

# LATEX ENVIRONMENTS



## TO IMAGE FORMAT

V2.0 — 2021-01-20\*

©2013–2021 by Pablo González L<sup>†</sup>

CTAN: <https://www.ctan.org/pkg/ltximg>

 <https://github.com/pablgonz/ltximg>

### Abstract

`ltximg` is a `perl` script that automates the process of extracting and converting environments provided by `TikZ`, `PStricks` and other packages from *(input file)* to image formats and standalone files using `ghostscript` and `poppler-utils`. Generates a file with only extracted environments and another with all extracted environments converted to `\includegraphics`.

## Contents

1	License	1	6	Image Formats	7
2	Motivation and Acknowledgments	1	7	How to use	8
3	Requirements for operation	2	7.1	Syntax	8
4	How it works	2	7.2	Command line interface	8
4.1	The input file	2	7.3	Options from input file	11
4.2	Verbatim contents	3	8	The way of <code>arara</code>	11
4.3	Steps process	4	9	Example usign <code>latexmk</code>	13
5	Extract content	6	10	Note for <code>dvisvgm</code> users	13
5.1	Default environments	6	11	Final notes	14
5.2	Extract with <code>docstrip</code> tags	7	12	Change history	14
5.3	Prevent extraction and remove	7	13	References	15
			14	Index of Documentation	16

## 1 License

This program is free software; you can redistribute it and/or modify it under the terms of the [GNU General Public License](#) as published by the [Free Software Foundation](#); either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the [GNU General Public License](#) for more details.

## 2 Motivation and Acknowledgments

The original idea was to extend the functionality of the `pst2pdf`[9] script to work with `tikzpicture` and other environments. The `TikZ`[2] package allows to *externalize* the environments, but, the idea was to be able to extend this to *any type* of environment covering three central points:

1. Generate a separate image files for environments.
2. Generate a standalone files with only the extracted environments.
3. Generate a file replacing the environments by `\includegraphics`.

From the side of  $\TeX$  there are some packages that cover several of these points such as the `preview`[1], `xcomment`[12], `extract`[13] and `cacheptic`[14] packages among others, but none covered all points.

In the network there are some solutions in `bash` that were able to extract and convert environments, but in general they presented problems when the document contained “*verbatim style*” code or were only available for [Linux](#).

\*This file describes a documentation for version 2.0, last revised 2021-01-20.

<sup>†</sup>E-mail: «[pablgonz@yahoo.com](mailto:pablgonz@yahoo.com)»

Analysed the situation the best thing was to create a new “*script*” that was able to cover the three points and was multi platform, the union of all these ideas is born **ltximg**.

This script would not be possible without the great work of Herbert Voß author of `pst2pdf`<sup>1</sup> and Heiko Oberdiek author of `pdfcrop`<sup>2</sup>. Several parts of the code have been taken and adapted from both scripts.

### 3 Requirements for operation

For the complete operation of **ltximg** you need to have a modern T<sub>E</sub>X distribution such as T<sub>E</sub>X Live or MiK<sub>T</sub>E<sub>X</sub>, have a version equal to or greater than 5.28 of `perl`, a version equal to or greater than 9.52 of `ghostscript`, a version equal to or greater than 1.40 of `pdfcrop` and have a version equal to or greater than 0.52 of `poppler-utils`.

The script uses only packages from the core of the `perl`, the distribution encapsulated in T<sub>E</sub>X Live 2020 for **Windows** does not have the `Win32::Console::ANSI` package, this does not affect the operation of the script, but it does affect the presentation of the messages when invoked from `cmd`, it is recommended to use a more modern (and comfortable) application such as **Windows Terminal**. MiK<sub>T</sub>E<sub>X</sub> users must install the appropriate software for full operation.

The script auto detects the `ghostscript`, but not `poppler-utils`. You should keep this in mind if you are using the script directly and not the version provided in your T<sub>E</sub>X distribution.

The script has been tested on **Windows** (v10), **cygwin** (v3.1.6) and **Linux** (fedora 33) using `ghostscript` v9.53.3, `poppler-utils` v0.90, `perl` v5.32 and the standard classes offers by L<sup>A</sup>T<sub>E</sub>X: `book`, `report`, `article` and `letter`. The `preview`[1] and `pst-pdf`[5] packages are required to process the *input file* and if an *output file* is generated, the `graphicx`[10] and `grfext`[11] packages will be needed.

### 4 How it works

It is important to have a general idea of how the “*extraction and conversion*” process works and the requirements that must be fulfilled so that everything works correctly, for this we must be clear about some concepts related to how to work with the *input file*, the *verbatim content* and the *steps process*.

#### 4.1 The input file

The *input file* must comply with *certain characteristics* in order to be processed, the content at the beginning and at the end of the *input file* is treated in a special way, before `\documentclass` and after `\end{document}` can go any type of content, internally the script will “*split*” the *input file* at this points.

If the *input file* contains files using `\input{file}` or `\include{file}` these will not be processed, from the side of the *script* they only represent lines within the file, if you want them to be processed it is better to use the `latexexpand`<sup>3</sup> first and then process the file.

Like `\input{file}` or `\include{file}`, blank lines, vertical spaces and tab characters are treated literally, for the *script* the *input file* is just a set of characters, as if it was a simple text file. It is advisable to format the source code *input file* using utilities such as `chktex`<sup>4</sup> and `latexindent`<sup>5</sup>, especially if you want to extract the source code of the environments.

Both `\thispagestyle{style}` and `\pagestyle{style}` are treated in a special way by the script, if they do not appear in the preamble then `\pagestyle{empty}` will be added and if they are present and `{style}` is different from `{empty}` this will be replaced by `{empty}`.

This is necessary for the image creation process, it does not affect the *output file*, but it does affect the *standalone* files. For the script the process of dividing the *input file* into four parts and then processing them:

```

1 % Part One: Everything before \documentclass
2 \documentclass{article}
3 % Part two: Everything between \documentclass and \begin{document}
4 \begin{document}
5 % Part three: : Everything between \begin{document} and \end{document}
6 \end{document}
7 % Part Four: Everything after \end{document}

```

If for some reason you have an environment `filecontents` before `\documentclass` or in the preamble of the *input file* that contains a *sub-document* or *environment* you want to extract, the script will ignore them. Similarly, the content after `\end{document}` is ignored in the extraction process.

<sup>1</sup><https://ctan.org/pkg/pst2pdf>

<sup>2</sup><https://ctan.org/pkg/pdfcrop>

<sup>3</sup><https://www.ctan.org/pkg/latexexpand>

<sup>4</sup><https://www.ctan.org/pkg/chktex>

<sup>5</sup><https://www.ctan.org/pkg/latexindent>

## 4.2 Verbatim contents

One of the greatest capabilities of this script is to “*skip*” the complications that *⟨verbatim content⟩* produces with the extraction of environments using tools outside the “ $\text{\TeX}$  world”<sup>6</sup>. In order to “*skip*” the complications, the *⟨verbatim content⟩* is classified into three types:

- Verbatim in line.
- Verbatim standard.
- Verbatim write.

### Verbatim in line

The small pieces of code written using a “*verbatim macro*” are considered *⟨verbatim in line⟩*, such as `\verb|⟨code⟩|` or `\verb*|⟨code⟩|` or `\macro{⟨code⟩}` or `\macro[⟨opts⟩]{⟨code⟩}`.

