

# Package ‘cleanr’

June 15, 2023

**Type** Package

**Title** Helps You to Code Cleaner

**Version** 1.4.0

**Description** Check your R code for some of the most common layout flaws. Many tried to teach us how to write code less dreadful, be it implicitly as B. W. Kernighan and D. M. Ritchie (1988) <ISBN:0-13-110362-8> in 'The C Programming Language' did, be it explicitly as R.C. Martin (2008) <ISBN:0-13-235088-2> in 'Clean Code: A Handbook of Agile Software Craftsmanship' did. So we should check our code for files too long or wide, functions with too many lines, too wide lines, too many arguments or too many levels of nesting. Note: This is not a static code analyzer like pylint or the like. Checkout <<https://cran.r-project.org/package=lintr>> instead.

**License** BSD\_2\_clause + file LICENSE

**URL** <https://gitlab.com/fvafrcu/cleanr>

**Depends** R (>= 3.3.0)

**Imports** checkmate,  
fritools,  
pkgload,  
rprojroot

**Suggests** devtools,  
rasciidoc,  
rmarkdown,  
RUnit,  
testthat,  
usethis

**VignetteBuilder** rasciidoc

**Encoding** UTF-8

**RoxygenNote** 7.2.3

**Collate** 'utils.R'  
'internals.R'  
'checks.R'

'cleanr-package.R'  
 'options.R'  
 'throw.R'  
 'wrappers.R'  
 'zzz.R'

## R topics documented:

cleanr-package . . . . .	2
check_directory . . . . .	3
check_file . . . . .	4
check_file_layout . . . . .	4
check_functions_in_file . . . . .	5
check_function_layout . . . . .	6
check_package . . . . .	7
file_checks . . . . .	8
function_checks . . . . .	9
get_cleanr_options . . . . .	11
set_cleanr_options . . . . .	12

<b>Index</b>	<b>14</b>
--------------	-----------

---

cleanr-package	<i>Helps You to Code Cleaner</i>
----------------	----------------------------------

---

## Description

Check your R code for some of the most common layout flaws.

## Details

Many tried to teach us how to write code less dreadful, be it implicitly as B. W. Kernighan and D. M. Ritchie in *The C Programming Language* did, be it explicitly as R.C. Martin in *Clean Code: A Handbook of Agile Software Craftsmanship* did.

So we should check our code for files too long or wide, functions with too many lines, too wide lines, too many arguments or too many levels of nesting.

## Note

This is not a static code analyzer like pylint or the like. If you're looking for a static code analyzer, check out `lintr` (<https://cran.r-project.org/package=lintr>).

## See Also

Packages `codetools` (<https://cran.r-project.org/package=codetools>), `formatR` (<https://cran.r-project.org/package=formatR>) and `lintr` (<https://cran.r-project.org/package=lintr>).

---

check_directory	<i>Check a Directory</i>
-----------------	--------------------------

---

## Description

Run [check\\_file](#) on files in a directory.

## Usage

```
check_directory(path, pattern = "\\.[rR]$", recursive = FALSE, ...)
```

## Arguments

path	Path to the directory to be checked.
pattern	A pattern to search files with, see <a href="#">list.files</a> .
recursive	Search the directory recursively? See <a href="#">list.files</a> .
...	Arguments to be passed to <a href="#">check_file</a> .

## Details

The function catches the messages of "cleanr"-conditions [thrown](#) by [check\\_file](#) and, if it caught any, [throws](#) them.

## Value

[Invisibly TRUE](#), but see *Details*.

## See Also

[check\\_package](#).

Other wrappers: [check\\_file\\_layout\(\)](#), [check\\_file\(\)](#), [check\\_function\\_layout\(\)](#), [check\\_functions\\_in\\_file\(\)](#), [check\\_package\(\)](#)

## Examples

```
# load internal functions first.
load_internal_functions("cleanr")
print(cleanr::check_directory(system.file("source", "R", package = "cleanr"),
                              max_num_arguments = 8, max_file_width = 90,
                              max_file_length = 350,
                              check_return = FALSE))
```

---

`check_file`*Check a File*

---

**Description**

Run `check_functions_in_file` and `check_file_layout` on a file.

**Usage**

```
check_file(path, ...)
```

**Arguments**

`path` Path to the file to be checked.  
`...` Arguments to be passed to `check_functions_in_file` or `check_file_layout`.

**Details**

The function catches the messages of "cleanr"-conditions thrown by `check_functions_in_file` and `check_file_layout` and, if it caught any, throws them.

**Value**

Invisibly TRUE, but see *Details*.

**See Also**

Other wrappers: `check_directory()`, `check_file_layout()`, `check_function_layout()`, `check_functions_in_file()`, `check_package()`

**Examples**

```
print(cleanr::check_file(system.file("source", "R", "utils.R",  
package = "cleanr")))
```

---

`check_file_layout`*Check a File's Layout*

---

**Description**

Run all `file_checks` on a file.

