Package 'reproducr'

June 20, 2026

Title Behavioural Reproducibility Auditing for R Projects

Version 0.2.0

Description Audits R scripts for behavioural reproducibility risk. Scans scripts for qualified package::function calls and checks them against a curated database of known silent breaking changes across popular CRAN packages. Flags stochastic calls lacking set.seed() and detects locale-sensitive operations that may produce different results across systems. Supports baseline certification of analytical outputs so that silent numerical drift can be detected across package upgrades or platform changes. Generates human-readable audit reports suitable for academic submission or pharmaceutical QC workflows. For more details see <https://github.com/repro-stats/reproducr>.

License MIT + file LICENSE

Encoding UTF-8

Language en-GB

RoxygenNote 7.3.3

Depends R (>= 4.0.0)

Imports utils

Suggests digest (>= 0.6.0), jsonlite, commonmark, testthat (>= 3.0.0), knitr, rmarkdown, covr

Config/testthat/edition 3

VignetteBuilder knitr

URL https://github.com/repro-stats/reproducr

BugReports https://github.com/repro-stats/reproducr/issues

NeedsCompilation no

Author Ndoh Penn [aut, cre] (ORCID: <https://orcid.org/0009-0003-9054-465X>)

Maintainer Ndoh Penn <ndohpenn9@gmail.com>

Repository CRAN

Date/Publication 2026-06-20 14:10:02 UTC

Contents

`reproducr-package` *reproducr: Behavioural Reproducibility Auditing for R Projects*

Description

You finish an analysis. The code runs. The numbers look right. But are they stable?

`reproducr` makes behavioural reproducibility risks visible and trackable. It scans your scripts for known silent breaking changes, flags stochastic calls missing `set.seed()`, certifies analytical outputs as baselines, and detects numerical drift across runs.

Workflow

Tier 1 – Scan & score

```
report <- audit_script("analysis.R")
risks  <- risk_score(report)
print(risks)
```

Tier 2 – Baseline & drift

```
model <- lm(mpg ~ wt, data = mtcars)
certify(list(coefs = coef(model)), tag = "submission-v1")

# Later, after any environment change:
check_drift(list(coefs = coef(model)), against = "submission-v1")
```

Tier 3 – Report & export

```
repro_report(report, risks, format = "html", style = "pharma")
repro_badge(report, risks, output = "README")
```

Key functions

Function	Purpose
audit_script()	Parse a script and extract all pkg::fn calls
risk_score()	Check calls against the breaking-changes database
certify()	Hash and store analytical outputs as a baseline
check_drift()	Compare current outputs against a stored baseline
list_certs()	List all certifications in a .reproducr file
repro_report()	Render a human-readable audit report
repro_badge()	Generate a reproducibility status badge
check_db_staleness()	Check database entries against current CRAN versions

The breaking-changes database

The internal database covers known silent breaking changes in: dplyr, tidyr, ggplot2, readr, purrr, stringr, broom, data.table, lme4, lubridate, and base R. Community contributions are welcome – see vignette("contributing-to-the-database").

The database is kept current via a weekly GitHub Actions workflow that calls check_db_staleness() and opens an issue automatically when any entry's to_version ceiling falls below the current CRAN release.

Author(s)

Maintainer: Ndoh Penn <ndohpenn9@gmail.com> (ORCID)

See Also

Useful links:

- https://github.com/repro-stats/reproducr
- Report bugs at https://github.com/repro-stats/reproducr/issues

---

audit_script *Audit an R script for reproducibility risks*

Description

Parses one or more R source files and extracts every qualified package::function call, resolving the installed version of each package. The resulting audit_report object is the entry point for the rest of the reproducr workflow.

