

Package: cranlint (via r-universe)

October 5, 2026

Title Static Checks for Common CRAN Submission Mistakes

Version 0.0.0.9000

Description A personal-use package that runs static checks against a package source tree to catch recurring mistakes made before CRAN submission, such as DESCRIPTION formatting issues, fixed random seeds, and leftover debugging statements. Complements tools like 'checkhelper' and 'rcmdcheck' rather than replacing them.

License MIT + file LICENSE

URL <https://github.com/djnavarro/cranlint>,
<https://cranlint.djnavarro.net/>

BugReports <https://github.com/djnavarro/cranlint/issues>

Encoding UTF-8

Roxygen list(markdown = TRUE)

Imports desc, tibble

Suggests checkhelper, pkgdown, spelling, testthat (>= 3.0.0)

Config/testthat/edition 3

Config/Needs/website djnavarro/waeponwifestre

Language en-US

Config/roxygen2/version 8.1.0

Repository <https://djnavarro.r-universe.dev>

Date/Publication 2026-10-05 01:16:20 UTC

RemoteUrl <https://github.com/djnavarro/cranlint>

RemoteRef HEAD

RemoteSha 9f529080087fe2a054d0514e0a5b943661b8083d

Contents

cl_check_authors_r	2
cl_check_description_length	3
cl_check_doi_formatting	4
cl_check_dontrun_usage	4
cl_check_global_env_write	6
cl_check_hardcoded_seed	6
cl_check_installed_packages	7
cl_check_license_file	8
cl_check_missing_value	9
cl_check_option_restoration	10
cl_check_quoted_function_names	11
cl_check_quoted_software_names	11
cl_check_title_case	12
cl_check_verbose_output	13
cl_check_warn_suppression	14
cl_example_pkg	15
lint_cran	16
Index	18

cl_check_authors_r	<i>Check DESCRIPTION's Authors@R usage</i>
--------------------	--

Description

Flags a missing Authors@R field, and flags manually-specified Author/Maintainer fields that disagree with what R would generate from Authors@R – per the CRAN Cookbook, that disagreement results in automatic rejection.

Usage

```
cl_check_authors_r(path = ".")
```

Arguments

path Path to the package root. Defaults to the current directory.

Value

A tibble following the cranlint check-result contract; see AGENTS.md.

Examples

```
pkg_dir <- cl_example_pkg(  
  description = c(  
    Author = "Someone Else",  
    Maintainer = "Someone Else <someone@example.com>"  
  )  
)  
cl_check_authors_r(pkg_dir)  
unlink(pkg_dir, recursive = TRUE)
```

`cl_check_description_length`

Check that DESCRIPTION's Description field is a full paragraph

Description

Flags a Description field that reads as a single short sentence rather than the short paragraph (2+ sentences) the CRAN Cookbook recommends. Sentence counting is a heuristic (splitting on ./!/? followed by whitespace or end-of-string) and can be thrown off by abbreviations; treat findings as a prompt to review, not gospel.

Usage

```
cl_check_description_length(path = ".")
```

Arguments

path	Path to the package root (the directory containing DESCRIPTION). Defaults to the current directory.
------	---

Value

A tibble following the cranlint check-result contract; see AGENTS.md.

Examples

```
pkg_dir <- cl_example_pkg(  
  description = c(Description = "Does a thing.")  
)  
cl_check_description_length(pkg_dir)  
unlink(pkg_dir, recursive = TRUE)
```

`cl_check_doi_formatting`*Check DOI/URL reference formatting in the Description field*

Description

Flags `<doi: ...>/<https: ...>` references in the Description field that have whitespace right after the opening angle bracket or right after the `doi:/https:` prefix, which breaks CRAN's auto-linking of the reference. This is a text-matching heuristic on `<...>` spans that look like a reference; it can't verify the reference is otherwise well-formed (e.g. a real DOI).

Usage

```
cl_check_doi_formatting(path = ".")
```

Arguments

`path` Path to the package root. Defaults to the current directory.

Value

A tibble following the `cranlint` check-result contract; see `AGENTS.md`.

