Package 'reprex'

January 26, 2018

Title Prepare Reproducible Example Code for Sharing

Version 0.1.2

Description Convenience wrapper that uses the 'rmarkdown' package to render small snippets of code to target formats that include both code and output. The goal is to encourage the sharing of small, reproducible, and runnable examples on code-oriented websites, such as <http://stackoverflow.com> and <https://github.com>, or in email. 'reprex' also extracts clean, runnable R code from various common formats, such as copy/paste from an R session.

Depends R (>= 3.0.2)

License MIT + file LICENSE

LazyData true

URL https://github.com/jennybc/reprex

BugReports https://github.com/jennybc/reprex/issues

Imports callr, knitr, rmarkdown, tools, utils, whisker

Suggests covr, devtools, formatR, fortunes, miniUI, rstudioapi, shiny, shinyjs, testthat

SystemRequirements pandoc (>= 1.12.3) - http://pandoc.org

RoxygenNote 5.0.1.9000

NeedsCompilation no

Author Jennifer Bryan [aut, cre], David Robinson [aut]

Maintainer ORPHANED

Repository CRAN

Date/Publication 2018-01-26 08:23:05 UTC

X-CRAN-Original-Maintainer Jennifer Bryan <jenny.f.bryan@gmail.com>

X-CRAN-Comment Orphaned on 2018-01-26 for policy violations, and offending code disabled.

R topics documented:

`reprex` *Render a reprex*

Description

Run a bit of R code using `rmarkdown::render()`. The goal is to make it easy to share a small reproducible example ("reprex"), e.g., in a GitHub issue. Reprex source can be

- read from clipboard
- read from current selection or active document ("Render reprex" RStudio addin)
- provided directly as expression, character vector, or string
- read from file

The usual "code + commented output" is returned invisibly, put on the clipboard, and written to file. An HTML preview displays in RStudio's Viewer pane, if available, or in the default browser, otherwise. Leading `"> "` prompts, are stripped from the input code.

Usage

```
reprex(x = NULL, venue = c("gh", "so", "r", "R"), si = FALSE,
  show = TRUE, input = NULL, outfile = NULL, comment = "#>",
  opts_chunk = NULL, opts_knit = NULL)
```

Arguments

`x`	An expression. If not given, `reprex()` looks for code in `input` or on the clipboard, in that order.
`venue`	"gh" for GitHub (default), "so" for StackOverflow, "r" or "R" for a runnable R script, with commented output interleaved.
`si`	Whether to include the results of `devtools::session_info()`, if available, or `sessionInfo()` at the end of the reprex. When `venue = "gh"` (the default), session info is wrapped in a collapsible details tag.
`show`	Whether to show rendered output in a viewer (RStudio or browser). Defaults to `TRUE`.
`input`	Character. If has length one and lacks a terminating newline, interpreted as the path to a file containing reprex code. Otherwise, assumed to hold reprex code as character vector (length greater than one) or string (with embedded newlines).

`outfile` Optional basename for output files. When `NULL` (default), reprex writes to temp files below the session temp directory. If `outfile = "foo"`, expect output files in current working directory, like `foo_reprex.R`, `foo_reprex.md`, and, if `venue = "R"`, `foo_rendered.R`. If `outfile = NA`, expect output files in current working directory with basename derived from the path in `input`, if sensible, otherwise from `tempfile()`.

`comment` Character. Prefix with which to comment out output, defaults to `"#>"`.

`opts_chunk, opts_knit` Named list. Optional knitr chunk and package options, respectively, to supplement or override reprex defaults. See Details.

Details

reprex sets specific knitr options, which you can supplement or override via the `opts_chunk` and `opts_knit` arguments or via explicit calls to knitr in your reprex code (see examples). If all you want to override is the `comment` option, use the dedicated argument, e.g.`commment = "#;-)"`.

- Chunk options default to `collapse = TRUE`, `comment = "#>"`, `error = TRUE`. These are options you normally set via `knitr::opts_chunk$set()`. Note that `error = TRUE`, because a common use case is bug reporting.
- Package options default to `upload.fun = knitr::imgur_upload`. These are options you normally set via `knitr::opts_knit$set()`. The `upload.fun` defaults to `imgur_upload` so figures produced by the reprex appear properly on GitHub.

Value

Character vector of rendered reprex, invisibly.