Usage

```
audit_script(path = ".", renv = TRUE, verbose = TRUE)

## S3 method for class 'audit_report'
print(x, ...)

## S3 method for class 'audit_report'
summary(object, ...)
```

Arguments

path	character(1). Path to a .R, .Rmd, or .qmd file **or** a directory. When a directory is supplied, all R-ish source files are scanned recursively, excluding renv/ and packrat/ subdirectories. Defaults to "." (the current working directory).
renv	logical(1). If TRUE and a renv.lock file exists in the current working directory, package versions are read from the lockfile rather than the currently installed library. Useful for stable version reporting in CI environments. Default TRUE.
verbose	logical(1). Whether to print progress messages. Default TRUE.
x	An audit_report object (for print).
...	Additional arguments (currently unused).
object	An audit_report object (for summary).

Value

An S3 object of class "audit_report", a list containing:

calls A data.frame with one row per detected pkg::fn call, columns file, line, pkg, fn, pkg_version.

env A list with R version, platform, OS, locale, and timezone.

renv_used logical – were versions sourced from a lockfile?

timestamp POSIXct timestamp of when the audit was run.

paths Character vector of files that were scanned.

Detection approach

audit_script() uses regular-expression matching on source text to extract qualified calls of the form pkg::fn or pkg:::fn. It intentionally skips comment lines (lines beginning with #, after trimming whitespace). For more robust analysis, tools that operate on the parse tree (e.g. lintr) should be used alongside reproducr.

What counts as a qualifying call?

Only *qualified* calls – those using :: or ::: – are detected. Unqualified calls (e.g. filter(df, x > 0) without dplyr::) are not detected because he package cannot be determined unambiguously from source text alone. This is by design: qualifying calls is also a reproducibility best practice.

See Also

risk_score() to check detected calls against the breaking-changes database; repro_report() to render the full audit; certify() to lock a set of outputs as a baseline.

Examples

```
# Write a temporary script to audit
script <- tempfile(fileext = ".R")
writeLines(c(
  "set.seed(237)",
  "x <- dplyr::filter(mtcars, cyl == 4)",
  "y <- dplyr::summarise(x, mean_mpg = mean(mpg))",
  "z <- stats::rnorm(nrow(y))"
), script)

report <- audit_script(script, renv = FALSE, verbose = FALSE)
print(report)

# See the detected calls as a data frame
report$calls
```

---

certify *Certify analytical outputs as a reproducibility baseline*

Description

Hashes a named list of R objects (model coefficients, summary statistics, key scalars, data frames) and saves them alongside full environment metadata to a local certification file (.reproducr.rds by default). Later runs can call check_drift() to verify that results have not changed.

Think of certify() as a "signed receipt" for a completed analysis run.

Usage

```
certify(outputs, tag, script = NULL, file = ".reproducr")
```

Arguments

outputs	A fully named list of R objects to certify. Each element is hashed using SHA-256 (or a base-R fallback if digest is not available). Common choices: coef(model), summary(model)$r.squared, a results data.frame, or any key scalar.
tag	character(1). A human-readable label for this certification, e.g. "submission-v1" or "pre-review". Tags must be unique within a certification file; passing a duplicate tag overwrites the existing record with a warning.
script	character(1) or NULL. Path to the script that produced these outputs. Used for documentation in the certification record only; not validated. Default NULL.
file	character(1). Base path for the certification store. The actual file written is paste0(file, ".rds"). Default ".reproducr", which writes .reproducr.rds in the current working directory. Commit this file to version control.

Value

Invisibly returns the certification record (a list). Prints a one-line summary to the console.

Certification store

All certifications for a project are accumulated in a single `.reproducr.rds` file. You can have multiple tags representing different stages (e.g. before and after peer review). Use `list_certs()` to inspect stored tags.

Version control

Commit `.reproducr.rds` to your project's version control repository. This makes the certification auditable and shareable with collaborators.

See Also

`check_drift()` to compare current outputs against a baseline; `list_certs()` to inspect stored certifications.

