Package 'reprex'

August 17, 2022

Title Prepare Reproducible Example Code via the Clipboard

Version 2.0.2

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 <https://stackoverflow.com> and <https://github.com>, or in email. The user's clipboard is the default source of input code and the default target for rendered output. 'reprex' also extracts clean, runnable R code from various common formats, such as copy/paste from an R session.

License MIT + file LICENSE

URL https://reprex.tidyverse.org, https://github.com/tidyverse/reprex#readme

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

Depends R (>= 3.4)

Imports callr (>= 3.6.0), cli (>= 3.2.0), clipr (>= 0.4.0), fs, glue, knitr (>= 1.23), lifecycle, rlang (>= 1.0.0), rmarkdown, rstudioapi, utils, withr (>= 2.3.0)

Suggests covr, fortunes, miniUI, mockr, rprojroot, sessioninfo, shiny, spelling, styler (>= 1.2.0), testthat (>= 3.0.2)

VignetteBuilder knitr

Config/Needs/website dplyr, tidyverse/tidytemplate

Config/testthat/edition 3

Encoding UTF-8

Language en-US

RoxygenNote 7.2.1

SystemRequirements pandoc (>= 2.0) - http://pandoc.org

NeedsCompilation no

Author Jennifer Bryan [aut, cre] (<https://orcid.org/0000-0002-6983-2759>),
Jim Hester [aut] (<https://orcid.org/0000-0002-2739-7082>),
David Robinson [aut],
Hadley Wickham [aut] (<https://orcid.org/0000-0003-4757-117X>),
Christophe Dervieux [aut] (<https://orcid.org/0000-0003-4474-2498>),
RStudio [cph, fnd]

Maintainer Jennifer Bryan <jenny@rstudio.com>

Repository CRAN

Date/Publication 2022-08-17 07:40:05 UTC

R topics documented:

`reprex` *Render a reprex*

Description

Run a bit of R code using `rmarkdown::render()` and write the rendered result to user's clipboard. If the clipboard is unavailable, the file containing the rendered result is opened for manual copy. 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
- provided directly as expression, character vector, or string
- read from file
- read from current selection or active document in RStudio

reprex can also be used for syntax highlighting (with or without rendering); see below for more.

Usage

```
reprex(
  x = NULL,
  input = NULL,
  wd = NULL,
  venue = c("gh", "r", "rtf", "html", "slack", "so", "ds"),
  render = TRUE,
  advertise = NULL,
  session_info = opt(FALSE),
  style = opt(FALSE),
  comment = opt("#>"),
  tidyverse_quiet = opt(TRUE),
  std_out_err = opt(FALSE),
  html_preview = opt(TRUE),
  outfile = deprecated(),
  show = deprecated(),
  si = deprecated()
)
```

Arguments

x	An expression. If not given, `reprex()` looks for code in `input`. If `input` is not provided, `reprex()` looks on the clipboard. When the clipboard is structurally unavailable, e.g., on RStudio Server or RStudio Cloud, `reprex()` consults the current selection instead of the clipboard.
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. When `input` specifies a filepath, it also determines the reprex working directory and the location of all resulting files.
wd	An optional filepath that is consulted when `input` is not a filepath. (By default, all work is done, quietly, in a subdirectory of the session temp directory.) The most common use of `wd` is to set `wd = ".."`, which means "reprex right HERE in the current working directory". Do this if you really must demonstrate something with local files.
venue	Character. Must be one of the following (case insensitive): • "gh" for GitHub-Flavored Markdown, the default • "r" for a runnable R script, with commented output interleaved. Also useful for Slack code snippets; select "R" from the "Type" drop-down menu to enjoy nice syntax highlighting. • "rtf" for Rich Text Format (not supported for un-reprexing) • "html" for an HTML fragment suitable for inclusion in a larger HTML document (not supported for un-reprexing) • "slack" for pasting into a Slack message. Optimized for people who opt out of Slack's WYSIWYG interface. Go to **Preferences > Advanced > Input options** and select "Format messages with markup". (If there is demand for a second Slack venue optimized for use with WYSIWYG, please open an issue to discuss.)

