

# Package ‘devtools’

March 14, 2026

**Title** Tools to Make Developing R Packages Easier

**Version** 2.5.0

**Description** Collection of package development tools.

**License** MIT + file LICENSE

**URL** <https://devtools.r-lib.org/>, <https://github.com/r-lib/devtools>

**BugReports** <https://github.com/r-lib/devtools/issues>

**Depends** R (>= 4.1), usethis (>= 3.2.1)

**Imports** cli (>= 3.6.5), desc (>= 1.4.3), ellipsis (>= 0.3.2), fs (>= 1.6.7), lifecycle (>= 1.0.5), memoise (>= 2.0.1), miniUI (>= 0.1.2), pak (>= 0.9.2), pkgbuild (>= 1.4.8), pkgdown (>= 2.2.0), pkgload (>= 1.5.0), profvis (>= 0.4.0), rcmdcheck (>= 1.4.0), rlang (>= 1.1.7), roxygen2 (>= 7.3.3), rversions (>= 3.0.0), sessioninfo (>= 1.2.3), testthat (>= 3.3.2), urlchecker (>= 1.0.1), withr (>= 3.0.2)

**Suggests** BiocManager (>= 1.30.18), callr (>= 3.7.1), covr (>= 3.5.1), curl (>= 4.3.2), digest (>= 0.6.29), DT (>= 0.23), foghorn (>= 1.4.2), gh (>= 1.3.0), httr2 (>= 1.0.0), knitr (>= 1.39), linter (>= 3.0.0), quarto (>= 1.5.1), remotes (>= 2.5.0), rmarkdown (>= 2.14), rstudioapi (>= 0.13), spelling (>= 2.2), xml2

**VignetteBuilder** knitr, quarto

**Config/Needs/website** tidyverse/tidytemplate

**Config/testthat/edition** 3

**Config/testthat/parallel** true

**Encoding** UTF-8

**Language** en-US

**RoxygenNote** 7.3.3

**NeedsCompilation** no

**Author** Hadley Wickham [aut],  
Jim Hester [aut],  
Winston Chang [aut],

Jennifer Bryan [aut, cre] (ORCID:  
 <<https://orcid.org/0000-0002-6983-2759>>),  
 Posit Software, PBC [cph, fnd] (ROR: <<https://ror.org/03wc8by49>>)

**Maintainer** Jennifer Bryan <jenny@posit.co>

**Repository** CRAN

**Date/Publication** 2026-03-14 16:40:28 UTC

## Contents

build . . . . .	2
build_manual . . . . .	4
build_readme . . . . .	5
build_site . . . . .	5
check . . . . .	6
check_doc_fields . . . . .	8
check_mac_release . . . . .	9
check_man . . . . .	10
check_win . . . . .	11
dev_sitrep . . . . .	12
document . . . . .	13
install . . . . .	14
install_deps . . . . .	15
lint . . . . .	17
load_all . . . . .	18
release . . . . .	20
run_examples . . . . .	20
save_all . . . . .	21
source_gist . . . . .	22
source_url . . . . .	23
spell_check . . . . .	24
test . . . . .	24
uninstall . . . . .	25
<b>Index</b>	<b>26</b>

---

build

*Build package*

---

## Description

Building converts a package source directory into a single bundled file. If `binary = FALSE` this creates a `tar.gz` package that can be installed on any platform, provided they have a full development environment (although packages without source code can typically be installed out of the box). If `binary = TRUE`, the package will have a platform specific extension (e.g. `.zip` for windows), and will only be installable on the current platform, but no development environment is needed.

**Usage**

```
build(
  pkg = ".",
  path = NULL,
  binary = FALSE,
  vignettes = TRUE,
  manual = FALSE,
  args = NULL,
  quiet = FALSE,
  ...
)
```

**Arguments**

pkg	The package to use, can be a file path to the package or a package object. See <a href="#">as.package()</a> for more information.
path	Path in which to produce package. If NULL, defaults to the parent directory of the package.
binary	Produce a binary ( <code>--binary</code> ) or source ( <code>--no-manual --no-resave-data</code> ) version of the package.
vignettes, manual	For source packages: if FALSE, don't build PDF vignettes ( <code>--no-build-vignettes</code> ) or manual ( <code>--no-manual</code> ).
args	An optional character vector of additional command line arguments to be passed to R CMD build if <code>binary = FALSE</code> , or R CMD install if <code>binary = TRUE</code> .
quiet	if TRUE suppresses output from this function.
...	Additional arguments passed to <code>pkgbuild::build</code> .

**Details****Configuration:**

DESCRIPTION *entries*:

- `Config/build/clean-inst-doc` can be set to FALSE to avoid cleaning up `inst/doc` when building a source package. Set it to TRUE to force a cleanup. See the `clean_doc` argument.
- `Config/build/copy-method` can be used to avoid copying large directories in R CMD build. It works by copying (or linking) the files of the package to a temporary directory, leaving out the (possibly large) files that are not part of the package. Possible values:
  - none: pkgbuild does not copy the package tree. This is the default.
  - copy: the package files are copied to a temporary directory before R CMD build.
  - link: the package files are symbolic linked to a temporary directory before R CMD build. Windows does not have symbolic links, so on Windows this is equivalent to copy.

You can also use the `pkg.build_copy_method` option or the `PKG_BUILD_COPY_METHOD` environment variable to set the copy method. The option is consulted first, then the DESCRIPTION entry, then the environment variable.

