Package: librarian (via r-universe)

October 26, 2024

Title Install, Update, Load Packages from CRAN, 'GitHub', and 'Bioconductor' in One Step

Version 1.8.1

Date 2021-07-12

Description Automatically install, update, and load 'CRAN', 'GitHub', and 'Bioconductor' packages in a single function call. By accepting bare unquoted names for packages, it's easy to add or remove packages from the list.

URL https://github.com/DesiQuintans/librarian

BugReports https://github.com/DesiQuintans/librarian/issues

Depends R (>= 3.5.0)

License GPL-3

Encoding UTF-8

Imports BiocManager, remotes, tools, utils

RoxygenNote 7.1.1

Suggests testthat, knitr, rmarkdown

Language en-GB

VignetteBuilder knitr

Repository https://desiquintans.r-universe.dev

RemoteUrl https://github.com/desiquintans/librarian

RemoteRef HEAD

RemoteSha ebf4743eaa304aed4dd96246b9fa2bdac8d513e9

Contents

`browse_cran` *Search for CRAN packages by keyword/regex*

Description

Inspired by my mysterious inability to remember what the RColorBrewer package is actually called. Lets you find relevant CRAN packages right from your terminal.

Usage

```
browse_cran(query, fuzzy = FALSE, echo = TRUE, ignore.case = TRUE)
```

Arguments

`query`	(Character) A string to `grep()` for.
`fuzzy`	(Logical) If `TRUE`, enables fuzzy orderless matching. Every word in `query` (i.e. every group of characters separated with a space) will be wrapped with a lookaround `(?=*KEYWORD)`. This will match keywords regardless of the order in which those words appear.
`echo`	(Logical) If `TRUE`, print the results to the console.
`ignore.case`	(Logical) If `TRUE`, ignore upper/lowercase differences while searching.

Details

When `browse_cran()` is run for the first time in a new session, it will take about 6-12 seconds to download and cache CRAN data. This only happens once per session; subsequent calls will use the cached copy.

Value

Invisibly returns a dataframe of the packages that matched the query together with their descriptions. Prints results to the console.

Examples

```
browse_cran("colorbrewer")  # Search by keyword

#> RColorBrewer
#>     Provides color schemes for maps (and other graphics) designed by Cynthia
#>     Brewer as described at http://colorbrewer2.org
#>
#> Redmonder
#>     Provide color schemes for maps (and other graphics) based on the color
#>     palettes of several Microsoft(r) products.


browse_cran("zero-inflat.*?(abund|count)")  # Search by regular expression

#> hurdlr
#>     When considering count data, it is often the case that many more zero
#>     counts than would be expected of some given distribution are observed.

# And five other matches...


browse_cran("network twitter api", fuzzy = TRUE)  # Order-agnostic (fuzzy) search

#> RKlout
#>     An interface of R to Klout API v2.
```

---

check_attached *Check if packages are attached*

Description

Check if packages are attached

Usage

```
check_attached(...)
```

Arguments

...	(Dots) Package names as bare names, strings, or a character vector. If left empty, lists all attached packages.

Value

If `dots` is empty, a character vector of all attached packages. Otherwise, return a named logical vector where `TRUE` means the package is attached

Examples

```
## Not run:
check_attached()

#>   [1] "librarian" "testthat"  "magrittr"  "stats" ...

check_attached(c("utils", "stats"))

#> utils stats
#> TRUE  TRUE

check_attached("datasets", "base", fakepkg)

#> datasets     base  fakepkg
#>     TRUE     TRUE    FALSE

## End(Not run)
```

`check_installed` *Check if packages are installed*

Description

Check if packages are installed

Usage

```
check_installed(...)
```

Arguments

...	(Dots) Package names as bare names, strings, or a character vector. If left empty, lists all installed packages.

Value

If `dots` is empty, a character vector of all installed packages. Otherwise, return a named logical vector where `TRUE` means the package is installed.

