# The bashful Package*

Yossi Gil†
Department of Computer Science
The Technion—Israel Institute of Technology
Technion City, Haifa 32000, Israel

2011/02/28‡

**Abstract**

*It is sometimes useful to "*escape-to-shell*" from within LaTeX. The most obvious application is when the document explains something about the working of a computer program. Your text would be more robust to changes, and easier to write, if all the examples it gives, are run directly from within LaTeX.*

*To facilitate this and other applications, package* bashful *provides a convenient interface to TeX's primitive* `\write18`*—the execution of shell commands from within your input files, also known as* shell escape*. Text between* `\bash` *and* `\END` *is executed by the* `bash`*, a popular Unix command line interpreter. Various flags control whether the executed commands and their output show up in the printed document, and whether they are saved to files.*

*Although provisions are made for using shells other than* `bash`*, this package may* not *operate without modifications on Microsoft's operating systems.*



†`mailto:yogi@cs.Technion.ac.IL`

‡This document describes bashful V 0.91.

# Contents



# 1 Introduction

At the time I run this document through LaTeX, the temperature in Jerusalem, Israel, was *14 °C*, while the weather condition was *cloudy*.

You may not care so much about these bits of truly ephemeral information, but you may be surprised that they were produced by the very process of LaTeXing.

How did I do that? Well, the first step was to write a series of shell commands that retrieve the current temperature, and another such series to obtain the current weather conditions. This task required connection to Google's weather service and minimal dexterity with Unix pipes and filters to process the output. My command series to obtain the current temperature was:

```
location=Jerusalem,Israel
server="http://www.Google.com/ig/api"
request="$server?weather=$location"
wget -q -O - $request |\
tr "<>" "\012\012" |\
grep temp_c |\
sed 's/[^0-9]//g'
```

while the weather condition was obtained by

```
location=Jerusalem,Israel
server="http://www.Google.com/ig/api"
request="$server?weather=$location"
wget -q -O - $request |\
tr "<>" "\012\012" |\
grep "condition data" |\
head -n 1 |\
sed -e 's/^.*="//' -e 's/"\/*//' |\
tr 'A-Z' 'a-z'
```

The second step was coercing LaTeX to run these commands while processing this file. To do that, I wrapped these two series within a `\bash`...`\END` pair. The `\bash` command, offered by this package, takes all subsequent lines, stopping at the closing `\END`, places these in a file, and then lets the `bash` shell interpreter execute this file.

Allowing LaTeX to run arbitrary shell commands can be dangerous—you never know whether that nice looking `tex` file you received by email was prepared by a friend or a foe. This is the reason that you have to tell LaTeX explicitly that shell escapes are allowed. The `-shell-esc` does that. Processing this file was thus done by executing, at the command line,

```
% latex -shell-escape bashful.tex
```

What I actually wrote in my `.tex` file to produce the temperature in Jerusalem, Israel was:

```
\bash[verbose,scriptFile=temperature.sh,stdoutFile=temperature.tex]
location=Jerusalem,Israel
server="http://www.Google.com/ig/api"
request="$server?weather=$location"
wget -q -O - $request |\
tr "<>" "\012\012" |\
grep temp_c |\
sed 's/[^0-9]//g'
\END
```

The flags passed to above to the `bash` control sequence instructed it:

1. to be verbose, typing out a detailed log of everything it did;
2. to save the shell commands in a script file named `temperature.sh`; and,
3. to store the standard output of the script in a file named `temperature.tex`.

To obtain the current weather condition in the capital I wrote:

```
\bash[verbose,scriptFile=condition.sh,stdoutFile=condition.tex]
location=Jerusalem,Israel
server="http://www.Google.com/ig/api"
request="$server?weather=$location"
wget -q -O - $request |\
tr "<>" "\012\012" |\
grep "condition data" |\
head -n 1 |\
sed -e 's/^.*="//' -e 's/"\/*//' |\
tr 'A-Z' 'a-z'
\END
```

I wrote these two just after my `\begin{document}`. When LaTeX encountered these, it executed the bash commands and created two files `temperature.tex` and `condition.tex`.

Subsequently, I could use the content of these files by writing:

```
At the time I run this document through \LaTeX{},
the temperature in Jerusalem, Israel,
was~\emph{\input{temperature.tex}\celsius},
while the weather condition was \emph{\input{condition}}\unskip.
```

# 2 An Easy to Digest Example

If you were intimidated by technicalities of the above description, let's try another example that might be easier to digest.

I will start by telling a simple story of writing, compiling and executing and a simple program. Then, I will explain how I used the `\bash` command to not only tell a story, but also to play it live: that is, authoring a simple C program, compiling it and executing it, all from within LaTeX.

## 2.1 A "Hello, World" Program

### 2.1.1 Authoring

Let's first write a simple Hello, World! program in the C programming language:

```
% rm -f hello.c; cat << EOF > hello.c
/*
** hello.c: My first C program; it prints
** "Hello, World!", and dies.
*/

#include <stdio.h>

int main()
{
  printf("Hello, World!\n");
  return 0;
}
EOF
```

### 2.1.2 Compiling

Now, let's compile this program:

```
% cc hello.c
```

### 2.1.3 Executing

Finally, we can execute this program, and see that indeed, it prints the "Hello, World!" string.

```
% ./a.out
Hello, World!
```

## 2.2 Behind the Scenes

### 2.2.1 Authoring

What I wrote in the input to produce the `hello.c` program was:

```
\bash[list]
rm -f hello.c; cat << EOF > hello.c
/*
** hello.c: My first C program; it prints
** "Hello, World!", and dies.
*/

#include <stdio.h>
```

```
int main()
{
  printf("Hello, World!\n");
  return 0;
}
EOF
\END
```

In doing so, all the text between the `\bash` and `\END` was sent to a temporary file, which was then sent to the `bash` bash shell for execution. The `list` flag instructed `\bash` to list this file in the main document. This listing was prefixed with `%␣` to make it clear that it was input to `bash`.