- `Config/build/extra-sources` can be used to define extra source files for `pkgbuild` to decide whether a package DLL needs to be recompiled in `needs_compile()`. The syntax is a comma separated list of file names, or globs. (See `utils::glob2rx()`.) E.g. `src/rust/src/*.rs` or `configure*`.
- `Config/build/bootstrap` can be set to `TRUE` to run `Rscript bootstrap.R` in the source directory prior to running subsequent build steps.
- `Config/build/never-clean` can be set to `TRUE` to never add `--preclean` to `R CMD INSTALL`, e.g., when header files have changed. This helps avoiding rebuilds that can take long for very large C/C++ codebases and can lead to build failures if object files are out of sync with header files. Control the dependencies between object files and header files by adding `include file.d` to `Makevars` for each `file.c` or `file.cpp` source file.

*Options:*

- `pkg.build_copy_method`: use this option to avoid copying large directories when building a package. See possible values above, at the `Config/build/copy-method DESCRIPTION` entry.
- `pkg.build_stop_for_warnings`: if it is set to `TRUE`, then `pkgbuild` will stop for `R CMD build` errors. It takes precedence over the `PKG_BUILD_STOP_FOR_WARNINGS` environment variable.

*Environment variables:*

- `PKG_BUILD_COLOR_DIAGNOSTICS`: set it to `false` to opt out of colored compiler diagnostics. Set it to `true` to force colored compiler diagnostics.
- `PKG_BUILD_COPY_METHOD`: use this environment variable to avoid copying large directories when building a package. See possible values above, at the `Config/build/copy-method DESCRIPTION` entry.

will stop for `R CMD build` errors. The `pkg.build_stop_for_warnings` option takes precedence over this environment variable.

**Value**

a string giving the location (including file name) of the built package

**Note**

The default `manual = FALSE` is not suitable for a CRAN submission, which may require `manual = TRUE`. Even better, use `submit_cran()` or `release()`.

---

build\_manual

*Create package pdf manual*

---

**Description**

Create package pdf manual

**Usage**

`build_manual(pkg = ".", path = NULL)`

**Arguments**

pkg	The package to use, can be a file path to the package or a package object. See <a href="#">as.package()</a> for more information.
path	path in which to produce package manual. If NULL, defaults to the parent directory of the package.

**See Also**

[Rd2pdf\(\)](#)

---

build_readme	<i>Build README</i>
--------------	---------------------

---

**Description**

Renders an executable README, i.e. README.qmd or README.Rmd, to README.md. Specifically, `build_readme()`:

- Installs a copy of the package's current source to a temporary library
- Renders the README in a clean R session

**Usage**

```
build_readme(path = ".", quiet = TRUE, ...)
```

**Arguments**

path	Path to the top-level directory of the source package.
quiet	If TRUE, suppresses most output. Set to FALSE if you need to debug.
...	Additional arguments passed to <a href="#">rmarkdown::render()</a> , in the case of README.Rmd. Not used for README.qmd

---

build_site	<i>Run pkgdown::build_site()</i>
------------	----------------------------------

---

**Description**

This is a thin wrapper around [pkgdown::build\\_site\(\)](#), used for generating static HTML documentation. Learn more at <https://pkgdown.r-lib.org>.

**Usage**

```
build_site(path = ".", ...)
```

**Arguments**

path	Path to the package to build the static HTML.
...	Additional arguments passed to <a href="#">pkgdown::build_site()</a> .

---

`check`*Build and check a package*

---

## Description

`check()` automatically builds and checks a source package, using all known best practices. `check_built()` checks an already-built package.

Passing R CMD check is essential if you want to submit your package to CRAN: you must not have any **ERRORS** or **WARNINGS**, and you want to ensure that there are as few **NOTES** as possible. If you are not submitting to CRAN, at least ensure that there are no **ERRORS** or **WARNINGS**: these typically represent serious problems.

`check()` automatically builds a package before calling `check_built()`, as this is the recommended way to check packages. Note that this process runs in an independent R session, so nothing in your current workspace will affect the process. Under-the-hood, `check()` and `check_built()` rely on `pkgbuild::build()` and `rcmdcheck::rcmdcheck()`.

## Usage

```
check(  
  pkg = ".",  
  document = NULL,  
  build_args = NULL,  
  ...,  
  manual = FALSE,  
  cran = TRUE,  
  remote = FALSE,  
  incoming = remote,  
  force_suggests = FALSE,  
  run_dont_test = FALSE,  
  args = "--timings",  
  env_vars = c(NOT_CRAN = "true"),  
  quiet = FALSE,  
  check_dir = NULL,  
  cleanup = deprecated(),  
  vignettes = TRUE,  
  error_on = c("never", "error", "warning", "note")  
)
```

```
check_built(  
  path = NULL,  
  cran = TRUE,  
  remote = FALSE,  
  incoming = remote,  
  force_suggests = FALSE,  
  run_dont_test = FALSE,  
  manual = FALSE,
```

```

  args = "--timings",
  env_vars = NULL,
  check_dir = tempdir(),
  quiet = FALSE,
  error_on = c("never", "error", "warning", "note")
)

