pst2pdf

Running a PSTricks document with (pdf/xe/lua)latex; v0.19 — 2020-08-17*

Herbert Voß
Pablo González L

pst2pdf is a Perl *script* which isolates all PostScript or PSTricks related parts of the TEX document, read all postscript, pspicture, psgraph and PSTexample environments and, extract source code in *standalone* files and converting them into image format pdf, eps, jpg, svg or png (default pdf). Create new file with all extracted environments converted to \includegraphics and runs (pdf/Xe/lua)latex.

Contents

* This file describes a documentation for version 0.19, last revised 2020-08-17.

1 Introduction

PSTricks as PostScript related package uses the programming language PostScript for internal calculations. This is an important advantage, because floating point arithmetic is no problem. Nearly all mathematical calculation can be done when running the DVI-file with GHOSTSCRIPT. However, creating a PDF file in a direct way with pdflatex is not possible. pdflatex cannot understand the PostScript related stuff.

Instead of running pdflatex one can use the *script* pst2pdf, it extracts all PSTricks related code into single documents with the same preamble as the original main document.

The pst2pdf *script* runs document, extract source code for all PSTricks as PostScript related parts, clips all whitespace around the image and creates a .pdf images of the PSTricks related code.

In a last run which is the pdflatex the PSTricks code in the main document is replaced by the created images.

2 License

3 Requirements for operation

For the complete operation of pst2pdf you need to have a modern TEX distribution such as TEX Live or MiKTEX, the packages preview[3], pst-pdf[4], graphicx[6] and grfext[5], have a version equal to or greater than 5.28 of perl, a version equal to or greater than 9.24 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 distribution of TEX Live 2020 for Windows includes pst2pdf and all requirements, MiKTEX 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 TEX distribution.

The script has been tested on Windows (v10) and Linux (fedora 32) using Ghostscript v9.52, poppler-utils v0.84, perl v5.30 and the standard classes offers by LATEX: book, report, article and letter.

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 `latexpand`[1] 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`[2] and `latexindent`[3], 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:

```
% Part One: Everything before \documentclass
\documentclass{article}
% 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 an environment `filecontens` 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.

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 "TEX world". 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`, `fancyvrb` and `listings` 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:

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

1 https://www.ctan.org/pkg/latexpand
2 https://www.ctan.org/pkg/chktex
3 https://www.ctan.org/pkg/latexindent

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` or `doc` 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*⟩:

```
\UndefineShortVerb{\|}
\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 8.2.

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:

- `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`

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

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

If any of the ⟨*verbatim standard*⟩ environments is not supported by default or can not detected, you can use the options described in 8.2.

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`
- `extikzpicture`
- `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 8.2.

4.3 Steps process

For creation of the image formats, extraction of source code of environments and creation of an ⟨*output file*⟩, `pst2pdf` need a various steps. Let's assume that the ⟨*input file*⟩ is `test.tex`, ⟨*output file*⟩ is `test-pdf.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`:

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

Validating Options

The first step is read and validated `[`⟨*options*⟩`]` from the command line, verifying that `test.tex` contains *some* environment to extract, 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 standalone 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 croping, 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 and. The `--shell` option activates `-shell-escape` for compatibility with packages such as `minted` or others.

Create output file

In the fifth step, the script creates the output file `test-pdf.tex` converting all extracted code to `\includegraphics`, remove all `PSTricks` packages and content betwen `%CleanPST ... %CleanPST`, then adding the following lines at end of preamble:

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

The script will try to detect whether the graphicx package and the \graphicspath command are in the preamble of the ⟨*output file*⟩. If it is not possible to find it, it will read the .log file generated by the temporary file. Once the detection is complete, the package grfext and \PrependGraphicsExtensions* will be added at the end of the preamble, then proceed to run:

```
[user@machine ~:]$ ⟨compiler⟩ -recorder -shell-escape test-pdf.tex
```

generating the file test-pdf.pdf.

Clean temporary files and dirs

In the sixth step, the script read the files test-fig-1981.fls and test-out.fls, 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-pdf.tex looks like this:

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

5 Default extracted environments

pst2pdf support fourth environments for extraction. Internally the script converts all environments to extract in preview environments. Is better comment this package in preamble unless the option --noprew is used.

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

Environment provide by pst-pdf[4], auto-pst-pdf[7] and auto-pst-pdf-lua[13] packages. Since the pst-pdf, auto-pst-pdf and auto-pst-pdf-lua packages internally use the preview package, is better comment this in preamble. Only the *content* of this environment is extracted and *"not"* the environment itself when using the --srcenv option. The postscript environment should always be used, when there is some code before a pspicture environment or for some code which is not inside of a pspicture environment.

\begin{**pspicture**}
⟨*env content*⟩
\end{**pspicture**}

Environment provide by pstricks[15] package. The plain TEX syntax \pspicture ... \endpspicture its converted to LATEX syntax \begin{pspicture} ... \end{pspicture} if not within the PSTexample or postscript environments.

\begin{**psgraph**}
⟨*env content*⟩
\end{**psgraph**}

Environment provide by pst-plot[16] package. The plain TEX syntax \psgraph ... \endpsgraph its converted to LATEX syntax \begin{psgraph} ... \end{psgraph} if not within the PSTexample or postscript environments.

