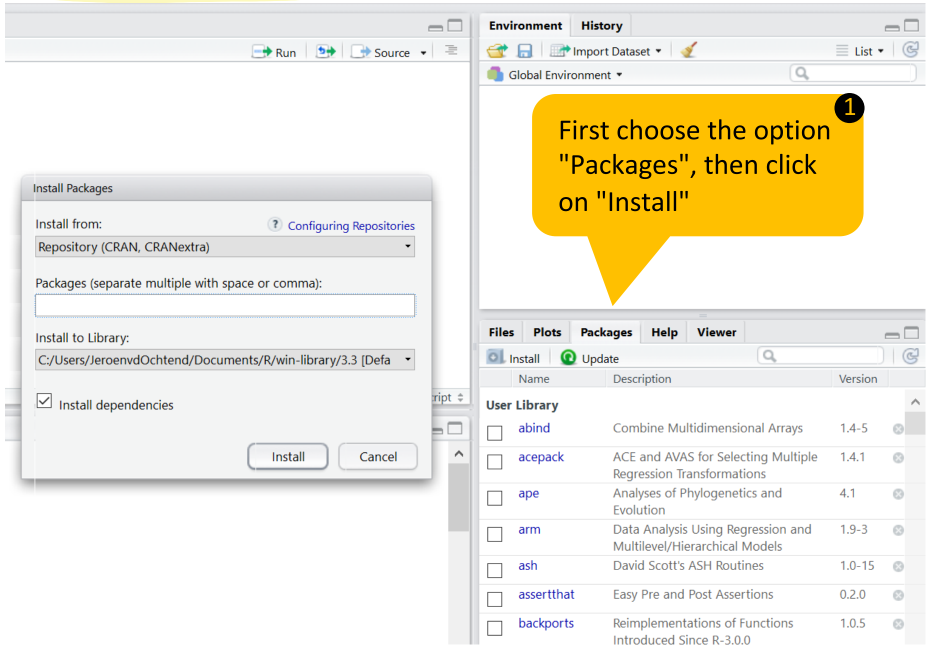
Installing R packages

How to install R packages: Two methods

1) Use the GUI in Rstudio



First choose the option "Packages", then click on "Install"

Install Packages

Install from: Configuring Repositories
Repository (CRAN, CRANextra)

Packages (separate multiple with space or comma):

Install to Library:
C:/Users/JeroenvdOchtend/Documents/R/win-library/3.3 [Defa

☑ Install dependencies

Install | Cancel

Name	Description	Version
User Library		
abind	Combine Multidimensional Arrays	1.4-5
acepack	ACE and AVAS for Selecting Multiple Regression Transformations	1.4.1
ape	Analyses of Phylogenetics and Evolution	4.1
arm	Data Analysis Using Regression and Multilevel/Hierarchical Models	1.9-3
ash	David Scott's ASH Routines	1.0-15
assertthat	Easy Pre and Post Assertions	0.2.0
backports	Reimplementations of Functions Introduced Since R-3.0.0	1.0.5

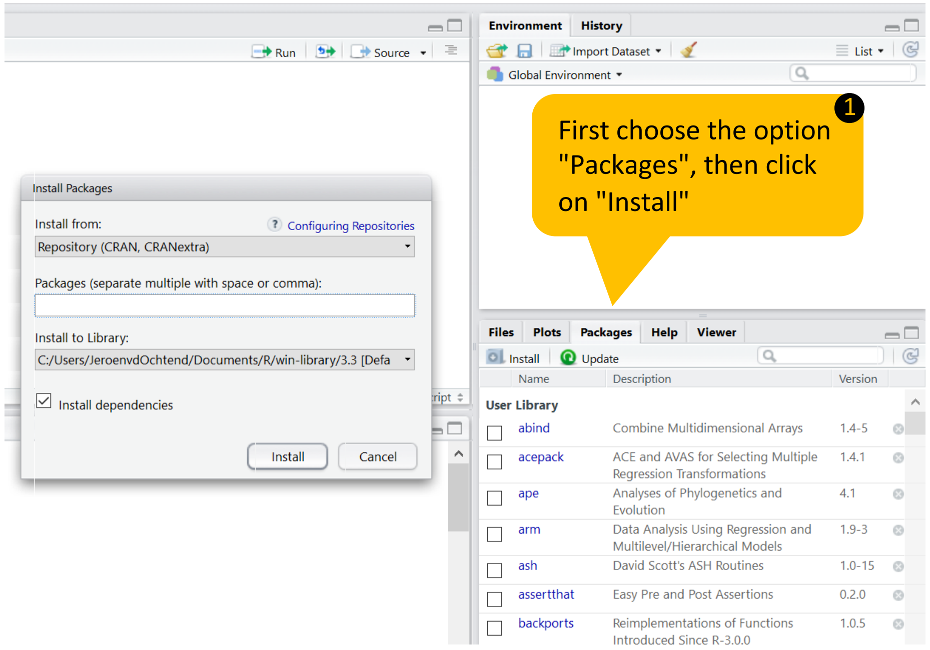
2) Use the R Shell in Rstudio

```
install.packages("data.table")
```

- Using R code instead of point and click is preferred for reproducibility reasons.