### 2.2.2 Compiling

Next, I wrote

```
\bash[list]
cc hello.c
\END
```

As before, in doing that, I achieved two objectives: first, when LaTeX processed the input, it also invokes the C compiler to compile file `hello.c`, the file which I just created. Second, thanks to the `list` flag, the command for compiling this program was included in the printed version of this document. Thanks to the `stdout` option, plain messages, i.e., not error messages, produced by the compiler would show on the printed version of this document. In this case, no such messages were produced.

### 2.2.3 Executing

Finally, I wrote

```
\bash[list,stdout]
./a.out
\END
```

to run the program I just wrote. The `stdout` adds to my listing the output that this execution produces, i.e., the string `Hello, World!` that this execution produces to the standard output.

## 3 Dealing with Shell Command Errors

Using bashful to demonstrate my *Hello, World!* program, made sure that the story I told is accurate: I really did everything I said I did. More accurately, the

`\bash` command acted as my proxy, and did it for me.

Luckily, my `hello.c` program was correct. But, if it was not, the `\bash` command would have detected the error, and would have stopped the LaTeX process, indicating that the compilation did not succeed. More specifically, the `\bash` command

1. collects all commands up to `\END`;
2. places these commands in a script file;
3. executes this script file, redirecting its standard output and its standard error streams to distinct files;
4. checks whether the exit code of the execution indicates an error (i.e., exit code which is different from 0), and if so, appends this exit code to the file containing the standard error;
5. checks whether the file containing the standard error is empty, and if not, pauses execution after displaying an error message;
6. lists, if requested to, the script file; and,
7. lists, if requested to, the file containing the standard output.

Let me demonstrate a situation in which the execution of the script generates an error. To do that, I will write a short LaTeX file, named `minimal.tex` which tries to use `\bash` to compile an incorrect C program. Since `minimal.tex` contains `\END`, I will have to author this file in three steps:

1. Creating the header of `minimal.tex`:

```
% cat << EOF > minimal.tex
\documentclass{minimal}
\usepackage{bashful}
\begin{document}
\bash
echo "main(){return int;}" > error.c
cc error.c
EOF
```

2. Adding `\END` to `minimal.tex`

```
% echo "\\END" >> minimal.tex
```

3. Finalizing `minimal.tex`

```
% echo "\\end{document}" >> minimal.tex
```

Let me now make sure `minimal.tex` was what I expect it to be:

```
% cat minimal.tex
\documentclass{minimal}
\usepackage{bashful}
\begin{document}
\bash
echo "main(){return int;}" > error.c
cc error.c
\END
\end{document}
```

I am now ready to run `minimal.tex` through LaTeX, but since I will not run the `latex` command myself, I will make sure a `q` character is sent to it when the anticipated error occurs.

```
% yes x | latex -shell-esc minimal.tex
This is pdfTeX, Version 3.1415926-1.40.11 (TeX Live 2010)
 \write18 enabled.
entering extended mode
(./minimal.tex
LaTeX2e <2009/09/24>
Babel <v3.8l> and hyphenation patterns for english, dumylang, nohyphenat
rman-x-2009-06-19, ngerman-x-2009-06-19, afrikaans, ancientgreek, ibycus
c, armenian, basque, bulgarian, catalan, pinyin, coptic, croatian, czech
h, dutch, ukenglish, usenglishmax, esperanto, estonian, ethiopic, farsi,
h, french, galician, german, ngerman, swissgerman, monogreek, greek, hun
 icelandic, assamese, bengali, gujarati, hindi, kannada, malayalam, mara
iya, panjabi, tamil, telugu, indonesian, interlingua, irish, italian, ku
 lao, latin, latvian, lithuanian, mongolian, mongolianlmc, bokmal, nynor
ish, portuguese, romanian, russian, sanskrit, serbian, slovak, slovenian
sh, swedish, turkish, turkmen, ukrainian, uppersorbian, welsh, loaded.
(/usr/local/texlive/2010/texmf-dist/tex/latex/base/minimal.cls
Document Class: minimal 2001/05/25 Standard LaTeX minimal class
) (./bashful.sty
(/usr/local/texlive/2010/texmf-dist/tex/latex/listings/listings.sty
(/usr/local/texlive/2010/texmf-dist/tex/latex/graphics/keyval.sty)
(/usr/local/texlive/2010/texmf-dist/tex/latex/listings/lstmisc.sty)
(/usr/local/texlive/2010/texmf-dist/tex/latex/listings/listings.cfg))
(/usr/local/texlive/2010/texmf-dist/tex/latex/xkeyval/xkeyval.sty
(/usr/local/texlive/2010/texmf-dist/tex/generic/xkeyval/xkeyval.tex)))
(./minimal.aux)bash minimal.sh exit code was 0

!
Error(s) executing minimal.sh. Here is how
file minimal.stderr begins:
error.c: In function   m a i n  :
but, you really ought to examine this file yourself.
\checkStderr@BL ... to examine this file yourself}
                                                    \please \examine \std
l.7 \END

? No pages of output.
Transcript written on minimal.log.
```

You can see that when LaTeX tried to process `minimal.tex`, it stopped execution while indicating that file `minimal.stderr` was not empty after the compilation. The first line of `minimal.stderr` was displayed, and I was advised to examine this file myself. Inspecting `minimal.stderr`, we see the C compiler error messages:

```
% cat minimal.stderr
error.c: In function    m a i n   :
error.c:1: error: expected expression before    i n t
```

# 4 Customization

## 4.1 Options

Options to `\bash` are passed using the xkeyval syntax:

`scriptFile` = ⟨*fileName*⟩ `\jobname.sh`
Name of file into which the script instructions are spilled prior to execution. The default is `\jobname.sh`; this file will be reused by all `\bash` commands in your documents. This is rarely a problem, since these scripts execute sequentially.