Most “*verbatim macro*” provide by packages `minted`[18], `fancyvrb`[16] and `listings`[17] have been tested and are fully supported. They are automatically detected the *verbatim macro* (including *\** argument) generates by `\newmint` and `\newmintinline` and the following list:

- |                        |                           |                            |
|------------------------|---------------------------|----------------------------|
| • <code>\mint</code>   | • <code>\verb</code>      | • <code>\pygment</code>    |
| • <code>\spverb</code> | • <code>\Verb</code>      | • <code>\Scontents</code>  |
| • <code>\qverb</code>  | • <code>\lstinline</code> | • <code>\tcboxverb</code>  |
| • <code>\fverb</code>  | • <code>\pyginline</code> | • <code>\mintinline</code> |

Some packages define abbreviated versions for “*verbatim macro*” as `\DefineShortVerb`, `\lstMakeShortInline` and `\MakeSpecialShortVerb`, will be detected automatically if are declared explicitly in *⟨input file⟩*.

The following consideration should be kept in mind for some packages that use abbreviations for verbatim macros, such as `shortvrb`[15] or `doc`[15] for example in which there is no explicit `\macro` in the document by means of which the abbreviated form can be detected, for automatic detection need to find `\DefineShortVerb` explicitly to process it correctly. The solution is quite simple, just add in *⟨input file⟩*:

```
\UndefinedShortVerb{\|}
\DefineShortVerb{\|}
```

depending on the package you are using. If your “*verbatim macro*” is not supported by default or can not detect, use the options described in 7.2 and 7.3.

### Verbatim standard

These are the “*classic*” environments for “*writing code*” are considered *⟨verbatim standard⟩*, such as `verbatim` and `lstlisting` environments. The following list (including *\** argument) is considered as *⟨verbatim standard⟩* environments:

- |                                   |                             |                              |                          |
|-----------------------------------|-----------------------------|------------------------------|--------------------------|
| • <code>Example</code>            | • <code>SaveVerbatim</code> | • <code>comment</code>       | • <code>pyglist</code>   |
| • <code>CenterExample</code>      | • <code>PSTcode</code>      | • <code>chklisting</code>    | • <code>program</code>   |
| • <code>SideBySideExample</code>  | • <code>LTXexample</code>   | • <code>verbatimtab</code>   | • <code>programL</code>  |
| • <code>PCenterExample</code>     | • <code>tcblisting</code>   | • <code>listingcont</code>   | • <code>programL</code>  |
| • <code>PSideBySideExample</code> | • <code>spverbatim</code>   | • <code>boxedverbatim</code> | • <code>programs</code>  |
| • <code>verbatim</code>           | • <code>minted</code>       | • <code>demo</code>          | • <code>programf</code>  |
| • <code>Verbatim</code>           | • <code>listing</code>      | • <code>sourcecode</code>    | • <code>programsc</code> |
| • <code>BVerbatim</code>          | • <code>lstlisting</code>   | • <code>xcomment</code>      | • <code>programt</code>  |
| • <code>LVerbatim</code>          | • <code>alltt</code>        | • <code>pygmented</code>     |                          |

They are automatically detected *⟨verbatim standard⟩* environments (including *\** argument) generates by commands:

- |   |                                |
|---|--------------------------------|
| • <code>\DefineVerbatimEnvironment</code> | • <code>\includecomment</code> |
| • <code>\NewListingEnvironment</code>     | • <code>\newtcblisting</code>  |
| • <code>\DeclareTCBListing</code>         | • <code>\NewTCBListing</code>  |
| • <code>\ProvideTCBListing</code>         | • <code>\newverbatim</code>    |
| • <code>\lstnewenvironment</code>         | • <code>\NewProgram</code>     |
| • <code>\newtabverbatim</code>            | • <code>\newminted</code>      |
| • <code>\specialcomment</code>            |                                |

If any of the *⟨verbatim standard⟩* environments is not supported by default or can not detected, you can use the options described in 7.2 and 7.3.

<sup>6</sup>Only  $\text{\TeX}$  can understand  $\text{\TeX}$ , all other languages and programs are just lines in a file.

## Verbatim write

Some environments have the ability to write “*external files*” or “*store content*” in memory, these environments are considered *⟨verbatim write⟩*, such as `scontents`, `filecontents` or `VerbatimOut` environments. The following list is considered (including `*` argument) as *⟨verbatim write⟩* environments:

- `scontents`
- `filecontents`
- `tcboutputlisting`
- `tcbexternal`
- `tcbwritetmp`
- `extcolorbox`
- `exttikzpicture`
- `VerbatimOut`
- `verbatimwrite`
- `filecontentsdef`
- `filecontentshere`
- `filecontentsdefmacro`
- `filecontentsdefstarred`
- `filecontentsgdef`
- `filecontentsdefmacro`
- `filecontentsgdefmacro`

They are automatically detected *⟨verbatim write⟩* (including `*` argument) environments generates by commands:

- `\renewtcbexternalizetcolorbox`
- `\renewtcbexternalizeenvironment`
- `\newtcbexternalizeenvironment`
- `\newtcbexternalizetcolorbox`
- `\newenvsc`

If any of the *⟨verbatim write⟩* environments is not supported by default or can not detected, you can use the options described in 7.2 and 7.3.

## 4.3 Steps process

For creation of the image formats, extraction of source code of environments and creation of an *⟨output file⟩*, `ltximg` need a various steps. Let’s assume that the *⟨input file⟩* is `test.tex`, *⟨output file⟩* is `test-out.tex`, the working directory are “`.`”, the directory for images are `./images`, the temporary directory is `/tmp` and we want to generate images in `pdf` format and *⟨standalone⟩* files for all environments extracted.

We will use the following code as `test.tex`:

```

1 % Some commented lines at begin file
2 \documentclass{article}
3 \usepackage{tikz}
4 \begin{document}
5 Some text
6 \begin{tikzpicture}
7   Some code
8 \end{tikzpicture}
9 Always use \verb|\begin{tikzpicture}| and \verb|\end{tikzpicture}| to open
10 and close environment
11 \begin{tikzpicture}
12   Some code
13 \end{tikzpicture}
14 Some text
15 \begin{verbatim}
16 \begin{tikzpicture}
17   Some code
18 \end{tikzpicture}
19 \end{verbatim}
20 Some text
21 \end{document}
22 Some lines that will be ignored by the script

```

## Validating Options

The first step is read and validated `[⟨options⟩]` from the command line and `test.tex`, verifying that `test.tex` contains *some* environment to extract, check the name and extension of `test-out.tex`, check the directory `./images` if it doesn’t exist create it and create a temporary directory `/tmp/hG45uVklv9`.

The entire `test.tex` file is loaded into memory and “*split*” to start the extraction process.

## Comment and ignore

In the second step, once the file `test.tex` is loaded and divided in memory, proceeds (in general terms) as follows:

Search the words `\begin{` and `\end{` in verbatim standard, verbatim write, verbatim in line and commented lines, if it finds them, converts to `\BEGIN{` and `\END{`, then places all code to extract inside the `\begin{preview} ... \end{preview}`.

At this point “all” the code you want to extract is inside `\begin{preview} ... \end{preview}`.

## Creating files and extracting

In the third step, the script generate *standalone* files: `test-fig-1.tex`, `test-fig-2.tex`, ... and saved in `./images` then proceed in two ways according to the `[<options>]` passed to generate a temporary file with a random number (1981 for example):

1. If script is call *without* `--noprew` options, the following lines will be added at the beginning of the `test.tex` (in memory):

```
\PassOptionsToPackage{inactive}{pst-pdf}%
\AtBeginDocument{%
\RequirePackage[inactive]{pst-pdf}%
\RequirePackage[active,tightpage]{preview}%
\renewcommand\PreviewBbAdjust{-60pt -60pt 60pt 60pt}%
% rest of input file
```

The different parts of the file read in memory are joined and save in a temporary file `test-fig-1981.tex` in “.”. This file will contain all the environments for extraction between `\begin{preview} ... \end{preview}` along with the rest of the document. If the document contains images, these must be in the formats supported by the *engine* selected to process the *input file*.