How to install R packages: Two methods

1) Use the GUI in Rstudio



First choose the option "Packages", then click on "Install"

Install Packages

Install from: Configuring Repositories
Repository (CRAN, CRANextra)

Packages (separate multiple with space or comma):

Install to Library:
C:/Users/JeroenvdOchtend/Documents/R/win-library/3.3 [Defa

☑ Install dependencies

Install | Cancel

User Library

	Name	Description	Version
☐	abind	Combine Multidimensional Arrays	1.4-5
☐	acepack	ACE and AVAS for Selecting Multiple Regression Transformations	1.4.1
☐	ape	Analyses of Phylogenetics and Evolution	4.1
☐	arm	Data Analysis Using Regression and Multilevel/Hierarchical Models	1.9-3
☐	ash	David Scott's ASH Routines	1.0-15
☐	assertthat	Easy Pre and Post Assertions	0.2.0
☐	backports	Reimplementations of Functions Introduced Since R-3.0.0	1.0.5

2) Use the R Shell in Rstudio

```
install.packages("data.table")
```

- Using R code instead of point and click is preferred for reproducibility reasons.

How to load/activate a package

```
library(data.table)
```

You have to reload your packages for every session you are using them.

How to load/activate a package

```
library(data.table)
```

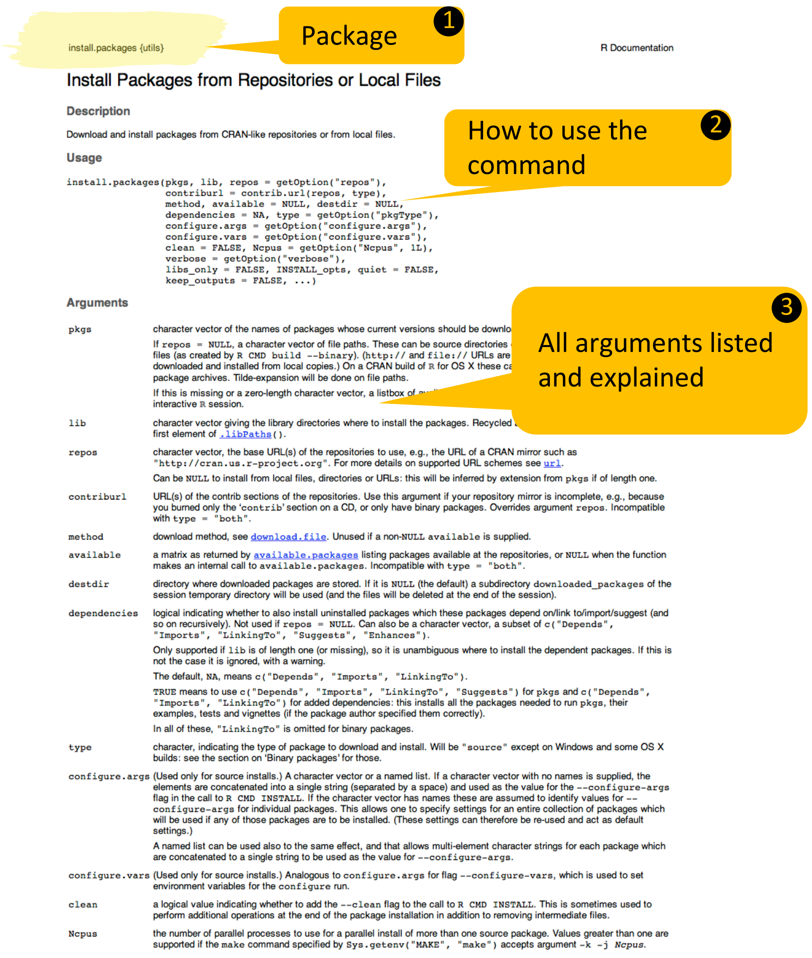
You have to reload your packages for every session you are using them.

How to load/activate a package

```
library(data.table)
```

You have to reload your packages for every session you are using them.

