

Package: checklist (via r-universe)

September 26, 2026

Type Package

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

Version 0.6.1

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, citeme (>= 0.1.4), cli, cyclocomp, desc, devtools (> 2.4.0), gert, hunspell, lintr (>= 3.3.0), pkgdown (>= 2.0.7), R6, rcmdcheck, rmarkdown, sessioninfo, utils, withr, yaml

Suggests bookdown, covr, curl, knitr, mockery, pak, renv, roxygen2, rstudioapi, showtext, sysfonts, testthat

VignetteBuilder knitr

Additional_repositories <https://inbo.r-universe.dev>

Config/citeme/communities inbo

Config/citeme/keywords quality control; documentation; publication; inbo_version

Config/roxygen2/version 8.0.0

Encoding UTF-8

Language en-GB

Roxygen list(markdown = TRUE)

SystemRequirements Pandoc (>= 1.17.2)

Config/pak/sysreqs cmake libfontconfig1-dev libfreetype6-dev libfribidi-dev git make libharfbuzz-dev libgit2-dev libicu-dev libjpeg-dev libpng-dev libtiff-dev libuv1-dev libwebp-dev libxml2-dev libssl-dev libx11-dev zlib1g-dev

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

Date/Publication 2026-06-28 12:25:12 UTC

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

RemoteRef HEAD

RemoteSha 5e93a3b6ae945aaac666e2a490ff2fd4f94244c7

Contents

add_agents	3
add_issue_templates	3
ask_rightsholder_funder	4
c_sort	4
check_cran	5
check_description	5
check_documentation	6
check_environment	7
check_filename	8
check_folder	9
check_license	10
check_lintr	10
check_package	11
check_project	12
check_source	13
check_spelling	13
checklist	14
clean_git	17
create_hexsticker	18
create_package	19
create_project	20
custom_dictionary	20
execshell	21
install_pak	21
is_repository	22
is_workdir_clean	22
new_branch	23
prepare_ghpages	23
print.checklist_spelling	24
read_checklist	24
set_license	25
set_tag	26
setup_package	26
setup_project	27
setup_source	28

ask_rightsholder_funder
Ask for rights holder or funder

Description

Ask for rights holder or funder

Usage

```
ask_rightsholder_funder(org, type = c("rightsholder", "funder", "publisher"))
```

Arguments

org	Organisation object
type	Character, either "rightsholder", "funder" or "publisher"

Value

A list with the selected names and the updated organisation object

See Also

Other utils: [c_sort\(\)](#), [create_hexsticker\(\)](#), [execshell\(\)](#), [install_pak\(\)](#), [yesno\(\)](#)

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_rightsholder_funder\(\)](#), [create_hexsticker\(\)](#), [execshell\(\)](#), [install_pak\(\)](#), [yesno\(\)](#)

check_cran	<i>Run all the package checks required by CRAN</i>
------------	--

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, time_out = 30)
```

Arguments

<code>x</code>	Either a checklist object or a path to the source code. Defaults to <code>..</code>
<code>quiet</code>	Whether to print check output during checking.
<code>time_out</code>	The time in seconds to wait for a response from the world clock API. Default is 15 seconds, but you can increase this if you have a slow internet connection. If the world clock API is not available, the system clock will be used without a warning, but the check will be less reliable.

Value

A checklist object.

See Also

Other package: [check_description\(\)](#), [check_documentation\(\)](#), [check_environment\(\)](#), [check_license\(\)](#), [check_package\(\)](#), [tidy_desc\(\)](#)

check_description	<i>Check the DESCRIPTION file</i>
-------------------	-----------------------------------

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

<code>x</code>	Either a checklist object or a path to the source code. Defaults to <code>..</code>
----------------	---

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_cran\(\)](#), [check_documentation\(\)](#), [check_environment\(\)](#), [check_license\(\)](#), [check_package\(\)](#), [tidy_desc\(\)](#)

check_documentation	<i>Check the documentation</i>
---------------------	--------------------------------

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_cran\(\)](#), [check_description\(\)](#), [check_environment\(\)](#), [check_license\(\)](#), [check_package\(\)](#), [tidy_desc\(\)](#)

check_environment	<i>Make sure that the required environment variables are set on GitHub</i>
-------------------	--

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_cran\(\)](#), [check_description\(\)](#), [check_documentation\(\)](#), [check_license\(\)](#), [check_package\(\)](#), [tidy_desc\(\)](#)

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 csl, eps, gif, jpg, jpeg, pdf, png, ps, svg, tiff, tif, wmf 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_agents\(\)](#), [add_issue_templates\(\)](#), [check_lintr\(\)](#), [check_spelling\(\)](#), [custom_dictionary\(\)](#), [print.checklist_spelling\(\)](#), [read_checklist\(\)](#), [update_citation\(\)](#), [write_checklist\(\)](#)

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 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	<i>Check the license of a package</i>
---------------	---------------------------------------

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 = ".", org)
```

