The TEXpert's Guide to Survival

Martin Helsø

16th February 2018

When any writing project grows large, the document becomes increasingly difficult to maintain. As a TEXpert, you have an arsenal of tools to help you keep organised. This is an exposition of LATEX features that make the writing process easier.

1 File Management

It is easier to keep organised if you break the document up into smaller, manageable files.

1.1 Create Your Own Package or Class

You should move most of your preamble to a separate file. This will make your main file less cluttered and make it easier to reuse the preamble in other projects. The separate file can be a `.sty` file (package) or a `.cls` file (class). Keep this separate file as organised as possible; it will save you a lot of headache in the long run.

In the `.sty` or `.cls` file, you need to write `\ProvidesPackage{name}` or `\ProvidesClass{name}`, respectively, to specify the name. The name must be the same as the name of the file. You should also end the file with `\endinput`. In the `.tex` file, you import the package or class with `\usepackage{name}` or `\documentclass{name}`, respectively. If you create a `.cls` file, then you probably want to use `\LoadClass{class}` to inherit properties from an existing class. There are more advanced things you can do with packages and classes, but this is all you need to move the preamble from the main file.

There is one further advantage to creating your own package or class. In order to protect certain commands, it is not possible to use "`@`" in macro names in a `.tex` file. If you need to use "`@`", then you must declare "`@`" as a letter with `\makeatletter` first and call `\makeatother` when you are done. In a `.sty` or `.cls` file, you are free to use "`@`" without any restrictions.

1.2 Include and Input

The \include command is used to include text that logically starts on a new page, such as a chapter. Say you have a file "filename.tex" with no preamble and no \begin{document}. Then \include{filename} is equivalent to copying the contents of "filename.tex" directly into the main file where \include is written, after a \clearpage. Note that \include{filename.tex} does not work; you cannot use the file extension.

The \include command should be used in conjunction with \includeonly. In the preamble, you enter a list of files in \includeonly. As the name suggests, only these files will be included when you compile. If you have compiled all the files previously, then \includeonly will ensure that you can still use cross references to files that are not currently included. It will also keep track of the page numbers, and section and environment counters. The moral is that you should never comment out \include, but rather files from the list in \includeonly.

The \input command is similar to \include, but works on a lower level. It does not automatically start a new page. Unlike \include, it can be nested; you can use \input or \include on a file which contains another \input command. You would normally use \input for smaller sections in a large chapter, or for instance to import a long table or Ti*k*Z code.

1.3 Standalone

In the process of creating an illustration, for instance with Ti*k*Z, you typically need to compile the drawing several times. To save time, you do not want to recompile the rest of the document each time. It is also easier to reuse the figure if it is contained in a separate file. In situations like this, it is helpful to use standalone, which is both a class and package.

The document class standalone is used for a single figure. It automatically crops the page to the content. If you have the standalone package in your main file, then you can include standalone files with \input. The preamble in the standalone file will then be ignored, so the preamble in the main file must contain the necessary packages to draw the figure. Also, the packages used in the standalone file must be imported *after* the standalone package in the main file. Therefore, standalone should be the first package you import.

If you do not wish to recompile the figure every time you compile the main document, then you can include the compiled PDF with \includegraphics instead of the .tex file with \input.

1.4 Graphicspath

The graphicx bundle provides the command \graphicspath. The syntax is as follows:

```
\graphicspath{{subdir1/}{subdir2/}{subdir3/}...{subdirn/}}
```

You enter a list of subdirectories where LaTeX should look for images to be included in the document. Instead of typing

```
\includegraphics{subdir1/figure}
```

You need only type

```
\includegraphics{figure}
```

There is one drawback: The search through the list of folders can be slow if there are many directories. This is a non-issue if you only keep one folder for images.

2 Draft and Final Options

Reflecting the various stages of writing, LaTeX is equipped with the document class options `draft` and `final`. When `draft` is enabled, overfull hboxes are marked with a black box in the right margin.

On that note, every document should import the package `microtype`. It makes subtle changes to the spaces between words. You will not notice the changes without looking closely, but this greatly reduces the occurrences of overfull hboxes.

