Package: checklist (via r-universe)

August 18, 2024

Type Package

Title A Thorough and Strict Set of Checks for R Packages and Source Code

Version 0.4.0

Description An opinionated set of rules for R packages and R source code projects.

License GPL-3

URL https://inbo.github.io/checklist/, https://github.com/inbo/checklist, https://doi.org/10.5281/zenodo.4028303

BugReports https://github.com/inbo/checklist/issues

Depends R (>= 4.1.0)

Imports assertthat, cli, codemetar, desc, devtools (> 2.4.0), fs, gert, httr, hunspell, jsonlite, knitr, lintr (>= 3.0.2.9000), pkgdown (>= 2.0.7), R6, rcmdcheck, renv, rmarkdown, sessioninfo, utils, withr, yaml

Suggests bookdown, covr, curl, mockery, roxygen2, rstudioapi, showtext, sysfonts, testthat, zen4R (>= 0.10)

VignetteBuilder knitr

Config/checklist/communities inbo

Config/checklist/keywords quality control; documentation; publication

Encoding UTF-8

Language en-GB

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

SystemRequirements Pandoc (>= 1.17.2)

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

RemoteUrl https://github.com/inbo/checklist

RemoteRef HEAD

RemoteSha 5649985b58693acb88337873ae14a7d5bc018d96

Contents

`add_badges` *add badges to a README*

Description

- `doi`: add a DOI badge
- `url`: add a website badge

Usage

```
add_badges(x = ".", ...)
```

Arguments

`x`	Either a `checklist` object or a path to the source code. Defaults to `.`.
`...`	Additional arguments

See Also

Other both: `check_filename()`, `check_lintr()`, `check_spelling()`, `custom_dictionary()`, `default_organisation()`, `print.checklist_spelling()`, `read_checklist()`, `read_organisation()`, `write_checklist()`, `write_organisation()`

Examples

```
## Not run:
  add_badges(url = "https://www.inbo.be")
  add_badges(doi = "10.5281/zenodo.8063503")
  add_badges(check_project = "inbo/checklist")
  add_badges(check_package = "inbo/checklist")
  add_badges(url = "https://www.inbo.be", doi = "10.5281/zenodo.8063503")

## End(Not run)
```

ask_yes_no *Function to ask a simple yes no question*

Description

Function to ask a simple yes no question

Usage

```
ask_yes_no(msg, default = TRUE, prompts = c("Yes", "No", "Cancel"), ...)
```

Arguments

msg	The prompt message for the user.
default	The default response.
prompts	Any of: a character vector containing 3 prompts corresponding to return values of TRUE, FALSE, or NA, or a single character value containing the prompts separated by / characters, or a function to call.
...	Additional parameters, ignored by the default function.

See Also

Other utils: bookdown_zenodo(), c_sort(), clean_git(), create_hexsticker(), execshell(), is_repository(), is_workdir_clean(), menu_first(), new_branch(), orcid2person(), store_authors(), use_author(), validate_email(), validate_orcid(), yesno()

bookdown_zenodo *Render a* bookdown *and upload to Zenodo*

Description

First clears all the existing files in the output_dir set in _bookdown_.yml. Then renders all required output formats and uploads them to Zenodo. bookdown_zenodo() creates a draft record when you don't specify a community in the yaml. Otherwise it creates a review request for the first community.

Usage

```
bookdown_zenodo(
  path,
  zip_format = c("bookdown::gitbook", "INBOmd::gitbook"),
 single_format = c("bookdown::pdf_book", "bookdown::epub_book", "INBOmd::pdf_report",
    "INBOmd::epub_book"),
  token,
  sandbox = TRUE,
  logger = "INFO"
)
```

Arguments

path	The root folder of the report
zip_format	A vector with output formats that generate multiple files. The function will bundle all the files in a zip file for every format.
single_format	A vector with output formats that generate a single output file. The output will remain as is.
token	the user token for Zenodo. By default an attempt will be made to retrieve token using zenodo_pat().
sandbox	When TRUE, upload a test version to https://sandbox.zenodo.org. When FALSE, upload the final version to https://zenodo.org.
logger	Type of logger for Zenodo upload. Defaults to "INFO" which provides minimal logs. Use NULL to hide the logs. "DEBUG" provides the full log.

See Also

Other utils: ask_yes_no(), c_sort(), clean_git(), create_hexsticker(), execshell(), is_repository(), is_workdir_clean(), menu_first(), new_branch(), orcid2person(), store_authors(), use_author(), validate_email(), validate_orcid(), yesno()

---

checklist *The checklist R6 class*

Description

A class which contains all checklist results.

Super class

checklist::spelling -> checklist

Public fields

package A logical indicating whether the source code refers to a package.

Active bindings

get_checked A vector with checked topics.

get_path The path to the package.

get_required A vector with the names of the required checks.

get_spelling Return the issues found by check_spelling()

fail A logical indicating if any required check fails.

template A list for a check list template.

Methods

Public methods:

- checklist$add_error()
- checklist$add_linter()
- checklist$add_motivation()
- checklist$add_notes()
- checklist$add_rcmdcheck()
- checklist$add_spelling()
- checklist$add_warnings()
- checklist$allowed()
- checklist$confirm_motivation()
- checklist$new()
- checklist$print()
- checklist$set_required()
- checklist$clone()

Method `add_error()`**:** Add errors

Usage:

```
checklist$add_error(errors, item, keep = TRUE)
```

Arguments:

`errors` A vector with errors.

`item` The item on which to store the errors.

`keep` Keep old results

Method `add_linter()`**:** Add results from `lintr::lint_package()`

Usage:

```
checklist$add_linter(linter)
```

Arguments:

`linter` A vector with linter errors.

Method `add_motivation()`**:** Add motivation for allowed issues.

Usage:

```
checklist$add_motivation(which = c("warnings", "notes"))
```

Arguments:

`which` Which kind of issue to add.

Method `add_notes()`**:** Add notes

Usage:

```
checklist$add_notes(notes, item)
```

Arguments:

`notes` A vector with notes.

`item` The item on which to store the notes.