Arguments

x	Either a checklist object or a path to the source code. Defaults to ..
org	An organisation object. If missing, the organisation will be read from the project.

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_cran\(\)](#), [check_description\(\)](#), [check_documentation\(\)](#), [check_environment\(\)](#), [check_package\(\)](#), [tidy_desc\(\)](#)

check_lintr	<i>Check the packages for linters</i>
-------------	---------------------------------------

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.

Details

When `check_lintr()` runs on a git repository, it checks whether the organisation has a custom `.lintr` file in their checklist repository. If so, it uses this file. Otherwise it looks for a local `.lintr` file in the project directory. If this file is not present, it uses the default `.lintr` file provided with the checklist package.

See Also

Other both: [add_agents\(\)](#), [add_issue_templates\(\)](#), [check_filename\(\)](#), [check_spelling\(\)](#), [custom_dictionary\(\)](#), [print.checklist_spelling\(\)](#), [read_checklist\(\)](#), [update_citation\(\)](#), [write_checklist\(\)](#)

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,  
  time_out = 30  
)
```

Arguments

<code>x</code>	Either a checklist object or a path to the source code. Defaults to <code>..</code> .
<code>fail</code>	Should the function return an error in case of a problem? Defaults to <code>TRUE</code> on a non-interactive session and <code>FALSE</code> on an interactive session.
<code>pkgdown</code>	Test pkgdown website. Defaults to <code>TRUE</code> on an interactive session and <code>FALSE</code> on a non-interactive session.
<code>quiet</code>	Whether to print check output during checking.
<code>time_out</code>	The time in seconds to wait for a response from the world clock API. Default is 15 seconds, but you can increase this if you have a slow internet connection. If the world clock API is not available, the system clock will be used without a warning, but the check will be less reliable.

Details

List of checks in order:

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

See Also

Other package: [check_cran\(\)](#), [check_description\(\)](#), [check_documentation\(\)](#), [check_environment\(\)](#), [check_license\(\)](#), [tidy_desc\(\)](#)

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

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

See Also

Other project: [check_folder\(\)](#), [check_source\(\)](#)

check_source	<i>Standardised test for an R source repository</i>
--------------	---

Description

Defunct function. Please use [check_project\(\)](#) instead.

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.

See Also

Other project: [check_folder\(\)](#), [check_project\(\)](#)

check_spelling	<i>Spell check a package or project</i>
----------------	---

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_agents\(\)](#), [add_issue_templates\(\)](#), [check_filename\(\)](#), [check_lintr\(\)](#), [custom_dictionary\(\)](#), [print.checklist_spelling\(\)](#), [read_checklist\(\)](#), [update_citation\(\)](#), [write_checklist\(\)](#)

 checklist

The checklist R6 class

Description

A class which contains all checklist results.

Super class

`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_gha_install` A vector with bash commands for GitHub Actions.

`get_path` The path to the package.

`get_pak` Packages to install with `pak::pkg_install()`.

`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_pak()`
- `checklist$set_gha_install()`
- `checklist$set_required()`

- `checklist$clone()`

`checklist$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

`checklist$add_linter()`: Add results from `lintr::lint_package()`

Usage:

```
checklist$add_linter(linter, error = FALSE)
```

Arguments:

`linter` A vector with linter errors.

`error` A logical indicating if the linter should be considered an error.

`checklist$add_motivation()`: Add motivation for allowed issues.

Usage:

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

Arguments:

`which` Which kind of issue to add.

`checklist$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.

`checklist$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.

`checklist$add_spelling()`: Add results from `check_spelling()`

Usage:

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

Arguments:

issues A data.frame with spell checking issues.

checklist\$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.

checklist\$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.

checklist\$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.

checklist\$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?

checklist\$print(): Print the checklist object. Add quiet = TRUE to suppress printing.

Usage:

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

Arguments:

... See description.

checklist\$set_pak(): Set packages to install with pak::pkg_install().

Usage:

```
checklist$set_pak(pkg = character(0))
```

Arguments:

pkg A vector of packages to install with `pak::pkg_install()`.

`checklist$set_gha_install()`: Set optional install commands for GitHub Actions

Usage:

`checklist$set_gha_install(commands)`

Arguments:

commands A vector with commands.

`checklist$set_required()`: set required checks

Usage:

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

Arguments:

checks a vector of required checks

`checklist$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: [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 <code>I()</code> . When using this parameter, always explicitly call by name (i.e. <code>repo = </code>) because future versions of gert may have additional parameters.
verbose	display some progress info while downloading

See Also

Other git: [is_repository\(\)](#), [is_workdir_clean\(\)](#), [new_branch\(\)](#), [set_tag\(\)](#)

create_hexsticker	<i>Make hexagonal logo for package</i>
-------------------	--

Description

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

Usage

```
create_hexsticker(
  package_name,
  filename = file.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_rightsholder_funder\(\)](#), [c_sort\(\)](#), [execshell\(\)](#), [install_pak\(\)](#), [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	<i>Create an R package according to INBO requirements</i>
----------------	---

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 = ".")
```

Arguments

package	Name of the new package.
path	Where to create the package directory.

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\(\)](#)

create_project	<i>Initialise a new R project</i>
----------------	-----------------------------------

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	<i>Add words to custom dictionaries</i>
-------------------	---

Description

Add words to custom dictionaries

Usage

```
custom_dictionary(issues)
```

Arguments

issues	The output of <code>check_spelling()</code> .
--------	---

See Also

Other both: [add_agents\(\)](#), [add_issue_templates\(\)](#), [check_filename\(\)](#), [check_lintr\(\)](#), [check_spelling\(\)](#), [print.checklist_spelling\(\)](#), [read_checklist\(\)](#), [update_citation\(\)](#), [write_checklist\(\)](#)

execshell	<i>Pass command lines to a shell</i>
-----------	--------------------------------------

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 <code>base::system()</code> or <code>base::shell()</code> .

See Also

Other utils: [ask_rightsholder_funder\(\)](#), [c_sort\(\)](#), [create_hexsticker\(\)](#), [install_pak\(\)](#), [yesno\(\)](#)

install_pak	<i>Install extra packages defined in checklist.yml</i>
-------------	--

Description

Install extra packages defined in `checklist.yml`

Usage

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

Arguments

x	Either a checklist object or a path to the source code. Defaults to ..
...	Additional arguments passed to <code>pak::pkg_install()</code>

See Also

Other utils: [ask_rightsholder_funder\(\)](#), [c_sort\(\)](#), [create_hexsticker\(\)](#), [execshell\(\)](#), [yesno\(\)](#)

is_repository	<i>Determine if a directory is in a git repository</i>
---------------	--

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 git: [clean_git\(\)](#), [is_workdir_clean\(\)](#), [new_branch\(\)](#), [set_tag\(\)](#)

is_workdir_clean	<i>Check if the current working directory of a repo is clean</i>
------------------	--

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. <code>repo = </code>) 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 git: [clean_git\(\)](#), [is_repository\(\)](#), [new_branch\(\)](#), [set_tag\(\)](#)

new_branch	<i>Create a new branch after cleaning the repo</i>
------------	--

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 <code>I()</code> . When using this parameter, always explicitly call by name (i.e. <code>repo = </code>) because future versions of gert may have additional parameters.

See Also

Other git: [clean_git\(\)](#), [is_repository\(\)](#), [is_workdir_clean\(\)](#), [set_tag\(\)](#)

prepare_ghpages	<i>Prepare a gh-pages branch with a place holder page</i>
-----------------	---

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 <code>..</code>
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_agents\(\)](#), [add_issue_templates\(\)](#), [check_filename\(\)](#), [check_lintr\(\)](#), [check_spelling\(\)](#), [custom_dictionary\(\)](#), [read_checklist\(\)](#), [update_citation\(\)](#), [write_checklist\(\)](#)