Examples

```
## Not run:
check_installed()

#>   [1] "addinslist"  "antiword" " ape"  "assertthat"  ...

check_installed(c("utils", "stats"))

#> utils stats
#> TRUE  TRUE

check_installed("datasets", "base", fakepkg)

#> datasets     base  fakepkg
#>     TRUE     TRUE    FALSE

## End(Not run)
```

`check_pkg_status` *Check if packages are installed or attached*

Description

Check if packages are installed or attached

Usage

```
check_pkg_status(..., status, use_list = FALSE)
```

Arguments

`...`	(Dots) Package names as bare names, strings, or a vector of strings. If left blank, returns a list of all packages that are installed/attached depending on the value of `status`.
`status`	(Character) `"installed"` checks if packages are installed. `"attached"` checks if packages are currently attached.
`use_list`	(Logical) If `TRUE`, a character vector of package names was passed in `..1`, so use that as the results list. This is for programming use; `nse_dots()` already detects if a char vector of length > 1 is in `..1` and uses it as the package list automatically, but it does not do that for char vectors of length 1 because the user can offer a mix of names and strings to `...` as a convenience.

Value

If `dots` is empty, a character vector of package names. Otherwise, return a named logical vector where `TRUE` means the package is installed or attached, depending on the value of `status`.

collapse_vec *Collapse a vector*

Description

I use this internally for turning a vector of package names into a string.

Usage

```
collapse_vec(..., wrap = "'", collapse = ", ", unique = TRUE)
```

Arguments

...	(...) Vectors that will be concatenated and coerced to Character.
wrap	(Character) Placed at the left and right sides of each vector element.
collapse	(Character) Placed between each element of the original vector(s).
unique	(Logical) If TRUE, duplicate entries in ... will be removed.

Value

A string.

Examples

```
## Not run:
collapse_vec(month.abb)
#> [1] "'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'"

## End(Not run)
```

dots1_is_pkglist *Is the 1st 'dots' arg a character vector with length > 1?*

Description

Is the 1st 'dots' arg a character vector with length > 1?

Usage

```
dots1_is_pkglist(...)
```

Arguments

...	(Dots)

Value

TRUE if ..1 is a vector or list with length > 1.

Examples

```
## Not run:
dots1_is_pkglist()

#> [1] FALSE

dots1_is_pkglist("hello", "hey", "hi")

#> [1] FALSE

dots1_is_pkglist(c("hello", "hey"), "hi")

#> [1] TRUE

dots1_is_pkglist(c(hello, hey), "hi")

#> [1] FALSE

# A common programming scenario:
pkg_list <- c("only_one_package")
dots1_is_pkglist(pkg_list)

#> [1] TRUE

## End(Not run)
```

---

dots_is_empty *Did the user pass arguments inside dots?*

Description

Did the user pass arguments inside dots?

Usage

```
dots_is_empty(...)
```

Arguments

...	(Dots)

Value

TRUE (dots is empty) or FALSE (dots is not empty).

Examples

```
## Not run:
is_dots_empty(package, names, here)

#> [1] FALSE

## End(Not run)
```

---

dots_length *How many items are in dots?*

Description

How many items are in dots?

Usage

```
dots_length(...)
```

Arguments

...	(Dots)

Value

An integer

Examples

```
## Not run:
dots_length(package, names, here)

#> [1] 3

## End(Not run)
```

fuzzy_needle *Turn a list of words into a fuzzy regex*

Description

A fuzzy regex is one that will match search terms in any order by using PERL lookaround. This is very slow, but often worth the cost to get more complete results.

Usage

```
fuzzy_needle(vec)
```

Arguments

vec	(Character) A string containing space-separated keywords to search for.

Value

A string where each word has been wrapped as a lookaround term.

Examples

```
## Not run:
fuzzy_needle("network centrality")
#> [1] "(?=.*network)(?=.*centrality)"

## End(Not run)
```

is_valid_url *Assert that a URL is complete and valid*

Description

Assert that a URL is complete and valid

Usage

```
is_valid_url(string)
```

Arguments

string	(Character) A URL to check.