2. If script is call *with* `--noprew` options, the `\begin{preview} ... \end{preview}` lines are only used as delimiters for extracting the content *without* using the package `preview`, the following lines will be added at the beginning of the `test.tex` (in memory):

```
\PassOptionsToPackage{inactive}{pst-pdf}%
\AtBeginDocument{%
\RequirePackage[inactive]{pst-pdf}%
% only environments extracted
```

Then it is joined with all extracted environments separated by `\newpage` and saved in a temporary file `test-fig-1981.tex` in “.”.

If `--norun` is passed, the temporary file `test-fig-1981.tex` is renamed to `test-fig-all.tex` and moved to `./images`.

## Generate image formats

In the fourth step, the script generating the file `test-fig-1981.pdf` with all code extracted and cropping, running:

```
[user@machine ~:]$ <compiler> -no-shell-escape -interaction=nonstopmode -recorder test-fig-1981.tex
[user@machine ~:]$ pdfcrop --margins 0 test-fig-1981.pdf test-fig-1981.pdf
```

Now move `test-fig-1981.pdf` to `/tmp/hG45uVklv9` and rename to `test-fig-all.pdf`, generate image files `test-fig-1.pdf` and `test-fig-2.pdf` and copy to `./images`, if the image files exist, they will be rewritten each time you run the script. The file `test-fig-1981.tex` is moved to the `./images` and rename to `test-fig-all.tex`.

Note the options passed to *compiler* always use `-no-shell-escape` and `-recorder`, to generate the `.fls` file which is used to delete temporary files and directories after the process is completed. The `--shell` option activates `-shell-escape` for compatibility with packages such as `minted` or others.

## Create output file

In the fifth step, the script apply the option `--clean`, remove all content between `%<remove> ... %</remove>` and try to detect whether the `graphicx` package and the `\graphicspath` command are in the preamble of the *output file* (in memory). If it is not possible to find it, it will read the `.log` file generated by the temporary file with only preamble. Once the detection is complete, the package `grfext` and `\PrependGraphicsExtensions*` will be added at the end of the preamble:

```

1 \usepackage{graphicx}
2 \graphicspath{{images/}}
3 \usepackage{grfext}
4 \PrependGraphicsExtensions*{.pdf}

```

Now converting all extracted code to `\includegraphics` and save `test-out.tex` in “.”, then proceed to run:

```
[user@machine ~:]$ <compiler> -recorder -no-shell-escape test-out.tex
```

generating the file `test-out.pdf`.

## Clean temporary files and dirs

In the sixth step, the script read the files `test-fig-1981.flx` and `test-out.flx`, extract the information from the temporary files and dirs generated in the process in “.” and then delete them together with the directory `/tmp/hG45uVklv9`.

Finally the output file `test-out.tex` looks like this:

```

1 % some commented lines at begin document
2 \documentclass{article}
3 \usepackage{tikz}
4 \graphicspath{{images/}}
5 \usepackage{grfext}
6 \PrependGraphicsExtensions*{.pdf}
7 \begin{document}
8 Some text
9 \includegraphics[scale=1]{test-fig-1}
10 Always use \verb|\begin{tikzpicture}| and \verb|\end{tikzpicture}| to open
11 and close environment
12 \includegraphics[scale=1]{test-fig-2}
13 Some text
14 \begin{verbatim}
15 \begin{tikzpicture}
16     Some code
17 \end{tikzpicture}
18 \end{verbatim}
19 Some text
20 \end{document}

```

## 5 Extract content

The script provides two ways to *extract* content from *input file*, using *environments* and *docstrip tags*. Some environment (including *\** argument) are supported by default. If environments are nested, the outermost one will be extracted.

### 5.1 Default environments

<code>\begin{preview}</code> <i>&lt;env content&gt;</i> <code>\end{preview}</code>	Environment provide by <code>preview</code> [1] package. If any <code>preview</code> environments found in the <i>input file</i> will be extracted and converted these. Internally the script converts all environments to extract in <code>preview</code> environments. Is better comment this package in preamble unless the option <code>-n,--noprew</code> is used. This environment is reserved for the internal process of extraction and conversion, it cannot be passed as an argument to the option <code>--skipenv</code> .
<code>\begin{postscript}</code> <i>&lt;env content&gt;</i> <code>\end{postscript}</code>	Environment provide by <code>pst-pdf</code> [5], <code>auto-pst-pdf</code> [6] and <code>auto-pst-pdf-lua</code> [7] packages. Since the <code>pst-pdf</code> , <code>auto-pst-pdf</code> and <code>auto-pst-pdf-lua</code> packages internally use the <code>preview</code> package, is better comment this in preamble. Only the <i>content</i> of this environment is extracted and “not” the environment itself when using the <code>--srcenv</code> or <code>--subenv</code> options.
<code>\begin{PSTexample}</code> <i>&lt;env content&gt;</i> <code>\end{PSTexample}</code>	Environment provide by <code>pst-exa</code> [8] packages. The script automatically detects the <code>\begin{PSTexample} ... \end{PSTexample}</code> environments and processes them as separately compiled files. The user should have loaded the package with the <code>[swpl]</code> or <code>[tcb]</code> option and run the script using <code>--latex</code> or <code>--xetex</code> . This environment is reserved for the internal process of extraction and conversion, it cannot be passed as an argument to the option <code>--skipenv</code> .
<code>\begin{pspicture}</code> <i>&lt;env content&gt;</i> <code>\end{pspicture}</code>	Environment provide by <code>PStricks</code> [3] package. The plain T <sub>E</sub> X syntax <code>\pspicture ... \endpspicture</code> its converted to L <sup>A</sup> T <sub>E</sub> X syntax <code>\begin{pspicture} ... \end{pspicture}</code> if not within the <code>PSTexample</code> or <code>postscript</code> environments.
<code>\begin{psgraph}</code> <i>&lt;env content&gt;</i> <code>\end{psgraph}</code>	Environment provide by <code>pst-plot</code> [4] package. The plain T <sub>E</sub> X syntax <code>\psgraph ... \endpsgraph</code> its converted to L <sup>A</sup> T <sub>E</sub> X syntax <code>\begin{psgraph} ... \end{psgraph}</code> if not within the <code>PSTexample</code> or <code>postscript</code> environments.

`\begin{tikzpicture}`  
`⟨env content⟩`  
`\end{tikzpicture}`

Environment provide by [TikZ\[2\]](#) package. The plain T<sub>E</sub>X syntax `\tikzpicture ... \tikzpicture` its converted to L<sup>A</sup>T<sub>E</sub>X syntax `\begin{tikzpicture} ... \end{tikzpicture}` but no a short syntax `\tikz ... ;`.

`\begin{pgfpicture}`  
`⟨env content⟩`  
`\end{pgfpicture}`

Environment provide by [pgf\[2\]](#) package. Since the script uses a “*recursive regular expression*” to extract environments, no presents problems if present `pgfinterruptpicture`.

If you need to extract other environments you can use one of the options described in [7.2](#) or [7.3](#).

## 5.2 Extract with docstrip tags

`%<*ltximg>`  
`⟨content⟩`  
`%</ltximg>`

All content included between `%<*ltximg> ... %</ltximg>` is extracted. The tags can *not* be nested and should be at the beginning of the line and in separate lines. Internally the script converts all this tags to `preview` environments.

```
% no space before open tag %<*
%<*ltximg>
code to extract
%</ltximg>
% no space before close tag %</
```

## 5.3 Prevent extraction and remove

Sometimes you do not want to “*extract all*” the environments from `⟨input file⟩` or you want to remove environments or arbitrary content. The script provides a convenient way to solve this situation.

