

Package ‘depguard’

October 6, 2026

Title Manifest-Based Dependency Conflict Detection for Sandboxed and Desktop R Sessions

Version 0.2.0

Description Lightweight, offline-first checking of R package dependencies against the currently installed environment, without requiring a full project lockfile. Verifies a declared manifest of package versions, including version constraints declared by transitive dependencies, reports session-level snapshot differences (including stale versions still loaded in a running session), detects packages shadowed by another library, and offers single-package version rollback. Designed for hosted notebooks (e.g. Kaggle, Colab, Binder) where 'renv'-style lockfile ownership is impractical, and equally usable on a normal desktop.

License MIT + file LICENSE

Encoding UTF-8

Language en-US

Depends R (>= 3.5)

Imports cli, stats, tools, utils

Suggests remotes, pak, testthat (>= 3.2.0), withr, knitr, rmarkdown, spelling

Config/testthat/edition 3

VignetteBuilder knitr

URL <https://github.com/sunraycodes/depguard>

BugReports <https://github.com/sunraycodes/depguard/issues>

NeedsCompilation no

Config/roxygen2/version 8.1.0

Author Samruddhi Amol Shah [aut, cre],
Kartik Patel [aut],
Amrit Pal [ctb]

Maintainer Samruddhi Amol Shah <samruddhi.bitnbyte@gmail.com>

Repository CRAN

Date/Publication 2026-10-06 07:30:09 UTC

Contents

dep_check	2
dep_diff	3
dep_env	4
dep_fix	5
dep_healthcheck	6
dep_libraries	7
dep_manifest	8
dep_manifest_freeze	9
dep_manifest_read	10
dep_snapshot	10
Index	12

dep_check	<i>Check the live environment against a dependency manifest</i>
-----------	---

Description

Verifies that the packages declared in a manifest are installed at the required versions, and (by default) walks their transitive dependency tree to catch *real* conflicts: for every package in the tree, each Imports/Depends/LinkingTo constraint such as cli ($\geq 3.4.0$) is compared against the version actually installed. This is the situation that bites in practice, e.g. a newer package was installed but a dependency was left at an older version.

Usage

```
dep_check(  
  manifest = NULL,  
  path = default_manifest_path(),  
  recursive = TRUE,  
  check_cran = FALSE,  
  stop_on_problem = FALSE  
)  
  
## S3 method for class 'depguard_check'  
print(x, all = FALSE, ...)
```

Arguments

- | | |
|-----------|---|
| manifest | A named character vector as returned by dep_manifest() / dep_manifest_read() . If NULL, the manifest is read from path. |
| path | File path to the manifest, used only if manifest is NULL. |
| recursive | Logical; if TRUE (default), also check transitive dependencies of each manifest package, not just the packages listed directly. |

check_cran	Logical; if TRUE, additionally query CRAN for the latest available version of each package (adds <code>cran_latest</code> and <code>outdated</code> columns). Requires network access; if CRAN cannot be reached a warning is shown and the rest of the result is returned unchanged.
stop_on_problem	Logical; if TRUE, raise an error when any requirement is not "ok". Useful in scripts and CI.
x	A <code>depguard_check</code> object.
all	Logical; show every row rather than only problem rows.
...	Passed on to the data frame print method.

Details

Everything is computed from locally installed package metadata, so it works with no internet connection. The network is only used if `check_cran = TRUE`.

The status column is one of:

- "ok": requirement satisfied;
- "mismatch": installed, but the version does not satisfy the requirement;
- "missing": not installed;
- "restart": the version on disk is fine, but the older version still loaded in this R session is not. Restart R to pick up the new one.

Value

Invisibly, a data frame (class `depguard_check`) with columns `package`, `required`, `installed`, `status`, `depth` ("direct"/"transitive"), `required_by`, `loaded` (version loaded in this session, or "-") and `reason`. Printing it shows only the problem rows; use `print(x, all = TRUE)` to see everything.

Examples

```
dep_check(manifest = c(cli = "1.0.0", utils = ""), recursive = FALSE)
```

dep_diff

Diff the current environment against a prior snapshot

Description

Compares package versions now against a snapshot taken earlier with `dep_snapshot()`. Changes are detected on disk, so an upgrade is caught even though R keeps running the old version in memory until restart.

Usage

```
dep_diff(snapshot = NULL)
```

Arguments

snapshot A depguard_snapshot from `dep_snapshot()`, a path to a snapshot saved with `dep_snapshot(path = )`, or NULL (default) to use the most recent snapshot taken in this session.

Details

Risk levels:

- "high": the package is loaded in this session *and* it changed on disk (or the loaded version no longer matches disk). Code already run used one version; code run after a restart will use another.
- "low": the package changed but is not loaded.
- "none": nothing changed.

Value

Invisibly, a data frame with one row per package and columns package, before, after, change ("upgraded", "downgraded", "added", "removed", "none"), changed, currently_loaded, loaded_version, session_stale (loaded version differs from disk) and risk.

dep_env

Describe the current R environment

Description

Detects whether R is running in a hosted notebook (Kaggle, Colab, Binder), RStudio, or a plain desktop/server session, and reports which package libraries exist and which are writable. depguard uses this to tailor its advice (for example, how to restart the session after an install).