```
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_agents\(\)](#), [add_issue_templates\(\)](#), [check_filename\(\)](#), [check_lintr\(\)](#), [check_spelling\(\)](#), [custom_dictionary\(\)](#), [print.checklist_spelling\(\)](#), [update_citation\(\)](#), [write_checklist\(\)](#)

set_license	<i>Set the proper license</i>
-------------	-------------------------------

Description

Set the proper license

Usage

```
set_license(x = ".", license, org)
```

Arguments

x	Either a checklist object or a path to the source code. Defaults to ..
license	the license to set. If missing, the user will be prompted to choose a license.
org	An organisation object. If missing, the organisation will be read from the project.

Value

Invisible NULL.

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 git: [clean_git\(\)](#), [is_repository\(\)](#), [is_workdir_clean\(\)](#), [new_branch\(\)](#)

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 = ".")
```

Arguments

path The path to the package. Defaults to ".".

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	<i>Set-up checklist on an existing R project</i>
---------------	--

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	<i>Add or update the checklist infrastructure to a repository with source files.</i>
--------------	--

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 = ".")
```

Arguments

path The path to the project. Defaults to ".".

See Also

Other setup: [create_package\(\)](#), [create_project\(\)](#), [prepare_ghpages\(\)](#), [set_license\(\)](#), [setup_package\(\)](#), [setup_project\(\)](#)

spelling	<i>The spelling R6 class</i>
----------	------------------------------

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()`

`spelling$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

`spelling$print()`: Print the spelling object.

Usage:

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

Arguments:

`...` currently ignored.

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

Usage:

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

Arguments:

`language` The language.

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

Usage:

```
spelling$set_exceptions()
```

`spelling$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.

`spelling$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.

`spelling$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](#)

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_cran\(\)](#), [check_description\(\)](#), [check_documentation\(\)](#), [check_environment\(\)](#), [check_license\(\)](#), [check_package\(\)](#)

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 both: [add_agents\(\)](#), [add_issue_templates\(\)](#), [check_filename\(\)](#), [check_lintr\(\)](#), [check_spelling\(\)](#), [custom_dictionary\(\)](#), [print.checklist_spelling\(\)](#), [read_checklist\(\)](#), [write_checklist\(\)](#)

write_checklist

Write a check list with allowed issues in the source code

Description

checklist stores its 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 be 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 another machine (e.g. GitHub actions).

See Also

Other both: `add_agents()`, `add_issue_templates()`, `check_filename()`, `check_lintr()`, `check_spelling()`, `custom_dictionary()`, `print.checklist_spelling()`, `read_checklist()`, `update_citation()`

`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_rightsholder_funder\(\)](#), [c_sort\(\)](#), [create_hexsticker\(\)](#), [execshell\(\)](#), [install_pak\(\)](#)