`stdoutFile` = ⟨*fileName*⟩ `\jobname.stdout`
Name of file into which the shell standard output stream is redirected.

`stderrFile` = ⟨*fileName*⟩ `\jobname.stderr`
Name of file into which the shell standard error stream is redirected.

`list` = ⟨`true`/`false`⟩ `false`
If `true`, the content of `scriptFile` is listed in the main document.

`stdout` = ⟨`true`/`false`⟩ `false`
If `true`, the content of `stdoutFile` is listed in the main document. If both `list` and `stdout` are `true`, then `scriptFile` is listed first, and leaving no vertical space, `stdoutFile` is listed next.

`prefix` = ⟨tokens⟩ `%␣`
String that prefixes the listing of `scriptFile`.

`verbose` = ⟨`true`/`false`⟩ `false`
If `true`, the package logs every step it takes.

## 4.2 Listings Styles

Package listing is used for all listing both the executed shell commands and their output.

### 4.2.1 Listings Style for Script File

Style `bashfulList` is used for displaying the executed shell commands (when option `list` is used). The current definition of this style is:

```
\lstdefinestyle{bashfulList}{
    basicstyle=\ttfamily,
```

```
  showstringspaces=false}
```

Redefine this style to match your needs.

### 4.2.2 Listings Style for Standard Output

Style `bashfulStdout` is used for displaying the output of the executed shell commands (when option `stdout` is used). The current definition is:

```
\lstdefinestyle{bashfulStdout}{
  basicstyle=\sl\ttfamily,
  showstringspaces=false
}
```

Redefine this style to match your needs.

# 5 Interaction with Other Packages

This packages tries to work around a bug in `polyglossia` by which `\texttt` is garbled upon switching to languages which do not use the Latin alphabet. Also, in case bidirectional TEXing is in effect, bashful forces the listing to be left-to-right.

# 6 History

**Version 0.91** Initial release

# 7 Future

The following may get implemented some day.

1. *Package options.* Currently all options are passed to the command itself.
2. *A `clean` option.* This option will automatically erase files generated for storing the script, and its standard output and standard error streams.
3. *A `noclobber` option.* This option will make this package safer, by reducing the risk of accidentally erasing existing files.
4. *A `ignoreExitCode` option.* When `true \bash` will consider an execution correct even if its exit code is not 0.
5. *A `ignoreStderr` option.* When `true \bash` will consider an execution correct even if produces output to the standard error stream.

# 8 Acknowledgments

The manner by which `\bash` collects its arguments is based on that of tobiShell. Martin Scharrer tips on TEX internals were invaluable.