- "so" for Stack Overflow Markdown. Note: this is just an alias for "gh", since Stack Overflow started to support CommonMark-style fenced code blocks in January 2019.
- "ds" for Discourse, e.g., community.rstudio.com. Note: this is currently just an alias for "gh".

`render` Logical. Whether to call `rmarkdown::render()` on the templated reprex, i.e. whether to actually run the code. Defaults to `TRUE`. Exists primarily for the sake of internal testing.

`advertise` Logical. Whether to include a footer that describes when and how the reprex was created. If unspecified, the option `reprex.advertise` is consulted and, if that is not defined, default is `TRUE` for venues `"gh"`, `"html"`, `"so"`, `"ds"` and `FALSE` for `"r"`, `"rtf"`, `"slack"`.

`session_info` Logical. Whether to include `sessioninfo::session_info()`, if available, or `sessionInfo()` at the end of the reprex. When `venue` is "gh", the session info is wrapped in a collapsible details tag. Read more about `opt()`.

`style` Logical. Whether to set the knitr chunk option `tidy = "styler"`, which re-styles code with the styler package. Read more about `opt()`.

`comment` Character. Prefix with which to comment out output, defaults to `"#>"`. Read more about `opt()`.

`tidyverse_quiet`
Logical. Sets the options `tidyverse.quiet` and `tidymodels.quiet`, which suppress (`TRUE`, the default) or include (`FALSE`) the startup messages for the tidyverse and tidymodels packages. Read more about `opt()`.

`std_out_err` Logical. Whether to append a section for output sent to stdout and stderr by the reprex rendering process. This can be necessary to reveal output if the reprex spawns child processes or `system()` calls. Note this cannot be properly interleaved with output from the main R process, nor is there any guarantee that the lines from standard output and standard error are in correct chronological order. See `callr::r()` for more. Read more about `opt()`.

`html_preview` Logical. Whether to show rendered output in a viewer (RStudio or browser). Always `FALSE` in a noninteractive session. Read more about `opt()`.

`outfile` **[Deprecated]** in favor of `wd` or providing a filepath to `input`. To reprex in current working directory, use `wd = "."` now, instead of `outfile = NA`.

`show` **[Deprecated]** in favor of `html_preview`, for greater consistency with other R Markdown output formats.

`si` **[Deprecated]** in favor of `session_info`.

Value

Character vector of rendered reprex, invisibly.

Details

The usual "code + commented output" is returned invisibly, written to file, and, whenever possible, put on the clipboard. 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. Read more at https://reprex.tidyverse.org/.

reprex sets specific knitr options:

- Chunk options default to `collapse = TRUE`, `comment = "#>"`, `error = TRUE`. Note that `error = TRUE`, because a common use case is bug reporting.
- reprex also sets knitr's `upload.fun`. It defaults to `knitr::imgur_upload()` so figures produced by the reprex appear properly on GitHub, Stack Overflow, Discourse, and Slack. Note that `imgur_upload()` requires the packages httr and xml2. When `venue = "r"`, `upload.fun` is set to `identity()`, so that figures remain local. In that case, you may also want to provide a filepath to `input` or set `wd`, to control where the reprex files are written. You can supplement or override these options with special comments in your code (see examples).

Error backtraces

reprex sets the rlang option `rlang_backtrace_on_error_report = "full"`. Combined with the knitr option `error = TRUE`, this means rlang errors are displayed with a full backtrace. This basically eliminates the need to call `rlang::last_error()` or `rlang::last_trace()` explicitly, although these functions can be used in a reprex.

Insert a line containing the special comment `#'` in between the error-causing code and the `last_error()` or `last_trace()` call, to fulfill the requirement of being in separate chunks:

```
f <- function() rlang::abort('foo')
f()
#'
rlang::last_error()
rlang::last_trace()
```

Read more in rlang's documentation: Errors in RMarkdown.