Method `add_rcmdcheck()`: Add results from `rcmdcheck::rcmdcheck`

Usage:

```
checklist$add_rcmdcheck(errors, warnings, notes)
```

Arguments:

`errors` A vector with errors.

`warnings` A vector with warning messages.

`notes` A vector with notes.

Method `add_spelling()`: Add results from `check_spelling()`

Usage:

```
checklist$add_spelling(issues)
```

Arguments:

`issues` A `data.frame` with spell checking issues.

Method `add_warnings()`: Add warnings

Usage:

```
checklist$add_warnings(warnings, item)
```

Arguments:

`warnings` A vector with warnings.

`item` The item on which to store the warnings.

Method `allowed()`: Add allowed warnings and notes

Usage:

```
checklist$allowed(
  warnings = vector(mode = "list", length = 0),
  notes = vector(mode = "list", length = 0)
)
```

Arguments:

`warnings` A vector with allowed warning messages. Defaults to an empty list.

`notes` A vector with allowed notes. Defaults to an empty list.

`package` Does the check list refers to a package. Defaults to `TRUE`.

Method `confirm_motivation()`: Confirm the current motivation for allowed issues.

Usage:

```
checklist$confirm_motivation(which = c("warnings", "notes"))
```

Arguments:

`which` Which kind of issue to confirm.

Method `new()`: Initialize a new `checklist` object.

Usage:

```
checklist$new(x = ".", language, package = TRUE)
```

Arguments:

x The path to the root of the project.

language The default language for spell checking.

package Is this a package or a project?

Method `print()`**:** Print the `checklist` object. Add `quiet = TRUE` to suppress printing.

Usage:

```
checklist$print(...)
```

Arguments:

`...` See description.

Method `set_required()`**:** set required checks

Usage:

```
checklist$set_required(checks = character(0))
```

Arguments:

`checks` a vector of required checks

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
checklist$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other class: `citation_meta`, `organisation`, `spelling`

`check_codemeta` *Check the package metadata*

Description

Use the checks from `codemetar::give_opinions()`.

Usage

```
check_codemeta(x = ".")
```

Arguments

x	Either a `checklist` object or a path to the source code. Defaults to `.`

Value

A `checklist` object.

See Also

Other package: check_cran(), check_description(), check_documentation(), check_environment(), check_license(), check_package(), set_tag(), tidy_desc(), update_citation(), write_citation_cff(), write_zenodo_json()

---

check_cran *Run all the package checks required by CRAN*

Description

CRAN imposes an impressive list of tests on every package before publication. This suite of test is available in every R installation. Hence we use this full suite of tests too. Notice that check_package() runs several additional tests.

Usage

```
check_cran(x = ".", quiet = FALSE)
```

Arguments

x	Either a checklist object or a path to the source code. Defaults to . .
quiet	Whether to print check output during checking.

Value

A checklist object.

See Also

Other package: check_codemeta(), check_description(), check_documentation(), check_environment(), check_license(), check_package(), set_tag(), tidy_desc(), update_citation(), write_citation_cff(), write_zenodo_json()

---

check_description *Check the* DESCRIPTION *file*

Description

The DESCRIPTION file contains the most important meta-data of the package. A good DESCRIPTION is tidy, has a meaningful version number, full author details and a clear license.

Usage

```
check_description(x = ".")
```

Arguments

x	Either a checklist object or a path to the source code. Defaults to . .

Details

This function ensures the DESCRIPTION is tidy, using tidy_desc().

The version number of the package must have either a 0.0 or a 0.0.0 format (see this discussion why we allow only these formats). The version number in every branch must be larger than the current version number in the main or master branch. New commits in the main or master must have a larger version number than the previous commit. We recommend to protect the main or master branch and to not commit into the main or master.

Furthermore we check the author information.

- Is INBO listed as copyright holder and funder?
- Has every author an ORCID?

We check the license through check_license().

See Also

Other package: check_codemeta(), check_cran(), check_documentation(), check_environment(), check_license(), check_package(), set_tag(), tidy_desc(), update_citation(), write_citation_cff(), write_zenodo_json()

---

check_documentation *Check the documentation*

Description

The function make sure that the documentation is up to date. Rules:

- You must use roxygen2 to document the functions.
- If you use a README.Rmd, it should be rendered. You need at least a README.md.
- Don't use a NEWS.Rmd but a NEWS.md.
- NEWS.md must contain an entry for the current package version.

Usage

```
check_documentation(x = ".", quiet = FALSE)
```

Arguments

x	Either a checklist object or a path to the source code. Defaults to . .
quiet	Whether to print check output during checking.

Details

The function generates the help files from the roxygen2 tag in the R code. Then it checks whether any of the help files changed. We use the same principle with the README.Rmd. If any file changed, the documentation does not match the code. Hence check_documentation() returns an error.

A side effect of running check_documentation() locally, is that it generates all the documentation. So the only thing left for you to do, is to commit these changes. Pro tip: make sure RStudio renders the roxygen2 tags whenever you install and restart the package. We describe this in vignette("getting_started") under "Prepare local setup".

Required format for NEWS.md

```
# package_name version

* Description of something that changed.
* Lines should not exceed 80 characters.
  Start a new line with two space to continue an item.
* Add a single blank line before and after each header.

## Second level heading

* You can use second level headings when you want to add more structure.

 # `package_name` version

 * Adding back ticks around the package name is allowed.
```

See Also

Other package: check_codemeta(), check_cran(), check_description(), check_environment(), check_license(), check_package(), set_tag(), tidy_desc(), update_citation(), write_citation_cff(), write_zenodo_json()

---

check_environment *Make sure that the required environment variables are set on GitHub*

Description

Some actions will fail when these environment variables are not set. This function does only work on GitHub.

Usage

```
check_environment(x = ".")
```

Arguments

x	Either a checklist object or a path to the source code. Defaults to . .

Value

An invisible checklist object.