Start working with an R package by looking at its help files



install.packages {utils} R Documentation

Install Packages from Repositories or Local Files

Description

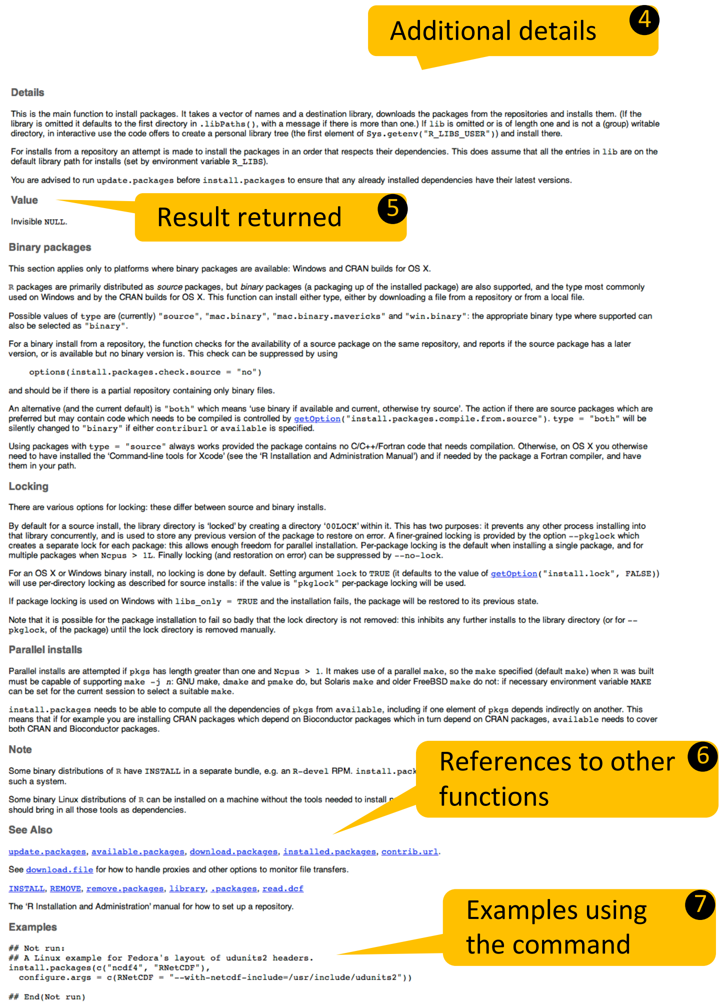
Download and install packages from CRAN-like repositories or from local files.

Usage

```
install.packages(pkgs, lib, repos = getOption("repos"),
                 contriburl = contrib.url(repos, type),
                 method, available = NULL, destdir = NULL,
                 dependencies = NA, type = getOption("pkgType"),
                 configure.args = getOption("configure.args"),
                 configure.vars = getOption("configure.vars"),
                 clean = FALSE, Ncpus = getOption("Ncpus", 1L),
                 verbose = getOption("verbose"),
                 libs_only = FALSE, INSTALL_opts, quiet = FALSE,
                 keep_outputs = FALSE, ...)
```