Syntax highlighting

[Experimental]

A secondary use case for reprex is to produce syntax highlighted code snippets, with or without rendering, to paste into applications like Microsoft Word, PowerPoint, or Keynote. Use `venue = "rtf"` for this.

This feature is experimental and requires the installation of the highlight command line tool. The `"rtf"` venue is documented in its own article

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)})
reprex({y <- 1:4; mean(y)}, style = TRUE)

# note that you can include newlines in those brackets
# in fact, that is often 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
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/tidyverse/reprex#readme) package!
  y <- 1:4
  mean(y)
}, advertise = FALSE)

# read reprex from file and write resulting files to that location
tmp <- file.path(tempdir(), "foofy.R")
writeLines(c("x <- 1:4", "mean(x)"), tmp)
reprex(input = tmp)
list.files(dirname(tmp), pattern = "foofy")

# clean up
file.remove(list.files(dirname(tmp), pattern = "foofy", full.names = TRUE))

# write reprex to file AND keep figure local too, i.e. don't post to imgur
tmp <- file.path(tempdir(), "foofy")
dir.create(tmp)
```

```
reprex({
  #+ setup, include = FALSE
  knitr::opts_knit$set(upload.fun = identity)

  #+ actual-reprex-code
  #' Some prose
  ## regular comment
  (x <- 1:4)
  median(x)
  plot(x)
  }, wd = tmp)
list.files(dirname(tmp), pattern = "foofy")

# clean up
unlink(tmp, recursive = TRUE)

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

## target venue = html
ret <- reprex({
  x <- 1:4
  y <- 2:5
  x + y
}, venue = "html")
ret

## include prompt and don't comment the output
## use this when you want to make your code hard to execute :)
reprex({
  #+ setup, include = FALSE
  knitr::opts_chunk$set(comment = NA, prompt = TRUE)

  #+ actual-reprex-code
  x <- 1:4
  y <- 2:5
  x + y
})

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

## End(Not run)
```

`reprex_addin` *Render a reprex, conveniently*

Description

`reprex_addin()` opens an RStudio gadget and addin that allows you to say where the reprex source is (clipboard? current selection? active file? other file?) and to control a few other arguments. Appears as "Render reprex" in the RStudio Addins menu.

`reprex_selection()` is an addin that reprexes the current selection, optionally customised by options. Appears as "Reprex selection" in the RStudio Addins menu. Heavy users might want to create a keyboard shortcut, which is described in `https://support.rstudio.com/hc/en-us/articles/206382178-Customizin`
Suggested shortcut: Cmd + Shift + R (macOS) or Ctrl + Shift + R (Windows).

Usage

```
reprex_addin()

reprex_selection(venue = getOption("reprex.venue", "gh"))
```

Arguments

`venue`	Character. Must be one of the following (case insensitive):

- "gh" for GitHub-Flavored Markdown, the default
- "r" for a runnable R script, with commented output interleaved. Also useful for Slack code snippets; select "R" from the "Type" drop-down menu to enjoy nice syntax highlighting.
- "rtf" for Rich Text Format (not supported for un-reprexing)
- "html" for an HTML fragment suitable for inclusion in a larger HTML document (not supported for un-reprexing)
- "slack" for pasting into a Slack message. Optimized for people who opt out of Slack's WYSIWYG interface. Go to **Preferences > Advanced > Input options** and select "Format messages with markup". (If there is demand for a second Slack venue optimized for use with WYSIWYG, please open an issue to discuss.)
- "so" for Stack Overflow Markdown. Note: this is just an alias for "gh", since Stack Overflow started to support CommonMark-style fenced code blocks in January 2019.
- "ds" for Discourse, e.g., community.rstudio.com. Note: this is currently just an alias for "gh".

`reprex_document` *reprex output format*

Description

This is an R Markdown output format designed specifically for making "reprexes", typically created via the `reprex()` function, which ultimately renders the document with `reprex_render()`. It is a heavily modified version of `rmarkdown::md_document()`. The arguments have different spheres of influence:

- venue potentially affects input preparation and reprex_render().
- Add content to the primary input, prior to rendering:
  - advertise
  - session_info
  - std_out_err (also consulted by reprex_render())
- Influence knitr package or chunk options:
  - style
  - comment
  - tidyverse_quiet

RStudio users can create new R Markdown documents with the reprex_document() format using built-in templates. Do *File* > *New File* > *R Markdown ...* > *From Template* and choose one of:

- reprex (minimal)
- reprex (lots of features)

Both include knit: reprex::reprex_render in the YAML, which causes the RStudio "Knit" button to use reprex_render(). If you render these documents yourself, you should do same.

Usage

