Package: runr (via r-universe)

July 27, 2026

Title Run R Experiments

Version 0.0.1

Description Contains a set of helper functions and scripts for running R-related experiments.

License MIT + file LICENSE

URL https://github.com/PRL-PRG/runr

BugReports https://github.com/PRL-PRG/runr/issues

Encoding UTF-8

Depends R (>= 3.5.0)

Imports callr, codetools, digest, dplyr, fs, knitr, lubridate, magrittr, progress, purrr, readr, remotes, rex, stats, stringr (>= 1.5.0), testthat, tibble, tools, utils, withr

Suggests rmarkdown, mutator

Remotes PRL-PRG/mutator@v0.2.1

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.0.0

Config/pak/sysreqs cmake git make libicu-dev libuv1-dev libx11-dev

Repository https://prl-prg.r-universe.dev

Date/Publication 2026-07-27 16:04:00 UTC

RemoteUrl https://github.com/PRL-PRG/runr

RemoteRef HEAD

RemoteSha c1eebb8bcd5510ebf05c54f8212f43aa9386dec6

Contents

`cloc` *Count lines of code with cloc*

Description

Runs the external `cloc` utility and parses its CSV output.

Usage

```
cloc(path, by_file = FALSE, r_only = FALSE, cloc_bin = "cloc")
```

Arguments

`path`	directory or file to count.
`by_file`	report one row per file rather than one row per language.
`r_only`	keep only the rows counting R code.
`cloc_bin`	name of, or path to, the `cloc` executable.

Value

A tibble with the `blank`, `comment` and `code` counts, identifying each row by `filename` when `by_file` is `TRUE` and by `language` and `files` otherwise (with the queried `path` prepended as a column), or `NULL` if `cloc` reported nothing.

current_script *Path of the script currently being executed*

Description

Works both under `Rscript` (via `--file=`) and when the file is `source()`d.

Usage

```
current_script()
```

Value

The normalised path of the running script, or `NULL` if it cannot be determined (for instance in an interactive session).

download_cran_package_source *Download and unpack a CRAN package's sources*

Description

Download and unpack a CRAN package's sources

Usage

```
download_cran_package_source(
  package,
  version = NULL,
  dest_dir = NULL,
  repos = getOption("repos")
)
```

Arguments

`package`	name of the package to download.
`version`	version to download, or `NULL` for the current one. Archived versions are resolved through the CRAN archive.
`dest_dir`	directory to extract into.
`repos`	the repositories to download from.

Value

The path to the extracted source directory. Errors if the sources could not be downloaded or did not extract where expected.

extract_kaggle_code *Extract the R code of a Kaggle notebook*

Description

Copies `.R` files as they are, knits `.Rmd` with `knitr::purl()`, and converts `.irnb`/`.ipynb` notebooks to `.Rmd` first, which needs the rmarkdown package.

Usage

```
extract_kaggle_code(source_file, target_file, quiet = TRUE)
```

Arguments

`source_file`	the notebook or script to extract from.
`target_file`	where to write the extracted R code.
`quiet`	suppress knitr's progress output.

Value

`target_file`, the path of the file written. Errors on an unsupported file type.

extract_package_code *Extract a package's runnable code into standalone R files*

Description

A package's executable code is not directly runnable: examples live inside `Rd` files, vignettes inside `Rmd`, and tests behind a `testthat.R` driver. This writes all of it out as plain `.R` files under `output_dir/<type>/`.