`\begin{nopreview}`  
`⟨env content⟩`  
`\end{nopreview}`

Environment provide by `preview` package. Internally the script converts all “*skip*” environments to `\begin{nopreview} ... \end{nopreview}`. Is better comment this package in preamble unless the option `-n,--noprew` is used. This environment is reserved for the internal process of extraction and conversion, it cannot be passed as an argument to the option `--extrenv`.

`%<*noltximg>`  
`⟨content⟩`  
`%</noltximg>`

All content between `%<*noltximg> ... %</noltximg>` are ignored and no extract. The tags can *not* be nested and should be at the beginning of the line and in separate lines. Internally the script converts all this tags to `nopreview` environments.

```
% no space before open tag %<*
%<*noltximg>
no extract this
%</noltximg>
% no space before close tag %</
```

`%<*remove>`  
`⟨content⟩`  
`%</remove>`

All content between `%<*remove> ... %</remove>` are deleted in the `⟨output file⟩`. The tags can *not* be nested and should be at the beginning of the line and in separate lines.

```
% no space before open tag %<*
%<*remove>
lines removed in output file
%</remove>
% no space before close tag %</
```

The content will be deleted if it is “*not*” within a `⟨verbatim⟩` or `⟨verbatim write⟩` environment. If you want to remove specific environments automatically you can use one of the options described in [7.2](#) or [7.3](#).

# 6 Image Formats

The `⟨image formats⟩` generated by the `ltximg` using `ghostscript` and `poppler-utils` are the following command lines:

**pdf** The image format generated using `ghostscript`. The line executed by the system is:

```
[user@machine ~:]$ gs -q -dNOSAFTER -sDEVICE=pdfwrite -dPDFSETTINGS=/prepress
```

**eps** The image format generated using `pdftops`. The line executed by the system is:

```
[user@machine ~:]$ pdftops -q -eps
```

**png** The image format generated using `ghostscript`. The line executed by the system is:

```
[user@machine ~:]$ gs -q -dNOSAFTER -sDEVICE=pngalpha -r150
```

**jpg** The image format generated using `ghostscript`. The line executed by the system is:

```
[user@machine ~:]$ gs -q -dNOSAFTER -sDEVICE=jpeg -r150 -dJPEGQ=100 \
-dGraphicsAlphaBits=4 -dTextAlphaBits=4
```



**ppm** The image format generated using **pdftoppm**. The line executed by the system is:

```
[user@machine ~:]$ pdftoppm -q -r 150
```

**tiff** The image format generated using **ghostscript**. The line executed by the system is:

```
[user@machine ~:]$ gs -q -dNOSAFTER -sDEVICE=tiff32nc -r150
```

**svg** The image format generated using **pdftocairo**. The line executed by the system is:

```
[user@machine ~:]$ pdftocairo -q -r 150
```

**bmp** The image format generated using **ghostscript**. The line executed by the system is:

```
[user@machine ~:]$ gs -q -dNOSAFTER -sDEVICE=bmp32b -r150
```

## 7 How to use

### 7.1 Syntax

The syntax for **ltximg** is simple, if your use the version provided in your T<sub>E</sub>X distribution:

```
[user@machine ~:]$ ltximg [<options>] [--] <input file>
```

If the development version is used:

```
[user@machine ~:]$ perl ltximg [<options>] [--] <input file>
```

The extension valid for *<input file>* are **.tex** or **.ltx**, relative or absolute paths for files and directories is not supported. If used without [*<options>*] the extracted environments are converted to **pdf** image format and saved in the **./images** directory using **pdflatex** and **preview** package.

### 7.2 Command line interface

The script provides a *command line interface* with short **-** and long **--** option, they may be given before the name of the *<input file>*, the order of specifying the options is not significant. Options that accept a *<value>* require either a blank space  or `=` between the option and the *<value>*. Multiple short options can be bundling and if the last option takes a *<comma separated list>* you need **--** at the end.

<b>-h, --help</b>	<i>&lt;boolean&gt;</i>	(default: off)
	Display a command line help and exit.	
<b>-l, --log</b>	<i>&lt;boolean&gt;</i>	(default: off)
	Write a <b>ltximg.log</b> file with all process information.	
<b>-v, --version</b>	<i>&lt;boolean&gt;</i>	(default: off)
	Display the current version (2.0) and exit.	
<b>-V, --verbose</b>	<i>&lt;boolean&gt;</i>	(default: off)
	Show verbose information of process in terminal.	
<b>-d, --dpi</b>	<i>&lt;integer&gt;</i>	(default: 150)
	Dots per inch for images files. Values are positive integers less than or equal to 2500.	
<b>-t, --tif</b>	<i>&lt;boolean&gt;</i>	(default: off)
	Create a <b>.tif</b> images files using <b>ghostscript</b> .	
<b>-b, --bmp</b>	<i>&lt;boolean&gt;</i>	(default: off)
	Create a <b>.bmp</b> images files using <b>ghostscript</b> .	
<b>-j, --jpg</b>	<i>&lt;boolean&gt;</i>	(default: off)
	Create a <b>.jpg</b> images files using <b>ghostscript</b> .	
<b>-p, --png</b>	<i>&lt;boolean&gt;</i>	(default: off)
	Create a <b>.png</b> transparent image files using <b>ghostscript</b> .	
<b>-e, --eps</b>	<i>&lt;boolean&gt;</i>	(default: off)
	Create a <b>.eps</b> image files using <b>pdftops</b> .	