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

Environment provide by 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 [swpl] or [tcb] option.

6 Remove PSTricks code

By design, the script remove all PSTricks packages in preamble of ⟨*output file*⟩, if you need delete other PSTricks code in preamble use:

%CleanPST ⟨*code*⟩ %CleanPST

All content betwen %CleanPST ... %CleanPST are deleted in preamble of the ⟨*output file*⟩. This lines can *not* be nested and should be at the beginning of the line and in separate lines.

7 Supported image formats

The ⟨*image formats*⟩ generated by the pst2pdf 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 -dNOSAFER -sDEVICE=pdfwrite -dPDFSETTINGS=/prepress
```

eps The image format generated using pdftoeps. 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 -dNOSAFER -sDEVICE=pngalpha -r150
```

jpg The image format generated using Ghostscript. The line executed by the system is:

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

8 How to use

8.1 Syntax

The syntax for pst2pdf is simple, if your use the version provided in your TEX distribution:

```
[user@machine ~:]$ pst2pdf [⟨options⟩] ⟨input file⟩
```

or

```
[user@machine ~:]$ pst2pdf ⟨input file⟩ [⟨options⟩]
```

If the development version is used:

```
[user@machine ~:]$ perl pst2pdf [⟨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 `latex»dvips»ps2pdf` and `preview` package for process ⟨*input file*⟩ and `pdflatex` for compiler ⟨*output file*⟩.

8.2 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*⟩. Some short options can be bundling.

`-h,--help` ⟨*bolean*⟩ (default: off)

Display a command line help and exit.

`-l,--log` ⟨*bolean*⟩ (default: off)

Write a `pst2pdf.log` file with all process information.

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

Display the current version (0.19) and exit.

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

Show verbose information of process in terminal.

`-d,--dpi` ⟨*integer*⟩ (default: 150)

Dots per inch for images files. Values are positive integers less than or equal to 2500.

`-t,--tif` ⟨*bolean*⟩ (default: off)

Create a `.tif` images files using `Ghostscript`.

`-b,--bmp` ⟨*bolean*⟩ (default: off)

Create a `.bmp` images files using `Ghostscript`.

`-j,--jpg` ⟨*bolean*⟩ (default: off)

Create a `.jpg` images files using `Ghostscript`.

`-p,--png` ⟨*bolean*⟩ (default: off)

Create a `.png` transparent image files using `Ghostscript`.

`-e,--eps` ⟨*bolean*⟩ (default: off)

Create a `.eps` image files using `pdftops`.

`-s,--svg` ⟨*bolean*⟩ (default: off)

Create a `.svg` image files using `pdftocairo`.

`-P,--ppm` ⟨*bolean*⟩ (default: off)

Create a `.ppm` image files using `pdftoppm`.

`-g,--gray` ⟨*bolean*⟩ (default: off)

Create a gray scale for all images using `Ghostscript`. The line behind this options is:

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

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

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

`-np,--noprew` ⟨*bolean*⟩ (default: off)

Create images files without `preview` package. The `\begin{preview}`...`\end{preview}` lines are only used as delimiters for extracting the content *without* using the package `preview`. Using this option *"only"* the extracted environments are processed and not the whole ⟨*input file*⟩, sometimes it is better to use it together with `--force`. Alternative name `--single`.

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

Set margins in bp for `pdfcrop`.

`-r,--runs` ⟨`1|2|3`⟩ (default: 1)

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

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

Set custom verbatim command `\myverb`. Just pass the ⟨*macro name*⟩ *without* "`\`".

`--ignore` ⟨*environment name*⟩ (default: empty)

Add a verbatim environment to internal list.

`--imgdir` ⟨*string*⟩ (default: images)

Set the name of directory for save generated files. Only the ⟨*name*⟩ of directory must be passed *without* relative or absolute paths.

`--srcenv` ⟨*bolean*⟩ (default: off)

Create separate files with *"only code"* for all extracted environments.

`--shell` ⟨*bolean*⟩ (default: off)

Enable `\write18`⟨*shell command*⟩.

`-ni,--norun` ⟨*bolean*⟩ (default: off)

Execute the script, but do not create image files. This option is designed to to generate standalone files or used in conjunction with `--srcenv` and to debug the ⟨*output file*⟩. Alternative name `--noimages`.

`--nopdf` ⟨*bolean*⟩ (default: off)

Don't create a `.pdf` image files.

`--nocrop` ⟨*bolean*⟩ (default: off)

Don't run `pdfcrop` in image files.

`-ns,--nosource` ⟨*bolean*⟩ (default: off)

Don't create standalone files.

`-x,--xetex` ⟨*bolean*⟩ (default: off)

Using `xelatex` compiler ⟨*input file*⟩ and ⟨*output file*⟩.

`--luatex` ⟨*bolean*⟩ (default: off)

Using `dvilualatex»dvips»ps2pdf` for compiler ⟨*input file*⟩ and `lualatex` for ⟨*output file*⟩.

`--arara` ⟨*bolean*⟩ (default: off)

Use `arara`[4] tool for compiler ⟨*output file*⟩. This option is designed to full process ⟨*output file*⟩, is mutually exclusive with `--latexmk` option.

4 https://ctan.org/pkg/arara

`--latexmk` ⟨*bolean*⟩ (default: off)

Using `latexmk`[5] for process ⟨*output file*⟩. This option is designed to full process ⟨*output file*⟩, is mutually exclusive with `--arara`.

`--zip` ⟨*bolean*⟩ (default: off)

Compress the files generated by the script in `./images` in `.zip` format. Does not include ⟨*output file*⟩.

`--tar` ⟨*bolean*⟩ (default: off)

Compress the files generated by the script in `./images` in `.tar.gz` format. Does not include ⟨*output file*⟩.

`--bibtex` ⟨*bolean*⟩ (default: off)

Run `bibtex` on the `.aux` file (if exists), is mutually exclusive with `--biber` option.

`--biber` ⟨*bolean*⟩ (default: off)

Run `biber` on the `.bcf` file (if exists), is mutually exclusive with `--bibtex` option.

8.3 Example of usage

An example of usage from command line:

```
[user@machine ~:]$ pst2pdf --luatex -e -p -j --imgdir pics test.ltx
```

Create a `./pics` directory (if it does not exist) with all extracted environments converted to image formats (`.pdf`, `.eps`, `.png`, `.jpg`) in individual files, an stanalone files (`.ltx`) for all environments extracted, an output file `test-pdf.ltx` with all extracted environments converted to `\includegraphics` and a single file `test-fig-all.ltx` with only the extracted environments using `dvilualatex»dvips»ps2pdf` and `preview` package for for process `test.ltx` and `lualatex` for `test-pdf.ltx`.

9 Working in another way

By design, the script generates separate images and files following a predetermined routine that has already been described in this documentation. Another way to generate images is as follows:

1. Execute the script using `--norun` to generate ⟨*standalone*⟩ files, move to `./images` and generate `.pdf` files runing:

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

2. Execute the script using `--norun`, move to `./images` `.pdf` file runing:

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

10 Example files

The `pst2pdf` documentation provides three example files `test1.tex`, `test2.tex` and `test3.tex` plus an image file `tux.jpg` to test and view the script in action. Copy these files to a directory you have write access to and execute:

```
[user@machine~:]$ pst2pdf [⟨options⟩] test1.tex
```

To see how this works.

5 https://www.ctan.org/pkg/latexmk

References

[1] Denis Girou. “Présentation de PSTricks”. In: *Cahier GUTenberg* 16 (Apr. 1994), pp. 21–70.

[2] Michel Goosens et al. *The LaTeX Graphics Companion*. 2nd ed. Reading, Mass.: Addison-Wesley Publishing Company, 2007.

[3] David Kastrup. *The `preview` package for LaTeX*. https://www.ctan.org/pkg/preview: CTAN, 2017.

[4] Rolf Niepraschk. *The `pst-pdf` package*. https://www.ctan.org/pkg/pst-pdf: CTAN, 2019.

[5] Heiko Oberdiek. *The `grfext` package*. https://www.ctan.org/pkg/grfext: CTAN, 2017.

[6] The LaTeX3 Project. *`graphics` - Enhanced support for graphics*. https://www.ctan.org/pkg/graphicx: CTAN, 2017.

[7] Will Robertson. *The `auto-pst-pdf` package*. https://www.ctan.org/pkg/auto-pst-pdf: CTAN, 2009.

[8] Herbert Voß. *`pst-exa` - Typeset `PSTricks` examples, with pdfTeX*. https://www.ctan.org/pkg/pst-exa: CTAN, 2017.

[9] Herbert Voß. *`pst-tools` – Helper functions*. CTAN:/graphics/pstricks/contrib/pst-tools: CTAN, 2012.

[10] Herbert Voß. *`PSTricks` – Grafik für TeX und LaTeX*. 7th ed. Heidelberg/Berlin: DANTE – Lehmanns, 2010.

[11] Herbert Voß. *PSTricks – Graphics for TeX and LaTeX*. Cambridge: UIT, 2011.

[12] Herbert Voß. *LaTeX quick reference*. Cambridge: UIT, 2012.

[13] Herbert Voß. *The `auto-pst-pdf-lua` package - Using LuaLaTeX with `PSTricks`*. https://www.ctan.org/pkg/auto-pst-pdf-lua: CTAN, 2018.

[14] Timothy van Zandt. *PSTricks - `PostScript` macros for generic TeX*. http://www.tug.org/application/PSTricks, 1993.

[15] Timothy van Zandt and Denis Girou. “Inside PSTricks”. In: *TUGboat* 15 (Sept. 1994), pp. 239–246.

[16] Timothy van Zandt and Herbert Voß. *`pst-plot` - Plot data using `PSTricks`*. https://www.ctan.org/pkg/pst-plot: CTAN, 2019.

Index