Usage

```
extract_package_code(
  pkg,
  pkg_dir = find.package(pkg),
  types = c("examples", "tests", "vignettes", "all"),
  output_dir,
  wrap = NULL,
  filter = NULL,
  split_testthat = FALSE,
  compute_sloc = FALSE,
  quiet = FALSE
)
```

Arguments

pkg	name of the package.
pkg_dir	the installed package directory to extract from.
types	which kinds of code to extract; "all" selects every kind.
output_dir	directory to write the extracted files to, created if needed.
wrap	NULL, a function(package, file, type, body) returning the new contents of each extracted file, or the path of a template file. A template is turned into such a function by wrap_using_template(), which substitutes the .PACKAGE., .FILE., .TYPE. and .BODY. placeholders.
filter	regexp on the extracted file names, or NULL to keep all.
split_testthat	emit one driver per test_*.R file instead of a single testthat.R, so a crashing test file does not take the suite down with it and each file gets its own exit code.
compute_sloc	also count lines of code, which requires cloc.
quiet	do not report progress.

Value

A tibble with one row per extracted file: file, relative to output_dir, and type. When compute_sloc is TRUE the blank, comment and code counts of the code the file came from are included, so for a testthat driver they describe the test file it runs rather than the driver itself.

---

file_sha1 *SHA1 digest of a file's contents*

Description

SHA1 digest of a file's contents

Usage

```
file_sha1(file)
```

Arguments

file	path to the file to hash.

Value

The SHA1 digest of the file contents, as a string.

impute_fun_srcref *Reconstruct source references inside a function*

Description

A function keeps a `srcref` for itself, but not for the expressions in its body. This recovers those from the parse data of the function's source file, so that coverage and tracing can attribute results to source locations.

Usage

```
impute_fun_srcref(fun)
```

Arguments

`fun`	the function to annotate. Must have been parsed with `keep.source` enabled, otherwise it is returned unchanged.

Value

`fun` with `srcref` attributes imputed on the expressions of its formals and body.

install_cran_packages *Install CRAN packages into a dedicated library*

Description

Installs the packages that are missing from `lib_dir` in a separate R process, keeping tests and source references so that the installed copies can be used for experiments.

Usage

```
install_cran_packages(
  packages,
  lib_dir = NULL,
  r_home = R.home(),
  dest_dir = NULL,
  mirror = "https://cloud.r-project.org/",
  force = FALSE,
  dependencies = TRUE,
  check = TRUE,
  install_opts = c("--example", "--install-tests", "--with-keep.source",
    "--no-multiarch"),
  n_cpus = floor(0.9 * parallel::detectCores())
)
```

Arguments

packages	names of the packages to install, or NULL for every package available from mirror.
lib_dir	library to install into, created if needed. NULL uses the default library.
r_home	the R_HOME of the R installation to install with.
dest_dir	directory to keep the downloaded sources in, created if needed.
mirror	CRAN mirror to install from.
force	install even the packages that are already present.
dependencies	passed to utils::install.packages().
check	also reinstall packages that are installed but fail to load.
install_opts	options passed to R CMD INSTALL.
n_cpus	number of parallel installation jobs.

Value

The paths to packages. This might be a subset of the requested packages, if some packages failed to install.

---

is_s3_dispatch_method *Does a function dispatch on S3?*

Description

Does a function dispatch on S3?

Usage

```
is_s3_dispatch_method(fun)
```

Arguments

fun	the function to inspect.

Value

TRUE if fun calls UseMethod() or NextMethod(), i.e. if it is an S3 generic or an S3 method that delegates further.

metadata_functions *Inventory of the functions in a package namespace*

Description

Loads package and describes every function bound in its namespace, including the ones it does not export.

Usage

```
metadata_functions(package, lib_loc = NULL)
```

Arguments

package	name of the package to inspect.
lib_loc	library path to load the package from, or NULL for the current one.

Value

A data frame with one row per function and the columns pkg_name, fun_name, exported, is_s3_dispatch (calls UseMethod() or NextMethod()), is_s3_method (registered as an S3 method) and params (the parameter names, ;-separated).

read_file *Read a whole file into a single string*

Description

Read a whole file into a single string

Usage

```
read_file(filename)
```

Arguments

filename	path to the file to read.

Value

The file contents as a length-one character vector, newlines included.