Details

The regex I use is "@stephenhay" from https://mathiasbynens.be/demo/url-regex because it's the shortest regex that matches every CRAN mirror at https://cran.r-project.org/mirrors.html.

Value

A logical value, TRUE if the URL is valid, FALSE if otherwise.

Examples

```
## Not run:
is_valid_url("http://rstudio.com")

## End(Not run)
```

---

lib_paths *Changing and viewing the package search paths*

Description

View and edit the list of folders that R will look inside when trying to find a package. Add an existing folder, create and add a new folder, or shuffle a folder to the front of the list so that it is used as the default installation location for new packages in the current session.

Usage

```
lib_paths(path, make_path = TRUE, ask = TRUE)
```

Arguments

path	(Character, or omit) A path to add to the library search path. Can be an absolute or relative path. If path has more than one element, only the first one will be kept. Tilde expansion is performed on the input, but wildcard expansion (globbing) is not. If path is omitted, return the current library search path.
make_path	(Logical) If TRUE, create path's directory structure if it doesn't exist.
ask	(Logical) If TRUE, ask before creating path's directory structure if make_path = TRUE. Ignored if make_path = FALSE.

Value

A character vector of the folders on the library search path. If path was not omitted, it will be the first element.

Examples

```
lib_paths()

#> [1] "D:/R/R-3.5.2/library"

lib_paths(file.path(tempdir(), "newlibraryfolder"), ask = FALSE)

#> [1] "C:/Users/.../Temp/Rtmp0Qbvgo/newlibraryfolder"
```

```
#> [2] "D:/R/R-3.5.2/library"
```

`lib_startup` *Set packages and library paths to automatically start-up with R*

Description

This function tells R to load packages and library folders at the start of every session (or on a per-project basis). It's best to keep this auto-load list to a minimum so that you don't forget to explicitly install/attach packages in scripts that need them.

Usage

```
lib_startup(..., lib = lib_paths(), global = TRUE)
```

Arguments

`...`	(Names) Packages as bare names. For packages that come from GitHub, you can keep the username/package format, or omit the username and provide just the package name. If you leave `...` blank, R will only load its default packages (see Details).
`lib`	(Character) The path where packages are installed. Can be an absolute or relative path. If `path` has more than one element, only the first one will be kept. Tilde expansion is performed on the input, but wildcard expansion (globbing) is not. Defaults to the current library search path.
`global`	(Logical) If `TRUE`, write these settings to a .Rprofile file in the home directory (on Windows, the My Documents folder). If `FALSE`, write them to a .Rprofile file that is in the current directory (i.e. the RStudio project's folder, or the current working directory). See Details for more.

Details

R's startup order is mentioned in `?Startup`, but briefly:

1. R tries to load the environmental variables file (Renviron.site)
2. R tries to load the site-wide profile (Rprofile.site)
3. R tries to load the user profile (.Rprofile), first in the current directory, and then in the user's home directory (on Windows, the My Documents folder). **Only one of these files is sourced into the workspace.**

Omitting `...` makes R load only its default packages. If these are not set in an environmental variable (`R_DEFAULT_PACKAGES`), then R will default to loading these packages: datasets, utils, grDevices, graphics, stats, and methods.

Value

A message listing the values that were written to the .Rprofile file.

Examples

```
#> lib_startup(librarian, magrittr, lib = "C:/Dropbox/My R Library")
```

`list_dependencies` *List the dependencies of selected packages*

Description

List the dependencies of selected packages

Usage

```
list_dependencies(of_pkgs, which = c("Depends", "Imports"))
```

Arguments

`of_pkgs`	(Character) Packages whose dependencies will be found.
`which`	(Character) The types of dependencies to find.

Value

A character vector of package names. Note that all dependencies of all requested packages will be placed into the one vector.

