Package: apicheck (via r-universe)

August 9, 2024

Title Explore the Historical API of R Packages

Version 0.1.0.9000

Description Check when functions were introduced and/or APIs changed in packages, using the 'MRAN' service.

License MIT + file LICENSE

Depends R (>= 3.3.0)

Imports desc, dplyr, glue, httr, jsonlite, memoise, parallel, purrr, rcheology, remotes, tibble, utils, withr, zeallot

Suggests clipr, covr, knitr, pkgapi, rlang, rmarkdown, testthat (>= 2.0.0), versions

VignetteBuilder knitr

Encoding UTF-8

LazyData true

Remotes hughjonesd/rcheology, r-lib/pkgapi

Roxygen list(markdown = TRUE)

RoxygenNote 6.0.1

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

RemoteUrl https://github.com/hughjonesd/apicheck

RemoteRef HEAD

RemoteSha 88128d380ebe08af980a3b4e08e99df2159dfb03

Contents

apicheck-package 2
api_same_at 2
available_versions 3
cached_install 4
call_with_namespace 5
compare_versions 6
fun_at 7

fun_exists_at 8
help_at 9
package_report 10
set_lib_dir 11
version_at_date 12
when_api_same 13

Index **15**

`apicheck-package` *apicheck: check function APIs in different versions of packages*

Description

This is a small package to check when functions were introduced and/or APIs changed in packages. It automatically installs different versions of a package in a separate directory and loads them without attaching them.

Details

Packages are cached within a session. To cache packages across sessions, use `set_lib_dir()` to point to a persistent directory.

By default, `apicheck`` uses theremotespackage to install source versions from CRAN. Alternatively, it can use theversions“ package to install different versions of a package from https://mran.microsoft.com/. To do this set `options(apicheck.use_mran = TRUE)`.

Be aware that functions can take a long time to return, as different versions of a package are installed and/or loaded.

Also, be aware that namespace loading and unloading can be unreliable. If this happens to you, try restarting your session.

Warning

Do not try to use `apicheck` on itself. This will lead to fiery elephants in the sky.

`api_same_at` *Test if a function's API is unchanged*

Description

`api_same_at` reports whether a function had the same API at a previous version or date.

Usage

```
api_same_at(fun, version = version_at_date(package, date), package,
  date = NULL, quiet = TRUE, current_fun = NULL, ...)
```

Arguments

fun	Function name as a character string. fun can be an S3 method; S4 methods aren't yet supported.
version	Version as a character string. If omitted, use the version available at date.
package	Package. Alternatively, specify the function name as e.g. "package::function".
date	Date, as a character string that can be read by as.Date() e.g. "2016-01-01".
quiet	Logical. Try to minimize output from package installation. (Some output comes from R CMD INSTALL and may be unavoidable.)
current_fun	Current function for comparison. By default, fun in the current version of the package (which is assumed to be already installed). This must be an actual function, not the name of one: use fun_at().
...	Arguments passed to versions::install.versions() or remotes::install_version(), and thence to install.packages(). Ncpus may be useful.

Details

If fun is not exported at version, api_same_at returns FALSE with a warning.

Value

TRUE or FALSE.

Examples

```
## Not run:
api_same_at("clipr::write_clip", version = "0.1.1")
# equivalently
api_same_at("write_clip", "clipr", "0.1.1")

## End(Not run)
```

---

available_versions *Report available versions*

Description

This returns packages ordered by date, using either MRAN or metacran (or rcheology for R core packages). Results are cached so as to relieve pressure on the server. If options("apicheck.use_mran") is TRUE (and package is non-core), then only versions available on MRAN (i.e. after 2014-09-17) will be returned; otherwise older versions will be returned too.

Usage

```
available_versions(package)
```

Arguments

package	Package name.

Value

A data frame with columns "version" and "date".

Speed

In my limited experience, metacran is much faster. YMMV.

Examples

```
## Not run:
available_versions("clipr")

## End(Not run)
```

---

cached_install *Install and/or load a version of a package*

Description

cached_install checks the package cache, installs the specified version if it is not already installed, and loads the versioned package namespace.

Usage

```
cached_install(package, version, return = c("namespace", "path"),
  cache = TRUE, quiet = TRUE, partial = TRUE, ...)
```

Arguments

package	Package name.
version	Version as a character string.
return	Return the file "path" to the installed package, or the "namespace" object?
cache	If FALSE, always reinstall the package.
quiet	Logical. Try to minimize output from package installation. (Some output comes from R CMD INSTALL and may be unavoidable.)
partial	Default TRUE. Passed to loadNamespace().
...	Arguments passed to versions::install.versions() or remotes::install_version(), and thence to install.packages(). Ncpus may be useful.