# A Source of `bashful.sty`

```
% Copyright (C) 2011 by Yossi Gil  yogi@cs.technion.ac.il
% ------------------------------------------------------------------------
% This work may be distributed and/or modified under the conditions of the
% LaTeX Project Public License (LPPL), either version 1.3 of this license or
% (at your option) any later version.  The latest version of this license is in
% http://www.latex-project.org/lppl.txt and version 1.3 or later is part of all
% distributions of LaTeX version 2005/12/01 or later.
%
% This work has the LPPL maintenance status `maintained'.
%
% The Current Maintainer of this work is Yossi Gil.
%
% This work consists of the files bashful.tex and bashful.sty and the derived
% bashful.pdf

\NeedsTeXFormat{LaTeX2e}%

% Auxiliary identification information
\newcommand\date@bashful{2011/02/28}%
\newcommand\version@bashful{V 0.91}%
\newcommand\author@bashful{Yossi Gil}%
\newcommand\mail@bashful{yogi@cs.technion.ac.il}%
\newcommand\signature@bashful{%
  bashful \version@bashful{} by
  \author@bashful{} \mail@bashful
}%

% Identify this package
\ProvidesPackage{bashful}[\date@bashful{} \signature@bashful:
  Write and execute a bash script within LaTeX, with, or
  without displaying the script and/or its output.
]
\PackageInfo{bashful}{This is bashful, \signature@bashful}%

% Use listing to display bash scripts.
\RequirePackage{listings}%
  % listings style for the script, can be redefined by client
  \lstdefinestyle{bashfulList}{
    basicstyle=\ttfamily,
    showstringspaces=false}%
  % listings style for the stdoutFile, can be redefined by client
  \lstdefinestyle{bashfulStdout}{
    basicstyle=\sl\ttfamily,
    showstringspaces=false
  }%

% Use xkeyval for retrieving parameters
\RequirePackage{xkeyval}%

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % Keys in alphabetical order:
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  % environment: String = \BL@environment: Which environment should we wrap the listings
  \def\BL@environment{none@BL}%
  \define@cmdkey{bashful}[BL@]{environment}{}%
```

```
    \newenvironment{none@BL}{}{} % Default, empty environment for wrapping the listings
58
    % scriptFile: String = Where should the listed script be saved?
    \def\BL@scriptFile{\jobname.lsh}%
    \define@cmdkey{bashful}[BL@]{scriptFile}{}%

    % list: Boolean = \ifBL@list: Should we list the script we generate?
    \define@boolkey{bashful}[BL@]{list}[true]{}%

    % prefix: String = \BL@prefix: What prefix should be printed before a listing.
    \def\BL@prefix{\@percentchar\space}%
68  \define@cmdkey{bashful}[BL@]{prefix}{}%

     % shell: String = \BL@shell: Which shell should be used for execution?
    \def\BL@shell{bash}%
    \define@cmdkey{bashful}[BL@]{shell}{}%

    % scriptFile: String = \BL@scriptFile: Where should the script be saved?
    \def\BL@scriptFile{\jobname.sh}%
    \define@cmdkey{bashful}[BL@]{scriptFile}{}%

78  % stderrFile: String = \BL@stderrFile: Where should the standard error be saved?
    \def\BL@stderrFile{\jobname.stderr}%
    \define@cmdkey{bashful}[BL@]{stderrFile}{}%

    % stdout: Boolean = \ifBL@stdout: Should we list the standard output?
    \define@boolkey{bashful}[BL@]{stdout}[true]{}%

    % stdoutFile: String = \BL@stdoutFile: Where should the standard output be saved?
    \def\BL@stdoutFile{\jobname.stdout}%
    \define@cmdkey{bashful}[BL@]{stdoutFile}{}%
88
    % verbose: Boolean = \ifBL@verbose: Log every step we do
    \define@boolkey{bashful}[BL@]{verbose}[true]{}%

  % \bash: the main command we define. It chains to \bashI which chains to
  % \bashII, etc.
  \begingroup
    \catcode`\^^M\active%
    \gdef\bash{%
          \log@BL{Beginning a group so that all cat code changes are local}%
98    \begingroup%
          \log@BL{Making \^\^M a true newline}%
      \catcode`\^^M\active%
      \def^^M{^^J}%
          \log@BL{Checking for optional arguments}%
      \@ifnextchar[\bashI{\bashI[]}%
    }%
  \endgroup

  % \bashI: Process the optional arguments and continue
108 \def\bashI[#1]{%
    \setKeys@BL{#1}%
    \bashII% And, continue processing.
  }%

  % \bashII: Set category codes of all to special, and proceed.
  \begingroup
    \catcode`\^^M\active%
    \gdef\bashII{%
          \log@BL{bashII: Making ^^M a true new line}%
118   \catcode`\^^M\active%
      \def^^M{^^J}%
      \log@BL{bashII: Making all characters other}%
      \let\do\@makeother%
      \dospecials%
      \bashIII%
    }%
```

```
\endgroup

% \bashIII: Consume all tokens until \END (but ignoring the preceding and
% terminating newline), and proceed.
\begingroup
  \catcode`\@=0\relax
  \catcode`\^^M\active
  @catcode`@\=12@relax%
  @gdef@bashIII^^M#1^^M\END{@bashIV{#1}}@relax%
@endgroup

% \bashIV: Process the tokens by storing them in a script file, executing this
% file, listing it if required, and presenting the standard output if required.
\newcommand\bashIV[1]{%
        \generateScriptFile@BL{#1}\relax
  \executeScriptFile@BL
  \checkStderr@BL
  \listScript@BL{#1}\relax%
  \listStdout@BL
  \log@BL{Ending group so that all cat codes changes disappear}\relax%
  \endgroup