Examples

```
pkg_dir <- cl_example_pkg(  
  description = c(  
    Description = paste(  
      "See < doi:10.1000/xyz123> for background.",  
      "It has no other dependencies."  
    )  
  )  
)  
cl_check_doi_formatting(pkg_dir)  
unlink(pkg_dir, recursive = TRUE)
```

`cl_check_dontrun_usage`*Check for \dontrun{} usage in .Rd examples*

Description

Flags every line containing `\dontrun{` in a package's `man/*.Rd` files. This is a soft, review-level finding rather than a hard fail: `\dontrun{}` is a legitimate way to mark example code that genuinely can't be run during checking (e.g. it needs credentials, a network resource, or is illustrative pseudo-code), but it's also easy to reach for out of habit for code that could run fine, or run via `\donttest{}` instead (which CRAN does run, just not during every regular check) – and code inside `\dontrun{}` is never executed by R CMD check, so it can silently rot. Static analysis can't tell which case applies, so every occurrence is surfaced for a human to judge.

Usage

```
cl_check_dontrun_usage(path = ".")
```

Arguments

path	Path to the package root. Defaults to the current directory.
------	--

Details

Matching is a plain line-based text search for the literal `\dontrun{` markup within an `\examples{}` block, not a full Rd parse – it doesn't distinguish an active `\dontrun{}` from one that's commented out with `%`, and reports one finding per line containing the markup rather than per `\dontrun{}` block.

Value

A tibble following the `cranlint` check-result contract; see `AGENTS.md`.

Examples

```
pkg_dir <- cl_example_pkg(  
  man_files = list(foo.Rd = c(  
    "\\name{foo}",  
    "\\alias{foo}",  
    "\\title{Foo}",  
    "\\examples{",  
    "\\dontrun{",  
    "foo()",  
    "}",  
    "}"  
  ))  
)  
cl_check_dontrun_usage(pkg_dir)  
unlink(pkg_dir, recursive = TRUE)
```

cl_check_global_env_write

Check for writes to enclosing/global environments via «- or -»

Description

Flags <<-/->> usage in R/. This operator writes to the nearest enclosing environment with an existing binding of the same name, or to .GlobalEnv if none exists – which the CRAN Repository Policy explicitly forbids. Package code that only uses <<- to update a variable in a known parent scope (e.g. a closure factory) is technically safe, but this check can't distinguish that case from an accidental global write without deeper scope analysis, so every use is flagged for review.

Usage

```
cl_check_global_env_write(path = ".")
```

Arguments

path Path to the package root. Defaults to the current directory.

Value

A tibble following the cranlint check-result contract; see AGENTS.md.

Examples

```
pkg_dir <- cl_example_pkg(
  r_files = list(foo.R = c(
    "set_flag <- function() {",
    "  flag <<- TRUE",
    "}"
  ))
)
cl_check_global_env_write(pkg_dir)
unlink(pkg_dir, recursive = TRUE)
```

cl_check_hardcoded_seed

Check for hardcoded random seeds in package code

Description

Flags set.seed() calls in R/ that pass a literal, unchangeable number (e.g. set.seed(42), set.seed(seed = 42L), set.seed(-5)) as any argument. Setting a seed a user can't override or opt out of, inside a function, is a common CRAN rejection reason. Calls in tests/, \examples, vignettes/, and demos are out of scope – and not scanned at all, since checks only look at R/ – because setting a seed there is expected and recommended for reproducibility.

Usage

```
cl_check_hardcoded_seed(path = ".")
```

Arguments

path Path to the package root. Defaults to the current directory.

Value

A tibble following the cranlint check-result contract; see AGENTS.md.

Examples

```
pkg_dir <- cl_example_pkg(  
  r_files = list(simulate.R = c(  
    "simulate <- function() {",  
    "  set.seed(42)",  
    "  rnorm(1)",  
    "}"  
  ))  
)  
cl_check_hardcoded_seed(pkg_dir)  
unlink(pkg_dir, recursive = TRUE)
```

cl_check_installed_packages

Check for calls to installed.packages()

Description

Flags any call to `installed.packages()` in R/. Per the CRAN Cookbook, this function can be slow (it reads several files per installed package) and shouldn't be used to test whether a package is available; `requireNamespace()/require()` are the recommended replacements (or `find.package()/system.file()` to locate one, `packageDescription()` for details of a small number of packages).

Usage

```
cl_check_installed_packages(path = ".")
```

Arguments

path Path to the package root. Defaults to the current directory.

Value

A tibble following the cranlint check-result contract; see AGENTS.md.