**Usage**

```
check_file_layout(  
  path,  
  max_file_length = get_cleanr_options("max_file_length"),  
  max_file_width = get_cleanr_options("max_file_width")  
)
```

**Arguments**

path Path to the file to be checked.  
max\_file\_length See [check\\_file\\_length](#).  
max\_file\_width See [check\\_file\\_width](#).

**Details**

The function catches the messages of "cleanr"-conditions [thrown](#) by [file\\_checks](#) and, if it caught any, [throws](#) them.

**Value**

Invisibly TRUE, but see *Details*.

**See Also**

Other wrappers: [check\\_directory\(\)](#), [check\\_file\(\)](#), [check\\_function\\_layout\(\)](#), [check\\_functions\\_in\\_file\(\)](#), [check\\_package\(\)](#)

**Examples**

```
print(cleanr::check_file_layout(system.file("source", "R", "checks.R",  
                                           package = "cleanr")))
```

---

check\_functions\_in\_file

*Check All Functions Defined in a File*

---

**Description**

Run [check\\_function\\_layout](#) on all functions defined in a file.

**Usage**

```
check_functions_in_file(path, ...)
```

## Arguments

path            Path to the file to be checked.  
...            Arguments to be passed to [check\\_function\\_layout](#).

## Details

The function catches the messages of "cleanr"-conditions thrown by [check\\_function\\_layout](#) and, if it caught any, throws them.

## Value

Invisibly TRUE, but see *Details*.

## See Also

Other wrappers: [check\\_directory\(\)](#), [check\\_file\\_layout\(\)](#), [check\\_file\(\)](#), [check\\_function\\_layout\(\)](#), [check\\_package\(\)](#)

## Examples

```
print(cleanr::check_functions_in_file(system.file("source", "R", "utils.R",  
                                                  package = "cleanr")))
```

---

check\_function\_layout *Check a Function's Layout*

---

## Description

Run all [function\\_checks](#) on a function.

## Usage

```
check_function_layout(  
  object,  
  function_name = NULL,  
  max_lines_of_code = get_cleanr_options("max_lines_of_code"),  
  max_lines = get_cleanr_options("max_lines"),  
  max_num_arguments = get_cleanr_options("max_num_arguments"),  
  max_nesting_depth = get_cleanr_options("max_nesting_depth"),  
  max_line_width = get_cleanr_options("max_line_width"),  
  check_return = get_cleanr_options("check_return")  
)
```

## Arguments

object	The function to be checked.
function_name	The name to be used for reporting. Stick with the default: If NULL, it is taken from the object given. Argument is used internally to pass function names retrieved via <code>get</code> in the wrapper function <code>check_functions_in_file</code> .
max_lines_of_code	See <code>check_num_lines_of_code</code> .
max_lines	See <code>check_num_lines</code> .
max_num_arguments	See <code>check_num_arguments</code> .
max_nesting_depth	See <code>check_nesting_depth</code> .
max_line_width	See <code>check_line_width</code> .
check_return	See <code>check_return</code> .

## Details

The functions catches the messages of "cleanr"-conditions `thrown` by `function_checks` and, if it caught any, `throws` them.

## Value

Invisibly TRUE, but see *Details*.

## See Also

Other wrappers: `check_directory()`, `check_file_layout()`, `check_file()`, `check_functions_in_file()`, `check_package()`

## Examples

```
print(cleanr::check_function_layout(cleanr::check_num_lines))
```

---

check\_package

*Check a Package*

---

## Description

Run `check_file` on a package's source.

## Usage

```
check_package(path, pattern = "\\.[rR]$", ...)
```

**Arguments**

path	Path to the package to be checked.
pattern	A pattern to search files with, see <a href="#">list.files</a> .
...	Arguments to be passed to <a href="#">check_file</a> .

**Details**

The function catches the messages of "cleanr"-conditions thrown by [check\\_file](#) and, if it caught any, [throws](#) them.

**Value**

Invisibly TRUE, but see *Details*.

**See Also**

Other wrappers: [check\\_directory\(\)](#), [check\\_file\\_layout\(\)](#), [check\\_file\(\)](#), [check\\_function\\_layout\(\)](#), [check\\_functions\\_in\\_file\(\)](#)

**Examples**

```
# create a fake package first:
package_path <- file.path(tempdir(), "fake")
usethis::create_package(package_path, fields = NULL,
                        rstudio = FALSE, open = FALSE)
directory <- system.file("runit_tests", "source", "R_s4",
                        package = "cleanr")
file.copy(list.files(directory, full.names = TRUE), file.path(package_path,
                                                              "R"))
RUnit::checkTrue(cleanr::check_package(package_path, check_return = FALSE))
```

---

file\_checks

*File Checks*


---

**Description**

A set of tiny functions to check that files adhere to a layout style. A file should have a clear layout, it should

- not have too many lines and
- not have lines too wide.