See Also

Other package: check_codemeta(), check_cran(), check_description(), check_documentation(), check_license(), check_package(), set_tag(), tidy_desc(), update_citation(), write_citation_cff(), write_zenodo_json()

---

check_filename *Check the style of file and folder names*

Description

A consistent naming schema avoids problems when working together, especially when working with different OS. Some OS (e.g. Windows) are case-insensitive whereas others (e.g. Linux) are case-sensitive. Note that the checklist GitHub Actions will test your code on Linux, Windows and MacOS.

Usage

```
check_filename(x = ".")
```

Arguments

x	Either a checklist object or a path to the source code. Defaults to . .

Details

The sections below describe the default rules. We allow several exceptions when the community standard is different. E.g. a package stores the function scripts in the R folder, while our standard enforces lower case folder names. Use the community standard, even if it does not conform with the checklist rules. Most likely checklist will have an exception for the name. If not, you can file an issue and motivate why you think we should add an exception.

Rules for folder names

- Folder names should only contain lower case letters, numbers, dashes (-) and underscores (_).
- They can start with a single dot (.).

Default rules for file names

- Base names should only contain lower case letters, numbers, dashes (-) and underscores (_).
- File extensions should only contain lower case letters and numbers. Exceptions: file extensions related to R must have an upper case R (.R, .Rd, .Rda, .Rnw, .Rmd, .Rproj). Exception to these exceptions: R/sysdata.rda.

Exceptions for some file formats

Underscores (_) causes problems for graphical files when using LaTeX to create pdf output. This is how we generate pdf output from rmarkdown. Therefore you need to use a dash (-) as separator instead of an underscores (_). Applies to files with extensions `eps`, `jpg`, `jpeg`, `pdf`, `png`, `ps`, `svg` and `.cls`.

We ignore files with `.otf` or `.ttf` extensions. These are fonts files which often require their own file name scheme.

See Also

Other both: `add_badges()`, `check_lintr()`, `check_spelling()`, `custom_dictionary()`, `default_organisation()`, `print.checklist_spelling()`, `read_checklist()`, `read_organisation()`, `write_checklist()`, `write_organisation()`

`check_folder` *Check the folder structure*

Description

For the time being, this function only checks projects. Keep in mind that R packages have requirements for the folder structure. That is one of things that `check_cran()` checks.

Usage

```
check_folder(x = ".")
```

Arguments

`x`	Either a `checklist` object or a path to the source code. Defaults to `.` .

Recommended folder structure

- `source`: contains all `R` scripts and `Rmd` files.
- `data`: contains all data files.

`source`

A simple project with only `R` scripts or only `Rmd` files can place all the files directly in the `source` folder.

More elaborate projects should place in the files in several folders under `source`. Place every `bookdown` document in a dedicated folder. And create an RStudio project for that folder.

`data`

Simple projects in which `source` has no subfolders, place `data` at the root of the project. For more elaborate project you must choose between either `data` at the root of the project or `data` as subfolder of the subfolders of `source`. E.g. `source/report/data`.

Place the data in an open file format. E.g. `csv`, `txt` or `tsv` for tabular data. We strongly recommend to use `git2rdata::write_vc()` to store such data. Use the geopackage format for spatial data. Optionally add description of the data as markdown files.

See Also

Other project: `check_project()`, `check_source()`

---

check_license *Check the license of a package*

Description

Every package needs a clear license. Without a license, the end-users have no clue under what conditions they can use the package. You must specify the license in the `DESCRIPTION` and provide a `LICENSE.md` file.

Usage

```
check_license(x = ".")
```

Arguments

`x`	Either a `checklist` object or a path to the source code. Defaults to `.`.

Details

This functions checks if the `DESCRIPTION` mentions one of the standard licenses. The `LICENSE.md` must match this license. Use `setup_package()` to add the correct `LICENSE.md` to the package.

Currently, following licenses are allowed:

- GPL-3
- MIT

We will consider pull requests adding support for other open source licenses.

See Also

Other package: `check_codemeta()`, `check_cran()`, `check_description()`, `check_documentation()`, `check_environment()`, `check_package()`, `set_tag()`, `tidy_desc()`, `update_citation()`, `write_citation_cff()`, `write_zenodo_json()`

check_lintr *Check the packages for linters*

Description

This functions does static code analysis. It relies on `lintr::lint_package()`. We recommend that you activate all code diagnostics in RStudio to help meeting the requirements. You can find this in the menu *Tools* > *Global options* > *Code* > *Diagnostics*. Please have a look at `vignette("philosophy")` for more details on the rules.

Usage

```
check_lintr(x = ".", quiet = FALSE)
```

Arguments

`x`	Either a `checklist` object or a path to the source code. Defaults to `.`.
`quiet`	Whether to print check output during checking.

See Also

Other both: `add_badges()`, `check_filename()`, `check_spelling()`, `custom_dictionary()`, `default_organisation()`, `print.checklist_spelling()`, `read_checklist()`, `read_organisation()`, `write_checklist()`, `write_organisation()`

check_package *Run the complete set of standardised tests on a package*

Description

A convenience function that runs all packages related tests in sequence. The details section lists the relevant functions. After fixing a problem, you can quickly check if it is solved by running only the related check. But we still recommend to run `check_package()` before you push to GitHub. And only push when the functions indicate that there are no problems. This catches most problems before sending the code to GitHub.

Usage

```
check_package(
  x = ".",
  fail = !interactive(),
  pkgdown = interactive(),
  quiet = FALSE
)
```

Arguments

x	Either a checklist object or a path to the source code. Defaults to . .
fail	Should the function return an error in case of a problem? Defaults to TRUE on a non-interactive session and FALSE on an interactive session.
pkgdown	Test pkgdown website. Defaults to TRUE on an interactive session and FALSE on a non-interactive session.
quiet	Whether to print check output during checking.

Details

List of checks in order:

1. check_cran()
2. check_lintr()
3. check_filename()
4. check_description()
5. check_documentation()
6. check_codemeta()

See Also

Other package: check_codemeta(), check_cran(), check_description(), check_documentation(), check_environment(), check_license(), set_tag(), tidy_desc(), update_citation(), write_citation_cff(), write_zenodo_json()

---

check_project *Run the required quality checks on a project*

Description

Set or update the required checks via setup_project().