`make_dirs` *Build a path, creating subfolders if needed*

Description

Whereas `base::file.path()` only concatenates strings to build a path, `make_dirs()` *also* makes sure those folders exist.

Usage

```
make_dirs(...)
```

Arguments

`...`	(Character) Arguments to send to `file.path()`. You can provide a complete path as a single string, or incrementally build a path with many strings.

Value

(Character) A file path. Automatically adds trailing slashes if required.

Authors

- Desi Quintans (http://www.desiquintans.com)

Source

- Desiderata package (https://github.com/DesiQuintans/desiderata)

Examples

```
## Not run:
make_dirs(tempdir(), "newfolder")

#> [1] "C:/Users/.../Temp/RtmpSwZA8X/newfolder"

## End(Not run)
```

---

nse_dots *Convert dots to package names*

Description

Convert dots to package names

Usage

```
nse_dots(..., keep_user = FALSE)
```

Arguments

...	(Dots) Package names provided as bare names or strings (of length 1). If a character vector is provided as the first argument, it will be used and all other arguments in dots will be ignored.
keep_user	(Logical) If FALSE, omit the username from a GitHub package reference.

Value

A character vector.

Examples

```
## Not run:
nse_dots(dplyr, DesiQuintans/desiderata, keep_user = FALSE)

#> [1] "dplyr"   "desiderata"

nse_dots(dplyr, DesiQuintans/desiderata, keep_user = TRUE)

#> [1] "dplyr"   "DesiQuintans/desiderata"

## End(Not run)
```

---

reshelf *Detach and then reattach packages to the search path*

Description

Convenience shortcut for force-unshelfing packages and then shelfing them again.

Usage

```
reshelf(...)
```

Arguments

...	(Names) Packages as bare names. For packages that come from GitHub, you can keep the username/package format, or omit the username and provide just the package name.

Value

Invisibly returns a named logical vector, where the names are the packages requested in ... and TRUE means that the package was successfully attached.

Examples

```
reshelf(datasets)

# reshelf() returns invisibly; bind its output to a variable or access the .Last.value.

print(.Last.value)

#> datasets
#>     TRUE
```

sentence *Keep the first sentence of a string.*

Description

Keep the first sentence of a string.

Usage

```
sentence(string)
```

Arguments

string	(Character) A string.

Value

The string with only the first sentence.

Examples

```
## Not run:
sentence("This is a sentence. And this is another sentence.")

#> [1] "This is a sentence."

sentence("This is just one sentence.")

#> [1] "This is just one sentence."

sentence("Is this a sentence? Or is this one. Maybe this one! What if there are lots of sentences?")

#> [1] "Is this a sentence?

## End(Not run)
```

shelf *Attach packages to the search path, installing them from CRAN, GitHub, or Bioconductor if needed*

Description

Attach packages to the search path, installing them from CRAN, GitHub, or Bioconductor if needed

Usage

```
shelf(
  ...,
  lib = NULL,
  update_all = FALSE,
  quiet = FALSE,
  ask = TRUE,
  cran_repo = getOption("repos"),
  bioc_repo = character()
)
```

Arguments

`...`	(Names) Packages as bare names. If the package is from GitHub, include both the username and package name as UserName/package (see examples).
`lib`	(Character) By R convention, packages are installed to the first folder in your library search path (`lib_paths()`). Here, you can set a specific folder to install new packages to instead. If the folder doesn't exist, you will be prompted to create it if `ask = TRUE`, otherwise it will be silently created. Can be an absolute or relative path. Tilde expansion is performed on the input, but wildcard expansion (globbing) is not. If `lib` has more than one element, only the first one will be kept. See the 'Details' section below for more information.
`update_all`	(Logical) If `TRUE`, the packages will be re-installed even if they are already in your library.
`quiet`	(Logical) If `TRUE`, suppresses most warnings and messages.
`ask`	(Logical) If `TRUE`, and `lib` points to a folder that doesn't exist, ask before creating the folder. If `FALSE`, the folder will be created silently.
`cran_repo`	(Character) In RStudio, a default CRAN repo can be set via *Options* > *Packages* > *Default CRAN Mirror*). Otherwise, provide the URL to CRAN or one of its mirrors. If an invalid URL is given, defaults to https://cran.r-project.org.
`bioc_repo`	(Character) If you use Bioconductor, you can set the repo URLs for it here. Defaults to Bioconductor's defaults (view them with `BiocInstaller::biocinstallRepos()`).