<b>-s, --svg</b>	<i>&lt;boolean&gt;</i>	(default: off)
Create a <b>.svg</b> image files using <b>pdftocairo</b> .		
<b>-P, --ppm</b>	<i>&lt;boolean&gt;</i>	(default: off)
Create a <b>.ppm</b> image files using <b>pdftoppm</b> .		
<b>-g, --gray</b>	<i>&lt;boolean&gt;</i>	(default: off)
Create a gray scale for all images using <b>ghostscript</b> . The line behind this options is:		
<pre>[user@machine ~:]\$ gs -q -dNOSAfer -sDEVICE=pdfwrite -dPDFSETTINGS=/prepress \ -sColorConversionStrategy=Gray -dProcessColorModel=/DeviceGray</pre>		
<b>-f, --force</b>	<i>&lt;boolean&gt;</i>	(default: off)
Try to capture <code>\psset{&lt;code&gt;}</code> and <code>\tikzset{&lt;code&gt;}</code> to extract. When using the <b>--force</b> option the script will try to capture <code>\psset{&lt;code&gt;}</code> or <code>\tikzset{&lt;code&gt;}</code> and leave it inside the <b>preview</b> environment, any line that is between <code>\psset{&lt;code&gt;}</code> and <code>\begin{pspicture}</code> or between <code>\tikzset{&lt;code&gt;}</code> and <code>\begin{tikzpicture}</code> will be captured.		
<b>-n, --noprew</b>	<i>&lt;boolean&gt;</i>	(default: off)
Create images files without <b>preview</b> package. The <code>\begin{preview}...</code> <code>\end{preview}</code> lines are only used as delimiters for extracting the content <i>without</i> using the package <b>preview</b> . Using this option “only” the extracted environments are processed and not the whole <i>&lt;input file&gt;</i> , sometimes it is better to use it together with <b>--force</b> .		
<b>-m, --margins</b>	<i>&lt;integer&gt;</i>	(default: 0)
Set margins in bp for <b>pdfcrop</b> .		
<b>-r, --runs</b>	<i>&lt;1 2 3&gt;</i>	(default: 1)
Set the number of times the <i>&lt;compiler&gt;</i> will run on the <i>&lt;input file&gt;</i> for environment extraction.		
<b>-o, --output</b>	<i>&lt;file name&gt;</i>	(default: empty)
Create <i>&lt;file name&gt;</i> with all extracted environments converted to <code>\includegraphics</code> . Only <i>&lt;file name&gt;</i> must be passed <i>without</i> relative or absolute paths.		
<b>--prefix</b>	<i>&lt;string&gt;</i>	(default: fig)
Set <i>&lt;prefix&gt;</i> append to each generated files.		
<b>--myverb</b>	<i>&lt;macro name&gt;</i>	(default: myverb)
Set custom verbatim command <code>\myverb</code> . Just pass the <i>&lt;macro name&gt;</i> <i>without</i> “\”.		
<b>--imgdir</b>	<i>&lt;string&gt;</i>	(default: images)
Set the name of directory for save generated files. Only the <i>&lt;name&gt;</i> of directory must be passed <i>without</i> relative or absolute paths.		
<b>--zip</b>	<i>&lt;boolean&gt;</i>	(default: off)
Compress the files generated by the script in <b>.images</b> in <b>.zip</b> format. Does not include <i>&lt;output file&gt;</i> .		
<b>--tar</b>	<i>&lt;boolean&gt;</i>	(default: off)
Compress the files generated by the script in <b>.images</b> in <b>.tar.gz</b> format. Does not include <i>&lt;output file&gt;</i> .		
<b>--srcenv</b>	<i>&lt;boolean&gt;</i>	(default: off)
Create separate files with “only code” for all extracted environments. This option is mutually exclusive with <b>--subenv</b> .		
<b>--subenv</b>	<i>&lt;boolean&gt;</i>	(default: off)
Create a <i>&lt;standalone&gt;</i> files (with “preamble and code”) for all extracted environments. This option is designed to generate “compilable files” for each extracted environment, is mutually exclusive with <b>--srcenv</b> .		
<b>--shell</b>	<i>&lt;boolean&gt;</i>	(default: off)
Enable <code>\write18&lt;shell command&gt;</code> .		
<b>--norun</b>	<i>&lt;boolean&gt;</i>	(default: off)
Execute the script, but do not create image files. This option is designed to be used in conjunction with <b>--srcenv</b> or <b>--subenv</b> and to debug the <i>&lt;output file&gt;</i> .		
<b>--nopdf</b>	<i>&lt;boolean&gt;</i>	(default: off)
Don’t create a <b>.pdf</b> image files.		
<b>--nocrop</b>	<i>&lt;boolean&gt;</i>	(default: off)
Don’t run <b>pdfcrop</b> in image files.		

<code>--arara</code>	<code>&lt;boolean&gt;</code>	(default: off)
Use <code>arara</code> <sup>7</sup> tool for compiler <code>&lt;input file&gt;</code> and <code>&lt;output file&gt;</code> . This option is designed to full process <code>&lt;input file&gt;</code> and <code>&lt;output file&gt;</code> , is mutually exclusive with “any” other <code>&lt;compiler&gt;</code> option. See 8 for more information.		
<code>--xetex</code>	<code>&lt;boolean&gt;</code>	(default: off)
Using <code>xelatex</code> compiler <code>&lt;input file&gt;</code> and <code>&lt;output file&gt;</code> .		
<code>--latex</code>	<code>&lt;boolean&gt;</code>	(default: off)
Using <code>latex»dvips»ps2pdf</code> compiler in <code>&lt;input file&gt;</code> and <code>pdflatex</code> for <code>&lt;output file&gt;</code> .		
<code>--dvips</code>	<code>&lt;boolean&gt;</code>	(default: off)
Using <code>latex»dvips»ps2pdf</code> for compiler <code>&lt;input file&gt;</code> and <code>&lt;output file&gt;</code> .		
<code>--dvi lua</code>	<code>&lt;boolean&gt;</code>	(default: off)
Using <code>dvi lua latex»dvips»ps2pdf</code> for compiler <code>&lt;input file&gt;</code> and <code>lua latex</code> for <code>&lt;output file&gt;</code> .		
<code>--dvi pdf</code>	<code>&lt;boolean&gt;</code>	(default: off)
Using <code>latex»dvi pdf mx</code> for compiler <code>&lt;input file&gt;</code> and <code>&lt;output file&gt;</code> .		
<code>--latexmk</code>	<code>&lt;boolean&gt;</code>	(default: off)
Using <code>latexmk</code> <sup>8</sup> for process <code>&lt;output file&gt;</code> . This option is designed to full process <code>&lt;output file&gt;</code> , is mutually exclusive with <code>--arara</code> .		
<code>--luatex</code>	<code>&lt;boolean&gt;</code>	(default: off)
Using <code>lua latex</code> for compiler <code>&lt;input file&gt;</code> and <code>&lt;output file&gt;</code> .		
<code>--clean</code>	<code>&lt;doc pst tkz all off&gt;</code>	(default: doc)
Removes specific content in <code>&lt;output file&gt;</code> . Valid values for this option are:		
<code>doc</code> All content after <code>\end{document}</code> is removed.		
<code>pst</code> All <code>\psset{&lt;code&gt;}</code> and <code>PStricks</code> package is removed in <code>&lt;preamble&gt;</code> and <code>&lt;body&gt;</code> .		
<code>tkz</code> All <code>\tikzset{&lt;code&gt;}</code> is removed in <code>&lt;body&gt;</code> .		
<code>all</code> Activates doc, pst and tkz.		
<code>off</code> Deactivate all.		
<code>--extenv</code>	<code>&lt;comma separated list&gt;</code>	(default: empty)
Add environments to extract, if it's the last option passed need <code>--</code> at the end. The environments <code>document</code> and <code>nopreview</code> are not supported in this option.		
<code>--skipenv</code>	<code>&lt;comma separated list&gt;</code>	(default: empty)
Add environments that should “not be extracted” and that the script supports by default, if it's the last option passed need <code>--</code> at the end. The environments <code>PSTexample</code> and <code>preview</code> are not supported in this option.		
<code>--verbenv</code>	<code>&lt;comma separated list&gt;</code>	(default: empty)
Add <code>&lt;verbatim standard&gt;</code> environment support, if it's the last option passed need <code>--</code> at the end.		
<code>--writenv</code>	<code>&lt;comma separated list&gt;</code>	(default: empty)
Add <code>&lt;verbatim write&gt;</code> environment support, if it's the last option passed need <code>--</code> at the end.		
<code>--deltenv</code>	<code>&lt;comma separated list&gt;</code>	(default: empty)
Add environments to deleted in <code>&lt;output file&gt;</code> . The environments are delete only in <code>&lt;body&gt;</code> of <code>&lt;output file&gt;</code> , if it's the last option passed need <code>--</code> at the end. The environment <code>document</code> is not supported in this option.		

## Passing options from command line

An example of usage from command line:

```
[user@machine ~:]$ ltximg --latex -s -o test-out test-in.ltx
```

Create a `./images` directory (if it does not exist) with all extracted environments converted to image formats (pdf, svg) in individual files, an output file `<test-out.ltx>` with all extracted environments converted to `\includegraphics` and a single file `<test-in-fig-all.ltx>` with only the extracted environments using `latex»dvips»ps2pdf` and `preview` package for for process `<test-in.ltx>` and `pdflatex` for `<test-out.ltx>`.