Usage

```
check_project(x = ".", fail = !interactive(), quiet = FALSE)
```

Arguments

x	Either a checklist object or a path to the source code. Defaults to . .
fail	Should the function return an error in case of a problem? Defaults to TRUE on a non-interactive session and FALSE on an interactive session.
quiet	Whether to print check output during checking.

See Also

Other project: check_folder(), check_source()

check_source *Standardised test for an R source repository*

Description

A convenience function that runs test on a project with only `.R` and `.Rmd` files. The details section lists the relevant functions. When you fixed a problem, you can speed things up by running only the related check. We still recommend to run `check_source()` before you push to GitHub. And only push when the functions indicate that there are no problems. This catches most problems before sending the code to GitHub.

Usage

```
check_source(x = ".", fail = !interactive())
```

Arguments

`x`	Either a `checklist` object or a path to the source code. Defaults to `.`.
`fail`	Should the function return an error in case of a problem? Defaults to `TRUE` on non-interactive session and `FALSE` on an interactive session.

Details

List of checks in order:

1. `check_lintr()`
2. `check_filename()`

See Also

Other project: `check_folder()`, `check_project()`

check_spelling *Spell check a package or project*

Description

This function checks by default any markdown (`.md`) or Rmarkdown (`.Rmd`) file found within the project. It also checks any R help file (`.Rd`) in the `man` folder. Use the `set_exceptions()` method of the `checklist` object to exclude files or use a different language. Have a look at `vignette("spelling", package = "checklist")` for more details.

Usage

```
check_spelling(x = ".", quiet = FALSE)
```

Arguments

x	Either a checklist object or a path to the source code. Defaults to . .
quiet	Whether to print check output during checking.

See Also

Other both: add_badges(), check_filename(), check_lintr(), custom_dictionary(), default_organisation(), print.checklist_spelling(), read_checklist(), read_organisation(), write_checklist(), write_organisation()

---

citation_meta *The* citation_meta *R6 class*

Description

A class which contains citation information.

Active bindings

get_errors Return the errors

get_meta Return the meta data as a list

get_notes Return the notes

get_type A string indicating the type of source.

get_path The path to the project.

get_warnings Return the warnings

Methods

Public methods:

- citation_meta$new()
- citation_meta$print()
- citation_meta$clone()

Method new(): Initialize a new citation_meta object.

Usage:

```
citation_meta$new(path = ".")
```

Arguments:

path The path to the root of the project.

Method print(): Print the citation_meta object.

Usage:

```
citation_meta$print(...)
```

Arguments:

... currently ignored.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
citation_meta$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other class: `checklist`, `organisation`, `spelling`

`clean_git` *Clean the git repository*

Description

- update local branches that are behind their counterpart on origin.
- list local branches that have diverged from their counterpart the origin.
- list local branches without counterpart on origin that have diverged from the main branch.
- remove local branches without counterpart on origin and fully merged into the main branch.
- remove local copies of origin branches deleted at the origin.

Usage

```
clean_git(repo = ".", verbose = TRUE)
```

Arguments

`repo`	The path to the git repository. If the directory is not a repository, parent directories are considered (see git_find). To disable this search, provide the filepath protected with `I()`. When using this parameter, always explicitly call by name (i.e. `repo = `) because future versions of gert may have additional parameters.
`verbose`	display some progress info while downloading

See Also

Other utils: `ask_yes_no()`, `bookdown_zenodo()`, `c_sort()`, `create_hexsticker()`, `execshell()`, `is_repository()`, `is_workdir_clean()`, `menu_first()`, `new_branch()`, `orcid2person()`, `store_authors()`, `use_author()`, `validate_email()`, `validate_orcid()`, `yesno()`

`create_hexsticker` *Make hexagonal logo for package*

Description

This function makes a hexagonal logo in INBO style for the provided package name.

Usage

```
create_hexsticker(
  package_name,
  filename = path("man", "figures", "logo.svg"),
  icon,
  x = 0,
  y = 0,
  scale = 1
)
```

Arguments

`package_name`	package name that should be mentioned on the hexagonal sticker.
`filename`	filename to save the sticker.
`icon`	optional filename to an `.svg` file with an icon.
`x`	number of pixels to move the icon to the right. Use negative numbers to move the icon to the left.
`y`	number of pixels to move the icon to the bottom. Use negative numbers to move the icon to the top.
`scale`	Scales the `icon`.

Value

A figure is saved in the working directory or provided path.

See Also

Other utils: `ask_yes_no()`, `bookdown_zenodo()`, `c_sort()`, `clean_git()`, `execshell()`, `is_repository()`, `is_workdir_clean()`, `menu_first()`, `new_branch()`, `orcid2person()`, `store_authors()`, `use_author()`, `validate_email()`, `validate_orcid()`, `yesno()`

Examples

```
## Not run:
# make tempfile to save logo (or just use (path and) filename)
#' output <- tempfile(pattern = "hexsticker", fileext = ".svg")
create_hexsticker("checklist", filename = output)

## End(Not run)
```

`create_package` *Create an R package according to INBO requirements*

Description

Creates a package template in a new folder. Use this function when you want to start a new package. Please DO READ `vignette("getting_started")` before running this function.

Usage

```
create_package(
  package,
  path = ".",
  title,
  description,
  keywords,
  language = "en-GB",
  license = c("GPL-3", "MIT"),
  communities = character(0),
  maintainer
)
```

Arguments

`package`	Name of the new package.
`path`	Where to create the package directory.
`title`	A single sentence with the title of the package.
`description`	A single paragraph describing the package.
`keywords`	A vector of keywords.
`language`	Language of the project in `xx-YY` format. `xx` is the two letter code for the language. `YY` is the two letter code for the language variant. E.g. `en-GB` for British English, `en-US` for American English, `nl-BE` for Belgian Dutch.
`license`	What type of license should be used? Choice between GPL-3 and MIT. Default GPL-3.
`communities`	An optional vector of Zenodo community id's.
`maintainer`	When missing, the function interactively lets you add the maintainer and other authors. Otherwise it must be the output of `utils::person()`.