```

## Arguments

pkg	The package to use, can be a file path to the package or a package object. See <a href="#">as.package()</a> for more information.
document	By default (NULL) will document if your installed roxygen2 version matches the version declared in the DESCRIPTION file. Use TRUE or FALSE to override the default.
build_args	Additional arguments passed to R CMD build
...	Additional arguments passed on to <code>pkgbuild::build()</code> .
manual	If FALSE, don't build and check manual ( <code>--no-manual</code> ).
cran	if TRUE (the default), check using the same settings as CRAN uses. Because this is a moving target and is not uniform across all of CRAN's machine, this is on a "best effort" basis. It is more complicated than simply setting <code>--as-cran</code> .
remote	Sets <code>_R_CHECK_CRAN_INCOMING_REMOTE_</code> env var. If TRUE, performs a number of CRAN incoming checks that require remote access.
incoming	Sets <code>_R_CHECK_CRAN_INCOMING_</code> env var. If TRUE, performs a number of CRAN incoming checks.
force_suggests	Sets <code>_R_CHECK_FORCE_SUGGESTS_</code> . If FALSE (the default), check will proceed even if all suggested packages aren't found.
run_dont_test	Sets <code>--run-donttest</code> so that examples surrounded in <code>\donttest{}</code> are also run. When <code>cran = TRUE</code> , this only affects R 3.6 and earlier; in R 4.0, code in <code>\donttest{}</code> is always run as part of CRAN submission.
args	Character vector of arguments to pass to R CMD check. Pass each argument as a single element of this character vector (do not use spaces to delimit arguments like you would in the shell). For example, to skip running of examples and tests, use <code>args = c("--no-examples", "--no-tests")</code> and not <code>args = "--no-examples --no-tests"</code> . (Note that instead of the <code>--output</code> option you should use the <code>check_dir</code> argument, because <code>--output</code> cannot deal with spaces and other special characters on Windows.)
env_vars	Environment variables set during R CMD check
quiet	if TRUE suppresses output from this function.
check_dir	Path to a directory where the check is performed. If this is not NULL, then the a temporary directory is used, that is cleaned up when the returned object is garbage collected.
cleanup	<b>[Deprecated]</b> See <code>check_dir</code> for details.
vignettes	If FALSE, do not build or check vignettes, equivalent to using <code>args = '--ignore-vignettes'</code> and <code>build_args = '--no-build-vignettes'</code> .

error_on	Whether to throw an error on R CMD check failures. Note that the check is always completed (unless a timeout happens), and the error is only thrown after completion. error_on is passed through to <code>rcmdcheck::rcmdcheck()</code> , which is the definitive source for what the different values mean. If not specified by the user, both <code>check()</code> and <code>check_built()</code> default to <code>error_on = "never"</code> in interactive use and "warning" in a non-interactive setting.
path	Path to built package.

**Value**

An object containing errors, warnings, notes, and more.

**Environment variables**

Devtools does its best to set up an environment that combines best practices with how check works on CRAN. This includes:

- The standard environment variables set by devtools: `r_env_vars()`. Of particular note for package tests is the `NOT_CRAN` env var, which lets you know that your tests are running somewhere other than CRAN, and hence can take a reasonable amount of time.
- Debugging flags for the compiler, set by `compiler_flags(FALSE)`.
- If `aspell` is found, `_R_CHECK_CRAN_INCOMING_USE_ASPELL_` is set to `TRUE`. If no spell checker is installed, a warning is issued.
- Environment variables, controlled by arguments `incoming`, `remote` and `force_suggests`.

**See Also**

[release\(\)](#) if you want to send the checked package to CRAN.

---

check\_doc\_fields      *Check for missing documentation fields*

---

**Description**

Checks all Rd files in `man/` and looks for any that have a `\usage` section (i.e. a function) but that *don't* have `\value` and `\examples` sections. These missing fields are flagged by CRAN on initial submission.

**Usage**

```
check_doc_fields(pkg = ".", fields = c("value", "examples"))
```

**Arguments**

pkg	The package to use, can be a file path to the package or a package object. See <a href="#">as.package()</a> for more information.
fields	A character vector of Rd field names to check for.

**Value**

A named list of character vectors, one for each field, containing the names of Rd files missing that field. Returned invisibly.

**Examples**

```
## Not run:  
check_doc_fields(".")  
  
## End(Not run)
```

---

check_mac_release	<i>Check a package on macOS</i>
-------------------	---------------------------------

---

**Description**

Check on either the released or development versions of R, using <https://mac.r-project.org/macbuilder/submit.html>.

**Usage**

```
check_mac_release(  
  pkg = ".",  
  dep_pkgs = character(),  
  args = NULL,  
  manual = TRUE,  
  quiet = FALSE,  
  ...  
)  
  
check_mac_devel(  
  pkg = ".",  
  dep_pkgs = character(),  
  args = NULL,  
  manual = TRUE,  
  quiet = FALSE,  
  ...  
)
```

**Arguments**

pkg	The package to use, can be a file path to the package or a package object. See <a href="#">as.package()</a> for more information.
dep_pkgs	Additional custom dependencies to install prior to checking the package.
args	An optional character vector of additional command line arguments to be passed to R CMD build if binary = FALSE, or R CMD install if binary = TRUE.

manual	Should the manual be built?
quiet	If TRUE, suppresses output.
...	Additional arguments passed to <code>pkgbuild::build()</code> .