Examples

```
model <- lm(mpg ~ wt, data = mtcars)

cert_file <- tempfile()

certify(
  outputs = list(
    coefs     = coef(model),
    r_squared = summary(model)$r.squared,
    n_obs     = nrow(mtcars)
  ),
  tag = "baseline-v1",
  script = "analysis.R",
  file = cert_file
)

# See what is stored
list_certs(file = cert_file)
```

---

check_db_staleness *Check whether breaking-changes database entries are stale*

Description

Compares the `to_version` ceiling and `from_version` floor of each entry in the breaking-changes database against the current version of that package on CRAN. Two types of staleness are detected:

- `stale_ceiling` – the package has released a new version above the `to_version` ceiling. The window may need extending.
- `stale_floor` – the current CRAN version is so far ahead of `from_version` that the window captures users who are already well past the breaking-change transition. The entry may need closing or the `from_version` floor raising.

This function is primarily intended for use by `reproducr` maintainers and contributors. It is also run as a scheduled GitHub Actions workflow on the `reproducr` repository to automatically open issues when staleness is detected.

Usage

```
check_db_staleness(
  packages = NULL,
  verbose = TRUE,
  source = "cran",
  from_version_major_threshold = 1L
)
```

Arguments

`packages`	`character` or `NULL`. Package names to check. If `NULL` (the default), all packages tracked in the breaking-changes database are checked.
`verbose`	`logical(1)`. Print progress messages. Default `TRUE`.
`source`	`character(1)`. Where to resolve current package versions. One of: `"cran"` Query the CRAN package database via `utils::available.packages()`. Requires an internet connection. `"installed"` Use locally installed versions via `utils::packageDescription()`. Fast and offline, but only reflects what is installed on the current machine. Default `"cran"`.
`from_version_major_threshold`	`integer(1)` or `Inf`. Number of full major versions the current CRAN release must be *ahead* of `from_version` before the entry is flagged as having a stale floor. Set to `Inf` to disable this check. Default `1L`.

Value

A `data.frame` of class `c("staleness_report", "data.frame")` with one row per database entry. Columns:

`key` The `pkg::fn` key.

`pkg` Package name.

`fn` Function name.

`from_version` The floor version currently in the database.

`to_version` The ceiling version currently in the database.

`current_version` The current version on CRAN or installed.

`status` One of `"ok"`, `"stale_ceiling"`, `"stale_floor"`, or `"unknown"`.

`gap` Description of the version gap. `NA` when status is `"ok"` or `"unknown"`.

Rows are ordered: stale_ceiling first, stale_floor second, then ok, then unknown.

See Also

`risk_score()` which uses the database at runtime; `vignette("contributing-to-the-database")` for the database schema and version window design principles.

Examples

```
# Check all tracked packages against CRAN
report <- check_db_staleness()
print(report)

# Check specific packages only
check_db_staleness(packages = c("dplyr", "tidyr"))

# Offline check using installed versions
check_db_staleness(source = "installed")

# Filter to stale entries only
report <- check_db_staleness()
report[report$status != "ok", ]
```

`check_drift` *Check analytical outputs for drift against a certified baseline*

Description

Re-hashes a set of named R objects and compares them against a previously stored certification. Reports which outputs are unchanged (`"ok"`), have changed (`"drifted"`), are present in the baseline but not supplied (`"missing"`), or are new outputs not in the baseline (`"new"`).

Usage

```
check_drift(
  outputs,
  against = "latest",
  file = ".reproducr",
  tolerance = 1e-10
)
```

Arguments

outputs	A fully named list of current R objects – the same names used in the certify() call being compared against.
against	character(1). The certification tag to compare against. Use "latest" (the default) to automatically select the most recently added certification.
file	character(1). Base path of the certification store. Default ".reproducr" (reads .reproducr.rds).
tolerance	numeric(1). Numeric tolerance applied to hash comparison. When > 0, outputs whose hashes differ are also compared element-wise (for numeric vectors/matrices), and flagged as "ok" if the maximum absolute difference is within tolerance. Set to 0 for exact matching only. Default 1e-10.

Value

Invisibly returns a data.frame of class c("drift_report", "data.frame") with columns output, status ("ok", "drifted", "missing", "new"), max_delta, and note. Also emits a summary via message().

See Also

certify() to create a baseline; list_certs() to see available tags.