Details

What you get with a checklist setup:

- minimal folder structure and files required for an R package using INBO guidelines with GPL-3 or MIT license.
- an RStudio project file

- a local git repository
- an initial `NEWS.md` file
- a template for an `README.Rmd`
- set-up for automated checks and releases of the package using GitHub Actions.
- a code of conduct and contributing guidelines.
- the set-up for a pkgdown website using the INBO corporate identity.

See Also

Other setup: `create_project()`, `prepare_ghpages()`, `set_license()`, `setup_package()`, `setup_project()`, `setup_source()`

Examples

```
# maintainer in `utils::person()` format
maintainer <- person(
  given = "Thierry",
  family = "Onkelinx",
  role = c("aut", "cre"),
  email = "thierry.onkelinx@inbo.be",
  comment = c(ORCID = "0000-0001-8804-4216")
)

# creating the package
path <- tempfile()
dir.create(path)
create_package(
  path = path, package = "packagename", title = "package title",
  description = "A short description.", maintainer = maintainer,
  language = "en-GB", license = "GPL-3", keywords = "keyword"
)
```

`create_project` *Initialise a new R project*

Description

This function creates a new RStudio project with `checklist` functionality.

Usage

```
create_project(path, project)
```

Arguments

`path`	The folder in which to create the project as a folder.
`project`	The name of the project.

See Also

Other setup: create_package(), prepare_ghpages(), set_license(), setup_package(), setup_project(), setup_source()

`custom_dictionary` *Add words to custom dictionaries*

Description

Add words to custom dictionaries

Usage

```
custom_dictionary(issues)
```

Arguments

issues	The output of check_spelling().

See Also

Other both: add_badges(), check_filename(), check_lintr(), check_spelling(), default_organisation(), print.checklist_spelling(), read_checklist(), read_organisation(), write_checklist(), write_organisation()

`c_sort` *Sort using the C locale*

Description

Setting the locale before sorting ensures a stable sorting order

Usage

```
c_sort(x, ...)
```

Arguments

x	for sort an R object with a class or a numeric, complex, character or logical vector. For sort.int, a numeric, complex, character or logical vector, or a factor.
...	arguments to be passed to or from methods or (for the default methods and objects without a class) to sort.int.

See Also

Other utils: ask_yes_no(), bookdown_zenodo(), clean_git(), create_hexsticker(), execshell(), is_repository(), is_workdir_clean(), menu_first(), new_branch(), orcid2person(), store_authors(), use_author(), validate_email(), validate_orcid(), yesno()

---

default_organisation *Write default organisation settings*

Description

Store the default organisation rules. First run `org <- organisation$new()` with the appropriate argument. Next you can store the configuration with `default_organisation(org)`.

Usage

```
default_organisation(org = organisation$new())
```

Arguments

org	An organisation object. Create it with organisation$new().

See Also

Other both: add_badges(), check_filename(), check_lintr(), check_spelling(), custom_dictionary(), print.checklist_spelling(), read_checklist(), read_organisation(), write_checklist(), write_organisation()

---

execshell *Pass command lines to a shell*

Description

Cross-platform function to pass a command to the shell, using either base::system() or (Windows-only) base::shell(), depending on the operating system.

Usage

```
execshell(commandstring, intern = FALSE, path = ".", ...)
```

Arguments

commandstring	The system command to be invoked, as a string. Multiple commands can be combined in this single string, e.g. with a multiline string.
intern	a logical (not NA) which indicates whether to capture the output of the command as an R character vector.
path	The path from where the command string needs to be executed
...	Other arguments passed to base::system() or base::shell().

See Also

Other utils: ask_yes_no(), bookdown_zenodo(), c_sort(), clean_git(), create_hexsticker(), is_repository(), is_workdir_clean(), menu_first(), new_branch(), orcid2person(), store_authors(), use_author(), validate_email(), validate_orcid(), yesno()

---

is_repository *Determine if a directory is in a git repository*

Description

The path arguments specifies the directory at which to start the search for a git repository. If it is not a git repository itself, then its parent directory is consulted, then the parent's parent, and so on.

Usage

```
is_repository(path = ".")
```

Arguments

path	the location of the git repository, see details.

Value

TRUE if directory is in a git repository else FALSE

See Also

Other utils: ask_yes_no(), bookdown_zenodo(), c_sort(), clean_git(), create_hexsticker(), execshell(), is_workdir_clean(), menu_first(), new_branch(), orcid2person(), store_authors(), use_author(), validate_email(), validate_orcid(), yesno()

---

is_workdir_clean *Check if the current working directory of a repo is clean*

Description

A clean working directory has no staged, unstaged or untracked files.

Usage

```
is_workdir_clean(repo)
```

Arguments

repo	The path to the git repository. If the directory is not a repository, parent directories are considered (see git_find). To disable this search, provide the filepath protected with I(). When using this parameter, always explicitly call by name (i.e. repo =) because future versions of gert may have additional parameters.

Value

TRUE when there are no staged, unstaged or untracked files. Otherwise FALSE

See Also

Other utils: ask_yes_no(), bookdown_zenodo(), c_sort(), clean_git(), create_hexsticker(), execshell(), is_repository(), menu_first(), new_branch(), orcid2person(), store_authors(), use_author(), validate_email(), validate_orcid(), yesno()

---

menu_first *Improved version of menu()*

Description

Improved version of menu()

Usage

```
menu_first(choices, graphics = FALSE, title = NULL)
```

Arguments

choices	a character vector of choices
graphics	a logical indicating whether a graphics menu should be used if available.
title	a character string to be used as the title of the menu. NULL is also accepted.

See Also

Other utils: ask_yes_no(), bookdown_zenodo(), c_sort(), clean_git(), create_hexsticker(), execshell(), is_repository(), is_workdir_clean(), new_branch(), orcid2person(), store_authors(), use_author(), validate_email(), validate_orcid(), yesno()

new_branch *Create a new branch after cleaning the repo*

Description

This functions first runs `clean_git()`. Then it creates the new branch from the (updated) main branch.