**Usage**

```
check_file_width(path, max_file_width = gco("max_file_width"))
```

```
check_file_length(path, max_file_length = gco("max_file_length"))
```



## Arguments

path	Path to the file to be checked.
max_file_width	The maximum line width accepted. Set (preferably via <a href="#">set_cleanr_options</a> ) to NULL or FALSE to disable the check.
max_file_length	The maximum number of lines accepted. Set (preferably via <a href="#">set_cleanr_options</a> ) to NULL or FALSE to disable the check.

## Details

In case of a fail all [file\\_checks](#) throw a condition of class `c("cleanr", "error", "condition")`.

## Value

Invisibly TRUE, but see *Details*.

## See Also

Other check functions: [function\\_checks](#)

## Examples

```
print(cleanr::check_file_width(system.file("source", "R", "checks.R",
                                           package = "cleanr")))
print(cleanr::check_file_length(system.file("source", "R", "checks.R",
                                           package = "cleanr"),
                                max_file_length = 300))
```

---

function_checks	<i>Function Checks</i>
-----------------	------------------------

---

## Description

A set of tiny functions to check that functions adhere to a layout style. A function should have a clear layout, it should

- not have too many arguments,
- not have too many levels of nesting,
- neither have too many lines nor
- have too many lines of code,
- not have lines too wide and
- explicitly [return](#) an object.

**Usage**

```

check_num_arguments(object, max_num_arguments = gco("max_num_arguments"))

check_nesting_depth(object, max_nesting_depth = gco("max_nesting_depth"))

check_num_lines(object, max_lines = gco("max_lines"))

check_num_lines_of_code(object, max_lines_of_code = gco("max_lines_of_code"))

check_line_width(object, max_line_width = gco("max_line_width"))

check_return(object, check_return = gco("check_return"))

```

**Arguments**

object	The function to be checked. Should have been sourced with <code>keep.source = TRUE</code> (see <a href="#">get_function_body</a> ).
max_num_arguments	The maximum number of arguments accepted. Set (preferably via <a href="#">set_cleanr_options</a> ) to NULL or FALSE to disable the check.
max_nesting_depth	The maximum nesting depth accepted. Set (preferably via <a href="#">set_cleanr_options</a> ) to NULL or FALSE to disable the check.
max_lines	The maximum number of lines accepted. Set (preferably via <a href="#">set_cleanr_options</a> ) to NULL or FALSE to disable the check.
max_lines_of_code	The maximum number of lines of code accepted. Set (preferably via <a href="#">set_cleanr_options</a> ) to NULL or FALSE to disable the check.
max_line_width	The maximum line width accepted. Set (preferably via <a href="#">set_cleanr_options</a> ) to NULL or FALSE to disable the check.
check_return	Set (preferably via <a href="#">set_cleanr_options</a> ) to NULL or FALSE to disable the check.

**Details**

In case of a fail all [function\\_checks](#) throw a condition of class `c("cleanr", "error", "condition")`.

**Value**

Invisibly TRUE, but see *Details*.

**Warning**

[check\\_return](#) just `greps` for a for a line starting with a `return` statement (ah, see the code for the real thing). This does not ensure that *all* `return` paths from the function are explicit and it may miss a `return` path after a semicolon. It just checks if you use `return` at all.

**See Also**

Other check functions: [file\\_checks](#)

## Examples

```
print(cleanr::check_num_arguments(cleanr::check_num_arguments))
print(cleanr::check_nesting_depth(cleanr::check_nesting_depth))
print(cleanr::check_num_lines(cleanr::check_num_lines))
print(cleanr::check_num_lines_of_code(cleanr::check_num_lines_of_code))
print(cleanr::check_return(cleanr::check_return))
# R reformats functions on import (see
# help(get_function_body, package = "cleanr")), so we need 90 characters:
print(cleanr::check_line_width(cleanr::check_line_width,
                               max_line_width = 90))
```

---

get\_cleanr\_options      *Get options for **cleanr***

---

## Description

A convenience function for [getOption](#).

## Usage

```
get_cleanr_options(..., remove_names = FALSE, flatten_list = TRUE)
```

```
gco(..., remove_names = FALSE, flatten_list = TRUE)
```

## Arguments

...	See <a href="#">getOption</a> .
remove_names	[boolean(1)] Remove the names?
flatten_list	[boolean(1)] Return a vector?

## Value

a (possibly named) list or a vector.

## Note

gco is just an alias for get\_cleanr\_options.