**Value**

The url with the check results (invisibly)

**See Also**

Other build functions: `check_win()`

---

check\_man

*Check documentation, as R CMD check does*

---

**Description**

This function attempts to run the documentation related checks in the same way that R CMD check does. Unfortunately it can't run them all because some tests require the package to be loaded, and the way they attempt to load the code conflicts with how devtools does it.

**Usage**

```
check_man(pkg = ".")
```

**Arguments**

pkg	The package to use, can be a file path to the package or a package object. See <a href="#">as.package()</a> for more information.
-----	---

**Value**

Nothing. This function is called purely for it's side effects: if no errors there will be no output.

**Examples**

```
## Not run:  
check_man("mypkg")  
  
## End(Not run)
```

---

`check_win`*Check a package on Windows*

---

### Description

This function first bundles a source package, then uploads it to <https://win-builder.r-project.org/>. Once the service has built and checked the package, an email is sent to address of the maintainer listed in DESCRIPTION. This usually takes around 30 minutes. The email contains a link to a directory with the package binary and check logs, which will be deleted after a couple of days.

### Usage

```
check_win_devel(  
  pkg = ".",  
  args = NULL,  
  manual = TRUE,  
  email = NULL,  
  quiet = FALSE,  
  webform = FALSE,  
  ...  
)  
  
check_win_release(  
  pkg = ".",  
  args = NULL,  
  manual = TRUE,  
  email = NULL,  
  quiet = FALSE,  
  webform = FALSE,  
  ...  
)  
  
check_win_oldrelease(  
  pkg = ".",  
  args = NULL,  
  manual = TRUE,  
  email = NULL,  
  quiet = FALSE,  
  webform = FALSE,  
  ...  
)
```

### Arguments

`pkg` The package to use, can be a file path to the package or a package object. See [as.package\(\)](#) for more information.

args	An optional character vector of additional command line arguments to be passed to R CMD build if binary = FALSE, or R CMD install if binary = TRUE.
manual	Should the manual be built?
email	An alternative email address to use. If NULL, the default is to use the package maintainer's email.
quiet	If TRUE, suppresses output.
webform	If TRUE, uses web form instead of passive FTP upload.
...	Additional arguments passed to <code>pkgbuild::build()</code> .

### Functions

- `check_win_devel()`: Check package on the development version of R.
- `check_win_release()`: Check package on the released version of R.
- `check_win_oldrelease()`: Check package on the previous major release version of R.

### See Also

Other build functions: [check\\_mac\\_release\(\)](#)

---

dev\_sitrep

*Report package development situation*

---

### Description

Call this function if things seem weird and you're not sure what's wrong or how to fix it. It reports:

- If R is up to date.
- If RStudio or Positron is up to date.
- If compiler build tools are installed and available for use.
- If devtools and its dependencies are up to date.
- If the package's dependencies are up to date.

### Usage

```
dev_sitrep(pkg = ".", debug = FALSE)
```

### Arguments

pkg	The package to use, can be a file path to the package or a package object. See <a href="#">as.package()</a> for more information.
debug	If TRUE, will print out extra information useful for debugging. If FALSE, it will use result cached from a previous run.

**Value**

A named list, with S3 class `dev_sitrep` (for printing purposes).

**Examples**

```
## Not run:  
dev_sitrep()  
  
## End(Not run)
```

---

document

*Use roxygen to document a package*

---

**Description**

This function is a wrapper for the `roxygen2::roxygenize()` function from the `roxygen2` package. See the documentation and vignettes of that package to learn how to use `roxygen`.

**Usage**

```
document(pkg = ".", roclets = NULL, quiet = FALSE)
```

**Arguments**

<code>pkg</code>	The package to use, can be a file path to the package or a package object. See <a href="#">as.package()</a> for more information.
<code>roclets</code>	Character vector of roclet names to use with package. The default, <code>NULL</code> , uses the <code>roxygen</code> <code>roclets</code> option, which defaults to <code>c("collate", "namespace", "rd")</code> .
<code>quiet</code>	if <code>TRUE</code> suppresses output from this function.

**See Also**

[roxygen2::roxygenize\(\)](#), [browseVignettes\("roxygen2"\)](#)

---

install	<i>Install a local development package</i>
---------	--

---

### Description

Uses R CMD INSTALL to install the package, after installing needed dependencies with `pak::local_install_deps()`.

To install to a non-default library, use `withr::with_libpaths()`.

### Usage

```
install(
  pkg = ".",
  reload = TRUE,
  quick = FALSE,
  build = !quick,
  args = getOption("devtools.install.args"),
  quiet = FALSE,
  dependencies = NA,
  upgrade = FALSE,
  build_vignettes = FALSE,
  keep_source = getOption("keep.source.pkgs") || !build,
  force = deprecated()
)
```

