Documentation for the Package `tobiShell.sty`

Tobias Nähring

December 3, 2007

Contents

1 Introduction

The package `tobiShell.sty`[1] allows to include script files for external programs within LaTeX documents. Inlined code is written to temporary files and the external program is called with the file name of the script as the last command line argument.
The default external program is the `bash`-shell. You may easily change the default program (e.g. to gnuplot) or specify some other executable for some given inline script.
The package has been written and tested for `Linux`. Maybe, with some modifications it runs also under other operating systems.

2 Installation

Copy `tobiShell.sty` to some place where LaTeX finds it. A good location is your personal `texmf` tree (e.g. `~/texmf/tex/latex/tobiShell.sty`).
For security reasons TeX does not run external programs from input files by default. This behavior can be changed by setting the variable `shell_escape` in `/etc/texmf/texmf.cnf` to `t`. You should not do so because of it is a security risk! It enables also mailware possibly hidden in some down-loaded TeX document.
It is better to enable the execution of external commands with the help of the TeX command line option `--shell-escape` (see section 3) only for safe LaTeX input files (i.e. input files you have written or at least inspected by yourself).

3 Usage

3.1 Most simple usage – the `bash`-shell

The most simple application of the package is to embed a `bash` shell script into your LaTeX document. Just write the shell script between the command sequences `\bShell` and `\eShell` in your LaTeX-source. When comiling the LaTeX-source pass the command line option `--shell-escape` to LaTeX to enable the execution of external commands. (Once more the advice to apply this command line option only with trustworthy LaTeX-sources.)

[1]The prefix `tobi` has been chosen to make the package name unique. There are just too many files named `shell.tex` in the net.

LaTeX will save the lines between `\bShell` and `\eShell` to a file named `shEsc.tmp` (by default) and send it to the shell.

`Bash` script example For an example let us write the current directory contents into some file `ls.output` from within a LaTeX-document. The code for this end is just:

```
\bShell
ls > ls.output
\eShell
```

You can easily include the output of the command into your LaTeX-source via

```
\input{./ls.output}
```

On my computer this results to: `compilabletext.txt defaults.gp errfct.eps gnuplot.tmp ls.output Makefile mex.pstex shEsc.tmp sinc.eps test.tex tmp.tex tobiShell1.pdf tobiShell.aux tobiShell.dvi tobiShell.log tobiShell.orig.tex tobiShell.pdf tobiShell.pdf2 tobiShell.ps tobiShell.sty tobiShell.tex tobiShell.toc`
If something goes wrong (e.g. you didn't issue the `--shell-escape` command line option) LaTeX will complain about the missing input file `ls.output`. This is fine if you take it as an error message. If you prefer LaTeX to run smoothly and to embed an error message into the LaTeX-document you can use the native LaTeX-macro `\IfFileExists` as demonstrated in the following example.

```
\IfFileExists{./ls.output}
 {%%% then branch %%%
  \catcode`\_=11 % underscores occur often in file names
  \input{./ls.output}
}{%%% else branch %%%
  The file \texttt{ls.output} has not been generated.
  Maybe you did not pass the command line option
  \texttt{--shell-escape} to \LaTeX.
}
```

3.2 Running other commands

Other commands than the `bash` shell can be used. For an example, we will apply the Linux command `sort` to some lines of text. The command to be used can be passed to `\bShell` by an optional argument in brackets. This argument is then completed by the name of the temporary script file (by default `shEsc.tmp`) containing the text between `\bShell[...]` and `\eShell`.

```
\bShell[sort -o tmp.tex]
the
quick
brown
fox
jumps
over
a
lazy
dog
\eShell
```

Below the shell script the outcome of the command can be included into the LaTeX-document via:

```
\input{./tmp.tex}
```

The result is in this example[2]:

[2]The `\obeylines` macro has been used to preserve the newlines.

a
brown
dog
fox
jumps
lazy
over
quick
the
Some programs produce TEX-output that can directly be read in by LATEX. As an example we use Gnu-Pari to derive the power series of $\sin(x)/x$:

```
\bShell[gp -q -f <]
system("rm -f pari.tex");
writetex("pari.tex",precision(sin(x)/x,10));
quit;
\eShell
\begin{equation}
\input{./pari.tex}
\end{equation}
```

and get

The file `pari.tex` has not been generated. Perhaps, the Gnu-Pari executable `gp` is not installed. (1)

Note, the character '`<`' in the optional argument to `\bShell` in this example. The meaning of this character becomes clear if one considers the completed command line:

```
gp -q -f < shEsc.tmp
```

It just redirects the content of the file `shEsc.tmp` to the standard input of Gnu-pari.