Examples

```
pkg_dir <- cl_example_pkg(
  r_files = list(foo.R = c(
    "has_pkg <- function(pkg) {",
    "  pkg %in% rownames(installed.packages())",
    "}"
  ))
)
cl_check_installed_packages(pkg_dir)
unlink(pkg_dir, recursive = TRUE)
```

cl_check_license_file *Check for unnecessary "+ file LICENSE" references*

Description

Flags a License field that references + file LICENSE alongside a base license that isn't one of the CRAN templates requiring an additional file. Per the CRAN Cookbook, most licenses are bundled with R itself and don't need a LICENSE file in the package; only MIT, BSD_2_clause, and BSD_3_clause are templates that do. A LICENSE file is otherwise only needed when there are additional attribution requirements or restrictions beyond the base license, which this check can't detect – so a finding here is a prompt to review, not an automatic removal.

Usage

```
cl_check_license_file(path = ".")
```

Arguments

path Path to the package root. Defaults to the current directory.

Value

A tibble following the cranlint check-result contract; see AGENTS.md.

Examples

```
pkg_dir <- cl_example_pkg(
  description = c(License = "GPL-3 + file LICENSE")
)
cl_check_license_file(pkg_dir)
unlink(pkg_dir, recursive = TRUE)
```

`cl_check_missing_value`*Check for missing \value/@return documentation*

Description

Flags exported functions and S3 methods whose generated .Rd file would have no (or an empty) \value tag – the cause of the CRAN Cookbook’s “**Missing \\value tags in .Rd files**” and the “Please add \value to .Rd files” rejection message. This check does not reimplement the detection logic itself (see AGENTS.md) – it wraps `checkhelper::audit_tags()` and reshapes its output into the standard cranlint check-result contract.

Usage

```
cl_check_missing_value(path = ".")
```

Arguments

`path` Path to the package root. Defaults to the current directory.

Details

`checkhelper::audit_tags()` works by actually loading the package’s namespace and re-running `roxygen2::roxygenise()` against it, which writes `NAMESPACE/man/*.Rd` and may touch `DESCRIPTION`. To keep this check read-only from the caller’s point of view (matching every other `cl_check_*`()), it runs against a disposable copy of `path` in a temporary directory rather than `path` itself; nothing under `path` is modified.

Value

A tibble following the cranlint check-result contract; see AGENTS.md.

Examples

```
pkg_dir <- cl_example_pkg(  
  description = c(Encoding = "UTF-8"),  
  r_files = list(foo.R = c(  
    "' Foo",  
    "'",  
    "' @export",  
    "foo <- function() 1"  
  )),  
  man_files = list()  
)  
dir.create(file.path(pkg_dir, "man"))  
cl_check_missing_value(pkg_dir)  
unlink(pkg_dir, recursive = TRUE)
```

cl_check_option_restoration

Check for unrestored par()/options()/setwd() changes

Description

Flags a `par()`, `options()`, or `setwd()` call in R/ that changes state without a paired `on.exit()` call in the same function frame to restore it. Per the CRAN Cookbook, packages shouldn't permanently change a user's graphical parameters, options, or working directory; if a function has to, it should save the old value and restore it with `on.exit()` immediately.

Usage

```
cl_check_option_restoration(path = ".")
```

Arguments

`path` Path to the package root. Defaults to the current directory.

Details

`setwd()` always counts as a change. For `par()/options()`, a bare call with no arguments, or a call whose only arguments are unnamed (e.g. `par("mfrow")`, or `par(oldpar)/options(oldopts)` restoring a previously-saved list) is treated as a query/restore rather than a change and isn't flagged; `par(no.readonly = TRUE)` is also excluded as it's the standard snapshot-for-later-restore idiom, not a change by itself.

This is a coarse heuristic: it doesn't check that the `on.exit()` call actually restores the *same* thing that was changed, or that the restore happens immediately after the change – only that both are present in the same function frame. A change with no enclosing function at all is always flagged, since `on.exit()` couldn't apply there anyway.

Value

A tibble following the `cranlint` check-result contract; see `AGENTS.md`.

Examples

```
pkg_dir <- cl_example_pkg(
  r_files = list(foo.R = c(
    "plot_thing <- function() {",
    "  par(mfrow = c(1, 2))",
    "  plot(1)",
    "}"
  ))
)
cl_check_option_restoration(pkg_dir)
unlink(pkg_dir, recursive = TRUE)
```

cl_check_quoted_function_names

Check for quoted function names in Title/Description

Description

Flags a single-quoted word in the Title or Description field that matches the name of a function defined in this package's own R/ files, or an exported function from base, stats, utils, or methods. CRAN's single-quote convention is for software/package names, not function names; quoting a function name is a common sign that the wrong thing got quoted (e.g. 'summary' where a software name was meant).

