LaTeX Environments to Image Format

v2.2 — 2026-11-27[*]

©2013–2026 by Pablo González L[†]

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 the ⟨*input file*⟩ to image formats and standalone files using `ghostscript` and `poppler-utils`. It generates a file with only the extracted environments and another one with all the extracted environments converted to `\includegraphics`.

Contents

1 License . . . 1
2 Motivation and Acknowledgments . . . 1
3 Requirements for operation . . . 2
3.1 Modules required . . . 2
4 How it works . . . 2
4.1 The input file . . . 2
4.2 Verbatim contents . . . 3
4.3 Steps process . . . 4
5 Extract content . . . 6
5.1 Default environments . . . 6
5.2 Extract with docstrip tags . . . 7
5.3 Prevent extraction and remove . . . 7
6 Image Formats . . . 7
7 How to use . . . 8
7.1 Syntax . . . 8
7.2 Command line interface . . . 8
7.3 Options from input file . . . 11
8 The way of `arara` . . . 12
9 Note for `dvisvgm` users . . . 12
10 Example using `latexmk` . . . 13
11 Final notes . . . 14
12 Change history . . . 14
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 you to *externalize* the environments, but the idea was to extend this to *any type* of environment, covering three central points:

1. Generate separate image files for the environments.
2. Generate standalone files with only the extracted environments.
3. Generate a file replacing the environments with `\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 cachepic[14] packages, among others, but none of them covered all the points.

On the internet there are some solutions written in `bash` that are able to extract and convert environments, but in general they have problems when the document contains *"verbatim style"* code, or they are only available for `Linux`.

[*] This file documents version 2.2, last revised 2026-11-27.
[†] E-mail: «pablgonz@yahoo.com»

After analysing the situation, the best option was to create a new *"script"* that covered the three points and was multi-platform. `ltximg` was born from the union of all these ideas.

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

3 Requirements for operation

For `ltximg` to work fully you need a modern TeX distribution such as TeX Live or MiKTeX, version 5.26 or later of `perl`, version 9.52 or later of `ghostscript`, version 1.40 or later of `pdfcrop` and version 0.52 or later of `poppler-utils`. MiKTeX and `cygwin` users must install the appropriate software for full operation.

`ltximg` automatically detects `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 TeX distribution.

The script is tested with TeX Live 2026 on `Linux` (Ubuntu 26.04, `ghostscript` 10.06, `poppler-utils` 26.01 and `perl` 5.40) and `Windows` (Windows Server 2025, `ghostscript` 10.08 and `poppler-utils` 26.09) using the standard classes provided by LaTeX: 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 are also needed.

3.1 Modules required

`ltximg` only uses core `perl` modules. The `perl` distribution included in TeX Live for `Windows` does not have the `Win32::Console::ANSI` module. This does not affect how the script works, but it does affect how messages are displayed when it is run from the *standard* `cmd`, so it is recommended to use a more modern (and comfortable) application such as `Windows Terminal`.

- `Getopt::Long`
- `File::Spec::Functions`
- `File::Basename`
- `Archive::Tar`
- `FileHandle`
- `IO::Compress::Zip`
- `File::Path`
- `File::Temp`
- `POSIX`
- `File::Copy`
- `File::Find`
- `Env`
- `autodie`
- `Config`
- `Cwd`
- `Term::ANSIColor`
- `Module::Load::Conditional`
- Only on `Windows`:
  - `Win32`
  - `Win32::Console::ANSI`
  - `Win32::TieRegistry`

4 How it works

It is important to have a general idea of how the *"extraction and conversion"* process works and of the requirements that must be met for everything to work correctly. For this, some concepts must be clear: how to work with the ⟨*input file*⟩, the ⟨*verbatim content*⟩ and the ⟨*steps process*⟩.

4.1 The input file

The ⟨*input file*⟩ must have *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: any type of content can go before `\documentclass` and after `\end{document}`, and internally the script will *"split"* the ⟨*input file*⟩ at these points.

If the ⟨*input file*⟩ contains files using `\input{`⟨*file*⟩`}` or `\include{`⟨*file*⟩`}` these will not be processed: for the *script* they are only lines within the file. If you want them to be processed, it is better to use `latexpand`[3] 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 were a plain text file. It is advisable to format the source code of the ⟨*input file*⟩ using utilities such as `chktex`[4] and `latexindent`[5], 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, `\pagestyle{`⟨*empty*⟩`}` will be added, and if they are present and `{`⟨*style*⟩`}` is different from `{`⟨*empty*⟩`}`, it 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. The script divides the ⟨*input file*⟩ into four parts and then processes them:

```
% Part One: Everything before \documentclass
\documentclass{article}
```

[1] https://ctan.org/pkg/pst2pdf
[2] https://ctan.org/pkg/pdfcrop
[3] https://www.ctan.org/pkg/latexpand
[4] https://www.ctan.org/pkg/chktex
[5] https://www.ctan.org/pkg/latexindent

©2013 – 2026 by Pablo González

```
% Part two: Everything between \documentclass and \begin{document}
\begin{document}
% Part three: : Everything between \begin{document} and \end{document}
\end{document}
% Part Four: Everything after \end{document}
```

If for some reason you have a `filecontents` environment 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 it. Similarly, the content after `\end{document}` is ignored in the extraction process.

