Package: origin (via r-universe)

October 18, 2024

Type Package

Title Explicitly Qualifying Namespaces by Automatically Adding 'pkg::' to Functions

Version 1.1.2

Author Matthias Nistler

Maintainer Matthias Nistler <m_nistler@web.de>

Description Automatically adding 'pkg::' to a function, i.e. mutate() becomes dplyr::mutate(). It is up to the user to determine which packages should be used explicitly, whether to include base R packages or use the functionality on selected text, a file, or a complete directory. User friendly logging is provided in the 'RStudio' Markers pane. Lives in the spirit of 'lintr' and 'styler'. Can also be used for checking which packages are actually used in a project.

License MIT + file LICENSE

URL https://github.com/mnist91/origin

BugReports https://github.com/mnist91/origin/issues

Depends R (>= 2.10)

Imports cli, rstudioapi, stats, utils

Suggests data.table, dplyr, knitr, purrr, rmarkdown, testthat

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.1

Repository https://mnist91.r-universe.dev

RemoteUrl https://github.com/mnist91/origin

RemoteRef HEAD

RemoteSha 42e2e0b8152b37cc46ec9bcf41aecfe08cf9487a

Contents

check_pkg_usage 2
get_exported_functions 3
get_local_functions 4
get_pkgs_from_description 4
originize_dir 5
originize_file 7
originize_pkg 9
originize_selection 11
print.pkg_usage 12

Index **14**

`check_pkg_usage` *Check which packages are actually used in a project*

Description

Provide a folder and a vector of package names to check, which packages are actually in use and which functions are used but not exported by the provided packages.

Usage

```
check_pkg_usage(
  path = getwd(),
  pkgs = getOption("origin.pkgs", .packages()),
  recursive = TRUE,
  exclude_files = NULL,
  path_to_local_functions = NULL,
  check_local_conflicts = TRUE,
  use_markers = TRUE
)
```

Arguments

`path`	a character vector of full path names; the default corresponds to the working directory, getwd()
`pkgs`	a character vector with package names. Defaults to the result of .packages but only if the option 'origin.pkgs' is not specified.
`recursive`	logical. Should the listing recurse into directories?
`exclude_files`	a character vector of file paths that should be excluded from being checked Helpful if all but a few files should be considered by origin.
`path_to_local_functions`	file path. Helpful if all project specific functions are defined in a specific folder. This folder might not be a sub directory of the current project so the default to just find all function definitions would not be sufficient.

check_local_conflicts
: if TRUE, it is checked whether locally defined functions inside of the project mask exported functions packages listed in pkgs. It avoids mistakenly adding pkg:: to a custom local function.

use_markers
: a boolean. If TRUE, the markers tab inn RStudio is used to track changes and show issues. FALSE prints the same information in the console.

Value

‘data.frame‘ invisibly, It consists of 5 columns. - ‘pkg‘: the package that exports this function - ‘fun‘: all functions in alphabetical order - ‘n_calls‘: how often the function has been used in the files - ‘namespaced‘: logical, whether the function has been called explicitly via ‘pkg::fct‘ or implicitly by an attached package - ‘conflict‘: whether this function is exported by multiple checked packages - ‘conflict_pkgs‘: in case of a conflict, which packages does export the same function but are masked Note that functions for that it is unknown from which package they are exported have an ‘NA‘ in the ‘pkg‘ column. Similarly, Packages that are checked but no functions from these are used are listed but have an ‘NA‘ in the ‘fun‘ column

Examples

```
## Not run:
check_pkg_usage()

## End(Not run)
```

---

get_exported_functions

Get All Exported Functions From a Package

Description

Get All Exported Functions From a Package

Usage

```
get_exported_functions(pkg)
```

Arguments

pkg
: a character string of a package name

Value

character vector of functions names

Examples

```
get_exported_functions("base")
```

get_local_functions *Find All User Defined functions in the Project*

Description

Find All User Defined functions in the Project

Usage

```
get_local_functions(path = ".")
```

Arguments

path	Path in which all defined function names should be found and retrieved. Defaults to the current working directory.