Usage

```
cl_check_quoted_function_names(path = ".")
```

Arguments

path Path to the package root. Defaults to the current directory.

Value

A tibble following the cranlint check-result contract; see AGENTS.md.

Examples

```
pkg_dir <- cl_example_pkg(
  description = c(
    Description = paste(
      "Provides a fast 'summary' method.",
      "It has no other dependencies."
    )
  )
)
cl_check_quoted_function_names(pkg_dir)
unlink(pkg_dir, recursive = TRUE)
```

cl_check_quoted_software_names

Check that the package's own dependencies are quoted in Title/Description

Description

Flags a package name from this package's own Depends/Imports/ Suggests/LinkingTo fields (excluding "R" itself) that appears unquoted in the Title or Description field. Per the CRAN Cookbook, software and package names should be wrapped in single quotes so the automatic spell check doesn't flag them.

Usage

```
cl_check_quoted_software_names(path = ".")
```

Arguments

path Path to the package root. Defaults to the current directory.

Details

The candidate list is deliberately narrow: rather than a general list of known software/API names (high false-positive risk – see `.agents/PLAN.md`), it's limited to packages this package actually declares a dependency on, which keeps precision high. It also means the check under-reports (e.g. it won't catch an unquoted "Python"). A plain word matching a dependency's name doesn't always refer to the package either, so findings are advisory rather than a firm "fix this."

Value

A tibble following the `cranlint` check-result contract; see `AGENTS.md`.

Examples

```
pkg_dir <- cl_example_pkg(
  description = c(
    Description = paste(
      "Wraps jsonlite for convenient parsing.",
      "It has no other dependencies."
    ),
    Imports = "jsonlite"
  )
)
cl_check_quoted_software_names(pkg_dir)
unlink(pkg_dir, recursive = TRUE)
```

cl_check_title_case	<i>Check that DESCRIPTION's Title field is in Title Case</i>
---------------------	--

Description

Compares the `Title` field against `tools::toTitleCase()`'s suggestion and flags a mismatch. Title Case judgement genuinely depends on author intent (proper nouns, quoted software names, etc. keep their original casing), so treat a finding as a prompt to review rather than an automatic rewrite.

Usage

```
cl_check_title_case(path = ".")
```

Arguments

path Path to the package root. Defaults to the current directory.

Value

A tibble following the cranlint check-result contract; see AGENTS.md.

Examples

```
pkg_dir <- cl_example_pkg(  
  description = c(Title = "an example package")  
)  
cl_check_title_case(pkg_dir)  
unlink(pkg_dir, recursive = TRUE)
```

`cl_check_verbose_output`

Check for print()/cat() calls outside print/format/summary methods

Description

Flags a `print()` or `cat()` call in R/ that isn't nested inside a function whose name looks like a `print.*/format.*/summary.*` S3 method. Per the CRAN Cookbook, these produce console output a user can't suppress; CRAN's own exception is printing inside `print`, `summary`, and similar methods, where it's the whole point of the function.

Usage

```
cl_check_verbose_output(path = ".")
```

Arguments

`path` Path to the package root. Defaults to the current directory.

Details

Two things this check can't detect, so it will over-report relative to what CRAN actually requires: a `cat()` call writing to a file/connection rather than the console (e.g. `cat(x, file = "out.txt")`) is still flagged, and the Cookbook's other accepted mitigation – gating the call behind a `verbose` argument, e.g. `if (verbose) cat(...)` – isn't recognized as an exemption. Both are worth a manual look before deciding a finding is a real problem.

Value

A tibble following the cranlint check-result contract; see AGENTS.md.

Examples

```
pkg_dir <- cl_example_pkg(
  r_files = list(foo.R = c(
    "process <- function(x) {",
    "  cat(\"processing...\n\")",
    "  x",
    "}"
  ))
)
cl_check_verbose_output(pkg_dir)
unlink(pkg_dir, recursive = TRUE)
```

```
cl_check_warn_suppression
  Check for options(warn = )
```

Description

Flags `options()` calls that set `warn` to a negative value, which globally suppresses all warnings for the rest of the session and can't be limited to a specific expression. Per the CRAN Cookbook, this is not allowed even when the option is immediately restored afterwards; `suppressWarnings()` around the specific expression is the recommended replacement.

Usage

```
cl_check_warn_suppression(path = ".")
```

Arguments

`path` Path to the package root. Defaults to the current directory.