}%

\def\setKeys@BL#1{%
  \log@BL{Processing key=val pairs in options string [#1]}\relax
  \setkeys{bashful}{#1}%
}%

% Store the list of tokens in the first argument into our script file
\newcommand\generateScriptFile@BL[1]{%
  \storeToFile@BL{#1}{\BL@scriptFile}%
}%

\newwrite\writer@BL
% Store the list of tokens in the first argument into the file given
% in the second argument
\newcommand\storeToFile@BL[2]{%
   \log@BL{Creating file #2 :=^^J#1^^J}%
  \immediate\openout\writer@BL#2
  \immediate\write\writer@BL{#1}%
  \immediate\closeout\writer@BL
}%

% Execute the content of our script file.
\newcommand\executeScriptFile@BL{%
  \def\command@BL{\BL@shell\space\BL@scriptFile}%
  \edef\command@BL{%
    \BL@shell\space -c
    "\command@BL \space > \BL@stdoutFile \space2> \BL@stderrFile \space
                          || echo '\command@BL' exit code was $? 2>>\BL@stderrFile"%
  }%
        \log@BL{Executing:^^J \command@BL}%
        \immediate\write18{\command@BL}%
}%

\newread\reader@BL
% Issue an error if stderr is not empty
\newcommand\checkStderr@BL{%
  \log@BL{Examining \BL@stderrFile}%
  \begingroup
  \newif\ifErrorsFound@\ErrorsFound@false
  \log@BL{Opening \BL@stderrFile}%
  \openin\reader@BL=\BL@stderrFile\relax
  \ifeof\reader@BL
    \BL@verbosetrue
    \log@BL{Strange... \BL@stderrFile{} does not exist (probably bashful bug)}%
    \log@BL{Switching to verbose mode}%
  \else
```

```
      \log@BL{Reading first line of \BL@stderrFile}%
      \catcode`\^^M=5
      \read\reader@BL to \firstErrorLine
      \ifeof\reader@BL
        \def\eoln{\par}
198     \ifx\firstErrorLine\eoln
          \log@BL{File \BL@stderrFile\space is empty}%
          \closein\reader@BL
        \else
          \log@BL{File \BL@stderrFile\space has one line [\firstErrorLine]}%
          \ErrorsFound@true
          \closein\reader@BL
        \fi
      \else
        \log@BL{File \BL@stderrFile\space has two lines or more}%
208     \ErrorsFound@true
        \closein\reader@BL
      \fi
    \fi
    \ifErrorsFound@
      \log@BL{Issuing an error message since \BL@stderrFile\space was not empty}%
      \let\please=\relax\let\examine=\relax\let\stderr=\relax
      \errmessage{^^JError(s) executing \BL@scriptFile. Here is how
       ^^Jfile \BL@stderrFile\space begins:
       ^^J\firstErrorLine
218                ^^Jbut, you really ought to examine this file yourself}
           \please
           \examine
           \stderr
    \else
      \log@BL{Proceeding as usual since \BL@stderrFile{} is empty}%
    \fi
        \endgroup
  }%

228 % List the content of the script file.
  \newcommand\listScript@BL[1]{%
    \ifBL@list{%
      \log@BL{Prepending prefix [[\BL@prefix]] to generate \BL@scriptFile}%
      \storeToFile@BL{\BL@prefix#1}{\BL@scriptFile}%
        \forceLTR@BL
        \fixPolyglossiaBug@BL
        \log@BL{Invoking \noexpand\lstinputlisting to list \BL@scriptFile}%
        \ifBL@stdout
          % trim bottom
238       \beginWrappingEnvironment@BL
            \lstinputlisting[style=bashfulList,belowskip=0pt]{\BL@scriptFile}%
          \else
            \beginWrappingEnvironment@BL
              \lstinputlisting[style=bashfulList]{\BL@scriptFile}%
              \endWrappingEnvironment@BL
          \fi
        }\else\relax\fi
  }%

248 \def\beginWrappingEnvironment@BL{%
    \expandafter\csname\BL@environment\endcsname
  }%

  \def\endWrappingEnvironment@BL{%
    \expandafter\csname end\BL@environment\endcsname
  }%

  % Display the standard output of the execution
  \newcommand\listStdout@BL{%
258   \ifBL@stdout{%
      \log@BL{Listing stdout file \BL@stdoutFile}%
        \forceLTR@BL
```

```
      \fixPolyglossiaBug@BL
      \log@BL{Invoking \noexpand\lstinputlisting to list \BL@stdoutFile}%
    \ifBL@list % trim top
          \lstinputlisting[style=bashfulStdout,aboveskip=0pt]{\BL@stdoutFile}%
          \endWrappingEnvironment@BL
      \else
          \beginWrappingEnvironment@BL
              \lstinputlisting[style=bashfulStdout]{\BL@stdoutFile}%
              \endWrappingEnvironment@BL
      \fi
      }\else\relax\fi
}%

\newcommand\fixPolyglossiaBug@BL{%
  \log@BL{Trying to fix a Polyglossia package bug}%
  \ifdefined\ttfamilylatin
    \log@BL{Replacing \noexpand\ttfamily with \noexpand\ttfamilylatin}%
    \let\ttfamily=\ttfamilylatin
    \log@BL{Replacing \noexpand\rmfamily with \noexpand\rmfamilylatin}%
    \let\rmfamily=\rmfamilylatin
    \log@BL{Replacing \noexpand\sffamily with \noexpand\sffamilylatin}%
    \let\sffamily=\sffamilylatin
    \log@BL{Replacing \noexpand\normalfont with \noexpand\normalfontlatin}%
    \let\normalfont=\normalfontlatin
  \else
    \log@BL{Polyglossia package probably not loaded}%
    \relax
 \fi
}%

\newcommand\forceLTR@BL{%
  \log@BL{Making sure we are not in right-to-left mode}%
  \ifdefined\setLTR
    \log@BL{Command \noexpand\setLTR is defined, invoking it}%
    \setLTR
  \else
    \log@BL{Command \noexpand\setLTR is not defined, we are probably LTR}%
    \relax
  \fi
}%

\newcommand\log@BL[1]{%
  \ifBL@verbose
    \typeout{L\the\inputlineno: #1}%
   \else
                         \relax
    \fi
}%
```