4.2 Verbatim contents

One of the greatest capabilities of this script is to *"skip"* the complications that ⟨*verbatim content*⟩ causes when environments are extracted using tools outside the "TEX world"[6]. In order to *"skip"* the complications, the ⟨*verbatim content*⟩ is classified into three types:

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

Verbatim in line

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 of the *"verbatim macros"* provided by the packages minted[18], fancyvrb[16] and listings[17] have been tested and are fully supported. The *verbatim macros* (including the `*` argument) generated by `\newmint` and `\newmintinline` and the following ones are detected automatically:

- `\mint`
- `\spverb`
- `\qverb`
- `\fverb`
- `\verb`
- `\Verb`
- `\lstinline`
- `\pyginline`
- `\pygment`
- `\Scontents`
- `\tcboxverb`
- `\mintinline`

Some packages define abbreviated versions of *"verbatim macros"*, such as `\DefineShortVerb`, `\lstMakeShortInline` and `\MakeSpecialShortVerb`; they will be detected automatically if they are declared explicitly in the ⟨*input file*⟩.

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

```
\UndefineShortVerb{\|}
\DefineShortVerb{\|}
```

depending on the package you are using. If your *"verbatim macro"* is not supported by default or cannot be detected, use the options described in 7.2 and 7.3.

Verbatim standard

The *"classic"* environments for *"writing code"* are considered ⟨*verbatim standard*⟩, such as the `verbatim` and `lstlisting` environments. The following environments (including the `*` argument) are considered ⟨*verbatim standard*⟩:

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

The ⟨*verbatim standard*⟩ environments (including the `*` argument) generated by the following commands are detected automatically:

- `\DefineVerbatimEnvironment`
- `\NewListingEnvironment`
- `\DeclareTCBListing`
- `\ProvideTCBListing`
- `\lstnewenvironment`
- `\newtabverbatim`
- `\specialcomment`
- `\includecomment`
- `\newtcblisting`
- `\NewTCBListing`
- `\newverbatim`
- `\NewProgram`
- `\newminted`

[6] Only TEX can understand TEX, all other languages and programs are just lines in a file.

©2013 – 2026 by Pablo González

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

Verbatim write

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

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

The ⟨*verbatim write*⟩ environments (including the `*` argument) generated by the following commands are detected automatically:

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

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

4.3 Steps process

To create the image formats, extract the source code of the environments and create an ⟨*output file*⟩, **`ltximg`** needs several steps. Let's assume that the ⟨*input file*⟩ is `test.tex`, the ⟨*output file*⟩ is `test-out.tex`, the working directory is "`./`", the directory for images is `./images`, the temporary directory is `/tmp` and we want to generate images in `pdf` format and ⟨*standalone*⟩ files for all the extracted environments.

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

```
% Some commented lines at begin file
\documentclass{article}
\usepackage{tikz}
\begin{document}
Some text
\begin{tikzpicture}
  Some code
\end{tikzpicture}
Always use \verb|\begin{tikzpicture}| and \verb|\end{tikzpicture}| to open
and close environment
\begin{tikzpicture}
  Some code
\end{tikzpicture}
Some text
\begin{verbatim}
\begin{tikzpicture}
  Some code
\end{tikzpicture}
\end{verbatim}
Some text
\end{document}
Some lines that will be ignored by the script
```

Validating Options

The first step is to read and validate the [⟨*options*⟩] from the command line and from `test.tex`, verify that `test.tex` contains *some* environment to extract, check the name and extension of `test-out.tex`, check the directory `./images` (and create it if it doesn't exist) 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, the script proceeds (in general terms) as follows:

©2013 – 2026 by Pablo González

Search for the words \begin{ and \end{ in verbatim standard, verbatim write, verbatim in line and commented lines; if it finds them, it converts them to \BEGIN{ and \END{, and then places all the code to extract inside \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 generates the ⟨*standalone*⟩ files test-fig-1.tex, test-fig-2.tex, ... and saves them in ./images. Then it proceeds in one of two ways, according to the [⟨*options*⟩] passed, to generate a temporary file with a random number (1981 for example):

1. If the script is called *without* the --noprew option, 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 into memory are joined and saved 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 the script is called *with* the --noprew option, the \begin{preview}...\end{preview} lines are only used as delimiters for extracting the content *without* using the preview package, and 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 the 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 generates the file test-fig-1981.pdf with all the extracted code and crops it, 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
```

Then test-fig-1981.pdf is moved to /tmp/hG45uVklv9 and renamed to test-fig-all.pdf, and the image files test-fig-1.pdf and test-fig-2.pdf are generated and copied to ./images. If the image files already exist, they will be overwritten each time you run the script. The file test-fig-1981.tex is moved to ./images and renamed to test-fig-all.tex.

Note that the options passed to the ⟨*compiler*⟩ always include -no-shell-escape and -recorder, to generate the .fls file that is used to delete temporary files and directories after the process is completed. The --shell option activates -shell-escape or -enable-write18 in MiKTeX for compatibility with packages such as minted or others.

Create output file

In the fifth step, the script applies the --clean option, removes all the content between %<*remove> ... %</remove> and tries to detect whether the graphicx package and the \graphicspath command are in the preamble of the ⟨*output file*⟩ (in memory). If they cannot be found, it reads the .log file generated by a temporary file with only the preamble. Once the detection is complete, the grfext package and \PrependGraphicsExtensions* are added at the end of the preamble:

```
\usepackage{graphicx}
\graphicspath{{images/}}
\usepackage{grfext}
\PrependGraphicsExtensions*{.pdf}
```

Now all the extracted code is converted to \includegraphics, test-out.tex is saved in "./" and then the script runs:

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

generating the file test-out.pdf.

©2013 – 2026 by Pablo González

Clean temporary files and dirs

In the sixth step, the script reads the files `test-fig-1981.fls` and `test-out.fls`, gets from them the list of temporary files generated in "`./`" and deletes them together with the directory `/tmp/hG45uVklv9`. Only the files created by the script (the temporary files and the auxiliary files of the ⟨*output file*⟩) are deleted; other files written by the document are kept.

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

```
% some commented lines at begin document
\documentclass{article}
\usepackage{tikz}
\graphicspath{{images/}}
\usepackage{grfext}
\PrependGraphicsExtensions*{.pdf}
\begin{document}
Some text
\includegraphics[scale=1]{test-fig-1}
Always use \verb|\begin{tikzpicture}| and \verb|\end{tikzpicture}| to open
and close environment
\includegraphics[scale=1]{test-fig-2}
Some text
\begin{verbatim}
\begin{tikzpicture}
  Some code
\end{tikzpicture}
\end{verbatim}
Some text
\end{document}
```

5 Extract content

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

5.1 Default environments

`\begin{preview}` ⟨*env content*⟩ `\end{preview}`

Environment provided by the preview[1] package. Any `preview` environments found in the ⟨*input file*⟩ will be extracted and converted. Internally, the script converts all the environments to extract into `preview` environments. It is better to comment out this package in the preamble unless the `-n`,`--noprew` option is used. This environment is reserved for the internal process of extraction and conversion, so it cannot be passed as an argument to the `--skipenv` option.

`\begin{postscript}` ⟨*env content*⟩ `\end{postscript}`

Environment provided by the pst-pdf[5], auto-pst-pdf[6] and auto-pst-pdf-lua[7] packages. Since the pst-pdf, auto-pst-pdf and auto-pst-pdf-lua packages use the preview package internally, it is better to comment them out in the preamble. Only the *content* of this environment is extracted and *"not"* the environment itself when using the `--srcenv` or `--subenv` options.

`\begin{PSTexample}` ⟨*env content*⟩ `\end{PSTexample}`

Environment provided by the pst-exa[8] package. The script automatically detects the `\begin{PSTexample}` `...\end{PSTexample}` environments and processes them as separately compiled files. The user should have loaded the package with the [swpl] or [tcb] option and run the script using `--latex` or `--xetex`. This environment is reserved for the internal process of extraction and conversion, so it cannot be passed as an argument to the `--skipenv` option.

©2013 – 2026 by Pablo González

\begin{pspicture} Environment provided by the PSTricks[3] package. The plain TeX syntax \pspicture ... \endpspicture is converted to the LaTeX syntax \begin{pspicture} ... \end{pspicture} if it is not within the PSTexample or postscript environments.
⟨env content⟩
\end{pspicture}

\begin{psgraph} Environment provided by the pst-plot[4] package. The plain TeX syntax \psgraph ... \endpsgraph is converted to the LaTeX syntax \begin{psgraph} ... \end{psgraph} if it is not within the PSTexample or postscript environments.
⟨env content⟩
\end{psgraph}

\begin{tikzpicture} Environment provided by the TikZ[2] package. The plain TeX syntax \tikzpicture ... \endtikzpicture is converted to the LaTeX syntax \begin{tikzpicture} ... \end{tikzpicture}, but the short syntax \tikz ... ; is not.
⟨env content⟩
\end{tikzpicture}

\begin{pgfpicture} Environment provided by the pgf[2] package. Since the script uses a *"recursive regular expression"* to extract environments, there are no problems if pgfinterruptpicture is present.
⟨env content⟩
\end{pgfpicture}

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> All content included between %<*ltximg> ... %</ltximg> is extracted. The tags can*not* be nested and must be at the beginning of the line and on separate lines. Internally, the script converts all these tags to preview environments.
⟨content⟩
%</ltximg>

```
% 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 the ⟨*input file*⟩, or you want to remove environments or arbitrary content. The script provides a convenient way to solve this situation.

\begin{nopreview} Environment provided by the preview package. Internally, the script converts all the *"skipped"* environments to \begin{nopreview} ...\end{nopreview}. It is better to comment out this package in the preamble unless the -n,--noprew option is used. This environment is reserved for the internal process of extraction and conversion, so it cannot be passed as an argument to the --extrenv option.
⟨env content⟩
\end{nopreview}

%<*noltximg> All content between %<*noltximg> ... %</noltximg> is ignored and not extracted. The tags can*not* be nested and must be at the beginning of the line and on separate lines. Internally, the script converts all these tags to nopreview environments.
⟨content⟩
%</noltximg>