```
reprex_document(
  venue = c("gh", "r", "rtf", "html", "slack", "so", "ds"),
  advertise = NULL,
  session_info = opt(FALSE),
  style = opt(FALSE),
  comment = opt("#>"),
  tidyverse_quiet = opt(TRUE),
  std_out_err = opt(FALSE),
  pandoc_args = NULL
)
```

Arguments

venue — Character. Must be one of the following (case insensitive):

- "gh" for GitHub-Flavored Markdown, the default
- "r" for a runnable R script, with commented output interleaved. Also useful for Slack code snippets; select "R" from the "Type" drop-down menu to enjoy nice syntax highlighting.
- "rtf" for Rich Text Format (not supported for un-reprexing)
- "html" for an HTML fragment suitable for inclusion in a larger HTML document (not supported for un-reprexing)
- "slack" for pasting into a Slack message. Optimized for people who opt out of Slack's WYSIWYG interface. Go to **Preferences > Advanced > Input options** and select "Format messages with markup". (If there is demand for a second Slack venue optimized for use with WYSIWYG, please open an issue to discuss.)

- "so" for Stack Overflow Markdown. Note: this is just an alias for "gh", since Stack Overflow started to support CommonMark-style fenced code blocks in January 2019.
- "ds" for Discourse, e.g., community.rstudio.com. Note: this is currently just an alias for "gh".

`advertise` Logical. Whether to include a footer that describes when and how the reprex was created. If unspecified, the option `reprex.advertise` is consulted and, if that is not defined, default is `TRUE` for venues `"gh"`, `"html"`, `"so"`, `"ds"` and `FALSE` for `"r"`, `"rtf"`, `"slack"`.

`session_info` Logical. Whether to include `sessioninfo::session_info()`, if available, or `sessionInfo()` at the end of the reprex. When venue is "gh", the session info is wrapped in a collapsible details tag. Read more about `opt()`.

`style` Logical. Whether to set the knitr chunk option `tidy = "styler"`, which restyles code with the styler package. Read more about `opt()`.

`comment` Character. Prefix with which to comment out output, defaults to `"#>"`. Read more about `opt()`.

`tidyverse_quiet` Logical. Sets the options `tidyverse.quiet` and `tidymodels.quiet`, which suppress (`TRUE`, the default) or include (`FALSE`) the startup messages for the tidyverse and tidymodels packages. Read more about `opt()`.

`std_out_err` Logical. Whether to append a section for output sent to stdout and stderr by the reprex rendering process. This can be necessary to reveal output if the reprex spawns child processes or `system()` calls. Note this cannot be properly interleaved with output from the main R process, nor is there any guarantee that the lines from standard output and standard error are in correct chronological order. See `callr::r()` for more. Read more about `opt()`.

`pandoc_args` Additional command line options to pass to pandoc

Value

An R Markdown output format to pass to `rmarkdown::render()`.

Examples

```
reprex_document()
```

---

reprex_locale *Render a reprex in a specific locale*

Description

Render a `reprex()`, with control over the localization of error messages and aspects of the locale. Note that these are related but distinct issues! Typical usage is for someone on a Spanish system to create a reprex that is easier for an English-speaking audience to follow.

Usage