Value

character vector of function names

Examples

```
get_local_functions(path = ".")
get_local_functions(path = rstudioapi::getActiveProject())
```

get_pkgs_from_description *Get Packages from the DESCRIPTION file*

Description

It looks for a DESCRIPTION file in the current project and returns all packages listed in Suggests, Imports, and Depends.

Usage

```
get_pkgs_from_description(path = NULL)
```

Arguments

path	Path to a DESC>RIPTION file, If 'NULL' (default), the functions searches for a description file in the current active project

Value

character vector of package names

Examples

```
# Only works inside of a package developing project
## Not run:
get_pkgs_from_description()

## End(Not run)
```

---

originize_dir *Originize a complete directory*

Description

To originize complete folders/projects, this function finds and originizes all R files within this folder and (by default) its subdirectories.

Usage

```
originize_dir(
  path = getwd(),
  pkgs = getOption("origin.pkgs", .packages()),
  recursive = TRUE,
  exclude_files = NULL,
  overwrite = getOption("origin.overwrite", TRUE),
 ask_before_applying_changes = getOption("origin.ask_before_applying_changes", TRUE),
  check_conflicts = getOption("origin.check_conflicts", TRUE),
  check_base_conflicts = getOption("origin.check_base_conflicts", TRUE),
  path_to_local_functions = getOption("origin.path_to_local_functions", NULL),
  check_local_conflicts = getOption("origin.check_local_conflicts", TRUE),
  add_base_packages = getOption("origin.add_base_packages", FALSE),
  excluded_functions = getOption("origin.excluded_functions", list()),
  verbose = getOption("origin.verbose", FALSE),
  use_markers = getOption("origin.use_markers_for_logging", TRUE)
)
```

Arguments

path	path to a directory. Defaults to the current working directory.
pkgs	a character vector with package names. Defaults to the result of .packages but only if the option 'origin.pkgs' is not specified.
recursive	logical. Should scripts be originized recursively, this means that all files in the subfolders will be searched as well. See list.files
exclude_files	a character vector of file paths that should be excluded excluded from being originized. Helpful if all but a few files should be considered by origin.
overwrite	if TRUE the file will be saved and overwritten. Otherwise, only the logging is triggered. Note that, depending on 'ask_before_applying_changes', the user is ask whether the result is as desired.

`ask_before_applying_changes`
: if TRUE, the user has to approve changes made by origin prior to applying them. Note that this mutes all checks, i.e. large number of files, local functions mask exported functions, and the presence and order of function conflicts.

`check_conflicts`
: if TRUE, possible namespace conflicts between functions exported by packages listed in pkgs are checked. See details.

`check_base_conflicts`
: if TRUE; native R functions are also included in checking for conflicts. See details.

`path_to_local_functions`
: Path to search for local functions that mask all exported functions from originizing. If NULL, defaults to the current RStudio Project root.

`check_local_conflicts`
: if TRUE, it is checked whether locally defined functions inside of the project mask exported functions packages listed in pkgs. It avoids mistakenly adding pkg:: to a custom local function.

`add_base_packages`
: a boolean. If TRUE, base R functions are handled like all other packages and added via '::'

`excluded_functions`
: a list. Either an unnamed list of function names as strings. These functions are excluded from all packages and never considered in origin. Or a named list with character vectors, Then the name of the list element refers to a package and the given functions are only excluded from this package. A very explicit way to handle namespace conflicts or highlighting popular infix functions like '%>%' or ':='.

`verbose`
: if TRUE, origin provides a logging output about its results.

`use_markers`
: a boolean. If TRUE, the markers tab inn RStudio is used to track changes and show issues. FALSE prints the same information in the console.