Details

You may choose to organise your library into folders to hold packages for different tasks or projects. If you specify a `lib` folder, it will be created (if needed) and attached to the package search path. R will look for packages by working through the package search path in order. You can view the folders that are on this path by calling `lib_paths()` with no arguments.

If you specify a new `lib` and use the argument `update_all = TRUE` to force an already-installed package to reinstall, a new copy of that package will be made in `lib` and then loaded from there. This means that you can potentially have several copies of the same package across many folders on your machine, each a different version. This allows you to maintain a different library folder for different projects, so that updated packages in Project B will not affect the package versions you rely on for Project A.

Value

Invisibly returns a named logical vector, where the names are the packages requested in ... and TRUE means that the package was successfully installed and attached.

Examples

```
shelf(fortunes, DesiQuintans/emptyRpackage, cowsay, lib = tempdir(), update_all = TRUE)

# shelf() returns invisibly; bind its output to a variable or access the .Last.value.

print(.Last.value)

#> fortunes emptyRpackage        cowsay
#>     TRUE          TRUE          TRUE
```

---

shhh *Suppresses console output, including printing*

Description

This is copied from my personal package, desiderata.

Usage

```
shhh(expr)
```

Arguments

expr	(Expression) An expression to evaluate.

Value

Evaluates expr.

---

stock *Install packages from CRAN, GitHub, or Bioconductor if needed/wanted*

Description

Install packages from CRAN, GitHub, or Bioconductor if needed/wanted

Usage

```
stock(
  ...,
  lib = NULL,
  update_all = FALSE,
  quiet = FALSE,
  ask = TRUE,
  cran_repo = getOption("repos"),
  bioc_repo = character()
)
```

Arguments

`...`	(Names) Packages as bare names. If the package is from GitHub, include both the username and package name as UserName/package (see examples).
`lib`	(Character) By R convention, packages are installed to the first folder in your library search path (`lib_paths()`). Here, you can set a specific folder to install new packages to instead. If the folder doesn't exist, you will be prompted to create it if `ask = TRUE`, otherwise it will be silently created. Can be an absolute or relative path. Tilde expansion is performed on the input, but wildcard expansion (globbing) is not. If `lib` has more than one element, only the first one will be kept. See the 'Details' section below for more information.
`update_all`	(Logical) If `TRUE`, the packages will be re-installed even if they are already in your library.
`quiet`	(Logical) If `TRUE`, suppresses most warnings and messages.
`ask`	(Logical) If `TRUE`, and `lib` points to a folder that doesn't exist, ask before creating the folder. If `FALSE`, the folder will be created silently.
`cran_repo`	(Character) In RStudio, a default CRAN repo can be set via *Options* > *Packages* > *Default CRAN Mirror*). Otherwise, provide the URL to CRAN or one of its mirrors. If an invalid URL is given, defaults to https://cran.r-project.org.
`bioc_repo`	(Character) If you use Bioconductor, you can set the repo URLs for it here. Defaults to Bioconductor's defaults (view them with `BiocInstaller::biocinstallRepos()`).

Details

You may choose to organise your library into folders to hold packages for different tasks or projects. If you specify a `lib` folder, it will be created (if needed) and attached to the package search path. R will look for packages by working through the package search path in order. You can view the folders that are on this path by calling `lib_paths()` with no arguments.

If you specify a new `lib` and use the argument `update_all = TRUE` to force an already-installed package to reinstall, a new copy of that package will be made in `lib` and then loaded from there. This means that you can potentially have several copies of the same package across many folders on your machine, each a different version. This allows you to maintain a different library folder for different projects, so that updated packages in Project B will not affect the package versions you rely on for Project A.