In themselves, the options `draft` and `final` do nothing more, but they are passed on to the imported packages. Here are some packages affected by this:

changes turns off mark-up of changes in `final` mode,

graphicx replaces images with a frame indicating where the pictures should be in `draft` mode,

hyperref disables linking features in `draft` mode,

listings does not include external files in `draft` mode,

microtype is disabled in `draft` mode,

showkeys is disabled in `final` mode,

todonotes is disabled in `final` mode if the package option `obeyFinal` is used.

You may manually overrule the global class options for each package. If you are using `draft` to detect overfull hboxes, then you want

```
\usepackage[final]{microtype}
```

to prevent changes in line-breaks. Moreover, if you are using `draft` to find overfull hboxes in a code listing, then you need

```
\usepackage[final]{listings}
```

3 Todonotes

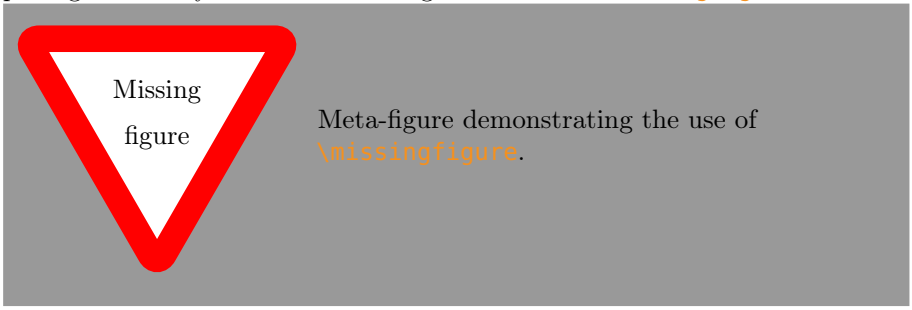
The package todonotes allows you to write marginal notes with the \todo{} command. The notes are placed directly in the text if you instead use \todo[inline]{}:

Make this clearer.

An inline note.

You can remove the line pointing to the text with \todo[noline]{}. The package enables you to indicate images-to-be with \missingfigure{}:

Rewrite!



Meta-figure demonstrating the use of \missingfigure.

You can also get a list of all that needs doing with \listoftodos:

Todo list

4 Cross References

You are probably not always writing in a linear fashion, so section, theorem and equation numbers will change in the process. To keep of track of these numbers in references, you should use automatic cross reference tools.

The holy trinity of cross reference packages are varioref, hyperref and cleveref. They should be imported in precisely that order and they should be the last packages you import. The hyperref package creates clickable links, cleveref defines the command \cref{}, which can tell what type of item it is referring to, and varioref defines \vref{}, which can tell where the item is. If the packages are imported in the correct order, then varioref learns from cleveref.

In order to reference something, you must give it a key with `\label{key}`. To create a cross reference, write `\cmd{key}`, where `\cmd` is your favourite cross reference command:

`\ref` is the most basic reference command and only prints the number,

`\eqref` is for equations only and adds parentheses around the number,

`\pageref` prints the page number,

`\cref` prints the type of object as well as number,

`\vref` prints the type of object, its location and number.

Both `\cref` and `\vref` add parentheses if they are referring to an equation.

You also have the command `\hyperref[key]{description}`, which is a bit different. You choose a description, which becomes clickable and points to the item belonging to the key.

The LaTeX way of dealing with tables and figures is letting them float. You will save yourself a lot of hassle by allowing this. Note that if you use `\vref`, then it will always be easy for the reader to find the floating object.

4.1 Label Keys

It can be difficult to come up with good keys for labels. Remember that keys are case-sensitive, but can be as long as you want and they may contain spaces. Therefore,

```
\label{A short story about a man called Jesus}
```

is a perfectly acceptable key. However, it is a good idea to use a prefix to indicate what kind of item the label belongs to, such as "thm" for theorems and "ex" for examples. If you have a section, theorem, example and equation that are all related to the same subject, then you may recycle the key for all of them and only change the prefix.

Since spaces are allowed in the keys, you cannot use a space to separate the keys when cross referencing multiple items at the same time. Hence `\cref{key1,key2}` is correct, but `\cref{key1, key2}` is not.

When it comes to tables and figures, note that it is the caption that is numbered, not the table or the figure. Therefore, `\label{key}` must be called *after* `\caption{}`.