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

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

```
% 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*⟩ are generated by **ltximg** using ghostscript and poppler-utils with the following command lines:

pdf Image format generated using ghostscript. The line executed by the system is:

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

eps Image format generated using pdftops. The line executed by the system is:

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

©2013 – 2026 by Pablo González

png Image format generated using ghostscript. The line executed by the system is:

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

jpg Image format generated using ghostscript. The line executed by the system is:

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

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

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

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

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

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

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

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

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

7 How to use

7.1 Syntax

The syntax for **ltximg** is simple. If you use the version provided in your TEX distribution:

```
[user@machine ~:]$ ltximg [⟨options⟩] [--] ⟨input file⟩
```

If the development version is used:

```
[user@machine ~:]$ perl ltximg [⟨options⟩] [--] ⟨input file⟩
```

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

7.2 Command line interface

The script provides a *command line interface* with short - and long -- options. They must be given before the name of the ⟨*input file*⟩, and the order in which they are given 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 bundled, and if the last option takes a ⟨*comma separated list*⟩ you need -- at the end.

-h, --help ⟨*boolean*⟩ (default: off)

Display the command line help and exit.

-l, --log ⟨*boolean*⟩ (default: off)

Write a ltximg.log file with all process information.

-v, --version ⟨*boolean*⟩ (default: off)

Display the current version (2.2) and exit.

-V, --verbose ⟨*boolean*⟩ (default: off)

Show verbose information about the process in the terminal.

-d, --dpi ⟨*integer*⟩ (default: 150)

Dots per inch for image files. Values are positive integers less than 2500.

-t, --tif ⟨*boolean*⟩ (default: off)

Create .tif image files using ghostscript.

-b, --bmp ⟨*boolean*⟩ (default: off)

Create .bmp image files using ghostscript.

-j, --jpg ⟨*boolean*⟩ (default: off)

Create .jpg image files using ghostscript.

-p, --png ⟨*boolean*⟩ (default: off)

Create transparent .png image files using ghostscript.

-e, --eps ⟨*boolean*⟩ (default: off)

©2013 – 2026 by Pablo González

Create .eps image files using pdftops.

-s, --svg ⟨*boolean*⟩ (default: off)

Create .svg image files using pdftocairo.

-P, --ppm ⟨*boolean*⟩ (default: off)

Create .ppm image files using pdftoppm.

-g, --gray ⟨*boolean*⟩ (default: off)

Create all images in gray scale using ghostscript. The line behind this option is:

```
[user@machine ~:]$ gs -q -dSAFER -sDEVICE=pdfwrite -dPDFSETTINGS=/prepress       \
                      -sColorConversionStrategy=Gray -dProcessColorModel=/DeviceGray
```

-f, --force ⟨*boolean*⟩ (default: off)

Try to capture \psset{⟨*code*⟩} and \tikzset{⟨*code*⟩} to extract. When using the --force option the script will try to capture \psset{⟨*code*⟩} or \tikzset{⟨*code*⟩} and leave it inside the preview environment; any line between \psset{⟨*code*⟩} and \begin{pspicture} or between \tikzset{⟨*code*⟩} and \begin{tikzpicture} will be captured.

-n, --noprew ⟨*boolean*⟩ (default: off)

Create image files without the preview package. The \begin{preview}...\end{preview} lines are only used as delimiters for extracting the content *without* using the preview package. With this option *"only"* the extracted environments are processed, not the whole ⟨*input file*⟩; sometimes it is better to use it together with --force. This option is used automatically when the ⟨*input file*⟩ has \DocumentMetadata, since the preview package does not work correctly with it.

-m, --margins ⟨*integer*⟩ (default: 0)

Set margins in bp for pdfcrop.

-r, --runs ⟨1|2⟩ (default: 1)

Set the number of times the ⟨*compiler*⟩ will run on the ⟨*input file*⟩ for environment extraction.

-o, --output ⟨*file name*⟩ (default: empty)

Create ⟨*file name*⟩ with all extracted environments converted to \includegraphics. Only the ⟨*file name*⟩ must be passed, *without* relative or absolute paths, and it can only contain letters, digits, ., +, - and _.

--prefix ⟨*string*⟩ (default: fig)

Set the ⟨*prefix*⟩ appended to each generated file. It can only contain letters, digits, ., +, - and _.

--myverb ⟨*macro name*⟩ (default: myverb)

Set custom verbatim command \myverb. Just pass the ⟨*macro name*⟩ *without* "\"; it can only contain letters.

--imgdir ⟨*string*⟩ (default: images)

Set the name of the directory where the generated files are saved. Only the ⟨*name*⟩ of the directory must be passed, *without* relative or absolute paths, and it can only contain letters, digits, ., +, - and _.

--zip ⟨*boolean*⟩ (default: off)

Compress the files generated by the script in ./images in .zip format. The ⟨*output file*⟩ is not included.

--tar ⟨*boolean*⟩ (default: off)

Compress the files generated by the script in ./images in .tar.gz format. The ⟨*output file*⟩ is not included.

--srcenv ⟨*boolean*⟩ (default: off)

Create separate files with *"only the code"* of all the extracted environments. This option is mutually exclusive with --subenv.

--subenv ⟨*boolean*⟩ (default: off)

Create ⟨*standalone*⟩ files (with *"preamble and code"*) for all the extracted environments. This option is designed to generate *"compilable files"* for each extracted environment and is mutually exclusive with --srcenv.

--shell ⟨*boolean*⟩ (default: off)

Enable \write18⟨*shell command*⟩. For security reasons this option is only accepted on the command line; it is ignored if it is passed from the ⟨*input file*⟩.

--norun ⟨*boolean*⟩ (default: off)

Execute the script, but do not create image files. This option is designed to be used in conjunction with --srcenv or --subenv and to debug the ⟨*output file*⟩.

--nopdf ⟨*boolean*⟩ (default: off)

©2013 – 2026 by Pablo González

Don't create .pdf image files.

--nocrop ⟨boolean⟩ (default: off)

Don't run pdfcrop on the image files.

--arara ⟨boolean⟩ (default: off)

Use the arara[7] tool to compile the ⟨*input file*⟩ and the ⟨*output file*⟩. This option is designed to fully process the ⟨*input file*⟩ and the ⟨*output file*⟩, and is mutually exclusive with *"any"* other ⟨*compiler*⟩ option. See 8 for more information.

--xetex ⟨boolean⟩ (default: off)

Use xelatex to compile the ⟨*input file*⟩ and the ⟨*output file*⟩. xelatex is called with the -no-pdf option, generating an .xdv file that is then processed with xdvipdfmx; this is only done to convert the environments to images *faster* and only for the ⟨*input file*⟩. When the pstricks package is loaded, xdvipdfmx uses the configuration file dvipdfmx-unsafe.cfg (the same as xelatex-unsafe) to support transparencies.

--latex ⟨boolean⟩ (default: off)

Using latex»dvips»ps2pdf for the ⟨*input file*⟩ and pdflatex for the ⟨*output file*⟩. To support transparencies in recent versions of ghostscript, the line executed by ps2pdf is:

```
[user@machine ~:]$ ps2pdf -dPDFSETTINGS=/prepress -dAutoRotatePages=/None -dALLOWPSTRANSPARENCY
```

--dvips ⟨boolean⟩ (default: off)

Using latex»dvips»ps2pdf for the ⟨*input file*⟩ and the ⟨*output file*⟩. To support transparencies in recent versions of ghostscript, the line executed by ps2pdf is:

```
[user@machine ~:]$ ps2pdf -dPDFSETTINGS=/prepress -dAutoRotatePages=/None -dALLOWPSTRANSPARENCY
```

--dvilua ⟨boolean⟩ (default: off)

Using dvilualatex»dvips»ps2pdf for the ⟨*input file*⟩ and lualatex for the ⟨*output file*⟩. To support transparencies in recent versions of ghostscript, the line executed by ps2pdf is:

```
[user@machine ~:]$ ps2pdf -dPDFSETTINGS=/prepress -dAutoRotatePages=/None -dALLOWPSTRANSPARENCY
```

--dvipdf ⟨boolean⟩ (default: off)

Using latex»dvips»ps2pdf for the ⟨*input file*⟩ and latex»dvipdfmx for the ⟨*output file*⟩. This is the same process used by --latexmk with -pdfdvi for the ⟨*output file*⟩. The bounding box of the images (.xbb files) is written using extractbb, so the ⟨*output file*⟩ is compiled without shell escape. If the ⟨*output file*⟩ does not set the dvipdfmx driver, it is passed to graphicx.

--latexmk ⟨boolean⟩ (default: off)

Use latexmk[8] to process the ⟨*output file*⟩. This option is designed to fully process the ⟨*output file*⟩ and is mutually exclusive with --arara. It can be combined with --xetex, --luatex, --dvilua, --dvips and --dvipdf to select the engine used by latexmk.

--luatex ⟨boolean⟩ (default: off)

Use lualatex to compile the ⟨*input file*⟩ and the ⟨*output file*⟩.

--clean ⟨doc|pst|tkz|all|off⟩ (default: doc)

Remove specific content in the ⟨*output file*⟩. Valid values for this option are:

- doc All content after \end{document} is removed.
- pst All \psset{⟨*code*⟩} and the PSTricks package are removed in the ⟨*preamble*⟩ and the ⟨*body*⟩.
- tkz All \tikzset{⟨*code*⟩} are removed in the ⟨*body*⟩.
- all Activates doc, pst and tkz.
- off Deactivates all of them.

--extrenv ⟨*comma separated list*⟩ (default: empty)

Add environments to extract. If it is the last option passed, you need -- at the end. The environments document and nopreview are not supported in this option.

--skipenv ⟨*comma separated list*⟩ (default: empty)

Add environments that should *"not be extracted"* among those the script supports by default. If it is the last option passed, you need -- at the end. The environments PSTexample and preview are not supported in this option.

--verbenv ⟨*comma separated list*⟩ (default: empty)

Add support for ⟨*verbatim standard*⟩ environments. If it is the last option passed, you need -- at the end.

[7] https://ctan.org/pkg/arara
[8] https://www.ctan.org/pkg/latexmk

©2013 – 2026 by Pablo González

`--writenv` ⟨*comma separated list*⟩ (default: empty)

Add support for ⟨*verbatim write*⟩ environments. If it is the last option passed, you need `--` at the end.

`--deltenv` ⟨*comma separated list*⟩ (default: empty)

Add environments to delete in the ⟨*output file*⟩. The environments are deleted only in the ⟨*body*⟩ of the ⟨*output file*⟩. If it is the last option passed, you need `--` at the end. The environment `document` is not supported in this option.

Passing options from command line

An example of usage from the command line:

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

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

7.3 Options from input file

Many of the ideas in this section are inspired by `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 in the ⟨*input file*⟩. They must be placed at the beginning of the file in commented lines, and each one must be on a single 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 tells the script which environments, not supported by default, are extracted.

`% ltximg:skipenv: {`⟨*environment one, environment two, environment three, ...*⟩`}`

This line tells the script which environments, among those supported by default, should not be extracted.

`% ltximg:verbenv: {`⟨*environment one, environment two, environment three, ...*⟩`}`

This line tells the script which environments are considered ⟨*verbatim standard*⟩.

`% ltximg:writenv: {`⟨*environment one, environment two, environment three, ...*⟩`}`

This line tells the script which environments are considered ⟨*verbatim write*⟩.

`% ltximg:deltenv: {`⟨*environment one, environment two, environment three, ...*⟩`}`

This line tells the script which environments are deleted.

`% ltximg:options: {`⟨*option one = value, option two = value, option three = value, ...*⟩`}`

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

All options passed from the ⟨*input file*⟩ are validated by `ltximg` after they are read, except `--shell`, which is ignored for security reasons. 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:

```
%<*remove>
% ltximg: options: { png, srcenv, xetex }
% ltximg: extrenv: { description }
%</remove>
```

Passing options from input file

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

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

and run:

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

This creates a `./pics` directory (if it does not exist) with all the extracted environments, except `tikzpicture`, converted to image formats (`pdf`) in individual files, an output file ⟨*file-out.tex*⟩ with all the extracted environments converted to `\includegraphics` and the `filecontents` environments removed, and a single file ⟨*file-in-env-all.tex*⟩

©2013 – 2026 by Pablo González

with *only* the extracted environments, using lualatex and the preview package to process ⟨*file-in.tex*⟩ and ⟨*file-out.tex*⟩.

8 The way of arara

By design, **ltximg** only runs *"one or more compilations"* of the ⟨*input file*⟩, but sometimes the ⟨*input file*⟩ needs to be processed in a specific way or with something other than the LaTeX, XeLaTeX, pdfLaTeX or LuaLaTeX engine. This is where arara[19] comes in: this *"great little tool"* can take 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 each "rule" to clean temporary files.
3. Avoid using `: clean: { extensions: [...] }`.
4. Don't set `-jobname` and `-output-directory` in any "rule".

When the `--arara` option is passed to the script, the line executed by the system is:

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

If you have several "rules" in the file, they will all be executed. To avoid this, add:

```
% arara: halt
```

after the last "rule" 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:

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

and run:

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

This creates a `./images` directory (if it does not exist) with all the extracted environments converted to image format (pdf) in individual files, an output file ⟨*file-out.tex*⟩ with all the extracted environments converted to `\includegraphics`, and a single file ⟨*test-tkz-all.tex*⟩ with only the extracted environments, using the preview package and lualatex *"twice"* to 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:

```
%<*remove>
% arara: lualatex: { options: [-recorder] }
% arara: lualatex: { options: [-recorder] }
% arara: halt
% ltximg: options: { arara, output = file-out, prefix = tkz}
%</remove>
% arara: xelatex: { options: [-recorder] }
% arara: xelatex: { options: [-recorder] }
```

The content between `%<*remove> ... %</remove>` is removed from the output file before compiling. Thus, the output file ⟨*file-out.tex*⟩ will be compiled using `xelatex` *"twice"*.

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 engine used 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 "rules" are used.

9 Note for `dvisvgm` users

By design, the image format svg is created using `pdftocairo` over the generated pdf file, but if you want good svg files that preserve the *typographic* fonts, it is best to use `dvisvgm`[9]. 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. Run the script using `--subenv` and `--norun` to generate the ⟨*standalone*⟩ files, move to `./images`, generate the `.dvi` or `.xdv` files and then run:

[9] https://ctan.org/pkg/dvisvgm

©2013 – 2026 by Pablo González

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

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

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

10 Example using `latexmk`

If you use `latexmk`, another great utility that automates the compilation process, keep in mind that it will only run on the ⟨*output file*⟩. Consider the following example adapted from How to get tikzmark to work and Draw an aircraft with Tikz to generate images in `svg`, `png` and `pdf` format from the `picture` environment using `lualatex` and `latexmk`.

```
%<*remove>
% ltximg: extrenv: {picture}
% ltximg: skipenv: {tikzpicture}
%</remove>
\documentclass{article}
\usepackage{tikz}
\usetikzlibrary{calc,tikzmark}
\setlength{\parindent}{0pt}
\begin{document}
\section{How to get Tikzmark to work}
By taking logarithms of both sides:
\[
  t = \frac{30\cdot\ln(3/22)}{\ln(15/22)}
  \tikzmark{calculator}\approx\tikzmark{otherside}
    156
\]
\begin{tikzpicture}[overlay,remember picture]
  \coordinate (target) at ($(pic cs:calculator)!1/2!(pic cs:otherside) - (0,.5ex)$);
  \draw[arrows=->] (target) ++(0,-2ex) node [anchor=north] {use calculator} -- (target);
\end{tikzpicture}
\section{Draw an aircraft with Tikz}
The best airplane ever drawn by David Carlise. No TikZ used, just the
classic and perhaps forgotten \verb|\begin{picture} ... \end{picture}|.