Arguments

`pkgs`	character vector of the names of packages whose current versions should be downlo If `repos = NULL`, a character vector of file paths. These can be source directories files (as created by `R CMD build --binary`). (`http://` and `file://` URLs are downloaded and installed from local copies.) On a CRAN build of R for OS X these ca package archives. Tilde-expansion will be done on file paths. If this is missing or a zero-length character vector, a listbox of av interactive R session.
`lib`	character vector giving the library directories where to install the packages. Recycled first element of `.libPaths()`.
`repos`	character vector, the base URL(s) of the repositories to use, e.g., the URL of a CRAN mirror such as `"http://cran.us.r-project.org"`. For more details on supported URL schemes see `url`. Can be `NULL` to install from local files, directories or URLs: this will be inferred by extension from `pkgs` if of length one.
`contriburl`	URL(s) of the contrib sections of the repositories. Use this argument if your repository mirror is incomplete, e.g., because you burned only the '`contrib`' section on a CD, or only have binary packages. Overrides argument `repos`. Incompatible with `type = "both"`.
`method`	download method, see `download.file`. Unused if a non-`NULL` `available` is supplied.
`available`	a matrix as returned by `available.packages` listing packages available at the repositories, or `NULL` when the function makes an internal call to `available.packages`. Incompatible with `type = "both"`.
`destdir`	directory where downloaded packages are stored. If it is `NULL` (the default) a subdirectory `downloaded_packages` of the session temporary directory will be used (and the files will be deleted at the end of the session).
`dependencies`	logical indicating whether to also install uninstalled packages which these packages depend on/link to/import/suggest (and so on recursively). Not used if `repos = NULL`. Can also be a character vector, a subset of `c("Depends", "Imports", "LinkingTo", "Suggests", "Enhances")`. Only supported if `lib` is of length one (or missing), so it is unambiguous where to install the dependent packages. If this is not the case it is ignored, with a warning. The default, `NA`, means `c("Depends", "Imports", "LinkingTo")`. `TRUE` means to use `c("Depends", "Imports", "LinkingTo", "Suggests")` for `pkgs` and `c("Depends", "Imports", "LinkingTo")` for added dependencies: this installs all the packages needed to run `pkgs`, their examples, tests and vignettes (if the package author specified them correctly). In all of these, `"LinkingTo"` is omitted for binary packages.
`type`	character, indicating the type of package to download and install. Will be `"source"` except on Windows and some OS X builds: see the section on 'Binary packages' for those.
`configure.args`	(Used only for source installs.) A character vector or a named list. If a character vector with no names is supplied, the elements are concatenated into a single string (separated by a space) and used as the value for the `--configure-args` flag in the call to `R CMD INSTALL`. If the character vector has names these are assumed to identify values for `--configure-args` for individual packages. This allows one to specify settings for an entire collection of packages which will be used if any of those packages are to be installed. (These settings can therefore be re-used and act as default settings.) A named list can be used also to the same effect, and that allows multi-element character strings for each package which are concatenated to a single string to be used as the value for `--configure-args`.
`configure.vars`	(Used only for source installs.) Analogous to `configure.args` for flag `--configure-vars`, which is used to set environment variables for the `configure` run.
`clean`	a logical value indicating whether to add the `--clean` flag to the call to `R CMD INSTALL`. This is sometimes used to perform additional operations at the end of the package installation in addition to removing intermediate files.
`Ncpus`	the number of parallel processes to use for a parallel install of more than one source package. Values greater than one are supported if the `make` command specified by `Sys.getenv("MAKE", "make")` accepts argument `-k -j` *Ncpus*.



Details

This is the main function to install packages. It takes a vector of names and a destination library, downloads the packages from the repositories and installs them. (If the library is omitted it defaults to the first directory in `.libPaths()`, with a message if there is more than one.) If `lib` is omitted or is of length one and is not a (group) writable directory, in interactive use the code offers to create a personal library tree (the first element of `Sys.getenv("R_LIBS_USER")`) and install there.

For installs from a repository an attempt is made to install the packages in an order that respects their dependencies. This does assume that all the entries in `lib` are on the default library path for installs (set by environment variable `R_LIBS`).

You are advised to run `update.packages` before `install.packages` to ensure that any already installed dependencies have their latest versions.

Value

Invisible `NULL`.

Binary packages

This section applies only to platforms where binary packages are available: Windows and CRAN builds for OS X.

R packages are primarily distributed as *source* packages, but *binary* packages (a packaging up of the installed package) are also supported, and the type most commonly used on Windows and by the CRAN builds for OS X. This function can install either type, either by downloading a file from a repository or from a local file.

Possible values of `type` are (currently) `"source"`, `"mac.binary"`, `"mac.binary.mavericks"` and `"win.binary"`: the appropriate binary type where supported can also be selected as `"binary"`.

For a binary install from a repository, the function checks for the availability of a source package on the same repository, and reports if the source package has a later version, or is available but no binary version is. This check can be suppressed by using

```
    options(install.packages.check.source = "no")
```

and should be if there is a partial repository containing only binary files.

An alternative (and the current default) is `"both"` which means 'use binary if available and current, otherwise try source'. The action if there are source packages which are preferred but may contain code which needs to be compiled is controlled by `getOption("install.packages.compile.from.source")`. `type = "both"` will be silently changed to `"binary"` if either `contriburl` or `available` is specified.

Using packages with `type = "source"` always works provided the package contains no C/C++/Fortran code that needs compilation. Otherwise, on OS X you otherwise need to have installed the 'Command-line tools for Xcode' (see the 'R Installation and Administration Manual') and if needed by the package a Fortran compiler, and have them in your path.

Locking

There are various options for locking: these differ between source and binary installs.

By default for a source install, the library directory is 'locked' by creating a directory '`00LOCK`' within it. This has two purposes: it prevents any other process installing into that library concurrently, and is used to store any previous version of the package to restore on error. A finer-grained locking is provided by the option `--pkglock` which creates a separate lock for each package: this allows enough freedom for parallel installation. Per-package locking is the default when installing a single package, and for multiple packages when `Ncpus > 1L`. Finally locking (and restoration on error) can be suppressed by `--no-lock`.

For an OS X or Windows binary install, no locking is done by default. Setting argument `lock` to `TRUE` (it defaults to the value of `getOption("install.lock", FALSE)`) will use per-directory locking as described for source installs: if the value is `"pkglock"` per-package locking will be used.