Examples

```
cert_file <- tempfile()
model <- lm(mpg ~ wt, data = mtcars)

certify(list(coefs = coef(model)), tag = "v1", file = cert_file)

# Same outputs -- should report "ok"
result <- check_drift(list(coefs = coef(model)),
  against = "v1", file = cert_file
)
print(result)

# Different model -- should report "drifted"
model2 <- lm(mpg ~ hp, data = mtcars)
check_drift(list(coefs = coef(model2)),
  against = "v1", file = cert_file
)
```

---

list_certs *List all certifications stored in a certification file*

Description

A convenience function to inspect what certification tags are stored and their key metadata, without needing to read the raw `.rds` file.

Usage

```
list_certs(file = ".reproducr")
```

Arguments

`file`	`character(1)`. Base path of the certification store. Default `".reproducr"`.

Value

A `data.frame` with columns `tag`, `timestamp`, `r_version`, `os`, `n_outputs`, `script` – one row per certification. Returns an empty data frame if no certifications exist.

Examples

```
cert_file <- tempfile()
model <- lm(mpg ~ wt, data = mtcars)

certify(list(coefs = coef(model)), tag = "v1", file = cert_file)
certify(list(coefs = coef(model)), tag = "v2", file = cert_file)

list_certs(file = cert_file)
```

`repro_badge` *Generate a reproducibility status badge*

Description

Produces a shields.io Markdown badge reflecting the current reproducibility status of a project. The badge is colour-coded:

- **Green** (`reproducible`) – no risks detected.
- **Yellow** (`caution`) – medium-severity risks only.
- **Red** (`at risk`) – one or more high-severity risks or drifted outputs.
- **Grey** (`unknown`) – no risk information supplied.

Can be inserted automatically into a `README.md` (e.g. from a GitHub Actions workflow).

Usage

```
repro_badge(
  audit,
  risks = NULL,
  drift = NULL,
  output = "markdown",
  readme_path = "README.md"
)
```

Arguments

audit	An audit_report from audit_script().
risks	A risk_report from risk_score(). Optional.
drift	A drift_report from check_drift(). Optional.
output	character(1). "markdown" (return the badge string) or "README" (insert/update the badge in README.md). Default "markdown".
readme_path	character(1). Path to the README file when output = "README". Default "README.md".

Value

Invisibly returns the badge Markdown string.

See Also

repro_report(), risk_score(), check_drift()

Examples

```
script <- tempfile(fileext = ".R")
writeLines("x <- dplyr::filter(mtcars, cyl == 4)", script)
report <- audit_script(script, renv = FALSE, verbose = FALSE)
risks <- risk_score(report)

badge <- repro_badge(report, risks)
cat(badge)
```

---

repro_report *Generate a human-readable reproducibility report*

Description

Renders a reproducibility audit report from an `audit_script()` result and optionally a `risk_score()` result and `check_drift()` result. Three style presets are available:

- `"minimal"` – compact summary suitable for console review or internal project documentation.
- `"academic"` – generates a ready-to-paste methods paragraph for journal submissions, listing all packages with versions and summarising risk findings.
- `"pharma"` – structured QC document with a risk register and sign-off fields, suitable for pharmaceutical or regulated analytical workflows.

Usage

```
repro_report(
  audit,
  risks = NULL,
  drift = NULL,
  format = "text",
  style = "minimal",
  output_file = NULL
)
```

Arguments

`audit`	An `audit_report` object from `audit_script()`. Required.
`risks`	A `risk_report` data frame from `risk_score()`. Optional but strongly recommended – without it, the report cannot assess reproducibility.
`drift`	A `drift_report` data frame from `check_drift()`. Optional.
`format`	`character(1)`. Output format: `"text"` (console), `"md"` (Markdown file), or `"html"` (HTML file). Default `"text"`.
`style`	`character(1)`. Report style: `"minimal"`, `"academic"`, or `"pharma"`. Default `"minimal"`.
`output_file`	`character(1)` or `NULL`. Output file path (used for `format = "md"` and `format = "html"`). If `NULL`, a sensible default name is used (`"reproducr_report.md"` / `"reproducr_report.html"`).

Value

Invisibly returns the report content as a character string. For file-based formats, the file is also written to disk.