5 Showkeys

sec:showkeys

The package `showkeys` displays labels and cite keys in the margin. This saves you from searching through the `.tex` files for the right key. The package will also print the key used by the various cross reference commands and `\cite`.

This can be turned off with the package options notref and notcite. All the functionality of showkeys is disabled when the class option final is used.

6 Comment

The package comment provides the environment comment, which is used to comment out multiple lines of code at once. Moreover, adding either the line \includecomment{comment} or \excludecomment{comment} to the preamble will turn the comments on or off. In fact, you can use those commands to define any comment-like environment. For instance, \includecomment{notes} defines the environment notes, which is printed as normal. Changing the line to \excludecomment{notes} removes the notes from the PDF. The appearance of notes can be further customised using the \specialcomments macro.

7 Changes

The changes package allows you to mark added and deleted text, which is handy if you are working on a document with other people. The mark-up is disabled when the class option final is in use.

Since changes requires you to manually mark the changes, it may be preferable to use an external version control program.

8 User-defined Commands

Creating your own macros can save you time typing, help you avoid misprints and make the code easier to read. Moreover, if you have not decided on the spelling of a word, you may postpone the decision by making a command with one spelling. If you change your mind, then you simply edit the macro definition.

There are several ways of specifying user-defined commands:

\newcommand only works when the macro you are trying to create is previously undefined.

\renewcommand is used to overwrite an existing command. You will get a compilation error if you attempt to overwrite an undefined macro.

\def defines the macro whether or not it already exists. This is potentially dangerous, and there is normally no reason to use it. However, \def is useful in package writing, where you do not have the luxury of checking if the command is defined.

\newenvironment is similar to \newcommand, but it defines an *environment*, which is initialised by \begin{environment} ... \end{environment}.

You should always check what a macro does before you overwrite it with \renewcommand. In some cases it is enough to simply call the macro to figure out what it does. However, you can also use \meaning for the precise definition. For instance, consider the \coloneqq command from the mathtools package. Typing

```
\meaning\coloneqq
```

produces the following output:

macro:->\vcentcolon \mathrel {\mkern -1.2mu}=

We see that \coloneqq consists of a vertically centred colon and an equal sign that are moved slightly closer together. For correct spacing, the combination of these two symbols is then classified as a relation symbol.

The syntax for \newcommand is as follows:

```
\newcommand{\<macro name>}[<number of arguments>]{<definition>}
```

If the macro takes arguments, then the n-th argument is accessed by $\#n$. Here is an example that takes one argument and prints it in bold red.

```
\newcommand{\red}[1]{\textbf{\textcolor{red}{#1}}}
```

If the macro is shorthand for some mathematical symbol, then you may consider adding \ensuremath to the definition. That way you may use the macro in normal text as well. Here is an example:

```
\newcommand{\R}{\ensuremath{\mathbb{R}}}
```

Now \R produces $\mathbb{R}$.

By default, a macro with no arguments will remove any space following it. This is for good reason; you may want to append to the output. Assume that you have defined the following:

```
\newcommand{\angstrom}{{\aa}ngstr\"om}
```

Suppose that you need to use the plural form "ångströms" at some point in the writing. You cannot type "\angstroms", since you have not defined a command \angstroms. However, "\angstrom s" produces the correct result. To use the \angstrom command for the singular form, you can write \angstrom{} or \angstrom\ to stop the removal of space.

The package xspace provides the \xspace command, which is a clever space. It is used for macros that you will never append to. If \xspace is followed by a letter, then it will insert a space, but it will not do anything if it is followed by a punctuation mark.

What about removing space in front of the macro? The following defines a command that removes space in order to insert a comma after the last word preceding "i.e.":

```
\newcommand{\ie}{\leavevmode\unskip, i.e.,\xspace}
```

8.1 Special Syntax

While you can use \newcommand for any macro, there are tools designed to make it easier to define certain types of commands. Here are some of them:

\DeclareMathOperator ensures that mathematical operators are written in an upright font. Unlike a text font, the operator is upright when placed in an italics environment, such as a theorem. It also adds a small space after the operator; this makes the difference between $\sin x$ and $\sin x$. To use \DeclareMathOperator, you need the package amsopn, which is automatically loaded by the package mathtools.