If package locking is used on Windows with `libs_only = TRUE` and the installation fails, the package will be restored to its previous state.

Note that it is possible for the package installation to fail so badly that the lock directory is not removed: this inhibits any further installs to the library directory (or for `--pkglock`, of the package) until the lock directory is removed manually.

Parallel installs

Parallel installs are attempted if `pkgs` has length greater than one and `Ncpus > 1`. It makes use of a parallel `make`, so the `make` specified (default `make`) when R was built must be capable of supporting `make -j` *n*: GNU make, `dmake` and `pmake` do, but Solaris `make` and older FreeBSD `make` do not: if necessary environment variable `MAKE` can be set for the current session to select a suitable `make`.

`install.packages` needs to be able to compute all the dependencies of `pkgs` from `available`, including if one element of `pkgs` depends indirectly on another. This means that if for example you are installing CRAN packages which depend on Bioconductor packages which in turn depend on CRAN packages, `available` needs to cover both CRAN and Bioconductor packages.

Note

Some binary distributions of R have `INSTALL` in a separate bundle, e.g. an `R-devel` RPM. `install.pack` such a system.

Some binary Linux distributions of R can be installed on a machine without the tools needed to install should bring in all those tools as dependencies.

See Also

`update.packages`, `available.packages`, `download.packages`, `installed.packages`, `contrib.url`.

See `download.file` for how to handle proxies and other options to monitor file transfers.

`INSTALL`, `REMOVE`, `remove.packages`, `library`, `.packages`, `read.dcf`

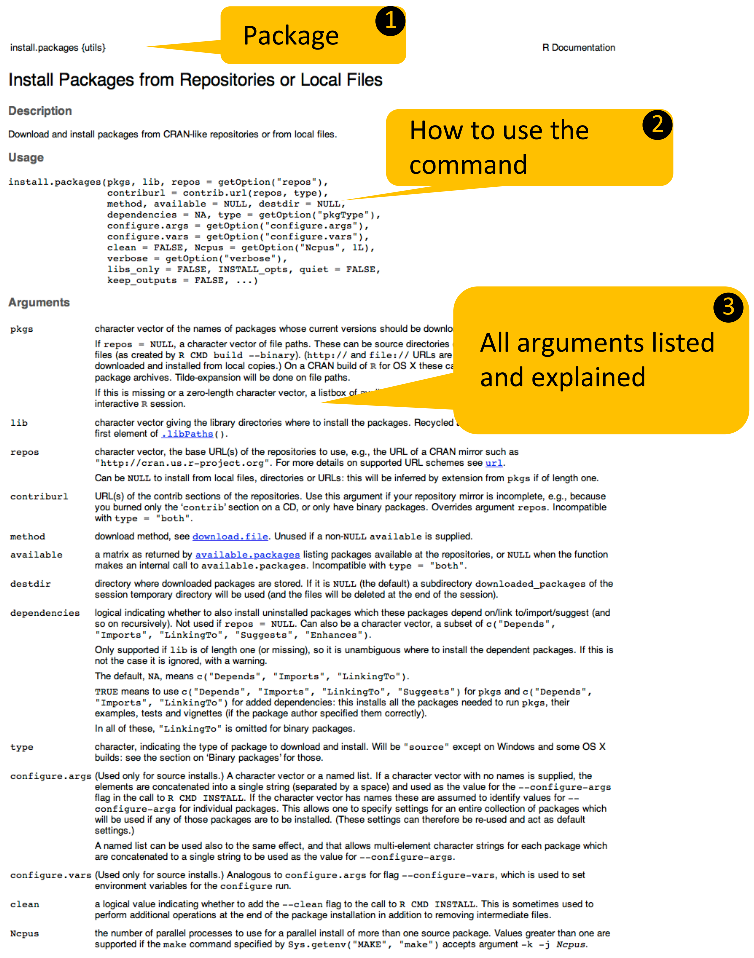
The 'R Installation and Administration' manual for how to set up a repository.