See Also

`audit_script()`, `risk_score()`, `check_drift()`, `repro_badge()`

Examples

```
script <- tempfile(fileext = ".R")
writeLines(c(
  "set.seed(237)",
  "x <- dplyr::filter(mtcars, cyl == 4)",
  "y <- stats::rnorm(10)"
), script)

report <- audit_script(script, renv = FALSE, verbose = FALSE)
risks <- risk_score(report)

# Console summary
repro_report(report, risks, format = "text", style = "minimal")

# Academic methods paragraph (printed, not written to file)
cat(repro_report(report, risks, format = "text", style = "academic"))
```

`risk_score` *Score function calls for reproducibility risk*

Description

Takes an `audit_report` and checks every detected `pkg::fn` call against three independent checks:

- `"changelog"` – matches against a curated database of known breaking changes in popular CRAN packages, flagging calls where the installed version falls in a known-risky version window.
- `"seed_check"` – flags stochastic functions (`rnorm`, `sample`, etc.) where no `set.seed()` appears within 50 lines above the call.
- `"locale_check"` – flags functions whose output is locale-sensitive (`sort()`, `format()`, `tolower()`, etc.).

Usage

```
risk_score(
  audit,
  methods = c("changelog", "seed_check", "locale_check"),
  min_risk = "low",
  major_version_grace = 1L
)

## S3 method for class 'risk_report'
print(x, ...)

## S3 method for class 'risk_report'
as.data.frame(x, ...)
```

```
## S3 method for class 'risk_report'
x[i, j, ...]
```

Arguments

audit	An audit_report object returned by audit_script().
methods	character. Which checks to run. Any combination of "changelog", "seed_check", "locale_check". Default: all three.
min_risk	character(1). Minimum risk level to include in the output. One of "low" (show all), "medium", or "high". Default "low".
major_version_grace	integer(1) or Inf. Number of full major versions the installed package must be *ahead* of from_version before the entry is suppressed entirely. When the installed version is this many or more major versions newer than from_version, the user is already past the breaking-change transition and the flag is a false positive – the entry is silently dropped from the results. Set to Inf to disable. Default 1L.
x	A risk_report object (for print, as.data.frame, and [).
...	Additional arguments (currently unused).
i	Row index.
j	Column index. When columns are subsetted and required columns are removed, the "risk_report" class is stripped so that print.risk_report() is not called on an incomplete object.

Value

A data.frame of class c("risk_report", "data.frame") with one row per flagged call. Columns:

file Source file path.

line Line number of the call.

call The pkg::fn string.

pkg_version Installed or lockfile-resolved version.

risk "high", "medium", or "low".

check Which check flagged it: "changelog", "seed_check", or "locale_check".

description Plain-English explanation of the risk.

reference URL to the relevant changelog or documentation.

Rows are ordered by risk severity (high first), then by file and line. If no risks are found, an empty data frame with the same columns is returned.

Version windows

The "changelog" check uses a half-open version window (from_ver, to_ver]: a call is flagged only if the installed version is *greater than* from_ver *and at most* to_ver. This means the risk is scoped to versions where the breaking change is known to apply.

Major version grace

When an installed version is `major_version_grace` or more major versions ahead of `from_version`, the entry is suppressed entirely. The user is already past the breaking-change transition – flagging it at any severity would be a false positive. The database staleness check (`check_db_staleness()`) handles the maintenance concern of identifying entries whose `from_version` floor is too old.

See Also

`audit_script()` to generate the input; `repro_report()` to render the results; `check_db_staleness()` to identify database entries with windows that are too wide.

Examples

```
script <- tempfile(fileext = ".R")
writeLines(c(
  "x <- dplyr::summarise(mtcars, n = dplyr::n())",
  "y <- stats::rnorm(100)",
  "z <- base::sort(letters)"
), script)

report <- audit_script(script, renv = FALSE, verbose = FALSE)
risks <- risk_score(report)
print(risks)

# High-severity items only
risk_score(report, min_risk = "high")

# Only the changelog check
risk_score(report, methods = "changelog")
```

Index