### Arguments

pkg	The package to use, can be a file path to the package or a package object. See <a href="#">as.package()</a> for more information.
reload	if TRUE (the default), will automatically attempt to reload the package after installing. Reloading is not always completely possible so see <a href="#">pkgload::unregister()</a> for caveats.
quick	if TRUE, skips some optional steps (e.g. help pre-rendering and multi-arch builds) to make installation as fast as possible.
build	<p>If TRUE (the default), first <a href="#">pkgbuild::build()</a>s the package. This ensures that the installation is completely clean, and prevents any binary artefacts (like <code>‘.o’</code>, <code>.so</code>) from appearing in your local package directory, but is considerably slower, because every compile has to start from scratch.</p> <p>One downside of installing from a built tarball is that the package is installed from a temporary location. This means that any source references will point to dangling locations and debuggers won't have direct access to the source for step-debugging. For development purposes, <code>build = FALSE</code> is often the better choice.</p> <p>If FALSE, the package is installed directly from its source directory. This is faster and can be favorable for preserving source references for debugging (see <code>keep_source</code>).</p>

args	An optional character vector of additional command line arguments to be passed to R CMD INSTALL. This defaults to the value of the option "devtools.install.args".
quiet	If TRUE, suppress output.
dependencies	What kinds of dependencies to install. Most commonly one of the following values: <ul style="list-style-type: none"> <li>• NA: only required (hard) dependencies,</li> <li>• TRUE: required dependencies plus optional and development dependencies,</li> <li>• FALSE: do not install any dependencies. (You might end up with a non-working package, and/or the installation might fail.) See <a href="#">Package dependency types</a> for other possible values and more information about package dependencies.</li> </ul>
upgrade	When FALSE, the default, pak does the minimum amount of work to give you the latest version(s) of pkg. It will only upgrade dependent packages if pkg, or one of their dependencies explicitly require a higher version than what you currently have. It will also prefer a binary package over to source package, even if the binary package is older. When upgrade = TRUE, pak will ensure that you have the latest version(s) of pkg and all their dependencies.
build_vignettes	if TRUE, will build vignettes. Normally it is build that's responsible for creating vignettes; this argument makes sure vignettes are built even if a build never happens (i.e. because build = FALSE).
keep_source	If TRUE will keep the srcfiles from an installed package. This is useful for debugging (especially inside of RStudio or Positron). Defaults to <code>getOption("keep.source.pkgs")    !build</code> , since srcfiles are most useful when the package is installed from its source directory, i.e. when build = FALSE.
force	<b>[Deprecated]</b> No longer used.

**See Also**

[with\\_debug\(\)](#) to install packages with debugging flags set.

Other package installation: [uninstall\(\)](#)

---

install\_deps

*Install package dependencies if needed*


---

**Description****[Deprecated]**

These functions are deprecated. Better alternatives:

- `pak::local_install_deps()` instead of `install_deps()`
- `pak::local_install_dev_deps()` instead of `install_dev_deps()`

**Usage**

```

install_deps(
  pkg = ".",
  dependencies = NA,
  repos = getOption("repos"),
  type = getOption("pkgType"),
  upgrade = c("default", "ask", "always", "never"),
  quiet = FALSE,
  build = TRUE,
  build_opts = c("--no-resave-data", "--no-manual", " --no-build-vignettes"),
  ...
)

install_dev_deps(
  pkg = ".",
  dependencies = TRUE,
  repos = getOption("repos"),
  type = getOption("pkgType"),
  upgrade = c("default", "ask", "always", "never"),
  quiet = FALSE,
  build = TRUE,
  build_opts = c("--no-resave-data", "--no-manual", " --no-build-vignettes"),
  ...
)

```

**Arguments**

pkg	The package to use, can be a file path to the package or a package object. See <a href="#">as.package()</a> for more information.
dependencies	Which dependencies do you want to check? Can be a character vector (selecting from "Depends", "Imports", "LinkingTo", "Suggests", or "Enhances"), or a logical vector.  TRUE is shorthand for "Depends", "Imports", "LinkingTo" and "Suggests". NA is shorthand for "Depends", "Imports" and "LinkingTo" and is the default. FALSE is shorthand for no dependencies (i.e. just check this package, not its dependencies).  The value "soft" means the same as TRUE, "hard" means the same as NA.  You can also specify dependencies from one or more additional fields, common ones include: <ul style="list-style-type: none"> <li>• Config/Needs/website - for dependencies used in building the pkgdown site.</li> <li>• Config/Needs/coverage for dependencies used in calculating test coverage.</li> </ul>
repos	A character vector giving repositories to use.
type	Type of package to update.
upgrade	Should package dependencies be upgraded? One of "default", "ask", "always", or "never". "default" respects the value of the R_REMOTES_UPGRADE environment variable if set, and falls back to "ask" if unset. "ask" prompts the user for which

	out of date packages to upgrade. For non-interactive sessions "ask" is equivalent to "always". TRUE and FALSE are also accepted and correspond to "always" and "never" respectively.
quiet	If TRUE, suppress output.
build	If TRUE build the package before installing.
build_opts	Options to pass to R CMD build, only used when build is TRUE.
...	Additional arguments passed to <code>remotes::install_deps()</code> .

### Examples

```
## Not run: install_deps(".")
```

---

lint	<i>Lint all source files in a package</i>
------	---

---

### Description

The default linters correspond to the style guide at <https://style.tidyverse.org/>, however it is possible to override any or all of them using the `linters` parameter.

### Usage

```
lint(pkg = ".", cache = TRUE, ...)
```

### Arguments

pkg	The package to use, can be a file path to the package or a package object. See <a href="#">as.package()</a> for more information.
cache	Store the lint results so repeated lints of the same content use the previous results. Consult the <code>lintr</code> package to learn more about its caching behaviour.
...	Additional arguments passed to <code>lintr::lint_package()</code> .