Value

Invisibly returns a named logical vector, where the names are the packages requested in ... and TRUE means that the package was successfully installed.

Examples

```
stock(fortunes, DesiQuintans/emptyRpackage, cowsay, lib = tempdir(), update_all = TRUE)

# shelf() returns invisibly; bind its output to a variable or access the .Last.value.

print(.Last.value)

#> fortunes emptyRpackage        cowsay
#>     TRUE          TRUE          TRUE

# And to confirm that they are installed but not attached:

check_attached(fortunes, DesiQuintans/emptyRpackage, cowsay)

#> fortunes emptyRpackage        cowsay
#>    FALSE         FALSE         FALSE
```

---

tell_user *Messages for the user*

Description

Messages for the user

Usage

```
tell_user(message, ...)
```

Arguments

message	(Character) An identifier string for a message.
...	(Dots) Data to pass into the message for sprintf().

Value

A string.

Examples

```
## Not run:
message(tell_user("not allowed to make path", "C:/fakefolder"))

## End(Not run)
```

`unshelf` *Detach (unload) packages from the search path*

Description

Packages can be detached by themselves, with their dependencies safely (i.e. as long as those dependencies are not being used by other packages), or with their dependencies unsafely (regardless of whether those dependencies are still needed). All non-default packages can be detached at once too, including Librarian itself.

Usage

```
unshelf(
  ...,
  everything = FALSE,
  also_depends = FALSE,
  safe = TRUE,
  quiet = TRUE
)
```

Arguments

`...`	(Names) Packages as bare names. For packages that come from GitHub, you can keep the username/package format, or omit the username and provide just the package name.
`everything`	(Logical) If `TRUE`, detach every non-default package including librarian. Any names in `...` are ignored. The default packages can be listed with `getOption("defaultPackages")`.
`also_depends`	(Logical) If `TRUE`, also detach the dependencies of the packages listed in `....` This can be slow.
`safe`	(Logical) If `TRUE`, packages won't be detached if they are needed by other packages that are **not** listed in `....`
`quiet`	(Logical) If `FALSE`, show a message when packages can't be detached because they are still needed by other packages.

Value

Invisibly returns a named logical vector, where the names are the packages and `TRUE` means that the package was successfully detached.

Examples

```
# These are the same:

#> unshelf(janitor, desiderata, purrr)
#> unshelf(janitor, DesiQuintans/desiderata, purrr)

# unshelf() returns invisibly; bind its output to a variable or access the .Last.value.

#> print(.Last.value)

#> desiderata    janitor      purrr
#>       TRUE       TRUE       TRUE

#> unshelf(everything = TRUE)
#> print(.Last.value)

#> librarian testthat
#> TRUE      TRUE
```

`wrap_text` *Produce a nicely-wrapped paragraph for console printing*

Description

Wrapping text needs to be done separately from actually printing it with `stop` or `warning` or `message`. This is because these functions typically also print some information about the environment where they were called.

Usage

```
wrap_text(...)
```

Arguments

`...`	Vectors to be coerced to Character.

Value

The text in `...` will be collapsed and wrapped.

Examples

```
## Not run:
wrapped <-
wrap_text(
    "Lorem ipsum dolor sit amet, ornare justo condimentum",
    "et sit lorem! Himenaeos, vel et sodales sit.",
```

```
    "Eu nulla. Magna ullamcorper nascetur placerat platea.\n\n",
    "Eleifend semper velit sed aliquam, ut ligula non commodo.")

cat(wrapped)

#>  Lorem ipsum dolor sit amet, ornare justo condimentum et sit lorem!
#>  Himenaeos, vel et sodales sit. Eu nulla. Magna ullamcorper
#>  nascetur placerat platea.
#>
#>  Eleifend semper velit sed aliquam, ut ligula non commodo.

## End(Not run)
```

Index