<sup>7</sup><https://ctan.org/pkg/arara>

<sup>8</sup><https://www.ctan.org/pkg/latexmk>

### 7.3 Options from input file

Many of the ideas in this section are inspired by the [arara](#). A very useful way to pass options to the script is to place them in commented lines at the beginning of the file, very much in the “style of [arara](#)”.

```
% ltximg: <argument>: {<option one, option two, option three, ...>}
%!ltximg: <argument>: {<option one, option two, option three, ...>}
```

The vast majority of the *<options>* can be passed into the *<input file>*. These should be put at the beginning of the file in commented lines and everything must be on the same line, the exclamation mark `!` deactivates the *<options>*. When passing options from the *<input file>* you should be aware that they must “not” contain `-` or `--`, the `=` sign between an option and its value is mandatory, short names are disabled and options found in the *<input file>* overwrite those passed on the command line. Valid values for *<argument>* are the following:

```
% ltximg: extrenv: {<environment one, environment two, environment three, ...>}
```

This line is to indicate to the script which environments, not supported by default, are extracted.

```
% ltximg: skipenv: {<environment one, environment two, environment three, ...>}
```

This line is to indicate to the script which environments, of the ones supported by default, should not be extracted.

```
% ltximg: verbenv: {<environment one, environment two, environment three, ...>}
```

This line is to indicate to the script which environments, its considerate a *<verbatim standard>*.

```
% ltximg: writenv: {<environment one, environment two, environment three, ...>}
```

This line is to indicate to the script which environments its consider *<verbatim write>*.

```
% ltximg: deltenv: {<environment one, environment two, environment three, ...>}
```

This line is to indicate to the script which environments are deleted.

```
% ltximg: options: {<option one = value, option two = value, option three = value, ...>}
```

This line is to indicate to the script which options (other than those listed above) need to process.

The options passed from the *<input file>* are validated by the script after they are read. If you are going to create an *<output file>* and you do not want these lines to remain, it is better to place them inside the `%<★remove> ... %</remove>`. Like this:

```
1 %<★remove>
2 % ltximg: options: { png, srcenv, xetex }
3 % ltximg: extrenv: { description }
4 %</remove>
```

### Passing options from input file

Adding the following lines to the beginning of the file `file-in.tex`:

```
1 % ltximg: options: { luatex, output = file-out, imgdir = pics, prefix = env }
2 % ltximg: skipenv: { tikzpicture }
3 % ltximg: deltenv: { filecontents }
```

and run:

```
[user@machine~:~]$ ltximg file-in.tex
```

Create a `./pics` directory (if it does not exist) with all extracted environments, except `tikzpicture`, converted to image formats (pdf) in individual files, an output file *<file-out.tex>* with all extracted environments converted to `\includegraphics` and environment `filecontents` removed, a single file *<file-in-env-all.ltx>* with *only* the extracted environments using `lualatex` and `preview` package for process *<file-in.tex>* and *<file-out.tex>*.

## 8 The way of arara

By design, the script only runs “one or more compilation” on top of the *<input file>*, but, sometimes you need to process in a specific mode the *<input file>* or needs to be processed with something other than `LTEX`, `XLLTEX`, `pdfLTEX` or `LuaLTEX` engine. This is where [arara](#)[19] comes in, this “great little tool”, is able to have complete control over the compilation of the *<input file>*, we just have to keep a few considerations in mind:

1. Read the documentation (this always comes first).
2. Add `{ options: [-recorder] }` to “rule” for clean temporary files.
3. Avoiding the use of: `clean: { extensions: [...] }`.
4. Don’t set `-jobname` and `-output-directory` in any “rule”.

When the `--arara` option is passed to the script, the line that runs in the system is:

```
[user@machine~:]$ arara --log file.tex
```

If you have several “rules” within the file they will all be executed, to avoid this we must add:

```
1 % arara: halt
```

After the last “rule” you have at the beginning of the file. With all these considerations in mind it is possible to extract and convert environments from *any file*.

For example, by adding these lines at the beginning of the file:

```
1 % arara: lualatex: { options: [-recorder] }
2 % arara: lualatex: { options: [-recorder] }
3 %<*remove>
4 % ltximg: options: { arara, output = file-out, prefix = tkz}
5 %</remove>
```

and run:

```
[user@machine~:]$ ltximg test.tex
```

Create a `./images` directory (if it does not exist) with all extracted environments converted to image format (pdf) in individual files, an output file `<file-out.tex>` with all extracted environments converted to `\includegraphics`, a single file `<test-tkz-all.tex>` with only the extracted environments using `preview` package and `lualatex` “two times” for process `<test.tex>` and `<file-out.tex>`.

Remember that the `<input file>` and `<output file>` will be compiled using the same “rule”. One *trick* to get around this situation is to use:

```
1 %<*remove>
2 % arara: lualatex: { options: [-recorder] }
3 % arara: lualatex: { options: [-recorder] }
4 % arara: halt
5 % ltximg: options: { arara, output = file-out, prefix = tkz}
6 %</remove>
7 % arara: xelatex: { options: [-recorder] }
8 % arara: xelatex: { options: [-recorder] }
```

The content between `%<*remove>` ... `%</remove>` are remove from output file before compiling. Thus, the output file `<file-out.tex>` will be compiled using `xelatex` “two times”.

As a final consideration, `ltximg` passes options to the `preview` package and the `pdfcrop` script according to the engine used. When using `--arara` it will “try” to detect the used engine by means of a regular expression, if the detection fails the default values will be used.

This does not affect the process of creating `<standalone>` files and can be prevented by using `--noprew` or `--nocrop` at the cost of not having the images cropped.

In this way we can `<compile>` and `<convert>` any document as long as the conditions of the `<input file>` are met and the correct “rule” are used.

## 9 Example usign latexmk

If you are a user of `latexmk`, another great utility that automates the compilation process, you must keep in mind that this will run only in the `<output file>`. Consider the following example adapted from [How to get tikzmark to work](#) and [Draw an aircraft with Tikz](#) to generate an image in `svg`, `png` and `pdf` format from environment `picture` using `lualatex` and `latexmk`.

```

1 %<*remove>
2 % ltximg: extrenv: {picture}
3 % ltximg: skipenv: {tikzpicture}
4 %</remove>
5 \documentclass{article}
6 \usepackage{tikz}
7 \usetikzlibrary{calc,tikzmark}
8 \setlength{\parindent}{0pt}
9 \begin{document}
10 \section{How to get Tikzmark to work}
11 By taking logarithms of both sides:
12
13 \[
14 t = \frac{30 \cdot \ln(3/22)}{\ln(15/22)}
15 \tikzmark{calculator}\approx\tikzmark{otherside}
16 156
17 \]
18 \begin{tikzpicture}[overlay,remember picture]
19 \coordinate (target) at ($(pic cs:calculator)!1/2!(pic cs:otherside) - (0,.5ex)$);
20 \draw[arrows=->] (target) ++(0,-2ex) node [anchor=north] {use calculator} -- (target);
21 \end{tikzpicture}
22
23 \section{Draw an aircraft with Tikz}
24 The best airplane ever drawn by David Carlisle. No TikZ used, just the
25 classic and perhaps forgotten \verb|\begin{picture} ... \end{picture}|.
26
27 \begin{picture}(200,100)
28 \put(30,40){\line(1,0){150}} \put(30,40){\line(0,1){60}}
29 \put(30,100){\line(1,0){20}} \put(50,100){\line(1,-4){10}}
30 \put(60,60){\line(1,0){100}} \put(160,60){\line(1,-1){20}}
31 \put(100,50){\line(0,-1){80}} \put(130,50){\line(0,-1){80}}
32 \put(100,-30){\line(1,0){30}} \put(100,61){\line(0,1){49}}
33 \put(130,61){\line(0,1){49}} \put(100,110){\line(1,0){30}}
34 \end{picture}
35 \end{document}

```

We now run:

```
[user@machine~:]$ ltximg --luatex --latexmk --svg --png -o file-out file-in.tex
```

Create a `./images` directory (if it does not exist) with all `picture` environments, except `tikzpicture`, converted to image formats (`svg`, `png`, `pdf`), an output file `<file-out.tex>` with all `picture` environments converted to `\includegraphics`, a single file `<file-in-fig-all.ltx>` with only environments `picture` extracted using `lualatex` and `preview` package for process `<file-in.tex>` and `latexmk` for full process `<file-out.tex>`.

