# Package 'pkgload'

November 30, 2021

**Title** Simulate Package Installation and Attach

**Version** 1.2.4

**Description** Simulates the process of installing a package and then attaching it. This is a key part of the 'devtools' package as it allows you to rapidly iterate while developing a package.

**License** GPL-3

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

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

**Imports** cli, crayon, desc, methods, rlang, rprojroot, rstudioapi, utils, withr

**Suggests** bitops, covr, pkgbuild, Rcpp, testthat

**Encoding** UTF-8

**RoxygenNote** 7.1.2

**NeedsCompilation** no

**Author** Hadley Wickham [aut],
Winston Chang [aut],
Jim Hester [aut],
Lionel Henry [aut, cre],
RStudio [cph],
R Core team [ctb] (Some namespace and vignette code extracted from base R)

**Maintainer** Lionel Henry <lionel@rstudio.com>

**Repository** CRAN

**Date/Publication** 2021-11-30 14:30:02 UTC

## R topics documented:

---

`dev_example` *Run a examples for an in-development function.*

---

## Description

`dev_example` is a replacement for `example`. `run_example` is a low-level function that takes a path to an Rd file.

## Usage

```
dev_example(topic, quiet = FALSE)

run_example(
  path,
  run_donttest = FALSE,
  run_dontrun = FALSE,
  env = new.env(parent = globalenv()),
  quiet = FALSE,
  macros = NULL,
  run,
  test
)
```

## Arguments

| | |
|---|---|
| `topic` | Name or topic (or name of Rd) file to run examples for |
| `quiet` | If `TRUE`, does not echo code to console. |
| `path` | Path to `.Rd` file |
| `run_donttest` | if `TRUE`, do run \donttest sections in the Rd files. out. |
| `run_dontrun` | if `TRUE`, do run \dontrun sections in the Rd files. |
| `env` | Environment in which code will be run. |
| `macros` | Custom macros to use to parse the `.Rd` file. See the `macros` argument of `tools::parse_Rd()`. If `NULL`, then the `tools::Rd2ex()` (and `tools::parse_Rd()`) default is used. |
| `run, test` | Deprecated, see `run_dontrun` and `run_donttest` above. |

## Examples

```
## Not run:
# Runs installed example:
library("ggplot2")
example("ggplot")

# Runs develoment example:
dev_example("ggplot")

## End(Not run)
```

---

dev_help *In-development help for package loaded with devtools*

---

## Description

dev_help searches for source documentation provided in packages loaded by devtools. To improve performance, the .Rd files are parsed to create to index once, then cached. Use dev_topic_index_reset to clear that index.

## Usage

```
dev_help(
  topic,
  dev_packages = NULL,
  stage = "render",
  type = getOption("help_type")
)

dev_topic_index_reset(pkg_name)
```

## Arguments

| | |
|---|---|
| topic | name of help to search for. |
| dev_packages | A character vector of package names to search within. If NULL, defaults to all packages loaded by devtools. |
| stage | at which stage ("build", "install", or "render") should \\Sexpr macros be executed? This is only important if you're using \\Sexpr macro's in your Rd files. |
| type | of html to produce: "html" or "text". Defaults to your default documentation type. |
| pkg_name | Name of package. |

## Examples

```
## Not run:
library("ggplot2")
help("ggplot") # loads installed documentation for ggplot

load_all("ggplot2")
dev_help("ggplot") # loads development documentation for ggplot

## End(Not run)
```

---

`help` *Drop-in replacements for help and ? functions*

---

## Description

The ? and `help` functions are replacements for functions of the same name in the utils package. They are made available when a package is loaded with `load_all()`.

## Usage

```
# help(topic, package = NULL, ...)

# ?e2
# e1?e2
```

## Arguments

| | |
|---|---|
| `topic` | A name or character string specifying the help topic. |
| `package` | A name or character string specifying the package in which to search for the help topic. If NULL, search all packages. |
| `...` | Additional arguments to pass to `utils::help()`. |
| `e1` | First argument to pass along to utils::¿‘. |
| `e2` | Second argument to pass along to utils::¿‘. |

## Details

The ? function is a replacement for `utils::?()` from the utils package. It will search for help in devtools-loaded packages first, then in regular packages.