```
reprex_locale(..., language = "en", locale = NULL)
```

Arguments

`...`	Inputs passed through to `reprex()`.
`language`	A string specifying the preferred language for messages. It is enacted via the `LANGUAGE` environment variable, for the duration of the `reprex()` call. Examples: `"en"` for English and `"fr"` for French. See Details for more.
`locale`	A named character vector, specifying aspects of the locale, in the `Sys.setlocale()` sense. It is enacted by setting one or more environment variables, for the duration of the `reprex()` call. See Details for more.

Value

Character vector of rendered reprex, invisibly.

`language`

Use the `language` argument to express the preferred language of error messages. The output of `dir(system.file(package = "translations"))` may provide some helpful ideas. The `language` should generally follow "XPG syntax": a two-letter language code, optionally followed by other modifiers.

Examples: `"en"`, `"de"`, `"en_GB"`, `"pt_BR"`.

`locale`

Use the `locale` argument only if you want to affect something like how day-of-the-week or month is converted to character. You are less likely to need to set this than the `language` argument. You may have more success setting specific categories, such as `"LC_TIME"`, than multi-category shortcuts like `"LC_ALL"` or `"LANG"`. The `locale` values must follow the format dictated by your operating system and the requested locale must be installed. On *nix systems, `locale -a` is a good way to see which locales are installed. Note that the format for `locale` and `language` are different from each other on Windows.

Examples: `"en_CA.UTF-8"` (macOS), `"French_France.1252"` (Windows).

See Also

- The Locale Names section of the GNU C docs, for more about XPG syntax
- The Internationalization and Localization section of the R Installation and Administration manual

Examples

```
## Not run:

# if all you want to do is make sure messages are in English
reprex_locale("a" / 2)
```

```
# change messages to a specific language
reprex_locale(
  {
    "a" / 2
  },
  language = "it"
)

reprex_locale(
  {
    "a" / 2
  },
  language = "fr_CA"
)

reprex_locale(
  {
    "a" / 2
  },
  language = "pt_BR"
)

# get day-of-week and month to print in French (not Windows)
reprex_locale(
  {
    format(as.Date(c("2019-01-01", "2019-02-01")), "%a %b %d")
  },
  locale = c(LC_TIME = "fr_FR")
)

# get day-of-week and month to print in French (Windows)
# assumes that the relevant language is installed on the system
# LC_TIME can also be specified as "French" or "French_France" here
reprex_locale(
  {
    format(as.Date(c("2019-01-01", "2019-02-01")), "%a %b %d")
  },
  locale = c(LC_TIME = "French_France.1252")
)

## End(Not run)
```

`reprex_options` *reprex options*

Description

Some `reprex()` behaviour can be controlled via an option, providing a way for the user to set personal defaults. The pattern for such option names is `reprex.<arg>`, where `<arg>` is an argument of `reprex()`. Here are the main ones:

- reprex.advertise
- reprex.session_info (previously, reprex.si)
- reprex.style
- reprex.html_preview (previously, reprex.show)
- reprex.comment
- reprex.tidyverse_quiet
- reprex.std_out_err

A few more options exist, but are only relevant to specific situations:

- reprex.venue: Can be used to control the venue used by the reprex_selection() addin.
- reprex.current_venue: Read-only option that is set during reprex_render(). Other packages that want to generate reprex-compatible output can consult it via getOption("reprex.current_venue"), if they want to tailor their output to the venue.
- reprex.clipboard: When FALSE, reprex makes no attempt to access the user's clipboard, ever. This exists mostly for internal use, i.e. we set it to FALSE when we detect use from RStudio Server. But a user could set this to FALSE to explicitly opt-out of clipboard functionality. A Linux user with no intention of installing xclip or xsel might also do this.
- reprex.highlight.hl_style: Only relevant to venue = "rtf". Details are in the article reprex venue RTF.
- reprex.highlight.font: See above.
- reprex.highlight.font_size: See above.
- reprex.highlight.other: See above.

Here's code you could put in .Rprofile to set reprex options. It would be rare to want non-default behaviour for all of these! We only do so here for the sake of exposition:

```
options(
  reprex.advertise       = FALSE,
  reprex.session_info    = TRUE,
  reprex.style           = TRUE,
  reprex.html_preview    = FALSE,
  reprex.comment         = "#;-)",
  reprex.tidyverse_quiet = FALSE,
  reprex.std_out_err     = TRUE,
  reprex.venue           = "html", # NOTE: only affects reprex_selection()!
  reprex.highlight.hl_style  = "acid", # NOTE: only affects RTF venue
  reprex.highlight.font      = "Andale Mono Regular",
  reprex.highlight.font_size = 35,
  reprex.highlight.other     = "--line-numbers"
)
```

The function usethis::edit_r_profile() is handy for creating and/or opening your .Rprofile.

Explaining the `opt()` helper

Arguments that appear like so in `reprex()`:

```
reprex(..., arg = opt(DEFAULT), ...)
```

get their value according to this logic:

```
user-specified value or, if not given,
  getOption("reprex.arg") or, if does not exist,
    DEFAULT
```

It's shorthand for:

```
f(..., arg = getOption("reprex.arg", DEFAULT), ...)
```

This is not an exported function and should not be called directly.

---

reprex_render *Render a document in a new R session*

Description

This is a wrapper around `rmarkdown::render()` that enforces the "reprex" mentality. Here's a simplified version of what happens:

```
callr::r(
  function(input) {
    rmarkdown::render(input, envir = globalenv(), encoding = "UTF-8")
  },
  args = list(input = input),
  spinner = is_interactive(),
  stdout = std_file, stderr = std_file
)
```

Key features to note

- `rmarkdown::render()` is executed in a new R session, by using `callr::r()`. The goal is to eliminate the leakage of objects, attached packages, and other aspects of session state from the current session into the rendering session. Also, the system and user-level `.Rprofiles` are ignored.
- Code is evaluated in the `globalenv()` of this new R session, which means that method dispatch works the way most people expect it to.
- The input file is assumed to be UTF-8, which is a knitr requirement as of v1.24.
- If the YAML frontmatter includes `std_err_out: TRUE`, standard output and error of the rendering R session are captured in `std_file`, which is then injected into the rendered result.

`reprex_render()` is designed to work with the `reprex_document()` output format, typically through a call to `reprex()`. `reprex_render()` may work with other R Markdown output formats, but it is not well-tested.

Usage

```
reprex_render(input, html_preview = NULL, encoding = "UTF-8")
```

Arguments

input	The input file to be rendered. This can be a .R script or a .Rmd R Markdown document.
html_preview	Logical. Whether to show rendered output in a viewer (RStudio or browser). Always FALSE in a noninteractive session. Read more about opt().
encoding	The encoding of the input file. Note that the only acceptable value is "UTF-8", which is required by knitr as of v1.24. This is exposed as an argument purely for technical convenience, relating to the "Knit" button in the RStudio IDE.

Value

The output of rmarkdown::render() is passed through, i.e. the path of the output file.

Examples

```
## Not run:
reprex_render("input.Rmd")

## End(Not run)
```

---

reprex_venue *Venue-specific shortcuts*

Description

These are thin wrappers around reprex() that incorporate the target venue as a suffix in the function name, for easier access via auto-completion.

Usage

```
reprex_html(...)

reprex_r(...)

reprex_rtf(...)

reprex_slack(...)
```

Arguments

...	Passed along to reprex().

un-reprex *Un-render a reprex*

Description

Recover clean, runnable code from a reprex captured in the wild and write it to user's clipboard. The code is also returned invisibly and optionally written to file. Three different functions address various forms of wild-caught reprex.

Usage