Details

check_conflicts checks whether multiple packages listed in pkgs export functions with the same name, e.g. lag() is both part of the dplyr and data.table namespace. If there are any conflicts actually present in any considered script, these conflicts are shown including how origin would solve them. User input is required to proceed. The order in pkgs determines the precedence, while those listed first have higher precedence than those listed later in the vector. This is consistent with function masking in R.

check_base_conflicts checks whether functions listed in pkgs mask R functions of R core packages (base, utils, stats, methods, graphics, grDevices, datasets). Even tough the user might not include those functions in the pkg::fct logic, potential conflicts require careful evaluation.

Value

No return value, called for side effects

Examples

```
## Not run:
originize_dir(path = "folder_to_originize",
              pkgs = c("dplyr", "data.table"),
              overwrite = TRUE,
              ask_before_applying_changes = TRUE,
              excluded_functions = list(dplyr = c("%>%", "tibble"),
                                        data.table = c(":=", "%like%"),
                                        # generally exclude
                                        c("last", "first")),
              exclude_files = c("dont_originize_this.R",
                                "dont_originize_that.R"),
              verbose = TRUE)

## End(Not run)
```

`originize_file` *Originize a specific file*

Description

Originize a specific file

Usage

```
originize_file(
  file,
  pkgs = getOption("origin.pkgs", .packages()),
  overwrite = getOption("origin.overwrite", TRUE),
 ask_before_applying_changes = getOption("origin.ask_before_applying_changes", TRUE),
  check_conflicts = getOption("origin.check_conflicts", TRUE),
  check_base_conflicts = getOption("origin.check_base_conflicts", TRUE),
  add_base_packages = getOption("origin.add_base_packages", FALSE),
  excluded_functions = getOption("origin.excluded_functions", list()),
  verbose = getOption("origin.verbose", FALSE),
  use_markers = getOption("origin.use_markers_for_logging", TRUE),
  path_to_local_functions = getOption("origin.path_to_local_functions", NULL),
  check_local_conflicts = getOption("origin.check_local_conflicts", TRUE)
)
```

Arguments

`file`	a path to a script
`pkgs`	a character vector with package names. Defaults to the result of .packages but only if the option 'origin.pkgs' is not specified.
`overwrite`	if TRUE the file will be saved and overwritten. Otherwise, only the logging is triggered. Note that, depending on 'ask_before_applying_changes', the user is ask whether the result is as desired.

`ask_before_applying_changes`
: if TRUE, the user has to approve changes made by origin prior to applying them. Note that this mutes all checks, i.e. large number of files, local functions mask exported functions, and the presence and order of function conflicts.

`check_conflicts`
: if TRUE, possible namespace conflicts between functions exported by packages listed in pkgs are checked. See details.

`check_base_conflicts`
: if TRUE; native R functions are also included in checking for conflicts. See details.

`add_base_packages`
: a boolean. If TRUE, base R functions are handled like all other packages and added via '::'

`excluded_functions`
: a list. Either an unnamed list of function names as strings. These functions are excluded from all packages and never considered in origin. Or a named list with character vectors, Then the name of the list element refers to a package and the given functions are only excluded from this package. A very explicit way to handle namespace conflicts or highlighting popular infix functions like '%>%' or ':='.

`verbose`
: if TRUE, origin provides a logging output about its results.

`use_markers`
: a boolean. If TRUE, the markers tab inn RStudio is used to track changes and show issues. FALSE prints the same information in the console.

`path_to_local_functions`
: Path to search for local functions that mask all exported functions from originizing. If NULL, defaults to the current RStudio Project root.

`check_local_conflicts`
: if TRUE, it is checked whether locally defined functions inside of the project mask exported functions packages listed in pkgs. It avoids mistakenly adding pkg:: to a custom local function.

Details

check_conflicts checks whether multiple packages listed in pkgs export functions with the same name, e.g. lag() is both part of the dplyr and data.table namespace. If there are any conflicts actually present in any considered script, these conflicts are shown including how origin would solve them. User input is required to proceed. The order in pkgs determines the precedence, while those listed first have higher precedence than those listed later in the vector. This is consistent with function masking in R.

check_base_conflicts checks whether functions listed in pkgs mask R functions of R core packages (base, utils, stats, methods, graphics, grDevices, datasets). Even tough the user might not include those functions in the pkg::fct logic, potential conflicts require careful evaluation.