### See Also

`lintr::lint_package()`, `lintr::lint()`

---

load_all	<i>Load complete package</i>
----------	------------------------------

---

## Description

`load_all()` loads a package. It roughly simulates what happens when a package is installed and loaded with `library()`, without having to first install the package. It:

- Loads all data files in `data/`. See `load_data()` for more details.
- Sources all R files in the R directory, storing results in environment that behaves like a regular package namespace. See `load_code()` for more details.
- Adds a shim from `system.file()` to `shim_system.file()` in the imports environment of the package. This ensures that `system.file()` works with both development and installed packages despite their differing directory structures.
- Adds shims from `help()` and `?` to `shim_help()` and `shim_question()` to make it easier to preview development documentation.
- Compiles any C, C++, or Fortran code in the `src/` directory and connects the generated DLL into R. See `pkgbuild::compile_dll()` for more details.
- Loads any compiled translations in `inst/po`.
- Runs `.onAttach()`, `.onLoad()` and `.onUnload()` functions at the correct times.
- If you use `testthat`, will load all test helpers so you can access them interactively. `devtools` sets the `DEVTOOLS_LOAD` environment variable to the package name to let you check whether the helpers are run during package loading.

`is_loading()` returns `TRUE` when it is called while `load_all()` is running. This may be useful e.g. in `.onLoad` hooks. A package loaded with `load_all()` can be identified with `is_dev_package()`.

## Usage

```
load_all(
  path = ".",
  reset = TRUE,
  recompile = FALSE,
  export_all = TRUE,
  helpers = TRUE,
  quiet = FALSE,
  ...
)
```

## Arguments

<code>path</code>	Path to a package, or within a package.
<code>reset</code>	<b>[Deprecated]</b> This is no longer supported because preserving the namespace requires unlocking its environment, which is no longer possible in recent versions of R.

recompile	DEPRECATED. force a recompile of DLL from source code, if present. This is equivalent to running <code>pkgbuild::clean_dll()</code> before <code>load_all()</code>
export_all	If TRUE (the default), export all objects. If FALSE, export only the objects that are listed as exports in the NAMESPACE file.
helpers	if TRUE loads <b>testthat</b> test helpers.
quiet	if TRUE suppresses output from this function.
...	Additional arguments passed to <code>pkgload::load_all()</code> .

### Differences to regular loading

`load_all()` tries its best to reproduce the behaviour of `loadNamespace()` and `library()`. However it deviates from normal package loading in several ways.

- `load_all()` doesn't install the package to a library, so `system.file()` doesn't work. `pkgload` fixes this for the package itself installing a shim, `shim_system.file()`. However, this shim is not visible to third party packages, so they will fail if they attempt to find files within your package. One potential workaround is to use `fs::path_package()` instead of `system.file()`, since that understands the mechanisms that devtools uses to load packages.
- `load_all()` loads all packages referenced in Imports at load time, but `loadNamespace()` and `library()` only load package dependencies as they are needed.
- `load_all()` copies all objects (not just the ones listed as exports) into the package environment. This is useful during development because it makes internal objects easy to access. To export only the objects listed as exports, use `export_all = FALSE`. This more closely simulates behavior when loading an installed package with `library()`, and can be useful for checking for missing exports.

### Controlling the debug compiler flags

`load_all()` delegates to `pkgbuild::compile_dll()` to perform the actual compilation, during which by default some debug compiler flags are appended. If you would like to produce an optimized build instead, you can opt out by either using `debug = FALSE`, setting the `pkg.build_extra_flags` option to FALSE, or setting the `PKG_BUILD_EXTRA_FLAGS` environment variable to FALSE. For further details see the Details section in `pkgbuild::compile_dll()`.

### Examples

```
## Not run:
# Load the package in the current directory
load_all("./")

# Running again loads changed files
load_all("./")

# With export_all=FALSE, only objects listed as exports in NAMESPACE
# are exported
load_all("./", export_all = FALSE)

## End(Not run)
```

---

release	<i>Release package to CRAN</i>
---------	--------------------------------

---

## Description

### [Deprecated]

`release()` is deprecated in favour of `usethis::use_release_issue()`. We no longer feel confident recommending `release()` because we don't use it ourselves, so there's no guarantee that it will track best practices as they evolve over time.

If you want to programmatically submit to CRAN, you can continue to use `submit_cran()`.

## Usage

```
release(pkg = ".", check = FALSE, args = NULL)
```

## Arguments

<code>pkg</code>	The package to use, can be a file path to the package or a package object. See <a href="#">as.package()</a> for more information.
<code>check</code>	if TRUE, run checking, otherwise omit it. This is useful if you've just checked your package and you're ready to release it.
<code>args</code>	An optional character vector of additional command line arguments to be passed to R CMD build.

## See Also

[usethis::use\\_release\\_issue\(\)](#) to create a checklist of release tasks that you can use in addition to or in place of `release`.

---

run_examples	<i>Run all examples in a package</i>
--------------	--------------------------------------

---

## Description

One of the most frustrating parts of R CMD check is getting all of your examples to pass - whenever one fails you need to fix the problem and then restart the whole process. This function makes it a little easier by making it possible to run all examples from an R function.

