iexec: LaTeX Package for Inputable Shell Executions*

Yegor Bugayenko
yegor256@gmail.com

2022-10-22, 0.11.0

1 Introduction

This package helps you execute shell commands right from the document and then put their output to the document:

Today is **22-Oct-2022**

```
\documentclass{article}
\usepackage{iexec}
\usepackage[paperwidth=3in]{geometry}
\pagestyle{empty}
\begin{document}
Today is \textbf{%
  \iexec{date +\%e-\%b-\%Y}}
\end{document}
```

`\iexec` The only command provided by this package is `\iexec [⟨options⟩] {⟨cmd⟩}`. Its only mandatory argument ⟨*cmd*⟩ is the command to be executed through the terminal shell (`bash`, or whatever is set as the default one in your console).

You have to run `pdflatex` (or just `latex`) with the `--shell-escape` flag in order to let shellesc execute your shell command.

2 Options

`quiet` If you don't want the output to be visible, use `\phantom\{\iexec{...}}`. Otherwise, you can use `quiet` option:

```
I just want to delete some file:
\iexec[quiet]{rm -f foo.txt}
```

In this case, whatever the shell command produces will not be included into the document.

`stdout` The output of your code is saved into the file provided as an optional argument of `\iexec` (the default value is `iexec.tmp`):

```
Today is \iexec[stdout=date.txt]{date +\%e-\%b-\%Y | tr -d '\\n'}.
```

The tailing part of the command here removes all ends-of-line.

`stderr` The error output of the code is saved into the file provided as an optional argument of `\iexec` (by default the error output is streamed into `stdout`):

```
Today is \iexec[stderr=my.txt]{broken-command}.
```

`exit` The exit code of the command it saved into a file. You can change the name of it using `exit` option:

```
Today is \iexec[exit=code.txt]{./broken-command.sh}.
```

`trace` The file specified will be deleted right after its usage. If you don't want this to happen, use `trace` package option: all files will remain in the directory where they were created. It's possible to turn tracing on globbaly, for the entire document, using `trace` option of the package:

```
\documentclass{article}
\usepackage[trace]{iexec}
\begin{document}
This file won't be deleted: \iexec[stdout=me.txt]{whoami}.
\end{document}
```

`append` The stdout produced will be appended to the file specified:

```
\documentclass{article}
\usepackage[trace]{iexec}
\begin{document}
\iexec[append,stdout=foo.txt,quiet]{echo 'Hello, '}
\iexec[append,stdout=foo.txt,quiet]{echo 'Jeffrey!'}
\input{foo.txt}
\end{document}
```

`log` The stdout produced will be printed in TEX log:

```
\iexec[log]{echo 'Hello, \\LaTeX!'}
```

`null` The stdout of the command will be sent to `/dev/null`:

```
\iexec[null]{rm some-file.txt}
```

`ignore` By default, we report an error if exit code is not equal to zero. You can suppress this with `ignore` option:

```
\iexec[ignore]{broken-command}
```

3 Implementation

First, we include shellesc package, which we use to execute shell commands:

```
\RequirePackage{shellesc}
```

Then, we parse package options:

```
\RequirePackage{xkeyval}
\makeatletter
\newif\ifiexec@trace
\DeclareOptionX{trace}{\iexec@tracetrue}
\ProcessOptionsX\relax
\makeatother
```

*The sources are in GitHub at yegor256/iexec

Then, we prepare to parse the options of `\iexec` command:

```
\RequirePackage{pgfkeys}
\makeatletter\pgfkeys{
  /iexec/.is family,
  /iexec,
  exit/.store in = \iexec@exit,
  exit/.default = iexec.ret,
  stdout/.store in = \iexec@stdout,
  stdout/.default = iexec.tmp,
  stderr/.store in = \iexec@stderr,
  trace/.store in = \iexec@traceit,
  append/.store in = \iexec@append,
  log/.store in = \iexec@log,
  null/.store in = \iexec@null,
  quiet/.store in = \iexec@quiet,
  ignore/.store in = \iexec@ignore,
  stdout,exit
}\makeatother
```

`\iexec@typeout` Then, we define an internal command `\iexec@typeout` for printing the content of a file, as suggested here:

```
\RequirePackage{expl3}
\makeatletter\ExplSyntaxOn
\NewDocumentCommand{\iexec@typeout}{m}{
  \iexec_typeout_file:n { #1 }}
\ior_new:N \g_iexec_typeout_ior
\cs_new_protected:Nn \iexec_typeout_file:n
{
  \ior_open:Nn \g_iexec_typeout_ior { #1 }
  \ior_str_map_inline:Nn \g_iexec_typeout_ior
    {\iow_term:n { ##1 }}
  \ior_close:N \g_iexec_typeout_ior
}
\ExplSyntaxOff\makeatother
```