Value

No return value, called for side effects

Examples

```
## Not run:
originize_file(file = "originize_me.R",
               pkgs = c("dplyr", "data.table"),
               overwrite = TRUE,
               ask_before_applying_changes = TRUE,
               excluded_functions = list(dplyr = c("%>%", "tibble"),
                                         data.table = c(":=", "%like%"),
                                         # generally exclude
                                         c("last", "first")),
               verbose = TRUE)

## End(Not run)
```

---

originize_pkg *Originize a Package Project*

Description

It shares the functionality of originize_dir but is designed to be used within R-package projects.

Usage

```
originize_pkg(
  path = rstudioapi::getActiveProject(),
  pkgs = getOption("origin.pkgs", get_pkgs_from_description()),
  recursive = TRUE,
  exclude_files = NULL,
  overwrite = getOption("origin.overwrite", TRUE),
 ask_before_applying_changes = getOption("origin.ask_before_applying_changes", TRUE),
  check_conflicts = getOption("origin.check_conflicts", TRUE),
  check_base_conflicts = getOption("origin.check_base_conflicts", TRUE),
  add_base_packages = getOption("origin.add_base_packages", FALSE),
  excluded_functions = getOption("origin.excluded_functions", list()),
  verbose = getOption("origin.verbose", FALSE),
  use_markers = getOption("origin.use_markers_for_logging", TRUE),
  path_to_local_functions = getOption("origin.path_to_local_functions", NULL),
  check_local_conflicts = getOption("origin.check_local_conflicts", TRUE)
)
```

Arguments

path	path to the package project root by getActiveProject
pkgs	a character vector of package names, defaults to packages mentioned in the DESCRIPTION file if the option 'origin.pkgs' is not set.
recursive	logical. Should scripts be originized recursively, this means that all files in the subfolders will be searched as well. See list.files

`exclude_files` a character vector of file paths that should be excluded from being originized. Helpful if all but a few files should be considered by origin.

`overwrite` if TRUE the file will be saved and overwritten. Otherwise, only the logging is triggered. Note that, depending on 'ask_before_applying_changes', the user is ask whether the result is as desired.

`ask_before_applying_changes`
if TRUE, the user has to approve changes made by origin prior to applying them. Note that this mutes all checks, i.e. large number of files, local functions mask exported functions, and the presence and order of function conflicts.

`check_conflicts`
if TRUE, possible namespace conflicts between functions exported by packages listed in pkgs are checked. See details.

`check_base_conflicts`
if TRUE; native R functions are also included in checking for conflicts. See details.

`add_base_packages`
a boolean. If TRUE, base R functions are handled like all other packages and added via '::'

`excluded_functions`
a list. Either an unnamed list of function names as strings. These functions are excluded from all packages and never considered in origin. Or a named list with character vectors, Then the name of the list element refers to a package and the given functions are only excluded from this package. A very explicit way to handle namespace conflicts or highlighting popular infix functions like '%>%' or ':='.

`verbose` if TRUE, origin provides a logging output about its results.

`use_markers` a boolean. If TRUE, the markers tab inn RStudio is used to track changes and show issues. FALSE prints the same information in the console.

`path_to_local_functions`
Path to search for local functions that mask all exported functions from originizing. If NULL, defaults to the current RStudio Project root.

`check_local_conflicts`
if TRUE, it is checked whether locally defined functions inside of the project mask exported functions packages listed in pkgs. It avoids mistakenly adding pkg:: to a custom local function.

Details

check_conflicts checks whether multiple packages listed in pkgs export functions with the same name, e.g. lag() is both part of the dplyr and data.table namespace. If there are any conflicts actually present in any considered script, these conflicts are shown including how origin would solve them. User input is required to proceed. The order in pkgs determines the precedence, while those listed first have higher precedence than those listed later in the vector. This is consistent with function masking in R.

check_base_conflicts checks whether functions listed in pkgs mask R functions of R core packages (base, utils, stats, methods, graphics, grDevices, datasets). Even tough the user might not include those functions in the pkg::fct logic, potential conflicts require careful evaluation.