# B Source of the Current Document

```
\documentclass{ltxdoc} % Process with xelatex -shell-escape
\usepackage{bashful}

\usepackage[colorlinks=true]{hyperref}
\usepackage{gensymb}
\usepackage{graphicx}
\usepackage{metalogo}
\usepackage{xkvview}
\usepackage{xspace}
\usepackage{amsmath}


\newcommand\me{bashful}
\newcommand\bashful{\textsf{\me}\xspace}
  \lstdefinestyle{input}{
    basicstyle=\ttfamily,
```

```
    showstringspaces=false,
    aboveskip=0pt,
    belowskip=0pt}%

\title{The \bashful Package\thanks{
  This work may be distributed and/or modified under the conditions of the
    \emph{GNU GENERAL PUBLIC LICENSE}{http://www.gnu.org/licenses/gpl.html}.
The latest version of this license is in
  \url{http://www.latex-project.org/lppl.txt} and version 1.3 or later
  is part of all distributions of \LaTeX{} version 2005/12/01 or later.
This work has the LPPL maintenance status `maintained'.
The Current Maintainer of this work is Yossi Gil.
This work consists of the files \texttt{\me.tex} and \texttt{\me.sty}
  and the derived file
  \texttt{\me.pdf}
}}

\author{Yossi Gil\thanks{\url{mailto:yogi@cs.Technion.ac.IL}}\\
   \normalsize Department of Computer Science\\
   \normalsize The Technion---Israel Institute of Technology\\
   \normalsize Technion City, Haifa 32000, Israel
}

\makeatletter
\date{\date@bashful\thanks{
      This document describes \bashful \version@bashful.}}
\makeatother

\begin{document}
\bash
cat << EOF > README
The bashful package, v 0.91

This package makes it possible to execute bash scripts from within LaTeX. The
main application is in writing computer-science texts, in which you want to
make sure the programs listed in the document are executed directly from the
input.

This package may be distributed and/or modified under the LaTeX Project Public
License, version 1.3 or higher (your choice). The latest version of this
license is at: http://www.latex-project.org/lppl.txt

This work is author-maintained (as per LPPL maintenance status)
by Yossi Gil, <yogi@cs.Technion.ac.i>
EOF
\END

\maketitle

\begin{abstract}
\parindent 1.5ex
\parskip 0.5em

\sl
It is sometimes useful to ``\emph{escape-to-shell}'' from within
        \LaTeX{}.
The most obvious application is when the document
        explains something about the working of a computer program.
Your text would be more robust to changes, and easier to write,
        if all the examples it gives, are run directly from
        within \LaTeX{}.

To facilitate this and other applications,
  package \bashful{} provides a convenient interface to \TeX's
        primitive \verb+\write18+---the execution of shell commands from within
        your input files, also known as \emph{shell escape}.
Text between \verb+\bash+ and \verb+\END+ is executed by
        the \href
```

```
                {http://en.wikipedia.org/wiki/Bash_%28Unix_shell%29}
                {\texttt{bash}},
        a popular Unix command line interpreter.
Various flags control whether the executed commands and their output
        show up in the printed document, and whether they are saved
        to files.
91
Although provisions are made for using shells other
        than \texttt{bash}, this package may \emph{not} operate without
        modifications on  Microsoft's operating systems.
\end{abstract}
\eject

\tableofcontents

\parindent 1.5ex
101 \parskip 0.5em

\section{Introduction}
\bash[verbose]
rm -f temperature.tex
rm -f condition.tex
\END

\bash[verbose,scriptFile=temperature.sh,stdoutFile=temperature.tex]
location=Jerusalem,Israel
111 server="http://www.Google.com/ig/api"
request="$server?weather=$location"
wget -q -O - $request |\
tr "<>" "\012\012" |\
grep temp_c |\
sed 's/[^0-9]//g'
\END

\bash[verbose,scriptFile=condition.sh,stdoutFile=condition.tex]
location=Jerusalem,Israel
121 server="http://www.Google.com/ig/api"
request="$server?weather=$location"
wget -q -O - $request |\
tr "<>" "\012\012" |\
grep "condition data" |\
head -n 1 |\
sed -e 's/^.*="//' -e 's/"\/*//' |\
tr 'A-Z' 'a-z'
\END

131 At the time I run this document through \LaTeX{},
        the temperature in Jerusalem, Israel,
        was~\emph{\input{temperature.tex}\celsius},
        while the weather condition was \emph{\input{condition}}\unskip.

You may not care so much about these bits of truly
  ephemeral information,
        but you may be surprised that they were produced
        by the very process of \LaTeX{}ing.

141 How did I do that? Well, the first step was to write
        a series of shell commands that retrieve the current temperature,
        and another such series to obtain the current
        weather conditions.
This task required connection to
\href{http://www.Google.com/support/forum/p/apps-apis/thread?tid=0c95e45bd80def1a&hl=en}%
{Google's weather service} and
        minimal dexterity with Unix pipes and filters to process the output.
My command series to obtain the current temperature was:

151 \begin{minipage}{\textwidth}
\begin{quote}
```

```
        \lstinputlisting[style=input]{temperature.sh}
\end{quote}
\end{minipage}

while the weather condition was obtained by

\begin{minipage}{\textwidth}
\begin{quote}
        \lstinputlisting[style=input]{condition.sh}
\end{quote}
\end{minipage}

The second step was coercing \LaTeX{} to run these commands
  while processing this file.
To do that, I wrapped these two series within
        a \verb+\bash+\ldots\verb+\END+ pair.
The \verb+\bash+ command, offered by this package,
        takes all subsequent lines, stopping at  the closing \verb+\END+,
        places these in a file, and then
        lets the \texttt{bash} shell interpreter execute this file.

Allowing \LaTeX{} to run arbitrary shell commands can be
        dangerous---you never know whether that nice looking \texttt{tex}
        file you received by email was prepared by a friend or
        a foe.
This is the reason that you have to tell \LaTeX{}
        explicitly that shell escapes
        are allowed.
The \texttt{-shell-esc} does that.
Processing this file was thus done by executing, at the command line,
\begin{quote}
        \tt
        \% latex -shell-escape \jobname.tex
\end{quote}

What I actually wrote in my \texttt{.tex} file
  to produce the temperature in
        Jerusalem, Israel was:

\begin{minipage}{\textwidth}
\begin{quote}
\noindent\verb+\bash[verbose,scriptFile=temperature.sh,stdoutFile=temperature.tex]+
\lstinputlisting[style=input]{temperature.sh}