**Usage**

```
run_examples(
  pkg = ".",
  start = NULL,
  show = deprecated(),
  run_donttest = FALSE,
  run_dontrun = FALSE,
  fresh = FALSE,
  document = TRUE,
  run = deprecated(),
  test = deprecated()
)
```

**Arguments**

pkg	The package to use, can be a file path to the package or a package object. See <a href="#">as.package()</a> for more information.
start	Where to start running the examples: this can either be the name of Rd file to start with (with or without extensions), or a topic name. If omitted, will start with the (lexicographically) first file. This is useful if you have a lot of examples and don't want to rerun them every time you fix a problem. To run only one example, use <a href="#">pkgload::run_example()</a> .
show	DEPRECATED.
run_donttest	if TRUE, do run <code>\donttest</code> sections in the Rd files.
run_dontrun	if TRUE, do run <code>\dontrun</code> sections in the Rd files.
fresh	if TRUE, will be run in a fresh R session. This has the advantage that there's no way the examples can depend on anything in the current session, but interactive code (like <a href="#">browser()</a> ) won't work.
document	if TRUE, <a href="#">document()</a> will be run to ensure examples are updated before running them.
run, test	Deprecated, see <code>run_dontrun</code> and <code>run_donttest</code> above.

**See Also**

[pkgload::run\\_example\(\)](#) to run a single example.

---

 save\_all

*Save all documents in an active IDE session*


---

**Description**

Helper function wrapping IDE-specific calls to save all documents in the active session. In this form, callers of `save_all()` don't need to execute any IDE-specific code. This function can be extended to include other IDE implementations of their equivalent `rstudioapi::documentSaveAll()` methods.

**Usage**

```
save_all()
```

---

```
source_gist
```

```
Run a script on gist
```

---

**Description**

“Gist is a simple way to share snippets and pastes with others. All gists are git repositories, so they are automatically versioned, forkable and usable as a git repository.” <https://gist.github.com/>

**Usage**

```
source_gist(id, ..., filename = NULL, sha1 = NULL, quiet = FALSE)
```

**Arguments**

id	either full url (character), gist ID (numeric or character of numeric).
...	other options passed to <a href="#">source()</a>
filename	if there is more than one R file in the gist, which one to source (filename ending in '.R')? Default NULL will source the first file.
sha1	The SHA-1 hash of the file at the remote URL. This is highly recommend as it prevents you from accidentally running code that's not what you expect. See <a href="#">source_url()</a> for more information on using a SHA-1 hash.
quiet	if FALSE, the default, prints informative messages.

**See Also**

[source\\_url\(\)](#)

**Examples**

```
## Not run:
# You can run gists given their id
source_gist(6872663)
source_gist("6872663")

# Or their html url
source_gist("https://gist.github.com/hadley/6872663")
source_gist("gist.github.com/hadley/6872663")

# It's highly recommend that you run source_gist with the optional
# sha1 argument - this will throw an error if the file has changed since
# you first ran it
source_gist(6872663, sha1 = "54f1db27e60")
# Wrong hash will result in error
source_gist(6872663, sha1 = "54f1db27e61")
```

```
#' # You can specify a particular R file in the gist
source_gist(6872663, filename = "hi.r")
source_gist(6872663, filename = "hi.r", sha1 = "54f1db27e60")

## End(Not run)
```

---

source\_url

*Run a script through some protocols such as http, https, ftp, etc*

---

## Description

If a SHA-1 hash is specified with the `sha1` argument, then this function will check the SHA-1 hash of the downloaded file to make sure it matches the expected value, and throw an error if it does not match. If the SHA-1 hash is not specified, it will print a message displaying the hash of the downloaded file. The purpose of this is to improve security when running remotely-hosted code; if you have a hash of the file, you can be sure that it has not changed. For convenience, it is possible to use a truncated SHA1 hash, down to 6 characters, but keep in mind that a truncated hash won't be as secure as the full hash.

## Usage

```
source_url(url, ..., sha1 = NULL)
```

## Arguments

<code>url</code>	url
<code>...</code>	other options passed to <code>source()</code>
<code>sha1</code>	The (prefix of the) SHA-1 hash of the file at the remote URL.

## See Also

[source\\_gist\(\)](#)

## Examples

```
## Not run:

source_url("https://gist.github.com/hadley/6872663/raw/hi.r")

# With a hash, to make sure the remote file hasn't changed
source_url("https://gist.github.com/hadley/6872663/raw/hi.r",
  sha1 = "54f1db27e60bb7e0486d785604909b49e8fef9f9")

# With a truncated hash
source_url("https://gist.github.com/hadley/6872663/raw/hi.r",
  sha1 = "54f1db27e60")

## End(Not run)
```

---

spell_check	<i>Spell checking</i>
-------------	-----------------------

---

**Description**

Runs a spell check on text fields in the package description file, manual pages, and optionally vignettes. Wraps the [spelling](#) package.

**Usage**

```
spell_check(pkg = ".", vignettes = TRUE, use_wordlist = TRUE)
```

**Arguments**

pkg	The package to use, can be a file path to the package or a package object. See <a href="#">as.package()</a> for more information.
vignettes	also check all rmd and rnw files in the pkg vignettes folder
use_wordlist	ignore words in the package <a href="#">WORDLIST</a> file

---

test	<i>Execute testthat tests in a package</i>
------	--

---

**Description**

- `test()` runs all tests in a package. It's a shortcut for `testthat::test_dir()`
- `test_active_file()` runs `test()` on the active file.
- `test_coverage()` computes test coverage for your package. It's a shortcut for `covr::package_coverage()` plus `covr::report()`.
- `test_coverage_active_file()` computes test coverage for the active file. It's a shortcut for `covr::file_coverage()` plus `covr::report()`.