Value

No return value, called for side effects

Examples

```
## Not run:
originize_pkg(path = rstudioapi::getActiveProject(),
              overwrite = TRUE,
              ask_before_applying_changes = TRUE,
              exclude_files = c("dont_originize_this.R",
                                "dont_originize_that.R"),
              verbose = TRUE)

## End(Not run)
```

`originize_selection` *Wrapper function to be used as an RStudio addin*

Description

Wrapper function to be used as an RStudio addin

Usage

```
originize_selection(
  context = rstudioapi::getSourceEditorContext(),
  pkgs = getOption("origin.pkgs", .packages()),
  overwrite = getOption("origin.overwrite"),
  ask_before_applying_changes = getOption("origin.ask_before_applying_changes"),
  check_conflicts = getOption("origin.check_conflicts"),
  check_base_conflicts = getOption("origin.check_base_conflicts"),
  add_base_packages = getOption("origin.add_base_packages"),
  excluded_functions = getOption("origin.excluded_functions"),
  verbose = getOption("origin.verbose"),
  use_markers = getOption("origin.use_markers_for_logging"),
  path_to_local_functions = getOption("origin.path_to_local_functions"),
  check_local_conflicts = getOption("origin.check_local_conflicts")
)
```

Arguments

`context`	information of marked editor section in RStudio
`pkgs`	a character vector with package names. Defaults to the result of .packages but only if the option 'origin.pkgs' is not specified.
`overwrite`	if TRUE the file will be saved and overwritten. Otherwise, only the logging is triggered. Note that, depending on 'ask_before_applying_changes', the user is ask whether the result is as desired.

ask_before_applying_changes
: if TRUE, the user has to approve changes made by origin prior to applying them. Note that this mutes all checks, i.e. large number of files, local functions mask exported functions, and the presence and order of function conflicts.

check_conflicts
: if TRUE, possible namespace conflicts between functions exported by packages listed in pkgs are checked. See details.

check_base_conflicts
: if TRUE; native R functions are also included in checking for conflicts. See details.

add_base_packages
: a boolean. If TRUE, base R functions are handled like all other packages and added via '::'

excluded_functions
: a list. Either an unnamed list of function names as strings. These functions are excluded from all packages and never considered in origin. Or a named list with character vectors, Then the name of the list element refers to a package and the given functions are only excluded from this package. A very explicit way to handle namespace conflicts or highlighting popular infix functions like '%>%' or ':='.

verbose
: if TRUE, origin provides a logging output about its results.

use_markers
: a boolean. If TRUE, the markers tab inn RStudio is used to track changes and show issues. FALSE prints the same information in the console.

path_to_local_functions
: Path to search for local functions that mask all exported functions from originizing. If NULL, defaults to the current RStudio Project root.

check_local_conflicts
: if TRUE, it is checked whether locally defined functions inside of the project mask exported functions packages listed in pkgs. It avoids mistakenly adding pkg:: to a custom local function.

Value

No return value, called for side effects

---

print.pkg_usage *Print the summary of check_pkg_usage*

Description

Print the summary of check_pkg_usage

Usage

```
## S3 method for class 'pkg_usage'
print(x, max_display = 10L, ...)
```

Arguments

x	a pkg_usage_object
max_display	maximum number of unknown functions or conflicts to print
...	passed to other methods

Value

x invisibly

Examples

```
## Not run:
result <- check_pkg_usage()
print(result)

## End(Not run)
```

Index

.packages, *2*, *5*, *7*, *11*

check_pkg_usage, 2

get_exported_functions, 3
get_local_functions, 4
get_pkgs_from_description, 4
getActiveProject, *9*
getwd, *2*

list.files, *5*, *9*

originize_dir, 5
originize_file, 7
originize_pkg, 9
originize_selection, 11

print.pkg_usage, 12