\verb+\END+\\
\end{quote}
\end{minipage}

The flags passed to above to the \verb+bash+ control sequence instructed it:
        \begin{enumerate}
                \item to be verbose, typing out a detailed log of everything it did;
                \item to save the shell commands in a script file named
                      \texttt{temperature.sh}; and,
                \item to store the standard output of the script in a file named
                      \texttt{temperature.tex}.
        \end{enumerate}

To obtain the current weather condition in the capital I wrote:

\begin{minipage}{\textwidth}
\begin{quote}
\noindent\verb+\bash[verbose,scriptFile=condition.sh,stdoutFile=condition.tex]+
\lstinputlisting[style=input]{condition.sh}
\verb+\END+
\end{quote}
\end{minipage}

I wrote these two just after my \verb+\begin{document}+.
```

```
When \LaTeX{} encountered these, it executed the bash commands and
        created two files \texttt{temperature.tex} and \texttt{condition.tex}.

Subsequently, I could use the content of these files by writing:


\begin{quote}
\begin{verbatim}
At the time I run this document through \LaTeX{},
        the temperature in Jerusalem, Israel,
        was~\emph{\input{temperature.tex}\celsius},
        while the weather condition was \emph{\input{condition}}\unskip.
\end{verbatim}
\end{quote}

\section{An Easy to Digest Example}
If you were intimidated by technicalities of the
  above description, let's try another example
  that might be easier to digest.

I will start by telling a simple story
  of writing, compiling and executing and
    a simple program.
Then, I will explain how I used the \verb+\bash+
  command to not only tell a story, but
  also to play it live: that is, authoring
  a simple~C program, compiling it and executing
  it, all from within \LaTeX{}.

\subsection{A ``Hello, World'' Program}

\subsubsection{Authoring}
Let's first write a simple
  \href{http://en.wikipedia.org/wiki/Hello_world_program}
    {Hello, World!} program in the
    \href{http://en.wikipedia.org/wiki/C_(programming_language)}
    {C programming language}:

\begin{quote}
\bash[verbose,list]
rm -f hello.c; cat << EOF > hello.c
/*
** hello.c: My first C program; it prints
** "Hello, World!", and dies.
*/

#include <stdio.h>

int main()
{
  printf("Hello, World!\n");
  return 0;
}
EOF
\END
\end{quote}

\subsubsection{Compiling}
Now, let's compile this program:
\begin{quote}
\bash[list,stdout]
cc hello.c
\END
\end{quote}

\subsubsection{Executing}
Finally, we can execute this program,
  and see that indeed, it prints the ``Hello, World!''
```

```
      string.
    \begin{quote}
291 \bash[list,stdout]
    ./a.out
    \END
    \end{quote}

    \subsection{Behind the Scenes}
    \subsubsection{Authoring}
    What I wrote in the input to produce the
      \texttt{hello.c} program was:
    \begin{quote}
301 \begin{verbatim}
    \bash[list]
    rm -f hello.c; cat << EOF > hello.c
    /*
    ** hello.c: My first C program; it prints
    ** "Hello, World!", and dies.
    */

    #include <stdio.h>

311 int main()
    {
      printf("Hello, World!\n");
      return 0;
    }
    EOF
    \END
    \end{verbatim}
    \end{quote}

321 In doing so, all the text between the \verb+\bash+
      and \verb+\END+ was sent to a temporary file,
        which was then sent to the \texttt{bash}
        bash shell for execution.
    The \texttt{list} flag instructed \verb+\bash+
      to list this file in the main document.
    This listing was prefixed with \verb*+% +
      to make it clear that it was input to \texttt{bash}.

    \subsubsection{Compiling}
331 Next, I wrote
    \begin{quote}
    \begin{verbatim}
    \bash[list]
    cc hello.c
    \END
    \end{verbatim}
    \end{quote}

    As before, in doing that, I achieved two objectives:
341   first, when \LaTeX{} processed
      the input, it also invokes the~C compiler to compile
      file \texttt{hello.c}, the file which I just created.
    Second, thanks to the \texttt{list} flag,
      the command for compiling this program
      was included in the printed version of
      this document.
    Thanks to the \texttt{stdout} option,
     plain messages, i.e., not error messages,
     produced by the compiler would show on
351   the printed version of this document.
    In this case, no such messages were produced.


    \subsubsection{Executing}
```

```
Finally, I wrote
\begin{quote}
\begin{verbatim}
\bash[list,stdout]
./a.out
\END
\end{verbatim}
\end{quote}
to run the program I just wrote.
The \texttt{stdout} adds to my listing
  the output that this execution produces, i.e.,
  the string \texttt{Hello, World!} that this
  execution produces to the standard output.

\section{Dealing with Shell Command Errors}
Using \bashful{} to demonstrate
  my \emph{Hello, World!} program, made
  sure that the story I told is accurate:
I really did everything I said I did.
More accurately, the \verb+\bash+ command
  acted as my proxy, and did it for me.

Luckily, my \texttt{hello.c} program was
  correct.
But, if it was not, the \verb+\bash+ command would have detected
  the error, and would have stopped the \LaTeX{} process,
  indicating that the compilation did not succeed.
More specifically, the \verb+\bash+ command
\begin{enumerate}
\item collects all commands up to \verb+\END+;
\item places these commands in a script file;
\item executes this script file, redirecting its standard output
      and its standard error streams to distinct files;
\item checks whether the exit code of the execution indicates an error
  (i.e., exit code which is different from~$0$), and if so,
    appends this exit code to the file containing the standard error;
\item checks whether the file containing the standard error is empty,
    and if not, pauses execution after displaying an error message;
\item lists, if requested to, the script file; and,
\item lists, if requested to, the file containing the standard output.
\end{enumerate}

Let me demonstrate a situation in which the execution of
  the script generates an error.
To do that, I will write a short \LaTeX{} file, named \texttt{minimal.tex}
  which tries to use \verb+\bash+ to compile an incorrect~C program.
Since \texttt{minimal.tex} contains \verb+\END+,
  I will have to author this file in three steps:
\begin{enumerate}
\item Creating the header of \texttt{minimal.tex}:
\bash[list]
cat << EOF > minimal.tex
\documentclass{minimal}