Examples

```
## Not run:
# put some code like this on the clipboard
# (y <- 1:4)
# mean(y)
reprex()

# provide code as an expression
reprex(rbinom(3, size = 10, prob = 0.5))
reprex({y <- 1:4; mean(y)})

# note that you can include newlines in those brackets
# in fact, that is probably a good idea
reprex({
  x <- 1:4
  y <- 2:5
  x + y
})

## provide code via character vector
reprex(input = c("x <- 1:4", "y <- 2:5", "x + y"))
```

```
## if just one line, terminate with '\n'
reprex(input = "rnorm(3)\n")

## customize the output comment prefix
reprex(rbinom(3, size = 10, prob = 0.5), comment = "#;-)")

# override a default chunk option, in general
reprex({(y <- 1:4); median(y)}, opts_chunk = list(collapse = FALSE))
# the above is simply shorthand for this and produces same result
reprex({
  #+ setup, include = FALSE
  knitr::opts_chunk$set(collapse = FALSE)

  #+ actual-reprex-code
  (y <- 1:4)
  median(y)
})

# add prose, use general markdown formatting
reprex({
  #' # A Big Heading
  #'
  #' Look at my cute example. I love the
  #' [reprex](https://github.com/jennybc/reprex#readme) package!
  y <- 1:4
  mean(y)
})

# read reprex from file
writeLines(c("x <- 1:4", "mean(x)"), "foofy.R")
reprex(input = "foofy.R")

# read from file and write to similarly-named outfiles
reprex(input = "foofy.R", outfile = NA)
list.files(pattern = "foofy")
file.remove(list.files(pattern = "foofy"))

# write rendered reprex to file
reprex({
  x <- 1:4
  y <- 2:5
  x + y
}, outfile = "foofy")
list.files(pattern = "foofy")
file.remove(list.files(pattern = "foofy"))

# write reprex to file AND keep figure local too, i.e. don't post to imgur
reprex({
  #' Some prose
  ## regular comment
  (x <- 1:4)
  median(x)
```

```
  plot(x)
  }, outfile = "blarg", opts_knit = list(upload.fun = identity))
list.files(pattern = "blarg")
unlink(list.files(pattern = "blarg"), recursive = TRUE)

## target venue = StackOverflow
## http://stackoverflow.com/editing-help
ret <- reprex({
  x <- 1:4
  y <- 2:5
  x + y
}, venue = "so")
ret

## target venue = R, also good for email or Slack snippets
ret <- reprex({
  x <- 1:4
  y <- 2:5
  x + y
}, venue = "R")
ret

## include prompt and don't comment the output
## use this when you want to make your code hard to execute :)
reprex({
  x <- 1:4
  y <- 2:5
  x + y
}, opts_chunk = list(comment = NA, prompt = TRUE))

## leading prompts are stripped from source
reprex(input = c("> x <- 1:3", "> median(x)"))

## End(Not run)
```

`reprex_addin` *Render a reprex*

Description

An RStudio gadget and addin to call `reprex()`. Appears as "Render reprex" in the RStudio Addins menu. Prepare in one of these ways:

1. Copy reprex source to clipboard.
2. Select reprex source.
3. Activate the file containing reprex source.

Call `reprex()` directly for more control via additional arguments.

Usage

```
reprex_addin()
```

`un-reprex` *Un-render a reprex*

Description

Recover clean, runnable code from a reprex captured in the wild. The code is printed, returned invisibly, and written to the clipboard, if possible. Pick the function that deals with your problem:

- `reprex_invert()` handles Markdown, with code blocks indicated with backticks or indentation, e.g., the direct output of `reprex(..., venue = "gh")` or `reprex(..., venue = "so")`.
- `reprex_clean()` assumes R code is top-level, possibly interleaved with commented output, e.g., a displayed reprex copied from GitHub or the direct output of `reprex(..., venue = "R")`.
- `reprex_rescue()` assumes R code lines start with a prompt and printed output is top-level, e.g., what you'd get by copying from the R Console.

Usage

```
reprex_invert(input = NULL, venue = c("gh", "so"), comment = "^#>")

reprex_clean(input = NULL, comment = "^#>")

reprex_rescue(input = NULL, prompt = getOption("prompt"))
```

Arguments

`input`	character, holding a wild-caught reprex as a character vector (length greater than one), string (length one with terminating newline), or file path (length one with no terminating newline). If not provided, the clipboard is consulted for input.
`venue`	"gh" for GitHub (default), "so" for StackOverflow, "r" or "R" for a runnable R script, with commented output interleaved.
`comment`	regular expression that matches commented output lines
`prompt`	character, the prompt at the start of R commands

Value

character vector holding just the clean R code, invisibly

Functions

- `reprex_invert`: Attempts to reverse the effect of `reprex()`. The input should be Markdown, presumably the output of `reprex()`. `venue` matters because, in GitHub-flavored Markdown, code blocks are placed within triple backticks. In other Markdown dialects, such as the one used on StackOverflow, code is indented by four spaces.
- `reprex_clean`: Removes lines of commented output from a displayed reprex, such as code copied from a GitHub issue or `reprex`'ed with `venue = "R"`.
- `reprex_rescue`: Removes lines of output and strips prompts from lines holding R commands. Typical input is copy/paste from R Console.

Examples

```
## a rendered reprex can be inverted, at least approximately
x <- reprex({
  #' Some text
  #+ chunk-label-and-options-cannot-be-recovered, message = TRUE
  (x <- 1:4)
  #' More text
  y <- 2:5
  x + y
}, show = FALSE)
writeLines(x)
reprex_invert(x)
## a displayed reprex can be cleaned of commented output
x <- c(
  "## a regular comment, which is retained",
  "(x <- 1:4)",
  "#> [1] 1 2 3 4",
  "median(x)",
  "#> [1] 2.5"
  )
reprex_clean(x)

## Not run:
## round trip with reprex(..., venue = "R")
code_in <- c("x <- rnorm(2)", "min(x)")
res <- reprex(input = code_in, venue = "R")
res
(code_out <- reprex_clean(res))
identical(code_in, code_out)

## End(Not run)
## rescue a reprex that was copied from a live R session
x <- c(
  "> ## a regular comment, which is retained",
  "> (x <- 1:4)",
  "[1] 1 2 3 4",
  "> median(x)",
  "[1] 2.5"
)
reprex_rescue(x)
```

Index