3.3 Changing the defaults

Assume you want to include a lot of graphical function plots into your document and you always use `gnuplot` for producing such plots.
Then it makes sense to change the default command from `bash` to `gnuplot` with the help of

```
\def\tobiShellCommand{gnuplot}
```

or

```
\renewcommand\tobiShellCommand{gnuplot}
```

(if you prefer the LATEX-style). Analogously, you can change the name of the generated shell script file from `shEsc.tmp` to whatever you like by

```
\def\tobiShellFileName{SomeFileName.SomeFileNameExtension}
```

For an example, you can set

```
\def\tobiShellFileName{gnuplot.tmp}
```

better indicating that this temporary file is used for gnuplot. After those definitions you can use embedded `gnuplot` scripts like the following to produce function plots.

```
\bShell
set terminal postscript eps
set output "sinc.eps"
plot [x=0:2*pi] sin(x)/x
\eShell
```

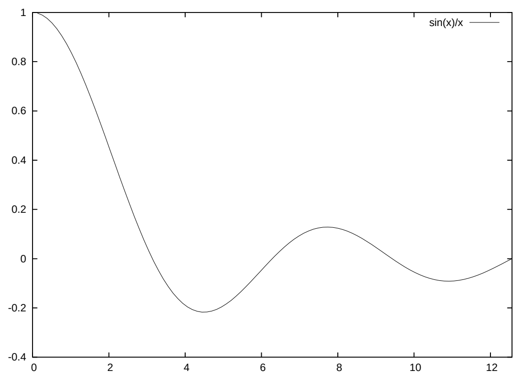

Figure 1: The function plot generated by `gnuplot` via an embedded script.

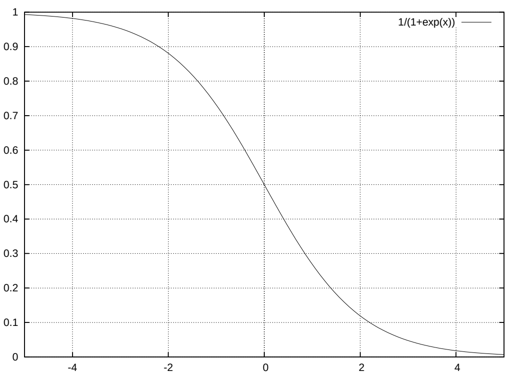

Figure 2: Function plot generated by `gnuplot` with default settings.

Below this script you may include the produced eps picture into some figure environment. For the current example the result is shown in Figure 1.
Sometimes you want to use the same fundamental `gnuplot` settings for a number of plots. Those settings can easily be included into the LaTeX-file with the help of the LaTeX-environment `\begin{filecontents*}...\end{filecontents*}` which must be used before the `\documentclass` statement. In the following example the default settings are written into a file named `./defaults.gp`

```
\begin{filecontents*}{./defaults.gp}
set style data lines
set grid
set zeroaxis xy
set terminal postscript eps
\end{filecontents*}
```

The file `defaults.gp` is then included into the gnuplot scripts via the gnuplot command `load` as shown in the following example:

```
\bShell
load "defaults.gp"
set output "errfct.eps"
plot [x=-5:5] 1/(1+exp(x))
\eShell
```

The resulting plot is represented in Figure 2.

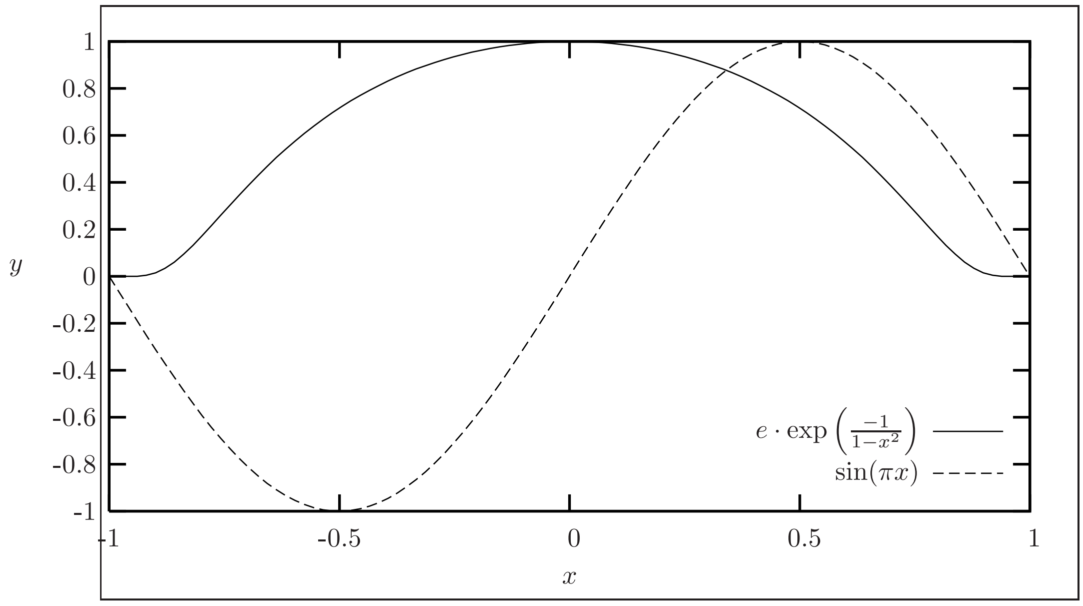

Figure 3: Using the `pslatex` driver of `gnuplot`

In the last example with the following inline script the `pslatex` driver of `gnuplot` is used which allows labeling of curves with LaTeX text (see Figure 3).

```
\bShell[gnuplot]
set term pslatex norotate
set output "mex.pstex"
f(x)=exp(-1/(1-x*x))
set lmargin 0
set xlabel "$x$"
set ylabel "$y$"
set key bottom spacing 2
plot [x=-1:1] f(x)/f(0) title \
"$e\\cdot\\exp\\left(\\frac{-1}{1-x^{2}}\\right)$", \
sin(pi*x) title "$\\sin(\\pi x)$
\eShell%
```