\usepackage{bashful}
\begin{document}
\bash
echo "main(){return int;}" > error.c
cc error.c
EOF
\END
\item Adding \verb+\END+ to \texttt{minimal.tex}
\bash[list]
echo "\\END" >> minimal.tex
\END
\item Finalizing \texttt{minimal.tex}
\bash[list]
echo "\\end{document}" >> minimal.tex
\END
```

```
\end{enumerate}

Let me now make sure \texttt{minimal.tex} was what I expect it to be:

\begin{minipage}{\textwidth}
\bash[list,stdout]
cat minimal.tex
\END
\end{minipage}

I am now ready to run \texttt{minimal.tex} through \LaTeX{},
  but since I will not run the \texttt{latex} command myself,
  I will make sure a \texttt{q} character is sent to it
  when the anticipated error occurs.

\begin{minipage}{\textwidth}
  \lstdefinestyle{bashfulStdoutFile}{
    showstringspaces=false,
    basicstyle=\scriptsize\ttfamily,
  }%
\bash[list,stdout]
yes x | latex -shell-esc minimal.tex
\END
\end{minipage}

You can see that when \LaTeX{} tried to process \texttt{minimal.tex},
  it stopped execution while indicating that file
  \texttt{minimal.stderr} was not
  empty after the compilation. The first line of \texttt{minimal.stderr}
  was displayed, and I was advised to examine this file myself.
Inspecting \texttt{minimal.stderr}, we see the C compiler error messages:

\begin{minipage}{\textwidth}
\bash[list,stdout]
cat minimal.stderr
\END
\end{minipage}

\section{Customization}

\subsection{Options}

Options to \verb+\bash+ are passed using the \textsf{xkeyval} syntax:

\newcommand\option[3]{%
      \noindent\(
          \text{\bfseries\texttt{#1}}
          =
          \langle\text{{#2}}\rangle
      \)
      \hfill\texttt{#3}\\}


\option{scriptFile}{\sl fileName}{\textbackslash jobname.sh}
Name of file into which the script instructions are spilled prior
  to execution.
The default is \verb+\jobname.sh+; this file
  will be reused by all \verb+\bash+ commands in your documents.
  This is rarely a problem, since these scripts
  execute sequentially.

\option{stdoutFile}{\sl fileName}{\textbackslash jobname.stdout}
Name of file into which the shell standard output stream is
  redirected.


\option{stderrFile}{\sl fileName}{\textbackslash jobname.stderr}
Name of file into which the shell standard error stream is
```

```
  redirected.

\option{list}{\texttt{true}/\texttt{false}}{\texttt{false}}
If \texttt{true}, the content of \texttt{scriptFile}
  is listed in the main document.

\option{stdout}{\texttt{true}/\texttt{false}}{\texttt{false}}
If \texttt{true}, the content of \texttt{stdoutFile}
  is listed in the main document.
If both \texttt{list} and \texttt{stdout} are
  \texttt{true}, then \texttt{scriptFile} is listed
  first, and leaving no vertical space,
  \texttt{stdoutFile} is listed next.

\option{prefix}{tokens}{\percentchar\textvisiblespace}
String that prefixes the listing of \texttt{scriptFile}.

\option{verbose}{\texttt{true}/\texttt{false}}{\texttt{false}}
If \texttt{true}, the package logs every step it takes.

\subsection{Listings Styles}
Package
  \href
    {ftp://ftp.tex.ac.uk/tex-archive/macros/latex/contrib/listings/listings.pdf}
    {\textsf{listing}}
  is used for all listing both the executed shell
  commands and their output.
\subsubsection{Listings Style for Script File}

Style \verb+bashfulList+ is used for displaying the executed shell
  commands (when option \texttt{list} is used).
The current definition of this style is:
\begin{verbatim}
\lstdefinestyle{bashfulList}{
    basicstyle=\ttfamily,
    showstringspaces=false}
\end{verbatim}
Redefine this style to match your needs.

\subsubsection{Listings Style for Standard Output}

Style \verb+bashfulStdout+ is used for displaying the output of the
  executed shell
  commands (when option \texttt{stdout} is used).
The current definition is:
\begin{verbatim}
  \lstdefinestyle{bashfulStdout}{
    basicstyle=\sl\ttfamily,
    showstringspaces=false
  }
\end{verbatim}
Redefine this style to match your needs.

\section{Interaction with Other Packages}
This packages tries to work around a bug in \texttt{polyglossia}
  by which \verb+\texttt+ is garbled upon
  switching to languages which do not use the Latin alphabet.
Also, in case bidirectional \TeX{}ing is in effect,
  \bashful forces the listing to be left-to-right.


\section{History}
\begin{description}
\item[Version 0.91] Initial release
\end{description}

\section{Future}
The following may get implemented some day.
```

```
561
     \begin{enumerate}
     \item \emph{Package options.} Currently all options are
         passed to the command itself.
     \item \emph{A \texttt{clean} option.} This option
       will automatically erase files
       generated for storing the script, and its standard
       output and standard error streams.

     \item \emph{A \texttt{noclobber} option.} This option
571    will make this package safer, by reducing the risk
       of accidentally erasing existing files.


     \item \emph{A \texttt{ignoreExitCode} option.} When
       \texttt{true} \verb+\bash+ will consider
         an execution correct even if its exit code
         is not 0.

     \item \emph{A \texttt{ignoreStderr} option.} When
581    \texttt{true} \verb+\bash+ will consider
         an execution correct even if produces
         output to the standard error stream.
     \end{enumerate}

     \section{Acknowledgments}
     The manner by which \verb+\bash+
       collects its arguments is based on that of
      \href
       {http://www.tn-home.de/Tobias/Soft/TeX/tobiShell.pdf}
591    {\textsf{tobiShell}}.
     Martin Scharrer tips on \TeX{} internals
       were invaluable.

     \appendix
     \section{Source of \texttt{\jobname.sty}}
             \lstinputlisting
               [             style=input,
                                 basicstyle=\scriptsize\ttfamily,
                                 numbers=left,
601                              stepnumber=10,
                                 numberstyle=\scriptsize\rmfamily\bfseries
                     ]
               {\jobname.sty}

     \section{Source of the Current Document}
             \lstinputlisting
               [             style=input,
                                 basicstyle=\scriptsize\ttfamily,
                                 numbers=left,
611                              stepnumber=10,
                                 numberstyle=\scriptsize\rmfamily\bfseries
                     ]
               {\jobname.tex}


     \end{document}
```