Examples

```
## Not run:
## A Linux example for Fedora's layout of udunits2 headers.
install.packages(c("ncdf4", "RNetCDF"),
  configure.args = c(RNetCDF = "--with-netcdf-include=/usr/include/udunits2"))

## End(Not run)
```

Start working with an R package by looking at its help files



install.packages {utils} R Documentation

Install Packages from Repositories or Local Files

Description

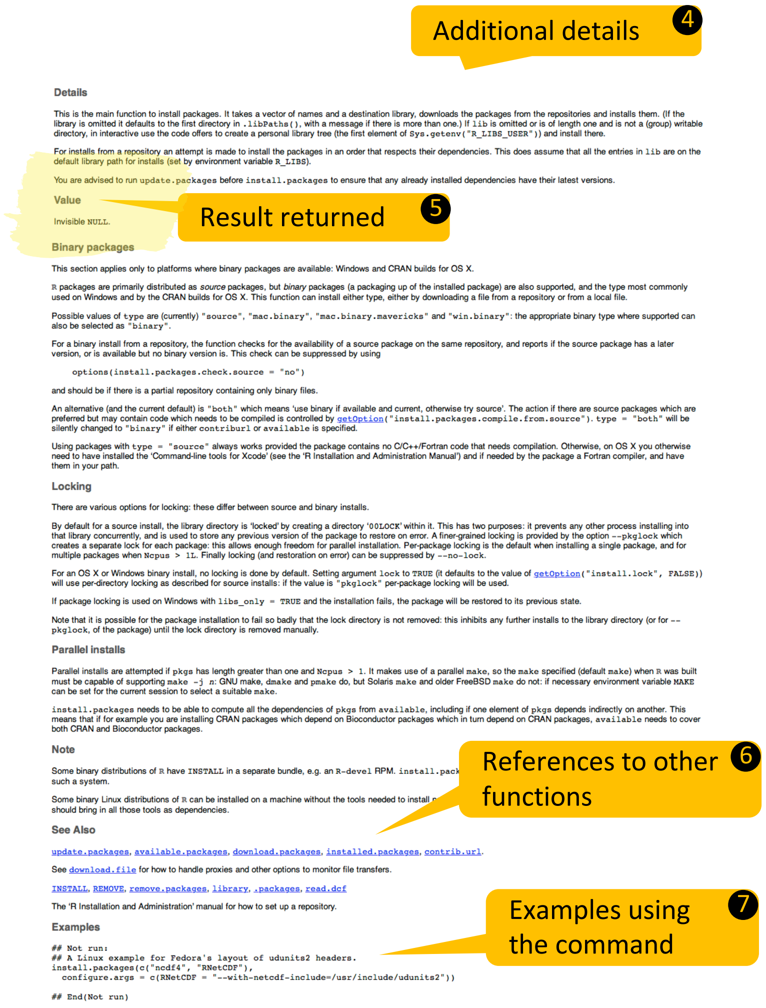
Download and install packages from CRAN-like repositories or from local files.

Usage

```
install.packages(pkgs, lib, repos = getOption("repos"),
                 contriburl = contrib.url(repos, type),
                 method, available = NULL, destdir = NULL,
                 dependencies = NA, type = getOption("pkgType"),
                 configure.args = getOption("configure.args"),
                 configure.vars = getOption("configure.vars"),
                 clean = FALSE, Ncpus = getOption("Ncpus", 1L),
                 verbose = getOption("verbose"),
                 libs_only = FALSE, INSTALL_opts, quiet = FALSE,
                 keep_outputs = FALSE, ...)
```