Index

- * **both**
 - add_agents, 3
 - add_issue_templates, 3
 - check_filename, 8
 - check_lintr, 10
 - check_spelling, 13
 - custom_dictionary, 20
 - print.checklist_spelling, 24
 - read_checklist, 24
 - update_citation, 31
 - write_checklist, 32
- * **class**
 - checklist, 14
 - spelling, 28
- * **git**
 - clean_git, 17
 - is_repository, 22
 - is_workdir_clean, 22
 - new_branch, 23
 - set_tag, 26
- * **package**
 - check_cran, 5
 - check_description, 5
 - check_documentation, 6
 - check_environment, 7
 - check_license, 10
 - check_package, 11
 - tidy_desc, 30
- * **project**
 - check_folder, 9
 - check_project, 12
 - check_source, 13
- * **setup**
 - create_package, 19
 - create_project, 20
 - prepare_ghpages, 23
 - set_license, 25
 - setup_package, 26
 - setup_project, 27
 - setup_source, 28
- * **utils**
 - ask_rightsholder_funder, 4
 - c_sort, 4
 - create_hexsticker, 18
 - execshell, 21
 - install_pak, 21
 - yesno, 33
- add_agents, 3
- add_agents(), 3, 9, 11, 13, 20, 24, 25, 31, 32
- add_issue_templates, 3
- add_issue_templates(), 3, 9, 11, 13, 20, 24, 25, 31, 32
- ask_rightsholder_funder, 4
- ask_rightsholder_funder(), 4, 19, 21, 33
- base::system(), 21
- c_sort, 4
- c_sort(), 4, 19, 21, 33
- check_cran, 5
- check_cran(), 6, 7, 10, 12, 30
- check_description, 5
- check_description(), 5, 7, 10, 12, 30
- check_documentation, 6
- check_documentation(), 5–7, 10, 12, 30
- check_environment, 7
- check_environment(), 5–7, 10, 12, 30
- check_filename, 8
- check_filename(), 3, 11, 13, 20, 24, 25, 31, 32
- check_folder, 9
- check_folder(), 13
- check_license, 10
- check_license(), 5–7, 12, 30
- check_lintr, 10
- check_lintr(), 3, 9, 13, 20, 24, 25, 31, 32
- check_package, 11
- check_package(), 5–7, 10, 30

check_project, 12
check_project(), 9, 13
check_source, 13
check_source(), 9, 13
check_spelling, 13
check_spelling(), 3, 9, 11, 20, 24, 25, 31, 32
checklist, 14, 30
clean_git, 17
clean_git(), 22, 23, 26
create_hexsticker, 18
create_hexsticker(), 4, 21, 33
create_package, 19
create_package(), 20, 24, 25, 27, 28
create_project, 20
create_project(), 19, 24, 25, 27, 28
custom_dictionary, 20
custom_dictionary(), 3, 9, 11, 13, 24, 25, 31, 32

execshell, 21
execshell(), 4, 19, 21, 33

git_find, 18, 22, 23

I(), 18, 22, 23
install_pak, 21
install_pak(), 4, 19, 21, 33
is_repository, 22
is_repository(), 18, 23, 26
is_workdir_clean, 22
is_workdir_clean(), 18, 22, 23, 26

lintr::lint_package(), 10

new_branch, 23
new_branch(), 18, 22, 23, 26

prepare_ghpages, 23
prepare_ghpages(), 19, 20, 25, 27, 28
print.checklist_spelling, 24
print.checklist_spelling(), 3, 9, 11, 13, 20, 25, 31, 32

read_checklist, 24
read_checklist(), 3, 9, 11, 13, 20, 24, 31, 32

set_license, 25
set_license(), 19, 20, 24, 27, 28
set_tag, 26
set_tag(), 18, 22, 23

setup_package, 26
setup_package(), 19, 20, 24, 25, 27, 28
setup_project, 27
setup_project(), 19, 20, 24, 25, 27, 28
setup_source, 28
setup_source(), 19, 20, 24, 25, 27
spelling, 14, 17, 28

tidy_desc, 30
tidy_desc(), 5–7, 10, 12

update_citation, 31
update_citation(), 3, 9, 11, 13, 20, 24, 25, 32

write_checklist, 32
write_checklist(), 3, 9, 11, 13, 20, 24, 25, 31

yesno, 33
yesno(), 4, 19, 21