Details

If the package is already loaded, `cached_install` will first attempt to unload it with a warning. This may not always work!

Note that the namespace is not attached. Partial loading is faster and safer when you are (un)loading multiple versions, but does not export functions etc.

Value

The namespace object or directory where the package is installed.

Examples

```
## Not run:
cached_install("clipr", "0.4.0")

## End(Not run)
```

`call_with_namespace` *Load a package namespace and pass it to a function*

Description

The package is downloaded and installed if necessary, and its namespace is loaded. Then `test(ns)` is called with the namespace object, and its value is returned. On exit, the namespace is unloaded.

Usage

```
call_with_namespace(package, version, test, quiet = TRUE, ...)
```

Arguments

`package`	Package name.
`version`	Version as a character string.
`test`	A one-argument function.
`quiet`	Logical. Try to minimize output from package installation. (Some output comes from `R CMD INSTALL` and may be unavoidable.)
`...`	Arguments passed to `versions::install.versions()` or `remotes::install_version()`, and thence to `install.packages()`. `Ncpus` may be useful.

Value

The value returned by `test`.

Examples

```
## Not run:
can_it_expand_urls <- function (namespace) "expand_urls" %in% names(namespace)
call_with_namespace("longurl", "0.3.0", test = can_it_expand_urls)

## End(Not run)
```

`compare_versions` *Compare versions of a package*

Description

`compare_versions` reports how functions and APIs changed between versions of a package.

Usage

```
compare_versions(package, version = previous_version(package, version2),
  version2, methods = FALSE, quiet = TRUE, ...)

## S3 method for class 'versions_report'
print(x, ...)

## S3 method for class 'versions_report'
summary(object, ...)
```

Arguments

`package`	Package name.
`version`	First version to compare. By default, the previous version to that currently installed.
`version2`	Second version to compare. By default, the current installed version.
`methods`	Logical: include non-exported S3 methods?
`quiet`	Logical. Try to minimize output from package installation. (Some output comes from `R CMD INSTALL` and may be unavoidable.)
`...`	Arguments passed to `versions::install.versions()` or `remotes::install_version()`, and thence to `install.packages()`. `Ncpus` may be useful.
`x, object`	An object of class `versions_report`.

Value

`compare_versions` returns a data frame of class `versions_report`, reporting functions that have been "Added", "Removed" or had "API changed", and details of function arguments. Extra information is in the `"package"` and `"versions"` attributes.

`summary` returns a string representation of changed function arguments.

Examples

```
## Not run:
compare_versions("clipr", "0.2.1", "0.3.0")

## End(Not run)
```

`fun_at` *Retrieve a function from a package version*

Description

Retrieve a function from a package version

Usage

```
fun_at(fun, version = version_at_date(package, date), package, date = NULL,
  quiet = TRUE, allow_core = FALSE, ...)
```

Arguments

`fun`	Function name as a character string. `fun` can be an S3 method; S4 methods aren't yet supported.
`version`	Version as a character string. If omitted, use the version available at `date`.
`package`	Package. Alternatively, specify the function name as e.g. `"package::function"`.
`date`	Date, as a character string that can be read by `as.Date()` e.g. "2016-01-01".
`quiet`	Logical. Try to minimize output from package installation. (Some output comes from `R CMD INSTALL` and may be unavoidable.)
`allow_core`	See below.
`...`	Arguments passed to `versions::install.versions()` or `remotes::install_version()`, and thence to `install.packages()`. `Ncpus` may be useful.

Details

By default, you cannot get functions from previous versions of R core packages, which are not available on CRAN/MRAN. If `allow_core` is `TRUE`, then a function will be returned from the rcheology dataset, with a null body but the same formal arguments as the historical function.

Value

The function itself.

Examples

```
## Not run:
fun_at("write_clip", "clipr", "0.1.1")

## End(Not run)
```

fun_exists_at *Test if a function exists at a given package version*

Description

fun_exists_at reports whether a function exists (i.e. is exported) from a package at a specific previous version or date.