Arguments

`pkgs`	character vector of the names of packages whose current versions should be downlo If `repos = NULL`, a character vector of file paths. These can be source directories files (as created by `R CMD build --binary`). (`http://` and `file://` URLs are downloaded and installed from local copies.) On a CRAN build of R for OS X these ca package archives. Tilde-expansion will be done on file paths. If this is missing or a zero-length character vector, a listbox of a interactive R session.
`lib`	character vector giving the library directories where to install the packages. Recycled first element of `.libPaths()`.
`repos`	character vector, the base URL(s) of the repositories to use, e.g., the URL of a CRAN mirror such as `"http://cran.us.r-project.org"`. For more details on supported URL schemes see `url`. Can be `NULL` to install from local files, directories or URLs: this will be inferred by extension from `pkgs` if of length one.
`contriburl`	URL(s) of the contrib sections of the repositories. Use this argument if your repository mirror is incomplete, e.g., because you burned only the '`contrib`' section on a CD, or only have binary packages. Overrides argument `repos`. Incompatible with `type = "both"`.
`method`	download method, see `download.file`. Unused if a non-`NULL` `available` is supplied.
`available`	a matrix as returned by `available.packages` listing packages available at the repositories, or `NULL` when the function makes an internal call to `available.packages`. Incompatible with `type = "both"`.
`destdir`	directory where downloaded packages are stored. If it is `NULL` (the default) a subdirectory `downloaded_packages` of the session temporary directory will be used (and the files will be deleted at the end of the session).
`dependencies`	logical indicating whether to also install uninstalled packages which these packages depend on/link to/import/suggest (and so on recursively). Not used if `repos = NULL`. Can also be a character vector, a subset of `c("Depends", "Imports", "LinkingTo", "Suggests", "Enhances")`. Only supported if `lib` is of length one (or missing), so it is unambiguous where to install the dependent packages. If this is not the case it is ignored, with a warning. The default, `NA`, means `c("Depends", "Imports", "LinkingTo")`. `TRUE` means to use `c("Depends", "Imports", "LinkingTo", "Suggests")` for `pkgs` and `c("Depends", "Imports", "LinkingTo")` for added dependencies: this installs all the packages needed to run `pkgs`, their examples, tests and vignettes (if the package author specified them correctly). In all of these, `"LinkingTo"` is omitted for binary packages.
`type`	character, indicating the type of package to download and install. Will be `"source"` except on Windows and some OS X builds: see the section on 'Binary packages' for those.
`configure.args`	(Used only for source installs.) A character vector or a named list. If a character vector with no names is supplied, the elements are concatenated into a single string (separated by a space) and used as the value for the `--configure-args` flag in the call to `R CMD INSTALL`. If the character vector has names these are assumed to identify values for `--configure-args` for individual packages. This allows one to specify settings for an entire collection of packages which will be used if any of those packages are to be installed. (These settings can therefore be re-used and act as default settings.) A named list can be used also to the same effect, and that allows multi-element character strings for each package which are concatenated to a single string to be used as the value for `--configure-args`.
`configure.vars`	(Used only for source installs.) Analogous to `configure.args` for flag `--configure-vars`, which is used to set environment variables for the `configure` run.
`clean`	a logical value indicating whether to add the `--clean` flag to the call to `R CMD INSTALL`. This is sometimes used to perform additional operations at the end of the package installation in addition to removing intermediate files.
`Ncpus`	the number of parallel processes to use for a parallel install of more than one source package. Values greater than one are supported if the `make` command specified by `Sys.getenv("MAKE", "make")` accepts argument `-k -j` *`Ncpus`*.



Details

This is the main function to install packages. It takes a vector of names and a destination library, downloads the packages from the repositories and installs them. (If the library is omitted it defaults to the first directory in `.libPaths()`, with a message if there is more than one.) If `lib` is omitted or is of length one and is not a (group) writable directory, in interactive use the code offers to create a personal library tree (the first element of `Sys.getenv("R_LIBS_USER")`) and install there.

For installs from a repository an attempt is made to install the packages in an order that respects their dependencies. This does assume that all the entries in `lib` are on the default library path for installs (set by environment variable `R_LIBS`).

You are advised to run `update.packages` before `install.packages` to ensure that any already installed dependencies have their latest versions.

Value

Invisible `NULL`.

Binary packages

This section applies only to platforms where binary packages are available: Windows and CRAN builds for OS X.

R packages are primarily distributed as *source* packages, but *binary* packages (a packaging up of the installed package) are also supported, and the type most commonly used on Windows and by the CRAN builds for OS X. This function can install either type, either by downloading a file from a repository or from a local file.

Possible values of `type` are (currently) `"source"`, `"mac.binary"`, `"mac.binary.mavericks"` and `"win.binary"`: the appropriate binary type where supported can also be selected as `"binary"`.

For a binary install from a repository, the function checks for the availability of a source package on the same repository, and reports if the source package has a later version, or is available but no binary version is. This check can be suppressed by using

```
    options(install.packages.check.source = "no")
```

and should be if there is a partial repository containing only binary files.

An alternative (and the current default) is `"both"` which means 'use binary if available and current, otherwise try source'. The action if there are source packages which are preferred but may contain code which needs to be compiled is controlled by `getOption("install.packages.compile.from.source")`. `type = "both"` will be silently changed to `"binary"` if either `contriburl` or `available` is specified.

Using packages with `type = "source"` always works provided the package contains no C/C++/Fortran code that needs compilation. Otherwise, on OS X you otherwise need to have installed the 'Command-line tools for Xcode' (see the 'R Installation and Administration Manual') and if needed by the package a Fortran compiler, and have them in your path.

Locking

There are various options for locking: these differ between source and binary installs.

By default for a source install, the library directory is 'locked' by creating a directory '`00LOCK`' within it. This has two purposes: it prevents any other process installing into that library concurrently, and is used to store any previous version of the package to restore on error. A finer-grained locking is provided by the option `--pkglock` which creates a separate lock for each package: this allows enough freedom for parallel installation. Per-package locking is the default when installing a single package, and for multiple packages when `Ncpus > 1L`. Finally locking (and restoration on error) can be suppressed by `--no-lock`.

For an OS X or Windows binary install, no locking is done by default. Setting argument `lock` to `TRUE` (it defaults to the value of `getOption("install.lock", FALSE)`) will use per-directory locking as described for source installs: if the value is `"pkglock"` per-package locking will be used.