\DeclarePairedDelimiter from mathtools defines flexible commands for delimiters. Say you have defined

```
\DeclarePairedDelimiter{\abs}{\lvert}{\rvert}
```

Then \abs{x} is equivalent to

```
\lvert x \rvert
```

While \abs*{x} corresponds to

```
\left\lvert x \right\rvert
```

You can also add specific size commands, such as \abs[\big]{x}. This is equal to

```
\big\lvert x \big\rvert
```

\declaretheorem is a special version of \newenvironment. It is introduced by thmtools, which extends amsthm. Both packages must be imported. The command provides an easy syntax for theorem-like environments. In addition to taking care of the formatting, \declaretheorem keeps track of numbering and makes it possible to make cross-references to the theorems. It is accompanied by the command \listoftheorems, which makes a list of theorems akin to the table of contents.

9 Memoir

While not an organising tool, the document class memoir deserves mention for taking care of so much that it will undoubtedly make your life easier. Its purpose is book design and it is therefore very flexible.

The idea behind memoir is to provide a unified solution to layout, rather than having to use a different package for each task. The class imports — or has code equivalent to — the following packages:

abstract	appendix	array	booktabs
ccaption	chngcntr	chngpage	dcolumn
delarray	enumerate	epigraph	fontenc
framed	ifmtarg	ifpdf	index
lmodern	makeidx	moreverb	needspace
newfile	nextpage	parskip	patchcmd
setspace	shortvrb	showidx	tabularx
titleref	titling	tocbibind	tocloft
verbatim	verse		

The packages `lmodern` and `fontenc` are only imported if the class option `extrafontsizes` is in use.

By default, `memoir` will look like the document class `book`, but with minor changes, such as blank pages being actually blank. It can easily emulate the other common document classes. For example, this guide is typeset using:

```
\documentclass[article, oneside]{memoir}
```

However, the argument for using `memoir` is not that it can look like existing classes, but that it is highly customisable. It comes with an array of different pre-defined styles that you may choose from. Click here to see the built-in chapter styles. The class is also equipped with many easy-to-use tools for defining your own layout and design. The class provides too much to cover here, but we demonstrate some of the possibilities.

A common mistake is thinking that the width of margins matters in itself. For legibility, the average number of characters per line is what matters. The margins in LaTeX are optimal for the default fontsize. However, many people make the margins smaller without increasing the fontsize to compensate. This issue is taken care of by `memoir`. If you change the fontsize, then `memoir` will calculate new, optimal margins for you.

Say that you are writing a doctoral dissertation that is to be printed on A4 stock paper and cut to 240 mm × 170 mm pages. It is of course possible to create an ordinary A4 document and let the press shrink the contents before printing. However, it is cumbersome to imagine how legible the text and figures will be when shrunk. More worryingly, the press decides where to cut, possibly not giving the margins the proportions you wanted.

You have complete control over the layout with `memoir`. Assuming that the class option `showtrims` is enabled, then the following code will display trim marks for 240 mm × 170 mm pages:

```
\stockaiv
\settrimmedsize{240mm}{170mm}{*}
\settrims{28.5mm}{20mm}
\setlrmarginsandblock{20mm}{30mm}{*}
\setulmarginsandblock{25mm}{25mm}{*}
\checkandfixthelayout
```

The first line defines the stock paper to be A4. The third line of code is not strictly necessary; it centres the trimmed page on the stock paper. The next two lines define the inner and outer margins to be 20 mm and 30 mm, respectively, and the upper and lower margins to 25 mm.

10 Non-TEXnical Tips

We finish with some tools that are not specific to LaTeX.

10.1 Editor

It is much easier to stay organised if you have a decent editor. It can take care of automatic indentation — making the code more structured — let you comment out blocks of text with a single macro, collapse sections that you are not working on, auto-complete commands and labels, highlight missing "&" in tables and much more.

There is a myriad of available editors and you should get one that you are comfortable with. Here is a discussion about editors with support for LaTeX.

10.2 Version Control

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. Not only is this useful as a backup for yourself, but it is also very effective for keeping track of the different contributions when you are collaborating with another author.

As mentioned, the package `changes` allows you to display changes in the PDF file. However, an external tools such as `git` is easier to use, since it automatically keeps tracks of the changes and lets you revert to an earlier version.