# Package 'depguard'

September 24, 2026

**Title** Manifest-Based Dependency Conflict Detection for Sandboxed R Sessions

**Version** 0.1.0

**Description** Provides lightweight, manifest-based checking of R package dependencies (including transitive dependencies) against the currently installed environment, without requiring a full project lockfile. Designed for sandboxed or ephemeral notebook environments (e.g. Kaggle, Colab, Binder) where 'renv'-style lockfile ownership is impractical. Includes session snapshot/diff tools (building on 'sessioninfo') to detect when an install silently changes the version of a package that is already loaded, and optional single-package version rollback.

**License** MIT + file LICENSE

**Encoding** UTF-8

**Depends** R (>= 3.5)

**Imports** tools, utils, sessioninfo, cli

**Suggests** remotes, pak, testthat (>= 3.0.0), 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** 2026-09-24 05:20:02 UTC

# Contents



---

`dep_check` *Check the live environment against a dependency manifest*

---

## Description

Walks the transitive dependency tree of each package declared in the manifest and compares installed versions against requirements. By default this is entirely local (reads installed package `DESCRIPTION` metadata) and makes no network calls, which matters in sandboxed notebooks with limited or no internet access.

## Usage

```
dep_check(
  manifest = NULL,
  path = default_manifest_path(),
  recursive = TRUE,
  check_cran = 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's live metadata for newer available versions. Requires network access and is `FALSE` by default since sandboxed notebooks may have flaky or no internet. |

## Value

A data frame with columns `package`, `required`, `installed`, `status` (`"ok"`, `"mismatch"`, `"missing"`), `depth` (`"direct"`/`"transitive"`), and `required_by`.

## dep_diff *Diff the current environment against a prior snapshot*

### Description

Compares package versions now against a snapshot taken earlier with `dep_snapshot()`, flagging anything that changed. Packages that changed *and* are still loaded in the current session are flagged as higher risk, since those are the ones most likely to cause silent downstream breakage.

### Usage

```
dep_diff(snapshot)
```

### Arguments

| | |
|---|---|
| `snapshot` | A `depguard_snapshot` object from `dep_snapshot()`. |

### Value

A data frame with columns `package`, `before`, `after`, `changed`, `currently_loaded`, `risk`.

## dep_fix *Roll back a single package to a specific version*

### 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("pak", "remotes"))
```

### Arguments

| | |
|---|---|
| `package` | Name of the package to roll back. |
| `version` | Target version string, e.g. `"1.5.0"`. |
| `method` | Which backend to use: `"pak"` (default, if available) or `"remotes"`. |

### Details

Requires the `remotes` or `pak` package (listed in Suggests, not Imports, so `depguard`'s core checking functions stay dependency-light).

## Value

Invisibly, TRUE on success.

## Examples

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

## End(Not run)
```

---

`dep_healthcheck` *Run a one-shot dependency health check*

---

## Description

Convenience entry point intended to be run at the top of a notebook session (e.g. on Kaggle or Colab). If a manifest exists (see `dep_manifest()`), checks the environment against it. Otherwise, falls back to reporting a plain snapshot of currently loaded package versions so you have a baseline to diff against later with `dep_diff()`.

## Usage

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

## Arguments

| | |
|---|---|
| `path` | File path to the manifest, used only if `manifest` is `NULL`. |
| `check_cran` | Logical; if `TRUE`, additionally query CRAN's live metadata for newer available versions. Requires network access and is `FALSE` by default since sandboxed notebooks may have flaky or no internet. |

## Value

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

## Examples

```
dep_healthcheck()
```

## dep_manifest *Declare a dependency manifest*

### Description

Records the packages (and minimum 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 where you don't own or persist the environment.

### Usage

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

### Arguments

| | |
|---|---|
| ... | Named arguments of the form package = "version", e.g. dplyr = "1.1.4". A version of NA or "" means "any version is acceptable, just check it's installed". |
| path | File path to store the manifest. Defaults to .depguard_manifest.rds in the current working directory. |

### 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)
```

## dep_manifest_read *Read an existing dependency manifest*

### Description

Read an existing dependency manifest

### Usage

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

### Arguments

| | |
|---|---|
| path | File path to the 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 *Snapshot currently loaded package versions*

---

## Description

Captures the versions of all currently loaded/attached packages, so a later call to dep_diff() can detect what an install silently changed. This is a thin wrapper around sessioninfo::session_info().

## Usage

```
dep_snapshot()
```

## Value

An object of class depguard_snapshot (invisibly printable), suitable for passing to dep_diff().

## Examples

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

---

print.depguard_snapshot *Print a depguard snapshot*

---

## Description

Print a depguard snapshot

## Usage

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

## Arguments

| | |
|---|---|
| x | A depguard_snapshot object. |
| ... | Ignored. |

## Value

The x object, returned invisibly.

# Index