cm}
\addtolength{\textwidth}{1cm}
```

In this context, you might want to look at the calc package. It allows you to use arithmetic operations in the argument of `\setlength` and other places where numeric values are entered into function arguments.

[3] `pkg/tools`

[4] I mean a real printed book produced by a reputable publisher.

## 6.5 More Fun With Lengths

Whenever possible, I avoid using absolute lengths in LaTeX documents. I rather try to base things on the width or height of other page elements. For the width of a figure this could be `\textwidth` in order to make it fill the page.

The following 3 commands allow you to determine the width, height and depth of a text string.

```
\settoheight{variable}{text}
\settodepth{variable}{text}
\settowidth{variable}{text}
```

The example below shows a possible application of these commands.

```
\flushleft
\newenvironment{vardesc}[1]{%
  \settowidth{\parindent}{#1:\ }
  \makebox[0pt][r]{#1:\ }}{}

\begin{displaymath}
a^2+b^2=c^2
\end{displaymath}

\begin{vardesc}{Where}$a$,
$b$ -- are adjacent to the right
angle of a right-angled triangle.

$c$ -- is the hypotenuse of
the triangle and feels lonely.

$d$ -- finally does not show up
here at all. Isn't that puzzling?
\end{vardesc}
```

$$a^2 + b^2 = c^2$$

Where: $a$, $b$ – are adjacent to the right angle of a right-angled triangle.

$c$ – is the hypotenuse of the triangle and feels lonely.

$d$ – finally does not show up here at all. Isn't that puzzling?

## 6.6 Boxes

LaTeX builds up its pages by pushing around boxes. At first, each letter is a little box, which is then glued to other letters to form words. These are again glued to other words, but with special glue, which is elastic so that a series of words can be squeezed or stretched as to exactly fill a line on the page.

I admit, this is a very simplistic version of what really happens, but the point is that TeX operates on glue and boxes. Letters are not the only things that can be boxes. You can put virtually everything into a box, including

other boxes. Each box will then be handled by LaTeX as if it were a single letter.

In earlier chapters you encountered some boxes, although I did not tell you. The `tabular` environment and the `\includegraphics`, for example, both produce a box. This means that you can easily arrange two tables or images side by side. You just have to make sure that their combined width is not larger than the textwidth.

You can also pack a paragraph of your choice into a box with either the

`\parbox[`*pos*`]{`*width*`}{`*text*`}`

command or the

`\begin{minipage}[`*pos*`]{`*width*`}` text `\end{minipage}`

environment. The `pos` parameter can take one of the letters `c, t` or `b` to control the vertical alignment of the box, relative to the baseline of the surrounding text. `width` takes a length argument specifying the width of the box. The main difference between a `minipage` and a `\parbox` is that you cannot use all commands and environments inside a `parbox`, while almost anything is possible in a `minipage`.

While `\parbox` packs up a whole paragraph doing line breaking and everything, there is also a class of boxing commands that operates only on horizontally aligned material. We already know one of them; it's called `\mbox`. It simply packs up a series of boxes into another one, and can be used to prevent LaTeX from breaking two words. As boxes can be put inside boxes, these horizontal box packers give you ultimate flexibility.

`\makebox[`*width*`][`*pos*`]{`*text*`}`

`width` defines the width of the resulting box as seen from the outside.[5] Besides the length expressions, you can also use `\width`, `\height`, `\depth`, and `\totalheight` in the width parameter. They are set from values obtained by measuring the typeset *text*. The *pos* parameter takes a one letter value: **c**enter, flush**l**eft, flush**r**ight, or **s**pread the text to fill the box.

The command `\framebox` works exactly the same as `\makebox`, but it draws a box around the text.

The following example shows you some things you could do with the `\makebox` and `\framebox` commands.

[5] This means it can be smaller than the material inside the box. You can even set the width to 0pt so that the text inside the box will be typeset without influencing the surrounding boxes.

```
\makebox[\textwidth]{%
    c e n t r a l}\par
\makebox[\textwidth][s]{%
    s p r e a d}\par
\framebox[1.1\width]{Guess I'm
    framed now!} \par
\framebox[0.8\width][r]{Bummer,
    I am too wide} \par
\framebox[1cm][l]{never
    mind, so am I}
Can you read this?
```

c e n t r a l

s p r e a d

Guess I'm framed now!

Bummer, I am too wide

never mind, so am I Can you read this?

Now that we control the horizontal, the obvious next step is to go for the vertical.[6] No problem for LaTeX. The

`\raisebox{`*lift*`}[`*extend-above-baseline*`][`*extend-below-baseline*`]{`*text*`}`

command lets you define the vertical properties of a box. You can use `\width`, `\height`, `\depth`, and `\totalheight` in the first three parameters, in order to act upon the size of the box inside the *text* argument.

```
\raisebox{0pt}[0pt][0pt]{\Large%
\textbf{Aaaa\raisebox{-0.3ex}{a}%
\raisebox{-0.7ex}{aa}%
\raisebox{-1.2ex}{r}%
\raisebox{-2.2ex}{g}%
\raisebox{-4.5ex}{h}}}
she shouted, but not even the next
one in line noticed that something
terrible had happened to her.
```

**Aaaaaaaargh** she shouted, but not even the next one in line noticed that something terrible had happened to her.

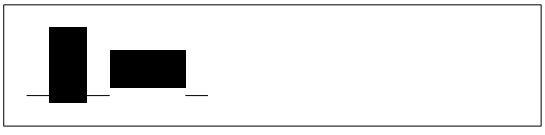
## 6.7 Rules

A few pages back you may have noticed the command

`\rule[`*lift*`]{`*width*`}{`*height*`}`

In normal use it produces a simple black box.

```
\rule{3mm}{.1pt}%
\rule[-1mm]{5mm}{1cm}%
\rule{3mm}{.1pt}%
\rule[1mm]{1cm}{5mm}%
\rule{3mm}{.1pt}
```



[6] Total control is only to be obtained by controlling both the horizontal and the vertical ...

This is useful for drawing vertical and horizontal lines. The line on the title page, for example, has been created with a `\rule` command.

The End.