`\iexec` Then, we define `\iexec` command. It is implemented with the help of `\ShellEscape` from `shellesc` package:

```
\makeatletter
\newread\iexec@exitfile
\newcommand\iexec[2][]{%
  \begingroup%
    \pgfqkeys{/iexec}{#1}%
```

First, we verify that `latex` is running with `--shell-escape` option, since without it nothing will work; so, it's better to throw an error earlier than later:

```
    \ifnum\ShellEscapeStatus=1\else%
      \PackageError{iexec}{You must run TeX processor with
      --shell-escape option}{}%
    \fi%
    \begingroup%
```

Then, start the log from a clean line:

```
      \ifdefined\iexec@log%
        \message{^^J}%
      \fi%
```

Then, we define a few special chars in order to escape them in the shell (the full list of them is in macros2e):

```
      \let\%\@percentchar%
      \let\\\@backslashchar%
      \let\{\@charlb%
      \let\}\@charrb%
```

Then, we execute it:

```
      \def\iexec@cmd{(#2)
        \ifdefined\iexec@append>\fi>
        \ifdefined\iexec@null/dev/null\else\iexec@stdout\fi
        \space\ifdefined\iexec@stderr2>\iexec@stderr\else2>&1\fi;
        /bin/echo -n $? >\iexec@exit}
      \ShellEscape{\iexec@cmd}%
```

Then, a message is printed to TeX log:

```
      \ifdefined\iexec@log%
        \message{iexec: [\iexec@cmd]^^J}%
      \fi%
    \endgroup%
```

Then, if required, the content of the stdout file will be printed to the log:

```
    \ifdefined\iexec@null\else%
    \ifdefined\iexec@log%
      \message{iexec: This is the content of \iexec@stdout:^^J}%
      \iexec@typeout{\iexec@stdout}%
      \message{<EOF>^^J}%
    \fi\fi%
```

Then, we check exit code, unless there is `ignore` option:

```
    \immediate\openin\iexec@exitfile=\iexec@exit%
    \read\iexec@exitfile to \iexec@code%
    \immediate\closein\iexec@exitfile%
    \ifnum\iexec@code=0\else%
      \ifdefined\iexec@ignore%
        \ifdefined\iexec@log%
          \message{iexec: Execution failure ignored,
            the exit code was \iexec@code^^J}%
        \fi%
      \else%
        \PackageError{iexec}{Execution failure,
          the exit code was \iexec@code}{}%
      \fi%
    \fi%
```

Then, include the produced output into the current document:

```
    \ifdefined\iexec@null\else%
    \ifdefined\iexec@quiet%
      \ifdefined\iexec@log%
        \message{iexec: Due to 'quiet' option we didn't read
        the content of '\iexec@stdout'
        \ifdefined\pdffilesize (\pdffilesize{\iexec@stdout}
        bytes)\fi^^J}%
      \fi%
    \else%
      \ifdefined\iexec@log%
```

```
        \message{iexec: We are going to include the content of
        '\iexec@stdout'\ifdefined\pdffilesize (\pdffilesize
        {\iexec@stdout} bytes)\fi...^^J}%
      \fi%
      \input{\iexec@stdout}%
      \message{iexec: The content of '\iexec@stdout'
      was included into the document^^J}%
    \fi\fi%
```

Finally, delete the file or leave it untouched:

```
    \ifdefined\iexec@null\else%
    \ifiexec@trace%
      \ifdefined\iexec@log%
        \message{iexec: Due to package option 'trace',
        the files '\iexec@stdout' and `\iexec@exit` were
        not deleted^^J}%
      \fi%
    \else%
      \ifdefined\iexec@traceit%
        \ifdefined\iexec@log%
          \message{iexec: Due to 'trace' package option,
          the files '\iexec@stdout' and '\iexec@exit'
          were not deleted^^J}%
        \fi%
      \else%
        \ShellEscape{rm \iexec@stdout}%
        \ifdefined\iexec@log%
          \message{iexec: The file '\iexec@stdout' was deleted^^J}%
        \fi%
        \ShellEscape{rm \iexec@exit}%
        \ifdefined\iexec@log%
          \message{iexec: The file '\iexec@exit' was deleted^^J}%
        \fi%
      \fi%
    \fi\fi%
  \endgroup
}\makeatother
```

Change History

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.