\begin{picture}(200,100)
  \put(30,40){\line(1,0){150}}  \put(30,40){\line(0,1){60}}
  \put(30,100){\line(1,0){20}}  \put(50,100){\line(1,-4){10}}
  \put(60,60){\line(1,0){100}}  \put(160,60){\line(1,-1){20}}
  \put(100,50){\line(0,-1){80}} \put(130,50){\line(0,-1){80}}
  \put(100,-30){\line(1,0){30}} \put(100,61){\line(0,1){49}}
  \put(130,61){\line(0,1){49}}  \put(100,110){\line(1,0){30}}
\end{picture}
\end{document}
```

We now run:

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

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

©2013 – 2026 by Pablo González

11 Final notes

The process and operations required to generate the various types of ⟨*image formats*⟩ or ⟨*standalone*⟩ files have been described throughout this documentation, but, as discussed in section 8, sometimes the requirements are a *little different*.

This is the best way to extend the capabilities of `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 *"good documentation"* is the great mission (which has not ended yet).

12 Change history

The most recent public release 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 welcome, 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 `ltximg`, along with the versions, both development (devp) and public (ctan).

v2.2 (ctan), 2026-11-27
- External commands are executed without the shell.
- The `shell` option is ignored in the input file.
- Validate `prefix`, `imgdir`, `output` and `myverb`.
- Use `-dSAFER` in `ghostscript`.
- `xdvipdfmx` uses `dvipdfmx-unsafe.cfg` with `pstricks`.
- Use `noprew` with `\DocumentMetadata`.
- Write `.xbb` files with `extractbb` for `dvipdf`.
- Fix `margins` option and file names with special chars.
- Fix the exit status and the cleaning of temporary files.
- Fix `latexmk` combined with other engine options.
- Fix the ⟨*output file*⟩ with `dvips` and `dvipdf`.
- Add regression tests and CI on Linux and Windows.

v2.1 (ctan), 2021-05-21
- Maintenance update.
- Fix internal `hash` for regex.

v2.0 (ctan), 2021-01-24
- Add `-dALLOWPSTRANSPARENCY` to `ps2pdf`.
- The `--xetex` option now uses `xelatex` and then `xdvipdfmx`.
- Fix module detection under TEX 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 `--dvilua` 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 `--dvipdfm(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.

©2013 – 2026 by Pablo González

13 References

[1] Kastrup, David. "The preview package for LaTeX". 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 TeX". 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 LuaLaTeX with PSTricks". Available from ctan, https://www.ctan.org/pkg/auto-pst-pdf-lua, 2018.

[8] Voß, Herbert. "pst-exa - Typeset PSTricks examples, with pdfTeX". Available from ctan, https://www.ctan.org/pkg/pst-exa, 2017.

[9] Voß, Herbert. "pst2pdf - A script to compile PSTricks documents via pdfTeX". Available from ctan, https://www.ctan.org/pkg/pst2pdf, 2017.

[10] The LaTeX3 Project. "graphics – Enhanced support for graphics". Available from ctan, https://www.ctan.org/pkg/graphicx, 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 LaTeX". 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 LaTeX". Available from ctan, https://www.ctan.org/pkg/minted, 2017.

[19] The Island of TeX. "arara - The cool TeX 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.

©2013 – 2026 by Pablo González

14 Index of Documentation

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

A

article (class) . . . 2
auto-pst-pdf (package) . . . 6
auto-pst-pdf-lua (package) . . . 6

B

book (class) . . . 2

C

cachepic (package) . . . 1
Compiler
- arara . . . 10
- dvilualatex . . . 10
- dvipdfmx . . . 10
- dvips . . . 10, 11
- extractbb . . . 10
- latex . . . 10, 11
- lualatex . . . 10, 12, 13
- pdflatex . . . 8, 10, 11
- xdvipdfmx . . . 10
- xelatex-unsafe . . . 10
- xelatex . . . 10, 12

Compiler options
- -enable-write18 . . . 5
- -no-pdf . . . 10
- -no-shell-escape . . . 5
- -recorder . . . 5
- -shell-escape . . . 5

D

\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

\DocumentMetadata . . . 9

E

Environments support by default
- PSTexample . . . 6
- nopreview . . . 7
- pgfpicture . . . 7
- postscript . . . 6
- preview . . . 6
- psgraph . . . 7