Usage

```
new_branch(branch, verbose = TRUE, checkout = TRUE, repo = ".")
```

Arguments

`branch`	name of branch to check out
`verbose`	display some progress info while downloading
`checkout`	move HEAD to the newly created branch
`repo`	The path to the git repository. If the directory is not a repository, parent directories are considered (see git_find). To disable this search, provide the filepath protected with `I()`. When using this parameter, always explicitly call by name (i.e. `repo = `) because future versions of gert may have additional parameters.

See Also

Other utils: `ask_yes_no()`, `bookdown_zenodo()`, `c_sort()`, `clean_git()`, `create_hexsticker()`, `execshell()`, `is_repository()`, `is_workdir_clean()`, `menu_first()`, `orcid2person()`, `store_authors()`, `use_author()`, `validate_email()`, `validate_orcid()`, `yesno()`

orcid2person *Defunct functions*

Description

Defunct functions

Usage

```
orcid2person()
```

See Also

Other utils: `ask_yes_no()`, `bookdown_zenodo()`, `c_sort()`, `clean_git()`, `create_hexsticker()`, `execshell()`, `is_repository()`, `is_workdir_clean()`, `menu_first()`, `new_branch()`, `store_authors()`, `use_author()`, `validate_email()`, `validate_orcid()`, `yesno()`

`organisation` *The organisation R6 class*

Description

A class with the organisation defaults

Active bindings

`as_person` The default organisation funder and rightsholder.

`get_community` The default organisation Zenodo communities.

`get_email` The default organisation email.

`get_funder` The default funder.

`get_github` The default GitHub organisation domain.

`get_organisation` The organisation requirements.

`get_rightsholder` The default rightsholder.

`template` A list for a check list template.

Methods

Public methods:

- `organisation$new()`
- `organisation$print()`
- `organisation$clone()`

Method `new()`**:** Initialize a new `organisation` object.

Usage:

```
organisation$new(...)
```

Arguments:

`...` The organisation settings. See the details.

Details:

- `github`: the name of the github organisation. Set to `NA_character_` in case you don't want a mandatory github organisation.
- `community`: the mandatory Zenodo community. Defaults to `"inbo"`. Set to `NA_character_` in case you don't want a mandatory community.
- `email`: the e-mail of the organisation. Defaults to `"info@inbo.be"`. Set to `NA_character_` in case you don't want an organisation e-mail.
- `funder`: the funder. Defaults to `"Research Institute for Nature and Forest (INBO)"`. Set to `NA_character_` in case you don't want to set a funder.
- `rightsholder`: the rightsholder. Defaults to `"Research Institute for Nature and Forest (INBO)"`. Set to `NA_character_` in case you don't want to set a rightsholder.

- `organisation`: a named list with one or more organisation default rules. The names of the element must match the e-mail domain name of the organisation. Every element should be a named list containing `affiliation` and `orcid`. `affiliation` is a character vector with the approved organisation names in one or more languages. `orcid = TRUE` indicated a mandatory ORCiD for every member. Use an empty list in case you don't want to set this.

Method `print()`**:** Print the `organisation` object.

Usage:

```
organisation$print(...)
```

Arguments:

`...` currently ignored.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
organisation$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other class: `checklist`, `citation_meta`, `spelling`

`prepare_ghpages` *Prepare a* `gh-pages` *branch with a place holder page*

Description

Prepare a `gh-pages` branch with a place holder page

Usage

```
prepare_ghpages(x = ".", verbose = TRUE)
```

Arguments

`x`	Either a `checklist` object or a path to the source code. Defaults to `.`
`verbose`	display some progress info while downloading

See Also

Other setup: `create_package()`, `create_project()`, `set_license()`, `setup_package()`, `setup_project()`, `setup_source()`

print.checklist_spelling *Display a* checklist_spelling *summary*

Description

Display a checklist_spelling summary

Usage

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

Arguments

x	The checklist_spelling object
...	currently ignored

See Also

Other both: add_badges(), check_filename(), check_lintr(), check_spelling(), custom_dictionary(), default_organisation(), read_checklist(), read_organisation(), write_checklist(), write_organisation()

read_checklist *Read the check list file from a package*

Description

The checklist package stores configuration information in the checklist.yml file in the root of a project. This function reads this configuration. It is mainly used by the other functions inside the package. If no checklist.yml file is found at the path, the function walks upwards through the directory structure until it finds such file. The function returns an error when it reaches the root of the disk without finding a checklist.yml file.

Usage

```
read_checklist(x = ".")
```

Arguments

x	Either a checklist object or a path to the source code. Defaults to . .

Value

A checklist object.

See Also

Other both: add_badges(), check_filename(), check_lintr(), check_spelling(), custom_dictionary(), default_organisation(), print.checklist_spelling(), read_organisation(), write_checklist(), write_organisation()

`read_organisation` *Read the organisation file*

Description

The checklist package stores organisation information in the `organisation.yml` file in the root of a project.

Usage

```
read_organisation(x = ".")
```

Arguments

`x`	Either a `checklist` object or a path to the source code. Defaults to `.`

Value

An `organisation` object.

See Also

Other both: add_badges(), check_filename(), check_lintr(), check_spelling(), custom_dictionary(), default_organisation(), print.checklist_spelling(), read_checklist(), write_checklist(), write_organisation()

`setup_package` *Add or update the checklist infrastructure to an existing package*

Description

Use this function when you have an existing package and you want to use the checklist functionality. Please keep in mind that the checklist is an opinionated list of checks. It might require some breaking changes in your package. Please DO READ `vignette("getting_started")` before running this function.

Usage

```
setup_package(path = ".", license = c("GPL-3", "MIT"))
```

Arguments

path	The path to the package. Defaults to ".".
license	What type of license should be used? Choice between GPL-3 and MIT. Default GPL-3.

Details

What you get with a checklist setup:

- minimal folder structure and files required for an R package using INBO guidelines with GPL-3 or MIT license.
- an RStudio project file
- a local git repository
- an initial NEWS.md file
- a template for an README.Rmd
- set-up for automated checks and releases of the package using GitHub Actions.
- a code of conduct and contributing guidelines.
- the set-up for a pkgdown website using the INBO corporate identity.