## 10 Note for dvisvgm users

By design, the image format `svg` is created using `pdftocairo` over the generated `pdf` file, but, if you want to have a good `svg` files that preserve our *typographic* fonts it is best to use `dvisvgm`<sup>9</sup>. The best results of `dvisvgm`[20] are obtained when processing the file in `.dvi` or `.xdv` format, there are two possible ways to do this:

1. Execute the script using `--subenv` and `--norun` to generate `<standalone>` files, move to `./images` and generate `.dvi` or `.xdv` files, then running:

```
[user@machine~:]$ for i in *.tex; do <compiler> [options] $i;done
[user@machine~:]$ for i in *.dvi; do dvisvgm [options] $i;done
```

2. Execute the script using `--norun`, move to `./images` and generate `.dvi` or `.xdv` file, then running:

<sup>9</sup><https://ctan.org/pkg/dvisvgm>

```
[user@machine~:]$ <compiler> [<options>] test-fig-all.tex
[user@machine~:]$ dvisvgm [<options>] test-fig-all.dvi
```

## 11 Final notes

The process and operations required to generate the various types of *<image formats>* or *<standalone>* files have been described throughout the documentation, but, as discussed in section 8, sometimes the requirements are a *little different*. This is the best way to extend the capabilities of the **ltximg**. Although many tasks can be *automated*, in the end only the user knows what the document contains and how it should be generated.

Finding the correct “*regular expressions*” and writing a “*good documentation*” would be the great mission (which does not end yet).

## 12 Change history

The most recent publicly released of **ltximg** is available at CTAN: <https://www.ctan.org/pkg/ltximg>. Historical and developmental versions are available at <https://github.com/pablgonz/ltximg>.

While general feedback via email is welcomed, specific bugs or feature requests should be reported through the issue tracker: <https://github.com/pablgonz/ltximg/issues>.

This is a short list of some of the notable changes in the history of the **ltximg** along with the versions, both development (devp) and public (ctan).

### v2.0 (ctan), 2021-01-20

- Add `-dALLOWPSTRANSARENCY` to `ps2pdf`.
- Fix module detection under T<sub>E</sub>X Live on Windows.
- Add `pod` and `man` documentation.

### v1.9 (ctan), 2020-08-22

- Fix `graphicx` detection.
- Fix typos in documentation.
- Add more contents to `.log` file.

### v1.8 (ctan), 2020-08-18

- It is now possible to extract any environment.
- Add `--log`, `--runs`, `--latexmk` and `--dviua` options.
- All calls to the system are captured.
- Re-write source code according to Perl v5.3x.
- Review of documentation.

### v1.7 (ctan), 2019-08-24

- Add `scontents` environment support.
- Add `filecontentsdefmacro` environment support.
- Fix `regex` in source code.
- Update documentation.

### v1.6 (ctan), 2019-07-13

- Add `--zip` and `--tar` options.
- Add new Verb from `fvextra`.
- Fix and update source code and documentation.

### v1.5 (ctan), 2018-04-12

- Use GitHub to control version.
- Rewrite and optimize code and options.

- Change `pdf2svg` for `pdftocairo`.
- Complete support for `pst-exa` package.
- Escape characters in `regex` according to Perl v5.2x.

### v1.4 (devp), 2016-11-29

- Remove and rewrite code for `regex` and `system` call.
- Add `--arara` compiler, `clean` and `comment` code.
- Add `--dvips` and `--dviu(x)` for creation images.
- Add `bmp`, `tiff` image format.

### v1.3 (devp), 2016-08-14

- Rewrite some part of code (`norun`, `nocrop`, `clean`).
- Support `minted` and `tcolorbox` package.
- Escape some characters in `regex` according to Perl v5.2x.
- All options read from command line and input file.
- Use `/tmp` dir for work process.

### v1.2 (ctan), 2015-04-22

- Remove unused modules.
- Add more image format.
- Fix `regex`.

### v1.1 (ctan), 2015-04-21

- Change `mogrify` to `gs` for image formats.
- Create output file.
- Rewrite source code and fix `regex`.
- Change format date to iso format.

### v1.0 (ctan), 2013-12-01

- First public release.

## 13 References

- [1] KASTRUP, DAVID. “The preview package for L<sup>A</sup>T<sub>E</sub>X”. Available from CTAN, <https://www.ctan.org/pkg/preview>, 2017.
- [2] TANTAU, TILL. “The TikZ and PGF Packages”. Available from CTAN, <https://www.ctan.org/pkg/pgf>, 2020.
- [3] VAN ZANDT, TIMOTHY. “PSTricks - PostScript macros for generic T<sub>E</sub>X”. Available from CTAN, <https://www.ctan.org/pkg/pstricks-base>, 2007.
- [4] VAN ZANDT, TIMOTHY. “pst-plot – Plot data using PSTricks”. Available from CTAN, <https://www.ctan.org/pkg/pst-plot>, 2019.
- [5] NIEPRASCHK, ROLF. “The pst-pdf Packages”. Available from CTAN, <https://www.ctan.org/pkg/pst-pdf>, 2019.
- [6] ROBERTSON, WILL. “The auto-pst-pdf Packages”. Available from CTAN, <https://www.ctan.org/pkg/auto-pst-pdf>, 2009.
- [7] VOß, HERBERT. “auto-pst-pdf-lua - Using LuaL<sup>A</sup>T<sub>E</sub>X with PSTricks”. Available from CTAN, <https://www.ctan.org/pkg/auto-pst-pdf-lua>, 2018.
- [8] VOß, HERBERT. “pst-exa - Typeset PSTricks examples, with pdfT<sub>E</sub>X”. Available from CTAN, <https://www.ctan.org/pkg/pst-exa>, 2017.
- [9] VOß, HERBERT. “pst2pdf - A script to compile PSTricks documents via pdfT<sub>E</sub>X”. Available from CTAN, <https://www.ctan.org/pkg/pst2pdf>, 2017.
- [10] THE L<sup>A</sup>T<sub>E</sub>X<sub>3</sub> PROJECT. “graphics – Enhanced support for graphics”. Available from CTAN, <https://www.ctan.org/pkg/graphics>, 2017.
- [11] OBERDIEK, HEIKO. “The grfext package”. Available from CTAN, <https://www.ctan.org/pkg/grfext>, 2019.
- [12] VAN ZANDT, TIMOTHY. “The xcomment package”. Available from CTAN, <https://www.ctan.org/pkg/xcomment>, 2010.
- [13] ADRIAENS, HENDRI. “The extract package”. Available from CTAN, <https://www.ctan.org/pkg/extract>, 2019.
- [14] TRZECIAK, TOMASZ M. “The cachepic package”. Available from CTAN, <https://www.ctan.org/pkg/cachepic>, 2009.
- [15] MITTELBACH, FRANK. “The doc and shortvrb Packages”. Available from CTAN, <https://www.ctan.org/pkg/doc>, 2020.
- [16] VAN ZANDT, TIMOTHY. “The fancyvrb package - Fancy Verbatims in L<sup>A</sup>T<sub>E</sub>X”. Available from CTAN, <https://www.ctan.org/pkg/fancyvrb>, 2020.
- [17] HOFFMANN, JOBST. “The listings package”. Available from CTAN, <https://www.ctan.org/pkg/listings>, 2020.
- [18] POORE, GEOFFREY M. “The minted package - Highlighted source code in L<sup>A</sup>T<sub>E</sub>X”. Available from CTAN, <https://www.ctan.org/pkg/minted>, 2017.
- [19] THE ISLAND OF T<sub>E</sub>X. “arara - The cool T<sub>E</sub>X automation tool”. Available from CTAN, <https://www.ctan.org/pkg/arara>, 2020.
- [20] GIESEKING, MARTIN. “dvisvgm - A fast DVI to SVG converter”. Available from CTAN, <https://ctan.org/pkg/dvisvgm>, 2020.