`read_files` *Read many files, turning failures into values*

Description

Bulk reader behind the other `read_parallel_*` functions. Shows a progress bar, and records a read failure as a value instead of aborting the batch, so that one unreadable file does not lose the whole run.

Usage

```
read_files(
  jobs,
  files,
  readf = read_lines,
  mapf = function(job, x) x,
  mapf_error = function(...) structure(list(...), class = "error"),
  reducef = identity,
  quiet = TRUE
)
```

Arguments

`jobs`	job names, parallel to `files`.
`files`	paths to read, parallel to `jobs`.
`readf`	function used to read one file.
`mapf`	`function(job, contents)` applied to each successful read.
`mapf_error`	function called with the job, the file and the error message when a read fails. Defaults to returning a condition object.
`reducef`	function applied to the whole list of results.
`quiet`	do not message about individual read failures.

Value

The result of `reducef` applied to the per-file results, a list named by `files` before reduction.

read_parallel_log *Read a GNU parallel job log*

Description

Read a GNU parallel job log

Usage

```
read_parallel_log(path)
```

Arguments

path	the --joblog file itself, or a directory containing a parallel.log.

Value

A tibble of the log with lower-cased column names: seq, host, starttime (a datetime), jobruntime (a period), send, receive, exitval, signal and command. Jobs retried with --retry-failed appear once per attempt; use read_parallel_results() to get one row per job.

read_parallel_results *Read the results of a GNU parallel run*

Description

Joins the job log with the per-job output directories under path. The join is driven by the directories, so a job retried with --retry-failed — which appends a second log entry but reuses its directory — contributes a single row, for the attempt whose output was kept.

Usage

```
read_parallel_results(path, stdout = TRUE, stderr = TRUE)
```

Arguments

path	the run directory, holding parallel.log and one subdirectory per job.
stdout	add the captured stdout.
stderr	add the captured stderr.

Value

A tibble of read_parallel_log() joined with job and path, plus a column per requested stream and a matching _error column holding the message when that file could not be read.

read_parallel_seq *Read the job sequence numbers of a GNU parallel run*

Description

Each job directory holds a `seq` file with the sequence number GNU parallel assigned it, which is what links a directory back to a log entry.

Usage

```
read_parallel_seq(path, quiet = TRUE)
```

Arguments

`path`	the run directory to scan recursively for `seq` files.
`quiet`	do not message about unreadable `seq` files.

Value

A tibble with one row per job directory and the columns `job`, `path` and `seq`.

run_all *Run every R file under a directory*

Description

Runs each `.R` file with `run_one()`. The individual files under a `testthat/` directory are skipped, because the extracted testthat drivers run them. By default the code is copied into a scratch directory first, so the corpus is left untouched even when `wrap_code_fun` rewrites files.

Usage

```
run_all(
  path,
  output_dir = getwd(),
  run_dir = tempfile(),
  filter = NULL,
  wrap_code_fun = NULL,
  clean = TRUE,
  quiet = TRUE,
  skip_if_out_exists = TRUE
)
```

Arguments

path	directory holding the R files to run.
output_dir	directory to write the per-file .out files to.
run_dir	scratch directory to copy the code into before running. Pass path to run in place.
filter	regexp on file names, or NULL to run everything.
wrap_code_fun	function(code) returning the new file contents, applied to each file just before it runs.
clean	remove run_dir afterwards.
quiet	do not report progress.
skip_if_out_exists	treat a file whose .out already exists as done.

Value

A data frame with one row per file and the columns file, out_file, exitval, time and error (the message if the file could not be run at all, NA otherwise).

---

run_one *Run a single R file in a fresh R subprocess*

Description

Runs file with the current R build under a fixed environment (LC_ALL=C, no browser, no PDF viewer, sources kept) so that runs are comparable across a corpus.