See Also

Other setup: create_package(), create_project(), prepare_ghpages(), set_license(), setup_project(), setup_source()

---

setup_project *Set-up* checklist *on an existing R project*

Description

Use this function to set-up or change the checklist infrastructure for an existing project. The function interactively asks questions to set-up the required checks.

Usage

```
setup_project(path = ".")
```

Arguments

path	the project root folder

See Also

Other setup: create_package(), create_project(), prepare_ghpages(), set_license(), setup_package(), setup_source()

setup_source — *Add or update the checklist infrastructure to a repository with source files.*

Description

This adds the required GitHub workflows to run `check_source()` automatically whenever you push commits to GitHub. It also adds a CC-BY 4.0 license file, a `CODE_OF_CONDUCT.md` and the checklist configuration file (`checklist.yml`).

Usage

```
setup_source(path = ".", language = "en-GB")
```

Arguments

`path`	The path to the project. Defaults to `"."`.
`language`	Language of the project in `xx-YY` format. `xx` is the two letter code for the language. `YY` is the two letter code for the language variant. E.g. `en-GB` for British English, `en-US` for American English, `nl-BE` for Belgian Dutch.

See Also

Other setup: `create_package()`, `create_project()`, `prepare_ghpages()`, `set_license()`, `setup_package()`, `setup_project()`

set_license — *Set the proper license*

Description

Set the proper license

Usage

```
set_license(x = ".")
```

Arguments

`x`	Either a `checklist` object or a path to the source code. Defaults to `.`.

See Also

Other setup: `create_package()`, `create_project()`, `prepare_ghpages()`, `setup_package()`, `setup_project()`, `setup_source()`

set_tag *Set a New Tag*

Description

This function is a part of the GitHub Action. Therefore it only works when run in a GitHub Action on the main or master branch. Otherwise it will only return a message. It sets a new tag at the current commit using the related entry from `NEWS.md` as message. This tag will turn into a release.

Usage

```
set_tag(x = ".")
```

Arguments

`x`	Either a `checklist` object or a path to the source code. Defaults to `.`.

See Also

Other package: `check_codemeta()`, `check_cran()`, `check_description()`, `check_documentation()`, `check_environment()`, `check_license()`, `check_package()`, `tidy_desc()`, `update_citation()`, `write_citation_cff()`, `write_zenodo_json()`

spelling *The spelling R6 class*

Description

A class with the configuration for spell checking

Active bindings

`default` The default language of the project.

`get_md` The markdown files within the project.

`get_r` The R files within the project.

`get_rd` The Rd files within the project.

`settings` A list with current spell checking settings.

Methods

Public methods:

- spelling$new()
- spelling$print()
- spelling$set_default()
- spelling$set_exceptions()
- spelling$set_ignore()
- spelling$set_other()
- spelling$clone()

Method `new()`**:** Initialize a new `spelling` object.

Usage:

```
spelling$new(language, base_path = ".")
```

Arguments:

`language` the default language.

`base_path` the base path of the project

Method `print()`**:** Print the `spelling` object.

Usage:

```
spelling$print(...)
```

Arguments:

`...` currently ignored.

Method `set_default()`**:** Define which files to ignore or to spell check in a different language.

Usage:

```
spelling$set_default(language)
```

Arguments:

`language` The language.

Method `set_exceptions()`**:** Define which files to ignore or to spell check in a different language.

Usage:

```
spelling$set_exceptions()
```

Method `set_ignore()`**:** Manually set the ignore vector. Only use this if you known what you are doing.

Usage:

```
spelling$set_ignore(ignore)
```

Arguments:

`ignore` The character vector with ignore file patterns.

Method `set_other()`**:** Manually set the other list. Only use this if you known what you are doing.

Usage:

```
spelling$set_other(other)
```

Arguments:

`other` a list with file patterns per additional language.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
spelling$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other class: `checklist`, `citation_meta`, `organisation`

`store_authors` *Store author details for later usage*

Description

Store author details for later usage

Usage

```
store_authors(x = ".")
```

Arguments

`x`	Either a `checklist` object or a path to the source code. Defaults to `.` .

See Also

Other utils: `ask_yes_no()`, `bookdown_zenodo()`, `c_sort()`, `clean_git()`, `create_hexsticker()`, `execshell()`, `is_repository()`, `is_workdir_clean()`, `menu_first()`, `new_branch()`, `orcid2person()`, `use_author()`, `validate_email()`, `validate_orcid()`, `yesno()`

tidy_desc *Make your DESCRIPTION tidy*

Description

A tidy `DESCRIPTION` uses a strict formatting and order of key-value pairs. This function reads the current `DESCRIPTION` and overwrites it with a tidy version.

Usage

```
tidy_desc(x = ".")
```

Arguments

`x`	Either a `checklist` object or a path to the source code. Defaults to `.`.

See Also

Other package: `check_codemeta()`, `check_cran()`, `check_description()`, `check_documentation()`, `check_environment()`, `check_license()`, `check_package()`, `set_tag()`, `update_citation()`, `write_citation_cff()`, `write_zenodo_json()`

update_citation *Create or update the citation files*

Description

The function extracts citation meta data from the project. Then it checks the required meta data. Upon success, it writes several files.

Usage

```
update_citation(x = ".", quiet = FALSE)
```

Arguments

`x`	Either a `checklist` object or a path to the source code. Defaults to `.`.
`quiet`	Whether to print check output during checking.

Details

- `.zenodo.json` contains the citation information in the format that Zenodo requires.
- `CITATION.cff` provides the citation information in the format that GitHub requires.
- `inst/CITATION` provides the citation information in the format that R packages require. It is only relevant for packages.

Value

An invisible `checklist` object.

Note

Source of the citation meta data:

- package: `DESCRIPTION`
- project: `README.md`

Should you want to add more information to the `inst/CITATION` file, add it to that file outside `# begin checklist entry` and `# end checklist entry`.

See Also

Other package: `check_codemeta()`, `check_cran()`, `check_description()`, `check_documentation()`, `check_environment()`, `check_license()`, `check_package()`, `set_tag()`, `tidy_desc()`, `write_citation_cff()`, `write_zenodo_json()`

---

use_author *Which author to use*

Description

Reuse existing author information or add a new author. Allows to update existing author information.

