# Package 'Rapp'

December 14, 2025

**Title** Easily Build Command Line Applications

**Version** 0.3.0

**Description** Run simple 'R' scripts as command line applications, with automatic robust and convenient support for command line arguments. This package provides 'Rapp', an alternative 'R' front-end similar to 'Rscript', that enables this.

**License** MIT + file LICENSE

**URL** https://github.com/r-lib/Rapp

**BugReports** https://github.com/r-lib/Rapp/issues

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**Suggests** testthat (>= 3.0.0), withr

**Config/testthat/edition** 3

**Config/testthat/parallel** true

**Config/testthat/start-first** help-snapshots, basics

**Imports** yaml

**NeedsCompilation** no

**Author** Tomasz Kalinowski [aut, cre]

**Maintainer** Tomasz Kalinowski <tomasz@posit.co>

**Repository** CRAN

**Date/Publication** 2025-12-14 19:00:02 UTC

## Contents

`install_pkg_cli_apps` *Install CLI launchers for package scripts*

## Description

`install_pkg_cli_apps()` scans an installed package's `exec/` directory for `.R` scripts whose shebang line invokes `Rapp` (for example, `#!/usr/bin/env Rapp`) or `Rscript` (for example, `#!/usr/bin/env Rscript`). Each discovered script gets a lightweight launcher in `destdir` that invokes `Rapp` or `Rscript` to run the app. The launcher encodes the absolute path to the R binary this function is called from.

## Usage

```
install_pkg_cli_apps(
  package = parent.pkg() %||% rownames(utils::installed.packages()),
  destdir = NULL,
  lib.loc = NULL,
  overwrite = NA
)

uninstall_pkg_cli_apps(package = parent.pkg(), destdir = NULL)
```

## Arguments

| | |
|---|---|
| `package` | Package names to process. Defaults to the calling package when run inside a package; otherwise all installed packages. |
| `destdir` | Directory to write launchers to. See Details for defaults. |
| `lib.loc` | Additional library paths forwarded to `base::system.file()` while locating package scripts. Discovery happens at install time; written launchers embed absolute script paths. |
| `overwrite` | Whether to replace an existing executable. `TRUE` always overwrites, `FALSE` never overwrites non-Rapp executables, and `NA` (the default) prompts interactively and otherwise skips. |

## Details

Optional `#| launcher:` front matter in the script lets authors tune the `Rscript` flags. By default, for both `Rscript` and `Rapp`, R is invoked with `--default-packages=base,<pkg>`, where `<pkg>` is the package providing the executable.

Launchers are regenerated each time `install_pkg_cli_apps()` is called, and any obsolete launchers for the same package are removed. `RAPP_INSTALL_DIR` overrides the default destination. Launchers are POSIX shell scripts on Unix-like systems and `.bat` files on Windows. Front-matter options such as `vanilla`, `no-environ`, and `default_packages` map directly to the corresponding `Rscript` arguments.

When `overwrite` is `NA`, files previously written by Rapp are always replaced while other executables trigger a confirmation prompt (skipped in non-interactive sessions). A warning is emitted when skipping an existing executable.

If `destdir` is not provided, it is resolved in this order:

- env var `RAPP_INSTALL_DIR`
- env var `XDG_BIN_HOME`
- env var `XDG_DATA_HOME/../bin`
- the default location:
  - macOS and Linux: `~/.local/bin`,
  - Windows: `%LOCALAPPDATA%\Programs\R\Rapp\bin`

On Windows, the resolved `destdir` is explicitly added to `PATH` (it generally is not by default). To disable adding it to the `PATH`, set envvar `RAPP_NO_MODIFY_PATH=1`.

On macOS or Linux, `~/.local/bin` is typically already on `PATH` if it exists. Note: some shells add `~/.local/bin` to `PATH` only if it exists at login. If `install_pkg_cli_apps()` created the directory, you may need to restart the shell for the new apps to be found on `PATH`.

Example setting `launcher` args:

```
#!/usr/bin/env Rapp
#| description: About this app
#| launcher:
#|   vanilla: true
#|   default-packages: [base, utils, mypkg]
```

## Value

Invisibly returns the paths of launchers that were (re)written.

## Examples

```
## Not run:
# Install the launcher for the Rapp package itself: `Rapp`
install_pkg_cli_apps("Rapp")

## End(Not run)
```

---

`run` *Run an R app.*

---

## Description

Run an R app.

## Usage

```
run(app, args = commandArgs(TRUE))
```

## Arguments

| | |
|---|---|
| app | A filepath to an Rapp. |
| args | character vector of command line args. |

## Details

See the package README for full details. https://github.com/r-lib/Rapp

## Value

Mainly called for its side effect. For advanced or testing use, it invisibly returns the evaluation environment where the app's expressions ran. If the app did not run (for example, when `--help` is used), it returns `NULL` invisibly.

## Examples

```
# For the example, place 'Rapp', the package examples, and 'R' on the PATH
old_path <- Sys.getenv("PATH")
Sys.setenv(PATH = paste(system.file("examples", package = "Rapp"),
                        system.file("exec", package = "Rapp"),
                        R.home("bin"),
                        old_path,
                        sep = .Platform$path.sep))

# Here is an example app:
# flip-coin.R
writeLines(readLines(
  system.file("examples/flip-coin.R", package = "Rapp")))

if(.Platform$OS.type != "windows") {
  # on macOS and Linux, you can call the app directly
  system("flip-coin.R")
  system("flip-coin.R --n 3")
} else {
  # On windows, there is no support for shebang '#!' style executables
  # but you can invoke 'Rapp' directly
  system("Rapp flip-coin.R")
  system("Rapp flip-coin.R --n 3")
}

# restore PATH
Sys.setenv(PATH = old_path)
```

# Index