Usage

```
dep_env(check_online = FALSE)

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

Arguments

check_online Logical; if TRUE, make a short (3 second timeout) request to CRAN to see whether the internet is reachable.

x A depguard_env object.

... Ignored.

Details

Detection looks at environment variables only and makes no network calls unless `check_online = TRUE`. Set `options(depguard.platform = "kaggle")` (or `"colab"`, `"binder"`, `"rstudio"`, `"desktop"`) to override it.

Value

An object of class `depguard_env`, a list with elements `platform`, `os`, `r_version`, `interactive`, `libs`, `writable_libs`, `work_dir` and `online` (NA unless `check_online = TRUE`).

Examples

```
dep_env()
```

dep_fix	<i>Roll back a single package to a specific version</i>
---------	---

Description

Reinstalls one package at a specific version, e.g. to undo a version that was silently pulled in as a side effect of another install. This performs a **single-package rollback only**: it does not resolve cascading conflicts that the rollback might introduce with other packages. For full dependency resolution, use `renv::restore()` or `pak`'s solver instead.

Usage

```
dep_fix(
  package,
  version,
  method = c("auto", "pak", "remotes", "archive"),
  lib = NULL,
  dry_run = FALSE
)
```

Arguments

<code>package</code>	Name of the package to roll back.
<code>version</code>	Target version string, e.g. <code>"1.5.0"</code> .
<code>method</code>	Which backend to use: <code>"auto"</code> (default), <code>"pak"</code> , <code>"remotes"</code> or <code>"archive"</code> . If the requested backend is not available, the next one in that order is used.
<code>lib</code>	Library to install into. Defaults to the first writable library on <code>.libPaths()</code> .
<code>dry_run</code>	Logical; if TRUE, show what would be done without installing anything.

Details

Three backends are supported. With `method = "auto"` the first one available is used: pak, then remotes (both in Suggests), then "archive", which needs nothing beyond base R: it downloads the source tarball from the CRAN archive and installs it. (Installing from source needs Rtools on Windows, and the Xcode command line tools on macOS if the package contains compiled code; Kaggle/Colab-style Linux images normally have what is needed.)

With `method = "auto"`, if the chosen backend fails (for example because the internet is flaky), `dep_fix()` falls back to the archive backend before giving up. Set `options(depguard.cran_mirror = "<url>")` to use a different CRAN mirror for the remotes and archive backends (pak uses its own repository settings).

After installing, the version on disk is verified. If the package is loaded in the current session, R keeps running the old version until it is restarted, and you are told so.

Value

Invisibly, TRUE on success; with `dry_run = TRUE`, a list describing the plan.

Examples

```
## Not run:
dep_fix("stringr", "1.5.0")
dep_fix("stringr", "1.5.0", dry_run = TRUE)

## End(Not run)
```

dep_healthcheck	<i>Run a one-shot dependency health check</i>
-----------------	---

Description

Convenience entry point intended to be run at the top of a notebook or script. It reports the environment (hosted notebook or desktop, writable libraries), warns about packages shadowed by another library, and then either checks the environment against your manifest (see [dep_manifest\(\)](#)) or, if there is none, captures a baseline snapshot you can compare against later with [dep_diff\(\)](#).

Usage

```
dep_healthcheck(
  path = default_manifest_path(),
  check_cran = FALSE,
  libraries = TRUE
)
```

Arguments

path	File path to the manifest, used only if manifest is NULL.
check_cran	Logical; if TRUE, additionally query CRAN for the latest available version of each package (adds cran_latest and outdated columns). Requires network access; if CRAN cannot be reached a warning is shown and the rest of the result is returned unchanged.
libraries	Logical; also run <code>dep_libraries()</code> to look for shadowed packages (default TRUE).

Value

Invisibly, either the result of `dep_check()` (if a manifest exists) or a `depguard_snapshot` object (if not).

Examples

```
dep_healthcheck()
```

dep_libraries	<i>Find packages installed in more than one library</i>
---------------	---

Description

R searches library paths in order and loads the *first* copy it finds. On hosted notebooks (a system library plus a user library) and on desktops with several libraries, an old copy can shadow the newer one you just installed, so `packageVersion()` and the version you think you installed disagree. This is one of the commonest causes of "I upgraded it but nothing changed".

Usage

```
dep_libraries()
```

Value

Invisibly, a data frame with one row per shadowed copy and columns `package`, `active_library`, `active_version`, `shadowed_library`, `shadowed_version` and `differs` (whether the versions are different). It has zero rows when nothing is shadowed.

Examples

```
dep_libraries()
```

dep_manifest	<i>Declare a dependency manifest</i>
--------------	--------------------------------------

Description

Records the packages and versions your project needs, so that `dep_check()` can later verify the live environment satisfies them. This is a lightweight alternative to a full `renv` lockfile, intended for sandboxed or ephemeral notebook sessions and for quick desktop projects where you don't want to manage a lockfile.

Usage