Usage

```
use_author(email)
```

Arguments

`email`	An optional email address. When given and it matches with a single person, the function immediately returns the information of that person.

Value

A data.frame with author information.

See Also

Other utils: `ask_yes_no()`, `bookdown_zenodo()`, `c_sort()`, `clean_git()`, `create_hexsticker()`, `execshell()`, `is_repository()`, `is_workdir_clean()`, `menu_first()`, `new_branch()`, `orcid2person()`, `store_authors()`, `validate_email()`, `validate_orcid()`, `yesno()`

validate_email *Check if a vector contains valid email*

Description

It only checks the format of the text, not if the email address exists.

Usage

```
validate_email(email)
```

Arguments

email	A vector with email addresses.

Value

A logical vector.

See Also

Other utils: ask_yes_no(), bookdown_zenodo(), c_sort(), clean_git(), create_hexsticker(), execshell(), is_repository(), is_workdir_clean(), menu_first(), new_branch(), orcid2person(), store_authors(), use_author(), validate_orcid(), yesno()

validate_orcid *Validate the structure of an ORCID id*

Description

Checks whether the ORCID has the proper format and the checksum.

Usage

```
validate_orcid(orcid)
```

Arguments

orcid	A vector of ORCID

Value

A logical vector with the same length as the input vector.

See Also

Other utils: `ask_yes_no()`, `bookdown_zenodo()`, `c_sort()`, `clean_git()`, `create_hexsticker()`, `execshell()`, `is_repository()`, `is_workdir_clean()`, `menu_first()`, `new_branch()`, `orcid2person()`, `store_authors()`, `use_author()`, `validate_email()`, `yesno()`

`write_checklist` *Write a check list with allowed issues in the source code*

Description

checklist stores it configuration as a `checklist.yml` file. `create_package()`, `setup_package()` and `setup_source()` generate a default file. If you need to allow some warnings or notes, you need to update the configuration.

Usage

```
write_checklist(x = ".")
```

Arguments

`x`	Either a `checklist` object or a path to the source code. Defaults to `.`

Details

First run `x <- checklist::check_package()` or `x <- checklist::check_source()`. These commands run the checks and store the `checklist` object in the variable `x`. Next you can store the configuration with `checklist::write_checklist(x)`. This will first list any existing allowed warnings or notes. For every one of them, choose whether you want to keep it or not. Next, the function presents every new warning or note which you may allow or not. If you choose to allow a warning or note, you must provide a motivation. Please provide a sensible motivation. Keep in mind that `checklist.yml` stores these motivations in plain text, so they are visible for other users. We use the `yesno()` function to make sure you carefully read the questions.

Caveat

When you allow a warning or note, this warning or note must appear. Otherwise you get a "missing warning" or "missing note" error. So if you fix an allowed warning or note, you need to rerun `checklist::write_checklist(x)` and remove the old version.

If you can solve a warning or note, then solve it rather than to allow it. Only allow a warning or note in case of a generic "problem" that you can't solve. The best example is the `checking CRAN incoming feasibility ... NOT` which appears when checking a package not on CRAN. That is should an allowed note as long as the package is not on CRAN. Or permanently when your package is not intended for CRAN.

Do not allow a warning or note to fix an issue specific to your machine. That will result in an error when checking the package on an other machine (e.g. GitHub actions).

See Also

Other both: add_badges(), check_filename(), check_lintr(), check_spelling(), custom_dictionary(), default_organisation(), print.checklist_spelling(), read_checklist(), read_organisation(), write_organisation()

---

write_citation_cff *Write a* CITATION.cff *file*

Description

This file format contains the citation information. It is supported by GitHub, Zenodo and Zotero. This function is super-seeded by update_citation().

Usage

```
write_citation_cff(x = ".", roles)
```

Arguments

x	Either a checklist object or a path to the source code. Defaults to . .
roles	No longer used.

Value

An invisible checklist object.

See Also

Other package: check_codemeta(), check_cran(), check_description(), check_documentation(), check_environment(), check_license(), check_package(), set_tag(), tidy_desc(), update_citation(), write_zenodo_json()

---

write_organisation *Write organisation settings*

Description

Store the organisation rules into organisation.yml file. First run org <- organisation$new() with the appropriate argument. Next you can store the configuration with write_organisation(org).

Usage

```
write_organisation(org, x = ".")
```

Arguments

org	An organisation object. Create it with organisation$new().
x	Either a checklist object or a path to the source code. Defaults to . .

See Also

Other both: add_badges(), check_filename(), check_lintr(), check_spelling(), custom_dictionary(), default_organisation(), print.checklist_spelling(), read_checklist(), read_organisation(), write_checklist()

---

write_zenodo_json *Write a* .zenodo.json *file*

Description

Zenodo uses the .zenodo.json file to define the citation information. See the Zenodo developers website for more information.

Usage

```
write_zenodo_json(x = ".")
```

Arguments

x	Either a checklist object or a path to the source code. Defaults to . .

Value

An invisible checklist object.

See Also

Other package: check_codemeta(), check_cran(), check_description(), check_documentation(), check_environment(), check_license(), check_package(), set_tag(), tidy_desc(), update_citation(), write_citation_cff()

yesno *A function that asks a yes or no question to the user*

Description

A function that asks a yes or no question to the user

Usage

```
yesno(...)
```

Arguments

...	Currently ignored

Value

A logical where TRUE implies a "yes" answer from the user.

Author(s)

Hadley Wickham Hadley@Rstudio.com Largely based on `devtools:::yesno()`. The user gets three options in an random order: 2 for "no", 1 for "yes". The wording for "yes" and "no" is random as well. This forces the user to carefully read the question.

See Also

Other utils: `ask_yes_no()`, `bookdown_zenodo()`, `c_sort()`, `clean_git()`, `create_hexsticker()`, `execshell()`, `is_repository()`, `is_workdir_clean()`, `menu_first()`, `new_branch()`, `orcid2person()`, `store_authors()`, `use_author()`, `validate_email()`, `validate_orcid()`

Index