- pspicture . . . 7
- tikzpicture . . . 7

Environments
- PSTexample . . . 7, 10
- document . . . 10, 11
- nopreview . . . 7, 10
- pgfinterruptpicture . . . 7
- picture . . . 13
- postscript . . . 7
- 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
- programsc . . . 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
- extikzpicture . . . 4
- filecontentsdefmacro . . . 4
- filecontentsdefstarred . . . 4
- filecontentsdef . . . 4
- filecontentsgdefmacro . . . 4
- filecontentsgdef . . . 4
- filecontentshere . . . 4
- filecontents . . . 3, 4, 11
- scontents . . . 4
- tcbexternal . . . 4
- tcboutputlisting . . . 4
- tcbwritetmp . . . 4
- verbatimwrite . . . 4

extract (package) . . . 1

F

fancyvrb (package) . . . 3
File
- ltximg.log . . . 8

©2013 – 2026 by Pablo González

File extensions
.bmp . . . 8
.dvi . . . 12, 13
.eps . . . 9
.fls . . . 5
.jpg . . . 8
.log . . . 5, 8
.ltx . . . 8
.pdf . . . 10
.png . . . 8
.ppm . . . 9
.svg . . . 9
.tar.gz . . . 9
.tex . . . 8
.tif . . . 8
.xbb . . . 10
.xdv . . . 10, 12, 13
.zip . . . 9
\fverb . . . 3

G
\graphicspath . . . 5
graphicx (package) . . . 2, 5, 10
grfext (package) . . . 2, 5

I
Image formats
bmp . . . 8
eps . . . 7
jpg . . . 8
pdf . . . 4, 7, 8, 11–13
png . . . 8, 13
ppm . . . 8
svg . . . 8, 11–13
tiff . . . 8
\include . . . 2
\includecomment . . . 3
\includegraphics . . . 1, 5, 9, 11–13
\input . . . 2

L
letter (class) . . . 2
listings (package) . . . 3
\lstinline . . . 3
\lstMakeShortInline . . . 3
\lstnewenvironment . . . 3

M
\MakeSpecialShortVerb . . . 3
\mint . . . 3
minted (package) . . . 3, 5
\mintinline . . . 3

N
\newenvsc . . . 4
\NewListingEnvironment . . . 3
\newmint . . . 3
\newminted . . . 3
\newmintinline . . . 3
\newpage . . . 5
\NewProgram . . . 3
\newtabverbatim . . . 3
\newtcbexternalizeenvironment . . . 4
\newtcbexternalizetcolorbox . . . 4
\NewTCBListing . . . 3
\newtcblisting . . . 3
\newverbatim . . . 3

O
Operating system
Linux . . . 1, 2
Windows . . . 2
ltximg options in command line
--arara . . . 10, 12
--bmp . . . 8
--clean . . . 5, 10
--deltenv . . . 11
--dpi . . . 8
--dvilua . . . 10
--dvipdf . . . 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 . . . 10, 12
--nopdf . . . 9
--noprew . . . 5–7, 9, 12
--norun . . . 5, 9, 12, 13
--output . . . 9
--png . . . 8
--ppm . . . 9
--prefix . . . 9
--runs . . . 9
--shell . . . 5, 9, 11
--skipenv . . . 6, 10
--srcenv . . . 6, 9
--subenv . . . 6, 9, 12
--svg . . . 9
--tar . . . 9
--tif . . . 8
--verbenv . . . 10
--verbose . . . 8
--version . . . 8
--writenv . . . 11
--xetex . . . 6, 10
--zip . . . 9
ltximg options in input file
deltenv . . . 11
extrenv . . . 11
options . . . 11
skipenv . . . 11
verbenv . . . 11
writenv . . . 11

P
Package options
swpl . . . 6
tcb . . . 6
Packages
PSTricks . . . 1, 7, 10
TikZ . . . 1, 7
auto-pst-pdf-lua . . . 6

©2013 – 2026 by Pablo González

auto-pst-pdf 6
cachepic 1
doc 3
extract 1
fancyvrb 3
graphicx 2, 5, 10
grfext 2, 5
listings 3
minted 3, 5
pgf 7
preview 1, 2, 5–9, 11–13
pst-exa 6
pst-pdf 2, 6
pst-plot 7
pstricks 10
shortvrb 3
xcomment 1
\pagestyle 2
Perl module
Archive::Tar 2
Config 2
Cwd 2
Env 2
File::Basename 2
File::Copy 2
File::Find 2
File::Path 2
File::Spec::Functions 2
File::Temp 2
FileHandle 2
Getopt::Long 2
IO::Compress::Zip 2
Module::Load::Conditional 2
POSIX 2
Term::ANSIColor 2
Win32::Console::ANSI 2
Win32::TieRegistry 2
Win32 2
autodie 2
pgf (package) 7
\PrependGraphicsExtensions* 5
preview (package) 1, 2, 5–9, 11–13
Programs
Windows Terminal 2
arara 12
chktex 2
cmd 2
cygwin 2
dvisvgm 12, 13
ghostscript 1, 2, 7–10
pdftocairo 8, 9, 12
pdftoppm 8, 9
pdftops 7, 9
perl 1, 2
poppler-utils 1, 2, 7
\ProvideTCBListing 3
pst-exa (package) 6
pst-pdf (package) 2, 6
pst-plot (package) 7
PSTricks (package) 1, 7, 10
pstricks (package) 10
\pyginline 3
\pygment 3

Q

\qverb 3

R

\renewtcbexternalizeenvironment 4
\renewtcbexternalizetcolorbox 4
report (class) 2

S

\Scontents 3
Scripts
latexindent 2
latexmk 10, 13
latexpand 2
pdfcrop 2, 9, 10, 12
ps2pdf 10, 11
pst2pdf 1, 2
shortvrb (package) 3
\specialcomment 3
\spverb 3
swpl (package option) 6

T

tcb (package option) 6
\tcboxverb 3
\thispagestyle 2
TikZ (package) 1, 7

V

\Verb 3
\verb 3

W

\write18 9

X

xcomment (package) 1

©2013 – 2026 by Pablo González