```
dep_manifest(..., path = default_manifest_path())
```

Arguments

<code>...</code>	Named arguments of the form <code>package = "requirement"</code> , e.g. <code>dplyr = "1.1.4"</code> or <code>ggplot2 = ">= 3.5.0"</code> .
<code>path</code>	File path to store the manifest. Defaults to <code>.depguard_manifest.rds</code> in the current working directory (or <code>getOption("depguard.manifest_path")</code> if set).

Details

Each requirement can be:

- a bare version such as `"1.1.4"`: **at least** that version;
- a version with an operator: `">= 1.1.4"`, `"> 1.1"`, `"== 1.1.4"` (exactly), `"< 2.0"`, `"<= 2.0"` or `"!= 1.2"`;
- NA or `""`: any version, as long as it is installed.

Use `"=="` when an *upgrade* would also be a conflict (see `dep_manifest_freeze()` to pin whatever you have now).

The file format follows the extension of `path`. `.rds` (the default) stores an R object. Any other extension (for example `depguard.txt`) writes a human-readable text file, one package (op version) per line, which is convenient to edit and to commit to version control.

Value

Invisibly, the manifest as a named character vector.

Examples

```
tmp <- tempfile(fileext = ".rds")
dep_manifest(dplyr = "1.1.4", ggplot2 = ">= 3.5.0", path = tmp)
unlink(tmp)

# Human-readable text format
```

```
txt <- tempfile(fileext = ".txt")
dep_manifest(cli = "== 3.6.2", path = txt)
readLines(txt)
unlink(txt)
```

dep_manifest_freeze *Pin the packages you are using right now*

Description

Writes a manifest from the versions currently installed, so a later `dep_check()` (for example at the top of a re-run notebook, or after installing something new) tells you if any of them moved.

Usage

```
dep_manifest_freeze(
  packages = NULL,
  exact = TRUE,
  path = default_manifest_path()
)
```

Arguments

packages	Character vector of packages to pin. Defaults to the packages attached in this session (the ones you called <code>library()</code> on), excluding base R packages and <code>depguard</code> itself.
exact	If TRUE (default) each package is pinned with "=", so both downgrades and upgrades are reported. If FALSE, versions are recorded as minimums.
path	File path to store the manifest. Defaults to <code>.depguard_manifest.rds</code> in the current working directory (or <code>getOption("depguard.manifest_path")</code> if set).

Value

Invisibly, the manifest as a named character vector.

Examples

```
tmp <- tempfile(fileext = ".txt")
dep_manifest_freeze("cli", path = tmp)
readLines(tmp)
unlink(tmp)
```

dep_manifest_read	<i>Read an existing dependency manifest</i>
-------------------	---

Description

Read an existing dependency manifest

Usage

```
dep_manifest_read(path = default_manifest_path())
```

Arguments

path	File path to the manifest (.rds, or a text file written by dep_manifest()). Defaults to .depguard_manifest.rds in the current working directory.
------	---

Value

A named character vector of package requirements, or NULL (invisibly) with a message if no manifest is found.

dep_snapshot	<i>Snapshot package versions</i>
--------------	----------------------------------

Description

Records, for every installed package, the version on disk (what a fresh R session would load) and the version currently loaded in this session. A later call to [dep_diff\(\)](#) compares against the snapshot to reveal what an install silently changed.

Usage

```
dep_snapshot(scope = c("all", "loaded"), path = NULL)
```

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

Arguments

scope	"all" (default) records every installed package, so silent changes to packages you have not loaded yet are caught too. "loaded" records only packages loaded in this session.
path	Optional file path (.rds) to save the snapshot to.
x	A depguard_snapshot object.
...	Ignored.

Details

The most recent snapshot is remembered for the rest of the session, so `dep_diff()` can be called without arguments. In hosted notebooks the session usually has to be restarted after an install; pass `path` to save the snapshot to disk so it survives the restart, then call `dep_diff("path/to/snapshot.rds")`.

Value

An object of class `depguard_snapshot`, suitable for passing to `dep_diff()`. Its `packages` element is a data frame with columns `package`, `disk_version`, `library` and `loaded_version`.

Examples

```
snap <- dep_snapshot()
dep_diff(snap)
```

Index

`.libPaths()`, [5](#)

`dep_check`, [2](#)
`dep_check()`, [7–9](#)
`dep_diff`, [3](#)
`dep_diff()`, [6](#), [10](#), [11](#)
`dep_env`, [4](#)
`dep_fix`, [5](#)
`dep_healthcheck`, [6](#)
`dep_libraries`, [7](#)
`dep_libraries()`, [7](#)
`dep_manifest`, [8](#)
`dep_manifest()`, [2](#), [6](#), [10](#)
`dep_manifest_freeze`, [9](#)
`dep_manifest_freeze()`, [8](#)
`dep_manifest_read`, [10](#)
`dep_manifest_read()`, [2](#)
`dep_snapshot`, [10](#)
`dep_snapshot()`, [3](#), [4](#)

`print.depguard_check (dep_check)`, [2](#)
`print.depguard_env (dep_env)`, [4](#)
`print.depguard_snapshot (dep_snapshot)`,
[10](#)