The `help` function is a replacement for `utils::help()` from the utils package. If `package` is not specified, it will search for help in devtools-loaded packages first, then in regular packages. If `package` is specified, then it will search for help in devtools-loaded packages or regular packages, as appropriate.

## Examples

```
## Not run:
# This would load devtools and look at the help for load_all, if currently
# in the devtools source directory.
load_all()
?load_all
help("load_all")

## End(Not run)

# To see the help pages for utils::help and utils::`?`:
help("help", "utils")
help("?", "utils")

## Not run:
# Examples demonstrating the multiple ways of supplying arguments
# NB: you can't do pkg <- "ggplot2"; help("ggplot2", pkg)
help(lm)
help(lm, stats)
help(lm, 'stats')
help('lm')
help('lm', stats)
help('lm', 'stats')
help(package = stats)
help(package = 'stats')
topic <- "lm"
help(topic)
help(topic, stats)
help(topic, 'stats')

## End(Not run)
```

---

`inst` *Get the installation path of a package*

---

## Description

Given the name of a package, this returns a path to the installed copy of the package, which can be passed to other devtools functions.

## Usage

```
inst(name)
```

## Arguments

| | |
|---|---|
| `name` | the name of a package. |

### Details

It searches for the package in `.libPaths()`. If multiple dirs are found, it will return the first one.

### Examples

```
inst("pkgload")
inst("grid")
```

---

## is_dev_package *Is the package currently under development?*

---

### Description

Returns `TRUE` or `FALSE` depending on if the package has been loaded by **pkgload**.

### Usage

```
is_dev_package(name)
```

### Arguments

| | |
|---|---|
| `name` | the name of a package. |

---

## load_all *Load complete package*

---

### Description

`load_all` loads a package. It roughly simulates what happens when a package is installed and loaded with `library()`.

### Usage

```
load_all(
  path = ".",
  reset = TRUE,
  compile = NA,
  export_all = TRUE,
  export_imports = export_all,
  helpers = TRUE,
  attach_testthat = uses_testthat(path),
  quiet = FALSE,
  recompile = FALSE,
  warn_conflicts = TRUE
)
```

## Arguments

| | |
|---|---|
| path | Path to a package, or within a package. |
| reset | clear package environment and reset file cache before loading any pieces of the package. This largely equivalent to running unload(), however the old namespaces are not completely removed and no .onUnload() hooks are called. Use reset = FALSE may be faster for large code bases, but is a significantly less accurate approximation. |
| compile | If TRUE always recompiles the package; if NA recompiles if needed (as determined by pkgbuild::needs_compile()); if FALSE, never recompiles. |
| export_all | If TRUE (the default), export all objects. If FALSE, export only the objects that are listed as exports in the NAMESPACE file. |
| export_imports | If TRUE (the default), export all objects that are imported by the package. If FALSE export only objects defined in the package. |
| helpers | if TRUE loads **testthat** test helpers. |
| attach_testthat | If TRUE, attach **testthat** to the search path, which more closely mimics the environment within test files. |
| quiet | if TRUE suppresses output from this function. |
| recompile | DEPRECATED. force a recompile of DLL from source code, if present. This is equivalent to running pkgbuild::clean_dll() before load_all |
| warn_conflicts | If TRUE, issues a warning if a function in the global environment masks a function in the package. This can happen when you accidentally source a .R file, rather than using load_all(), or if you define a function directly in the R console. This is frustrating to debug, as it feels like the changes you make to the package source aren't having the expected effect. |

## Details

Currently load_all:

- 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 below and load_code() for more details.
- Compiles any C, C++, or Fortran code in the src/ directory and connects the generated DLL into R. See pkgload::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 "true" to let you check whether the helpers are run during package loading.

### Namespaces

The namespace environment <namespace:pkgname>, is a child of the imports environment, which has the name attribute `imports:pkgname`. It is in turn is a child of <namespace:base>, which is a child of the global environment. (There is also a copy of the base namespace that is a child of the empty environment.)

The package environment <package:pkgname> is an ancestor of the global environment. Normally when loading a package, the objects listed as exports in the NAMESPACE file are copied from the namespace to the package environment. However, `load_all` by default will copy all objects (not just the ones listed as exports) to the package environment. This is useful during development because it makes all 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.