Usage

```
run_one(file, out_file, cwd = TRUE, quiet = TRUE, stats = TRUE)
```

Arguments

file	path to the R file to run.
out_file	where to write the combined stdout and stderr, or NULL to let it go to the console.
cwd	run from the file's own directory rather than the current one.
quiet	do not print the command being run.
stats	also report the elapsed time, recovered from the trailing proc.time() in the output.

Value

A one-row data frame with the exitval, and the elapsed time when stats is TRUE (NA if the run failed or the timing could not be read).

`run_r_file` *Run an R file with a specific R build*

Description

Runs `file` in a subprocess started from `r_home`, from the file's own directory. Use this rather than `run_one()` when the experiment is about a particular R build or library; `callr` cannot be told which R to use.

Usage

```
run_r_file(
  file,
  timeout,
  r_envir = c(R_TESTS = "", R_BROWSER = "false", R_PDFVIEWER = "false"),
  r_args = c("--no-save", "--quiet", "--no-readline"),
  r_home = R.home(),
  lib_path = NULL,
  keep_output = c("always", "never", "on_error"),
  quiet = TRUE
)
```

Arguments

`file`	path to the R file to run.
`timeout`	kill the process after this many seconds.
`r_envir`	named character vector of environment variables to set.
`r_args`	command line arguments passed to R.
`r_home`	the `R_HOME` of the R installation to run.
`lib_path`	library to run against, exported as `R_LIBS`, or `NULL` to inherit the default.
`keep_output`	whether to return the captured output `"always"`, `"never"`, or only `"on_error"`.
`quiet`	unused, kept for symmetry with the other runners.

Value

A one-row tibble with `file`, the exit `status`, the `elapsed` time in seconds and the combined stdout/stderr `output` (`NA` when `keep_output` suppresses it).

run_test_dir *Run a testthat directory the way test_check would*

Description

Simulates `testthat::test_check()`, otherwise `testthat::test_dir()` might skip some tests. Based on `testthat:::test_package_dir`.

Usage

```
run_test_dir(package, path, ...)
```

Arguments

package	name of the package under test.
path	directory containing the testthat tests.
...	passed on to `testthat::test_dir()`.

Value

The value of `testthat::test_dir()`.

run_test_env *Create a package environment with access to all package functions*

Description

Based on `testthat:::test_pkg_env`.

Usage

```
run_test_env(package)
```

Arguments

package	name of the package whose namespace to expose.

Value

An environment containing all bindings of the package namespace.

search_function_calls *Find calls to given functions in an expression*

Description

Walks `expr` recursively and collects the calls to any of `functions`, matching both the bare name (`fun(...)`) and the namespace-qualified forms (`pkg::fun(...)` and `pkg:::fun(...)`).

Usage

```
search_function_calls(expr, functions)
```

Arguments

`expr`	is the expression in which run the search
`functions`	is a string vector in the form of package:::function_name

Value

A list of the matching calls, or `NULL` if there are none.

wrap_files *Rewrite files in place with a wrapper function*

Description

Vectorised over `file` and `type`. A file that cannot be rewritten is reported and skipped, so one bad file does not stop the rest.

Usage

```
wrap_files(package, file, type, wrap_fun, quiet = TRUE)
```

Arguments

`package`	name of the package the files came from.
`file`	paths of the files to rewrite.
`type`	the kind of code each file holds, parallel to `file`.
`wrap_fun`	`function(package, file, type, body)` returning the new contents, for instance from `wrap_using_template()`.
`quiet`	do not report each file as it is rewritten.

Value

Called for its side effect of rewriting `file`.

`wrap_using_template` *Build a wrapper function from a template*

Description

Turns a template string into the `function(package, file, type, body)` that `extract_package_code()` and `wrap_files()` expect.

Usage

```
wrap_using_template(template)
```

Arguments

`template`	the template. The first occurrence of each of `.PACKAGE.`, `.FILE.`, `.TYPE.` and `.BODY.` is replaced with the corresponding argument.

Value

A `function(package, file, type, body)` returning the filled-in template.

Index