

# Package: easypackages (via r-universe)

September 6, 2024

**Title** Easy Loading and Installing of Packages

**Version** 0.1.0

**Description** Easily load and install multiple packages from different sources, including CRAN and GitHub. The `libraries` function allows you to load or attach multiple packages in the same function call. The `packages` function will load one or more packages, and install any packages that are not installed on your system (after prompting you). Also included is a `from_import` function that allows you to import specific functions from a package into the global environment.

**Depends** R (>= 3.0.0)

**Imports** assertthat, devtools, utils

**License** MIT + file LICENSE

**LazyData** true

**RoxygenNote** 5.0.1

**Suggests** knitr, rmarkdown

**VignetteBuilder** knitr

**Repository** <https://jakesherman.r-universe.dev>

**RemoteUrl** <https://github.com/jakesherman/easypackages>

**RemoteRef** HEAD

**RemoteSha** e22ee20bf2cbdc655f0852b90b33b9b00abb065c

## Contents

<code>from_import</code> . . . . .	2
<code>install_package</code> . . . . .	2
<code>install_packages</code> . . . . .	3
<code>is.package_obj</code> . . . . .	3
<code>libraries</code> . . . . .	4
<code>load_package</code> . . . . .	4
<code>package</code> . . . . .	5
<code>packages</code> . . . . .	6

**Index**[7](#)


---

from_import	<i>from_import()</i>
-------------	----------------------

---

**Description**

Imports one or more functions from a package into the parent environment, overwriting any existing objects in the parent environment sharing the name of the function(s). The package is loaded but not attached to the search list. Inspired by Python.

**Usage**

```
from_import(package, ..., verbose = TRUE)
```

**Arguments**

package	the package name (length one character vector)
...	one or more function names (as-is, or as a list)
verbose	give a warning when overwriting existing objects? TRUE by default.

**Examples**

```
## Not run:
from_import("dplyr", "select", "filter")
from_import("dplyr", list("select", "filter"))

## End(Not run)
```

---

install_package	<i>install_package()</i> Function to install a package given a package object
-----------------	---

---

**Description**

install\_package() Function to install a package given a package object

**Usage**

```
install_package(pack)
```

**Arguments**

pack	a package_obj object
------	----------------------

---

install_packages	<i>install_packages()</i>
------------------	---------------------------

---

**Description**

Installs one or more packages. Similar to `utils::install.packages`, but you may supply either package names and `package_obj` objects to this function. You can install Github packages by supplying `username/repo` to this function, or `username$repo` for Bitbucket packages.

**Usage**

```
install_packages(...)
```

**Arguments**

... one or more package names or `package_obj` objects

**Details**

`package_obj` allows you to supply it an install function if the package isn't on CRAN or in a public GitHub or Bitbucket repo.

**Examples**

```
## Not run:  
install_packages("dplyr", "ggplot2", "rvest", "magrittr")  
  
## End(Not run)
```

---

<code>is.package_obj</code>	<i>is.package_obj()</i> Returns TRUE if obj is a package_obj object
-----------------------------	---

---

**Description**

`is.package_obj()` Returns TRUE if obj is a `package_obj` object

**Usage**

```
is.package_obj(obj)
```

**Arguments**

obj an R object

---

libraries	<i>libraries()</i>
-----------	--------------------

---

**Description**

A vectorized version of the `library` function that accepts one or more package names to call `library` on. Unlike `library`, the `libraries` function does not use non-standard evaluation (NSE) on its inputs, meaning that you must supply your package names as strings, or as variables.

**Usage**

```
libraries(...)
```

**Arguments**

... one or more package names or `package_obj` objects

**Details**

You may supply either package names and `package_obj` objects to this function. You may also add `::` to the end of a package name to load the package instead of attaching the package (this means that the package will not be added to the search list, but will be available to access via the `::` operator).

**Examples**

```
## Not run:
libraries("dplyr", "ggplot2", "rvest", "magrittr")
libraries("dplyr::", "Rdatatable/data.table")

## End(Not run)
```

---

load_package	<i>load_package()</i> Function to load a package given a package object
--------------	---

---

**Description**

`load_package()` Function to load a package given a package object

**Usage**

```
load_package(pack)
```

**Arguments**

pack a `package_obj` object

---

package	<i>package()</i>
---------	------------------

---

## Description

A package object that contains at minimum the name of a package. If the package exists on CRAN, the name of the package is used to install that package from CRAN. A forward slash may be used in the package name to indicate a GitHub username and repo that a package is located in. A dollar sign may be used in the package name to indicate a Bitbucket username and repo that a package is located in.

## Usage

```
package(name, install = utils::install.packages, ..., load_type = "attach")
```

## Arguments

name	the name of a package, or, if the package is on Github, the username and repo of the package, ex. Rdatatable/data.table, where Rdatatable is the GitHub username, and data.table is the repo name. The same process works with Bitbucket, except \$ is used instead of / to separate the username and repo.
install	(optional) a function used to install a package.
...	(optional) additional arguments to the install function. Another way of supplying additional parameters to install is to use the purrr::partial function.
load_type	(default is "attach") should the package be loaded, or attached? See <a href="http://r-pkgs.had.co.nz/namespace.html">http://r-pkgs.had.co.nz/namespace.html</a> to learn more.

## Details

If a package is not intended to be installed from CRAN, Github (public), or Bitbucket (public) you may optionally supply a function to the `install` argument that installs the package, with additional arguments to the function supplied via the `...` argument.

## Examples

```
## Not run:
this_package <- package("dplyr")
github_package <- package("Rdatatable/data.table")
that_package <- package("jakePackage", devtools::install_bitbucket,
  repo = "repo", username = "user", password = "password")

## End(Not run)
```

---

packages	<i>packages()</i>
----------	-------------------

---

### Description

Loads one or more packages, and installs them after a user prompt if they are not already installed. You may supply either package names or `package_obj` objects to this function. You can install Github packages by supplying `username/repo` to this function, or `username$repo` for Bitbucket packages.

### Usage

```
packages(..., prompt = TRUE)
```

### Arguments

<code>...</code>	one or more package names or <code>package_obj</code> objects
<code>prompt</code>	(default is TRUE) prompt the user before installing packages? For interactive use keeping the default makes the most sense. For interactive scripts, or scripts that you are sharing with those you trust, it may make more sense to switch this to FALSE.

### Details

`package_obj` allows you to supply it an install function if the package isn't on CRAN or in a public GitHub or Bitbucket repo.

You may also add `::` to the end of a package name to load the package instead of attaching the package (this means that the package will not be added to the search list, but will be available to access via the `::` operator).

### Examples

```
## Not run:  
packages("dplyr", "ggplot2", "rvest", "magrittr")  
packages("dplyr::", "Rdatatable/data.table")  
  
## End(Not run)
```

# Index

`from_import`, [2](#)

`install_package`, [2](#)

`install_packages`, [3](#)

`is.package_obj`, [3](#)

`libraries`, [4](#)

`load_package`, [4](#)

`package`, [5](#)

`packages`, [6](#)