If package locking is used on Windows with `libs_only = TRUE` and the installation fails, the package will be restored to its previous state.

Note that it is possible for the package installation to fail so badly that the lock directory is not removed: this inhibits any further installs to the library directory (or for `--pkglock`, of the package) until the lock directory is removed manually.

Parallel installs

Parallel installs are attempted if `pkgs` has length greater than one and `Ncpus > 1`. It makes use of a parallel `make`, so the `make` specified (default `make`) when R was built must be capable of supporting `make -j` *n*: GNU make, `dmake` and `pmake` do, but Solaris `make` and older FreeBSD `make` do not: if necessary environment variable `MAKE` can be set for the current session to select a suitable `make`.

`install.packages` needs to be able to compute all the dependencies of `pkgs` from `available`, including if one element of `pkgs` depends indirectly on another. This means that if for example you are installing CRAN packages which depend on Bioconductor packages which in turn depend on CRAN packages, `available` needs to cover both CRAN and Bioconductor packages.

Note

Some binary distributions of R have `INSTALL` in a separate bundle, e.g. an `R-devel` RPM. `install.pack` such a system.

Some binary Linux distributions of R can be installed on a machine without the tools needed to install should bring in all those tools as dependencies.

See Also

`update.packages`, `available.packages`, `download.packages`, `installed.packages`, `contrib.url`.

See `download.file` for how to handle proxies and other options to monitor file transfers.

`INSTALL`, `REMOVE`, `remove.packages`, `library`, `.packages`, `read.dcf`

The 'R Installation and Administration' manual for how to set up a repository.

Examples

```
## Not run:
## A Linux example for Fedora's layout of udunits2 headers.
install.packages(c("ncdf4", "RNetCDF"),
  configure.args = c(RNetCDF = "--with-netcdf-include=/usr/include/udunits2"))

## End(Not run)
```

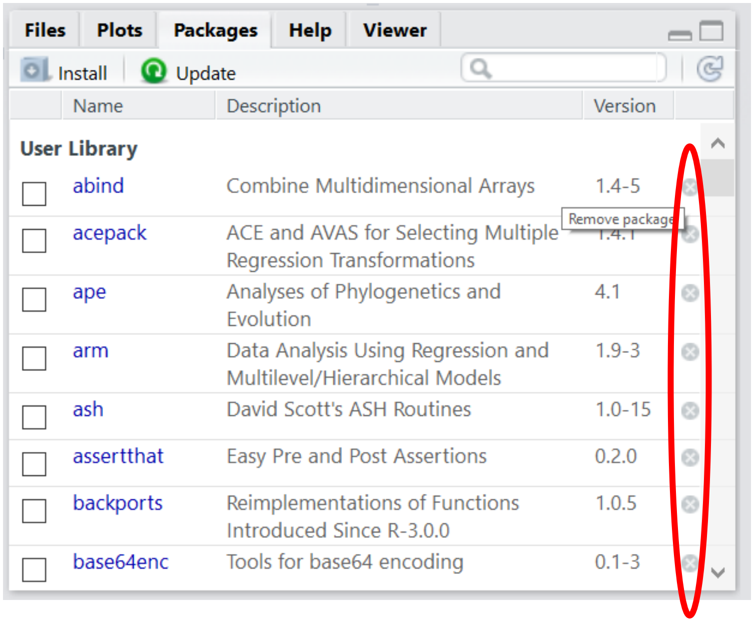
Sidenote: How to find answers in RStudio

RStudio offers multiple ways to help you with your questions:

- **`?commandName`**: shortcut for the regular `help()` command
- **`example(commandName)`**: list with examples of the command being used
- **`args(commandName)`**: list of a command's arguments
- **`help.search("search term")`**: search through R's help documentation for a specific term
- **`??"search term"`**: shortcut for the `help.search()` command

Remove an R package
Two methods

1) Use the GUI in RStudio



Name	Description	Version
User Library		
abind	Combine Multidimensional Arrays	1.4-5
acepack	ACE and AVAS for Selecting Multiple Regression Transformations	1.4.1
ape	Analyses of Phylogenetics and Evolution	4.1
arm	Data Analysis Using Regression and Multilevel/Hierarchical Models	1.9-3
ash	David Scott's ASH Routines	1.0-15
assertthat	Easy Pre and Post Assertions	0.2.0
backports	Reimplementations of Functions Introduced Since R-3.0.0	1.0.5
base64enc	Tools for base64 encoding	0.1-3

2) Use the R Shell in RStudio

```
remove.packages(“data.table”)
```