## 14 Index of Documentation

The italic numbers denote the pages where the corresponding entry is described.

<b>A</b>		
article (class)	2	
auto-pst-pdf (package)	6	
auto-pst-pdf-lua (package)	6	
<b>B</b>		
book (class)	2	
<b>C</b>		
cachepic (package)	1	
Compiler		
arara	10	
dvilualatex	10	
dviptdpmx	10	
dvips	10	
latex	10	
lualatex	10–13	
pdfplatex	8, 10	
xelatex	10, 12	
Compiler options		
-no-shell-escape	5	
-recorder	5	
-shell-escape	5	
<b>D</b>		
\DeclareTCBListing	3	
\DefineShortVerb	3	
\DefineVerbatimEnvironment	3	
doc (package)	3	
Docstrip tag		
ltximg	7	
noltximg	7	
remove	7	
Document class		
article	2	
book	2	
letter	2	
report	2	
<b>E</b>		
Environments suport by default		
PSTexample	6	
nopreview	7	
pgfpicture	7	
postscript	6	
preview	6	
psgraph	6	
pspicture	6	
tikzpicture	7	
Environments		
PSTexample	6, 10	
document	10	
nopreview	7, 10	
pgfinterruptpicture	7	
picture	13	
postscript	6	
preview	6, 7, 9, 10	
tikzpicture	1, 11, 13	
Environments verbatim		
BVerbatim	3	
CenterExample	3	
Example	3	
LTXexample	3	
LVerbatim	3	
PCenterExample	3	
PSTcode	3	
PSideBySideExample	3	
SaveVerbatim	3	
SideBySideExample	3	
Verbatim	3	
alltt	3	
boxedverbatim	3	
chklisting	3	
comment	3	
demo	3	
listingcont	3	
listing	3	
lstlisting	3	
minted	3	
programL	3	
programf	3	
programl	3	
programscl	3	
programs	3	
programt	3	
program	3	
pyglist	3	
pygmented	3	
sourcecode	3	
spverbatim	3	
tcblisting	3	
verbatimtab	3	
verbatim	3	
xcomment	3	
Environments verbatim write		
VerbatimOut	4	
extcolorbox	4	
exttikzpicture	4	
filecontents	2	
filecontentsdefmacro	4	
filecontentsdefstarred	4	
filecontentsdef	4	
filecontentsgdefmacro	4	
filecontentsgdef	4	
filecontentshere	4	
filecontents	4, 11	
scontents	4	
tcbboxexternal	4	
tcbboxputlisting	4	
tcbboxwritetmp	4	
verbatimwrite	4	
extract (package)	1	

<b>F</b>	
fancyvrb (package)	3
File	
ltximg.log	8
File extentions	
.bmp	8
.dvi	13
.eps	8
.fls	5
.jpg	8
.log	5, 8
.ltx	8
.pdf	9
.png	8
.ppm	9
.svg	9
.tar.gz	9
.tex	8
.tif	8
.xdv	13
.zip	9
\fverb	3

<b>G</b>	
\graphicspath	5
graphicx (package)	2, 5
grfext (package)	2, 5

<b>I</b>	
Image formats	
bmp	8
eps	7
jpg	7
pdf	4, 7, 8, 10–13
png	7, 13
ppm	8
svg	8, 10, 13
tiff	8
\include	2
\includecomment	3
\includegraphics	1, 6, 9–13
\input	2

<b>L</b>	
letter (class)	2
listings (package)	3
\lstinline	3
\lstMakeShortInline	3
\lstnewenvironment	3

<b>M</b>	
\MakeSpecialShortVerb	3
\mint	3
minted (package)	3, 5
\mintinline	3

<b>N</b>	
\newenvsc	4
\NewListingEnvironment	3
\newmint	3
\newminted	3
\newmintinline	3

\newpage	5
\NewProgram	3
\newtabverbatim	3
\newtcexternalizeenvironment	4
\newtcexternalizetcolorbox	4
\NewTCBListing	3
\newtcblisting	3
\newverbatim	3

<b>O</b>	
Operating system	
Linux	1, 2
Windows	2
cygwin	2

ltximg options in command line	
--arara	10, 12
--bmp	8
--clean	5, 10
--deltenv	10
--dpi	8
--dvi lua	10
--dvi pdf	10
--dvips	10
--eps	8
--extrenv	7, 10
--force	9
--gray	9
--help	8
--imgdir	9
--jpg	8
--latexmk	10
--latex	6, 10
--log	8
--luatex	10
--margins	9
--myverb	9
--nocrop	9, 12
--nopdf	9
--noprew	5–7, 9, 12
--norun	5, 9, 13
--output	9
--png	8
--ppm	9
--prefix	9
--runs	9
--shell	5, 9
--skipenv	6, 10
--srcenv	6, 9
--subenv	6, 9, 13
--svg	9
--tar	9
--tif	8
--verbenv	10
--verbose	8
--version	8
--writenv	10
--xetex	6, 10
--zip	9

ltximg options in input file	
deltenv	11
extrenv	11

options	11	\ProvideTCBListing	3
skipenv	11	pst-exa (package)	6
verbenv	11	pst-pdf (package)	2, 6
writenv	11	pst-plot (package)	6
		PStricks (package)	1, 6, 10
		\pyginline	3
		\pygment	3
<b>P</b>		<b>Q</b>	
Package options		\qverb	3
swpl	6		
tcb	6	<b>R</b>	
Packages		\renewtcexternalizeenvironment	4
PStricks	1, 6, 10	\renewtcexternalizetcolorbox	4
TikZ	1, 7	report (class)	2
auto-pst-pdf-lua	6		
auto-pst-pdf	6	<b>S</b>	
cachepic	1	\Scontents	3
doc	3	Scripts	
extract	1	latexindent	2
fancyvrb	3	latexmk	10, 13
graphicx	2, 5	latexpand	2
grfext	2, 5	pdfcrop	2, 9, 12
listings	3	ps2pdf	10
minted	3, 5	pst2pdf	1, 2
pgf	7	shortvrb (package)	3
preview	1, 2, 5–13	\specialcomment	3
pst-exa	6	\spverb	3
pst-pdf	2, 6	swpl (package option)	6
pst-plot	6		
shortvrb	3	<b>T</b>	
xcomment	1	tcb (package option)	6
\pagestyle	2	\tcboxverb	3
pgf (package)	7	\thispagestyle	2
\PrependGraphicsExtensions*	5	TikZ (package)	1, 7
preview (package)	1, 2, 5–13		
Programs		<b>V</b>	
Windows Terminal	2	\Verb	3
arara	12	\verb	3
chktex	2		
dvisvgm	13, 14	<b>W</b>	
ghostscript	1, 2, 7–9	\write18	9
pdftocairo	8, 9, 13		
pdftoept	7	<b>X</b>	
pdftoppm	8, 9	xcomment (package)	1
pdftops	8		
perl	1, 2		
poppler-utils	1, 2, 7		