```
reprex_invert(
  input = NULL,
  wd = NULL,
  venue = c("gh", "r"),
  comment = opt("#>"),
  outfile = deprecated()
)

reprex_clean(
  input = NULL,
  wd = NULL,
  comment = opt("#>"),
  outfile = deprecated()
)

reprex_rescue(
  input = NULL,
  wd = NULL,
  prompt = getOption("prompt"),
  continue = getOption("continue"),
  outfile = deprecated()
)
```

Arguments

input	Character. If has length one and lacks a terminating newline, interpreted as the path to a file containing the reprex. Otherwise, assumed to hold the reprex as a character vector. If not provided, the clipboard is consulted for input. If the clipboard is unavailable and we're in RStudio, the current selection is used.
wd	An optional filepath that is consulted when input is not a filepath. (By default, all work is done, quietly, in a subdirectory of the session temp directory.) The most common use of wd is to set wd = ".", which means "reprex right HERE in the current working directory". Do this if you really must demonstrate something with local files.
venue	Character. Must be one of the following (case insensitive):

- "gh" for GitHub-Flavored Markdown, the default
- "r" for a runnable R script, with commented output interleaved. Also useful for Slack code snippets; select "R" from the "Type" drop-down menu to enjoy nice syntax highlighting.
- "rtf" for Rich Text Format (not supported for un-reprexing)
- "html" for an HTML fragment suitable for inclusion in a larger HTML document (not supported for un-reprexing)
- "slack" for pasting into a Slack message. Optimized for people who opt out of Slack's WYSIWYG interface. Go to **Preferences > Advanced > Input options** and select "Format messages with markup". (If there is demand for a second Slack venue optimized for use with WYSIWYG, please open an issue to discuss.)
- "so" for Stack Overflow Markdown. Note: this is just an alias for "gh", since Stack Overflow started to support CommonMark-style fenced code blocks in January 2019.
- "ds" for Discourse, e.g., community.rstudio.com. Note: this is currently just an alias for "gh".

`comment` regular expression that matches commented output lines

`outfile` **[Deprecated]** in favor of `wd` or providing a filepath to `input`. To reprex in current working directory, use `wd = "."` now, instead of `outfile = NA`.

`prompt` character, the prompt at the start of R commands

`continue` character, the prompt for continuation lines

Value

Character vector holding just the clean R code, invisibly

Functions

- `reprex_invert()`: Attempts to reverse the effect of `reprex()`. When `venue = "r"`, this just calls `reprex_clean()`.
- `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")`. This function removes commented output.
- `reprex_rescue()`: Assumes R code lines start with a prompt and that printed output is top-level, e.g., what you'd get from copy/paste from the R Console. Removes lines of output and strips prompts from lines holding R commands.

Examples

```
## Not run:
# a roundtrip: R code --> rendered reprex, as gfm --> R code
original <- file.path(tempdir(), "original.R")
writeLines(glue::glue("
  #' Some text
  #+ chunk-label-and-options-cannot-be-recovered, message = TRUE
  (x <- 1:4)
```

```
  #' More text
  y <- 2:5
  x + y"), con = original)
reprex(input = original, html_preview = FALSE, advertise = FALSE)
reprexed <- sub("[.]R$", "_reprex.md", original)
writeLines(readLines(reprexed))
unreprexed <- reprex_invert(input = reprexed)
writeLines(unreprexed)

# clean up
file.remove(
  list.files(dirname(original), pattern = "original", full.names = TRUE)
)

## End(Not run)
## Not run:
# a roundtrip: R code --> rendered reprex, as R code --> original R code
code_in <- c(
  "# a regular comment, which is retained",
  "(x <- 1:4)",
  "median(x)"
)
reprexed <- reprex(input = code_in, venue = "r", advertise = FALSE)
writeLines(reprexed)
code_out <- reprex_clean(input = reprexed)
writeLines(code_out)
identical(code_in, code_out)

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

## End(Not run)
```

Index