Usage

```
fun_exists_at(fun, version = version_at_date(package, date), package,
  date = NULL, quiet = TRUE, ...)
```

Arguments

fun	Function name as a character string. fun can be an S3 method; S4 methods aren't yet supported.
version	Version as a character string. If omitted, use the version available at date.
package	Package. Alternatively, specify the function name as e.g. "package::function".
date	Date, as a character string that can be read by as.Date() e.g. "2016-01-01".
quiet	Logical. Try to minimize output from package installation. (Some output comes from R CMD INSTALL and may be unavoidable.)
...	Arguments passed to versions::install.versions() or remotes::install_version(), and thence to install.packages(). Ncpus may be useful.

Value

TRUE or FALSE.

Examples

```
## Not run:
fun_exists_at("clipr::dr_clipr", version = "0.3.1")
# or
fun_exists_at("dr_clipr", "clipr", "0.3.1")

## End(Not run)
```

help_at *Get help for a function at a package version*

Description

Get help for a function at a package version

Usage

```
help_at(fun, version = version_at_date(package, date), package, date = NULL,
  quiet = TRUE, ...)
```

Arguments

fun	Function name as a character string. fun can be an S3 method; S4 methods aren't yet supported.
version	Version as a character string. If omitted, use the version available at date.
package	Package. Alternatively, specify the function name as e.g. "package::function".
date	Date, as a character string that can be read by as.Date() e.g. "2016-01-01".
quiet	Logical. Try to minimize output from package installation. (Some output comes from R CMD INSTALL and may be unavoidable.)
...	Arguments passed to versions::install.versions() or remotes::install_version(), and thence to install.packages(). Ncpus may be useful.

Value

The help object (text format only).

See Also

help()

Examples

```
## Not run:
help_at("clipr::write_clip", "0.1.1")
help_at("clipr::write_clip", "0.2.0")

## End(Not run)
```

package_report *Report on backwards compatibility of a source package*

Description

package_report lists all external function calls from a source package using pkgapi::map_package(). It then checks backward-compatibility of each call with previous versions.

Usage

```
package_report(path = ".", include, exclude = core_packages(),
  cutoff_date = Sys.Date() - 2 * 365, max_n_versions = NULL,
  parallel = FALSE, progress = !parallel, ...)
```

Arguments

path	Path to the root of an R source package.
include	Packages to include. By default, all are included except core packages and the package at path itself.
exclude	Packages to exclude from checking. Overrides include.
cutoff_date	Don't check versions before this date (default: 2 years ago). Set to NULL for no cutoff.
max_n_versions	Check at most this number of previous versions. Set to NULL for no limit.
parallel	Run in parallel.
progress	Show a progress bar.
...	Arguments passed to call_with_namespace().

Value

A data frame with three rows:

- package for the external package
- version for the latest version which caused problems or NA if there were no problems
- funs a list-column of function names where the API changed

Parallelism

For parallel search, you can set up your own parallel cluster by using parallel::setDefaultCluster(); otherwise one will be created, using getOption("cl.cores") cores if that is set. If you set up your own cluster, it will not be stopped automatically (see parallel::stopCluster()).

Examples

```
## Not run:
package_report(".")
# to include base packages also:
package_report(".", exclude = character(0))

## End(Not run)
```

---

set_lib_dir *Location of the package cache*

Description

set_lib_dir() specifies where packages will be downloaded to. get_lib_dir() returns this directory. clear_lib_dir() deletes all downloaded packages.

Usage

```
set_lib_dir(lib_dir, create = FALSE)

get_lib_dir()

clear_lib_dir()
```

Arguments

lib_dir	Path to a directory, or NULL to unset.
create	Logical. Try to create the directory if it doesn't exist.

Details

If lib_dir is set to NULL, a subdirectory of tempdir() will be used. lib_dir will be normalized via normalizePath().

The package cache is under the directory specified by getOption("apicheck.lib_dir"), or, if that is unset, in a per-session temporary directory. You should use set_lib_dir() to change this rather than setting the option directly. Within this directory, subdirectories are named like package-version, e.g. longurl-0.3.0. Within these subdirectories are the actual installed libraries. So, lib_dir is not appropriate for passing to functions like library. To load a library from the cache yourself, do e.g. library("blah", lib.loc = file.path(get_lib_dir(), "blah-0.1.0")).

Value

set_lib_dir invisibly returns the old library location, or NULL if none was set in options.

get_lib_dir returns the actual library location, whether or not an option has been set.

clear_lib_dir invisibly returns TRUE if all files and directories could be removed, FALSE otherwise.