### Shim files

`load_all` also inserts shim functions into the imports environment of the loaded package. It presently adds a replacement version of `system.file` which returns different paths from `base::system.file`. This is needed because installed and uninstalled package sources have different directory structures. Note that this is not a perfect replacement for `base::system.file`.

### Examples

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

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

# With reset=TRUE, unload and reload the package for a clean start
load_all("./", TRUE)

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

## End(Not run)
```

---

## load_code *Load R code.*

---

### Description

Load all R code in the R directory. The first time the code is loaded, `.onLoad` will be run if it exists.

### Usage

```
load_code(path = ".")
```

### Arguments

| | |
|---|---|
| path | Path to a package, or within a package. |

---

## load_data *Load data.*

---

### Description

Loads all .RData files in the data subdirectory.

### Usage

```
load_data(path = ".")
```

### Arguments

| | |
|---|---|
| path | Path to a package, or within a package. |

---

## load_dll *Load a compiled DLL*

---

### Description

Load a compiled DLL

### Usage

```
load_dll(path = ".")
```

### Arguments

| | |
|---|---|
| path | Path to a package, or within a package. |

## packages *Helper functions for working with development packages.*

### Description

All functions search recursively up the directory tree from the input path until they find a DESCRIPTION file.

### Usage

```
pkg_path(path = ".")

pkg_name(path = ".")

pkg_desc(path = ".")

pkg_version(path = ".")

pkg_ns(path = ".")
```

### Arguments

| | |
|---|---|
| path | Path to a package, or within a package. |

### Functions

- pkg_path: Return the normalized package path.
- pkg_name: Return the package name.
- pkg_desc: Return the package DESCRIPTION as a desc::desc() object.
- pkg_version: Return the package version.
- pkg_ns: Return the package namespace.

## package_file *Find file in a package.*

### Description

It always starts by finding by walking up the path until it finds the root directory, i.e. a directory containing DESCRIPTION. If it cannot find the root directory, or it can't find the specified path, it will throw an error.

### Usage

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

### Arguments

| | |
|---|---|
| ... | Components of the path. |
| path | Place to start search for package directory. |

### Examples

```
## Not run:
package_file("figures", "figure_1")

## End(Not run)
```

---

## system.file *Replacement version of system.file*

---

### Description

This function is meant to intercept calls to base::system.file(), so that it behaves well with packages loaded by devtools. It is made available when a package is loaded with load_all().

### Usage

```
shim_system.file(..., package = "base", lib.loc = NULL, mustWork = FALSE)
```

### Arguments

| | |
|---|---|
| ... | character vectors, specifying subdirectory and file(s) within some package. The default, none, returns the root of the package. Wildcards are not supported. |
| package | a character string with the name of a single package. An error occurs if more than one package name is given. |
| lib.loc | a character vector with path names of R libraries. See 'Details' for the meaning of the default value of NULL. |
| mustWork | logical. If TRUE, an error is given if there are no matching files. |

### Details

When system.file is called from the R console (the global environment), this function detects if the target package was loaded with load_all(), and if so, it uses a customized method of searching for the file. This is necessary because the directory structure of a source package is different from the directory structure of an installed package.

When a package is loaded with load_all, this function is also inserted into the package's imports environment, so that calls to system.file from within the package namespace will use this modified version. If this function were not inserted into the imports environment, then the package would end up calling base::system.file instead.

`unload` *Unload a package*

## Description

This function attempts to cleanly unload a package, including unloading its namespace, deleting S4 class definitions and unloading any loaded DLLs. Unfortunately S4 classes are not really designed to be cleanly unloaded, and so we have to manually modify the class dependency graph in order for it to work - this works on the cases for which we have tested but there may be others. Similarly, automated DLL unloading is best tested for simple scenarios (particularly with `useDynLib(pkgname)` and may fail in other cases. If you do encounter a failure, please file a bug report at https://github.com/r-lib/pkgload/issues.

## Usage

```
unload(package = pkg_name(), quiet = FALSE)
```

## Arguments

| | |
|---|---|
| `package` | package name. |
| `quiet` | if `TRUE` suppresses output from this function. |

## Examples

```
## Not run:
# Unload package that is in current directory
unload()

# Unload package that is in ./ggplot2/
unload(pkg_name("ggplot2/"))

library(ggplot2)
# unload the ggplot2 package directly by name
unload("ggplot2")

## End(Not run)
```

# Index