Package 'suggests'

July 23, 2025

Type Package

Title Declare when Suggested Packages are Needed

Version 0.1.0

Date 2023-07-14

Description By adding dependencies to the ``Suggests" field of a package's DESCRIPTION file, and then declaring that they are needed within any dependent functionality, it is often possible to significantly reduce the number of ``hard" dependencies required by a package. This package provides a minimal way to declare when a suggested package is needed.

License MIT + file LICENSE

URL https://github.com/owenjonesuob/suggests

BugReports https://github.com/owenjonesuob/suggests/issues

Imports utils

Suggests covr, testthat (>= 3.0.0)

Encoding UTF-8

RoxygenNote 7.2.3

Config/testthat/edition 3

NeedsCompilation no

Author Owen Jones [aut, cre] (ORCID: <https://orcid.org/0000-0002-8410-9585>)

Maintainer Owen Jones <owenjonesuob@gmail.com>

Repository CRAN

Date/Publication 2023-08-08 14:30:02 UTC

Contents

`find_deps` *List places where dependencies are used*

Description

A quick-and-dirty diagnostic tool to find dependency usage within top-level expressions (e.g. declared functions) in R scripts within a development package.

Usage

```
find_deps(path = ".", threshold = NULL)
```

Arguments

`path`	Path to the base directory of a package.
`threshold`	Only report on dependencies used in fewer than this many top-level expressions.

Details

This might be useful for package developers hoping to use `need()` in their package, and looking for good candidates for dependencies which could be moved from `Imports` to `Suggests` in the `DESCRIPTION` file.

Dependencies are searched for in two ways:

- `import()` and `importFrom()` statements in the package's `NAMESPACE` file, such as those created by `@import` and `@importFrom` tags if creating package documentation with roxygen2
- Functions called by using `::` or `:::` to access a dependency's namespace directly

This approach isn't perfect, but it should capture most dependency uses.

Value

A data frame, with one row per distinct top-level expression where a package is used. Packages used in the fewest places are listed first.

Examples

```
find_deps(system.file("demopkg", package = "suggests"))
```

`is_installed` *Check whether packages are installed*

Description

Initially, `utils::packageVersion()` is used to try to retrieve a version from a package's `DESCRIPTION` file. This is a fast method, but doesn't categorically guarantee that the package is actually available to use.

If `load = TRUE`, then `base::requireNamespace()` is used to try to load the namespace of each package in turn. This is much slower, but is the closest we can get to ensuring that the package is genuinely usable.

Usage

```
is_installed(pkgs, load = FALSE, lib.loc = NULL)
```

Arguments

`pkgs`	A character vector of package names. You can check for a minimum version by appending `>=[version]` to a package name - see Examples.
`load`	Whether to make sure packages can be loaded - significantly slower, but gives an extra level of certainty.
`lib.loc`	Passed to `utils::packageVersion()`.

Value

A logical vector of the same length as `pkgs`, where each element is `TRUE` if the package is installed, and `FALSE` otherwise.

Examples

```
is_installed("base")
is_installed(c("base", "utils"))

is_installed("base>=3.0.0")
is_installed(c(
  "base>=3.0.0",
  "utils"
))
```

need *Declare that packages are needed*

Description

Declare that one or more packages are required by subsequent functionality; and if they're missing, either prompt the user to install them, or exit with an informative error message.

Usage

```
need(
  ...,
  msg = NULL,
  install_cmd = NULL,
  ask = interactive(),
  load = FALSE,
  lib.loc = NULL
)
```

Arguments

`...`	Names of required packages, as character strings. You can require a minimum version by appending `>=[version]` to a package name - see Examples.
`msg`	Custom message to display; if `NULL`, an informative one will be constructed.
`install_cmd`	Installation command to run, as a call (i.e. probably wrapped with `quote()` or `substitute()`). If `NULL`, `install.packages()` will be used for package installation.
`ask`	Whether to give the user the option of installing the required packages immediately.
`load`	Whether to make sure packages can be loaded - significantly slower, but gives an extra level of certainty.
`lib.loc`	Passed to `utils::packageVersion()`.

Value

Invisibly, any package names from `...` which were installed.

Examples

```
## Not run:
  need("dplyr")
  need("dplyr", "tidyr")

  # All unnamed arguments will be combined into one list of package names
  shared_deps <- c("dplyr", "tidyr")
  need(shared_deps, "stringr") # same as need("dplyr", "tidyr", "stringr")
```

```
# You can require a minimum version for some or all packages
need("dplyr>=1.0.0", "tidyr")


# Typically you'll want to use need() within a function
read_data <- function(path, clean_names = FALSE) {

  # Call need() as early as possible, to avoid wasted work
  if (isTRUE(clean_names))
    suggests::need("janitor")

  output <- utils::read.csv(path)

  if (isTRUE(clean_names))
    output <- janitor::clean_names(output)

  output
}


# You can provide a custom message and/or installation command if needed
need(
  "dplyr",
  msg = "We need the development version of dplyr, for now!",
  install_cmd = quote(remotes::install_github("tidyverse/dplyr"))
)


## End(Not run)
```

Index