Examples

```
## Not run:
set_lib_dir("~/.apicheck")

## End(Not run)
get_lib_dir()
## Not run:
clear_lib_dir()

## End(Not run)
```

`version_at_date` *Return the current version of a package at a given date*

Description

Return the current version of a package at a given date

Usage

```
version_at_date(package, date)
```

Arguments

package	Package. Alternatively, specify the function name as e.g. "package::function".
date	Date, as a character string that can be read by as.Date() e.g. "2016-01-01".

Value

A version string.

Examples

```
## Not run:
version_at_date("huxtable", "2017-01-01")

## End(Not run)
```

when_api_same *Compare APIs across package versions*

Description

when_api_same reports package versions where the API of a function was the same as now (or the same as current_fun).

when_fun_exists reports package versions where a function exists.

Usage

```
when_api_same(fun, package, current_fun = NULL, search = c("binary",
  "forward", "backward", "all", "parallel"), report = c("full", "brief"),
  quiet = TRUE, progress = interactive() && search != "parallel",
  min_version = NULL, max_version = NULL, ...)

when_fun_exists(fun, package, search = c("binary", "forward", "backward",
  "all", "parallel"), report = c("full", "brief"), quiet = TRUE,
  progress = interactive() && search != "parallel", min_version = NULL,
  max_version = NULL, ...)
```

Arguments

fun	Function name as a character string. fun can be an S3 method; S4 methods aren't yet supported.
package	Package. Alternatively, specify the function name as e.g. "package::function".
current_fun	Current function for comparison. By default, fun in the current version of the package (which is assumed to be already installed). This must be an actual function, not the name of one: use fun_at().
search	"binary", "forward", "backward", "all" or "parallel". See Search strategies.
report	"brief" or "full". See Value.
quiet	Logical. Try to minimize output from package installation. (Some output comes from R CMD INSTALL and may be unavoidable.)
progress	Print a progress bar.
min_version	Lowest version to check.
max_version	Highest version to check.
...	Arguments passed to versions::install.versions() or remotes::install_version(), and thence to install.packages(). Ncpus may be useful.

Details

"Same API" means having the same function arguments, as reported by formals().

Value

If `report` is "brief", the earliest "known good" version. Otherwise, a data frame of versions with a full results column.

Speed

This function may download and install multiple versions from MRAN, so it is likely to be slow when first used (and even afterwards if library loading is slow). Using `search = "parallel"` may help, but not if the network is the bottleneck: see https://hughjonesd.github.io/apicheck/performance2.html for details.

Search strategies

- `"forward"` (`"backward"`) searches incrementally from the earliest (latest) version.
- `"binary"` does a binary search from the midpoint.

These strategies assume that API changes happen just once - i.e. once a function exists or API is the same as now, it will stay so in future versions. This allows them to stop before searching every version.

- `"all"` searches every version.
- `"parallel"` searches every version in parallel using `parallel::parLapply()`.

Parallelism

For parallel search, you can set up your own parallel cluster by using `parallel::setDefaultCluster()`; otherwise one will be created, using `getOption("cl.cores")` cores if that is set. If you set up your own cluster, it will not be stopped automatically (see `parallel::stopCluster()`).

Examples

```
## Not run:
when_api_same("read.dta", "foreign")

## End(Not run)
## Not run:
when_fun_exists('read.dta', 'foreign')

## End(Not run)
```

Index

api_same_at, 2
apicheck-package, 2
as.Date(), *3*, *7–9*, *12*
available_versions, 3

cached_install, 4
call_with_namespace, 5
call_with_namespace(), *10*
clear_lib_dir *(*set_lib_dir*)*, 11
compare_versions, 6

formals(), *13*
fun_at, 7
fun_at(), *3*, *13*
fun_exists_at, 8

get_lib_dir *(*set_lib_dir*)*, 11

help(), *9*
help_at, 9

install.packages(), *3–9*, *13*

loadNamespace(), *4*

normalizePath(), *11*

package_report, 10
parallel::parLapply(), *14*
parallel::setDefaultCluster(), *10*, *14*
parallel::stopCluster(), *10*, *14*
pkgapi::map_package(), *10*
print.versions_report
*(*compare_versions*)*, 6

remotes::install_version(), *3–9*, *13*

set_lib_dir, 11
set_lib_dir(), *2*, *11*
summary.versions_report
*(*compare_versions*)*, 6

version_at_date, 12
versions::install.versions(), *3–9*, *13*

when_api_same, 13
when_fun_exists *(*when_api_same*)*, 13