## See Also

Other option wrappers: [set\\_cleanr\\_options\(\)](#)

Other option wrappers: [set\\_cleanr\\_options\(\)](#)

## Examples

```
cleanr::get_cleanr_options("max_lines")
cleanr::get_cleanr_options("max_lines", remove_names = TRUE)
cleanr::get_cleanr_options("max_lines", flatten_list = TRUE)
cleanr::get_cleanr_options("max_lines", flatten_list = TRUE,
                           remove_names = TRUE)
cleanr::get_cleanr_options(flatten_list = TRUE, remove_names = TRUE)
cleanr::get_cleanr_options(c("max_lines", "max_lines_of_code"))
```

---

set\_cleanr\_options      *Set Options for **cleanr***

---

## Description

A convenience function for [options](#).

## Usage

```
set_cleanr_options(..., reset = FALSE, overwrite = TRUE)
```

## Arguments

...	See <a href="#">options</a> . See <a href="#">function_checks</a> and <a href="#">file_checks</a> for options to be set with <b>cleanr</b> . Use <a href="#">get_cleanr_options</a> to get the current values. <b>cleanr</b> 's standard defaults are: max_file_width=80 max_file_length=300 max_lines=65 max_lines_of_code=50 max_num_arguments=5 max_nesting_depth=3 max_line_width=80 check_return=TRUE
reset	[boolean(1)] Reset all <b>cleanr</b> options to the package's defaults?
overwrite	[boolean(1)] Overwrite options already set? Is set to FALSE on package loading to ensure your previously set <b>cleanr</b> options won't get overridden. Just ignore that argument.

## Details

**cleanr** loads a couple of options as defaults for its functions. The defaults are stored in a list element of [options](#). All checks (see [function\\_checks](#) and [file\\_checks](#)) can be disabled by setting the corresponding option list item to NULL or FALSE.

## Value

Invisibly TRUE.

**See Also**

Other option wrappers: [get\\_cleanr\\_options\(\)](#)

**Examples**

```
# R.C. Martin's Clean Code recommends monadic argument lists.
cleanr::set_cleanr_options(max_num_arguments = 1)
# R.C. Martin's Clean Code recommends functions less than 20 lines long.
cleanr::set_cleanr_options(max_lines = 30, max_lines_of_code = 20)
# same as above:
cleanr::set_cleanr_options(list(max_lines = 30, max_lines_of_code = 20))
cleanr::get_cleanr_options(flatten_list = TRUE)
# we delete all options and set some anew
options("cleanr" = NULL)
options("cleanr" = list(max_lines = 30, max_lines_of_code = 20))
# fill the missing options with the package's defaults:
cleanr::set_cleanr_options(overwrite = FALSE)
cleanr::get_cleanr_options(flatten_list = TRUE)
# reset to the package's defaults:
cleanr::set_cleanr_options(reset = TRUE)
cleanr::get_cleanr_options(flatten_list = TRUE)
```

# Index

- \* **check functions**
    - file\_checks, 8
    - function\_checks, 9
  - \* **option wrappers**
    - get\_cleanr\_options, 11
    - set\_cleanr\_options, 12
  - \* **package**
    - cleanr-package, 2
  - \* **wrappers**
    - check\_directory, 3
    - check\_file, 4
    - check\_file\_layout, 4
    - check\_function\_layout, 6
    - check\_functions\_in\_file, 5
    - check\_package, 7
- check\_directory, 3, 4–8
- check\_file, 3, 4, 5–8
- check\_file\_layout, 3, 4, 4, 6–8
- check\_file\_length, 5
- check\_file\_length (file\_checks), 8
- check\_file\_width, 5
- check\_file\_width (file\_checks), 8
- check\_function\_layout, 3–6, 6, 8
- check\_functions\_in\_file, 3–5, 5, 7, 8
- check\_line\_width, 7
- check\_line\_width (function\_checks), 9
- check\_nesting\_depth, 7
- check\_nesting\_depth (function\_checks), 9
- check\_num\_arguments, 7
- check\_num\_arguments (function\_checks), 9
- check\_num\_lines, 7
- check\_num\_lines (function\_checks), 9
- check\_num\_lines\_of\_code, 7
- check\_num\_lines\_of\_code  
(function\_checks), 9
- check\_package, 3–7, 7
- check\_return, 7, 10
- check\_return (function\_checks), 9
- cleanr-package, 2
- file\_checks, 4, 5, 8, 9, 10, 12
- function\_checks, 6, 7, 9, 9, 10, 12
- gco (get\_cleanr\_options), 11
- get, 7
- get\_cleanr\_options, 11, 12, 13
- get\_function\_body, 10
- getOption, 11
- grep, 10
- Invisibly, 3–10, 12
- list.files, 3, 8
- options, 12
- return, 9, 10
- set\_cleanr\_options, 9–11, 12
- throw, 3–10
- TRUE, 3–10, 12