**Usage**

```
test(pkg = ".", filter = NULL, stop_on_failure = FALSE, export_all = TRUE, ...)
```

```
test_active_file(file = find_active_file(), ...)
```

```
test_coverage(pkg = ".", report = NULL, ...)
```

```
test_coverage_active_file(
  file = find_active_file(),
  filter = TRUE,
  report = NULL,
  export_all = TRUE,
  ...
)
```

**Arguments**

pkg	The package to use, can be a file path to the package or a package object. See <a href="#">as.package()</a> for more information.
filter	If not NULL, only tests with file names matching this regular expression will be executed. Matching is performed on the file name after it's stripped of "test-" and ".R".
stop_on_failure	If TRUE, throw an error if any tests fail.
export_all	If TRUE (the default), export all objects. If FALSE, export only the objects that are listed as exports in the NAMESPACE file.
...	additional arguments passed to wrapped functions.
file	One or more source or test files. If a source file the corresponding test file will be run. The default is to use the active file in RStudio (if available).
report	How to display the coverage report. <ul style="list-style-type: none"> <li>• "html" opens an interactive report in the browser.</li> <li>• "zero" prints uncovered lines to the console.</li> <li>• "silent" returns the coverage object without display.</li> </ul> Defaults to "html" if interactive; otherwise to "zero".

uninstall

*Uninstall a local development package***Description**

Uses `remove.packages()` to uninstall the package. To uninstall a package from a non-default library, use in combination with `withr::with_libpaths()`.

**Usage**

```
uninstall(pkg = ".", unload = TRUE, quiet = FALSE, lib = .libPaths()[[1]])
```

**Arguments**

pkg	The package to use, can be a file path to the package or a package object. See <a href="#">as.package()</a> for more information.
unload	if TRUE (the default), ensures the package is unloaded, prior to uninstalling.
quiet	If TRUE, suppress output.
lib	a character vector giving the library directories to remove the packages from. If missing, defaults to the first element in <code>.libPaths()</code> .

**See Also**

[with\\_debug\(\)](#) to install packages with debugging flags set.

Other package installation: [install\(\)](#)

# Index

- \* **build functions**
  - check\_mac\_release, 9
  - check\_win, 11
- \* **example functions**
  - run\_examples, 20
- \* **package installation**
  - install, 14
  - uninstall, 25
- \* **programming**
  - run\_examples, 20
- .libPaths, 25
- as.package(), 3, 5, 7–14, 16, 17, 20, 21, 24, 25
  
- browser(), 21
- build, 2
- build\_manual, 4
- build\_readme, 5
- build\_site, 5
  
- check, 6
- check\_built(check), 6
- check\_doc\_fields, 8
- check\_mac\_devel(check\_mac\_release), 9
- check\_mac\_release, 9, 12
- check\_man, 10
- check\_win, 10, 11
- check\_win\_devel(check\_win), 11
- check\_win\_oldrelease(check\_win), 11
- check\_win\_release(check\_win), 11
- compiler\_flags(FALSE), 8
- covr::file\_coverage(), 24
- covr::package\_coverage(), 24
- covr::report(), 24
  
- dev\_sitrep, 12
- document, 13
- document(), 21
  
- fs::path\_package(), 19
  
- install, 14, 25
- install\_deps, 15
- install\_dev\_deps(install\_deps), 15
- is\_dev\_package(), 18
  
- library(), 18, 19
- lint, 17
- lintr::lint(), 17
- lintr::lint\_package(), 17
- load\_all, 18
- load\_code(), 18
- load\_data(), 18
- loadNamespace(), 19
  
- Package dependency types, 15
- pak::local\_install\_deps(), 14
- pkgbuild::build, 3
- pkgbuild::build(), 6, 7, 10, 12, 14
- pkgbuild::clean\_dll(), 19
- pkgbuild::compile\_dll(), 18, 19
- pkgdown::build\_site(), 5
- pkgload::load\_all(), 19
- pkgload::run\_example(), 21
- pkgload::unregister(), 14
  
- r\_env\_vars(), 8
- rcmdcheck::rcmdcheck(), 6, 8
- Rd2pdf(), 5
- release, 20
- release(), 4, 8
- remotes::install\_deps(), 17
- rmarkdown::render(), 5
- roxygen2::roxygenize(), 13
- run\_examples, 20
  
- save\_all, 21
- shim\_help(), 18
- shim\_question(), 18
- shim\_system.file(), 18, 19
- source(), 22, 23

source\_gist, [22](#)  
source\_gist(), [23](#)  
source\_url, [23](#)  
source\_url(), [22](#)  
spell\_check, [24](#)  
spelling, [24](#)  
submit\_cran(), [4](#), [20](#)  
system.file(), [18](#), [19](#)

test, [24](#)  
test\_active\_file(test), [24](#)  
test\_coverage(test), [24](#)  
test\_coverage\_active\_file(test), [24](#)  
testthat::test\_dir(), [24](#)

uninstall, [15](#), [25](#)  
usethis::use\_release\_issue(), [20](#)  
utils::glob2rx(), [4](#)

with\_debug(), [15](#), [25](#)  
withr::with\_libpaths(), [14](#), [25](#)  
WORDLIST, [24](#)