Value

A tibble following the `cranlint` check-result contract; see `AGENTS.md`.

Examples

```
pkg_dir <- cl_example_pkg(
  r_files = list(foo.R = c(
    "quietly <- function() {",
    "  options(warn = -1)",
    "}"
  ))
)
cl_check_warn_suppression(pkg_dir)
unlink(pkg_dir, recursive = TRUE)
```

cl_example_pkg

Scaffold a minimal package on disk for trying out cranlint checks

Description

Writes a valid, minimal package (a DESCRIPTION file, and optionally R/ and man/ files) to a new temporary directory. This exists to support the runnable @examples on `cl_check_*/lint_cran()`, and to make it easy to try cranlint out interactively, without needing a real package on disk – it is not part of the actual linting logic.

Usage

```
cl_example_pkg(description = character(), r_files = list(), man_files = list())
```

Arguments

description	A named character vector of DESCRIPTION fields to add or override on top of a minimal default (Package, Title, Version, Authors@R, Description, License).
r_files	A named list mapping a filename (e.g. "foo.R") to a character vector of lines, written under R/.
man_files	A named list mapping a filename (e.g. "foo.Rd") to a character vector of lines, written under man/.

Value

The path to the scaffolded package (a tempdir). Callers should `unlink(path, recursive = TRUE)` once done with it.

Examples

```
pkg_dir <- cl_example_pkg(
  description = c(Description = "Does a thing."),
  r_files = list(simulate.R = c(
    "simulate <- function() {",
    "  set.seed(42)",
    "  rnorm(1)",
    "}"
  ))
)
lint_cran(pkg_dir)
unlink(pkg_dir, recursive = TRUE)
```

lint_cran	<i>Run all cranlint checks against a package</i>
-----------	--

Description

Runs every `cl_check_*()` check (DESCRIPTION issues like title case and Authors@R formatting, and code/doc issues like hardcoded seeds and `.GlobalEnv` writes) against path and combines their results into a single tibble. This is `cranlint`'s single top-level entry point, mirroring the role `lintr::lint()` plays for that package.

Usage

```
lint_cran(path = ".")
```

Arguments

`path` Path to the package root. Defaults to the current directory.

Details

If a check errors – for example, because path has no DESCRIPTION file at all – that error propagates rather than being caught and turned into a result row, since it signals something more fundamental than an individual finding. A single unparseable R file, by contrast, is already handled gracefully (skipped with a warning) and does not stop the other checks from running.

Every check other than `cl_check_missing_value()` only reads path. `cl_check_missing_value()` wraps `checkhelper::audit_tags()`, which needs to actually load the package and re-run `roxygen2::roxygenise()`; to keep `lint_cran()` itself side-effect-free on the caller's copy of the package, that check runs against a disposable copy of path rather than path directly (see `cl_check_missing_value()`'s documentation for details). It also relies on the `checkhelper` package (Suggests); if that isn't installed, `lint_cran()` emits a warning and skips just that check (contributing no rows) rather than erroring out entirely, so `lint_cran()` stays usable without it. Calling `cl_check_missing_value()` directly still errors in that case.

Value

A tibble following the `cranlint` check-result contract (see `AGENTS.md`), combining every check's findings. Zero rows if no check reports any findings.

Examples

```
pkg_dir <- cl_example_pkg(
  description = c(Description = "Does a thing."),
  r_files = list(simulate.R = c(
    "simulate <- function() {",
    "  set.seed(42)",
    "  rnorm(1)",
    "}"
  ))
))
```

```
)  
lint_cran(pkg_dir)  
unlink(pkg_dir, recursive = TRUE)
```

Index

`cl_check_authors_r`, [2](#)
`cl_check_description_length`, [3](#)
`cl_check_doi_formatting`, [4](#)
`cl_check_dontrun_usage`, [4](#)
`cl_check_global_env_write`, [6](#)
`cl_check_hardcoded_seed`, [6](#)
`cl_check_installed_packages`, [7](#)
`cl_check_license_file`, [8](#)
`cl_check_missing_value`, [9](#)
`cl_check_option_restoration`, [10](#)
`cl_check_quoted_function_names`, [11](#)
`cl_check_quoted_software_names`, [11](#)
`cl_check_title_case`, [12](#)
`cl_check_verbose_output`, [13](#)
`cl_check_warn_suppression`, [14](#)
`cl_example_pkg`, [15](#)

`lint_cran`, [16](#)