Package 'lintr'

October 1, 2019

Title A 'Linter' for R Code

Version 2.0.0

URL https://github.com/jimhester/lintr

BugReports https://github.com/jimhester/lintr/issues

Description Checks adherence to a given style, syntax errors and possible semantic issues. Supports on the fly checking of R code edited with 'RStudio IDE', 'Emacs', 'Vim', 'Sublime Text' and 'Atom'.

Depends R (>= 3.2)

Imports rex,
crayon,
codetools,
cyclocomp,
stringdist,
testthat,
digest,
rstudioapi (>= 0.2),
httr (>= 1.2.1),
jsonlite,
knitr,
stats,
utils,
xml2 (>= 1.0.0),
xmlparsedata (>= 1.0.3)

Suggests rmarkdown,
mockery

License MIT + file LICENSE

LazyData true

Encoding UTF-8

VignetteBuilder knitr

RoxygenNote 6.1.1

Collate 'T_and_F_symbol_linter.R'
'utils.R'
'aaa.R'
'addins.R'
'assignment_linter.R'

'cache.R'
'closed_curly_linter.R'
'commas_linter.R'
'comment_linters.R'
'comments.R'
'cyclocomp_linter.R'
'declared_functions.R'
'object_name_linters.R'
'deprecated.R'
'equals_na_lintr.R'
'exclude.R'
'expect_lint.R'
'extract.R'
'extraction_operator_linter.R'
'function_left_parentheses.R'
'get_source_expressions.R'
'ids_with_token.R'
'implicit_integer_linter.R'
'infix_spaces_linter.R'
'line_length_linter.R'
'lint.R'
'methods.R'
'namespace.R'
'no_tab_linter.R'
'object_usage_linter.R'
'open_curly_linter.R'
'paren_brace_linter.R'
'path_linters.R'
'pipe_continuation_linter.R'
'semicolon_terminator_linter.R'
'seq_linter.R'
'settings.R'
'single_quotes_linter.R'
'spaces_inside_linter.R'
'spaces_left_parentheses_linter.R'
'trailing_blank_lines_linter.R'
'trailing_whitespace_linter.R'
'tree-utils.R'
'undesirable_function_linter.R'
'undesirable_operator_linter.R'
'unneeded_concatenation_linter.R'
'with_id.R'
'zzz.R'

R topics documented:

`all_undesirable_functions` *Default undesirable functions and operators*

Description

Lists of function names and operators for `undesirable_function_linter` and `undesirable_operator_linter`. There is a list for the default elements and another that contains all available elements. Use `with_defaults` to produce a custom list.

Usage

```
all_undesirable_functions

default_undesirable_functions

all_undesirable_operators

default_undesirable_operators
```

Format

A named list of character strings.

checkstyle_output *Checkstyle Report for lint results*

Description

Generate a report of the linting results using the Checkstyle XML format.

Usage

```
checkstyle_output(lints, filename = "lintr_results.xml")
```

Arguments

lints	the linting results.
filename	the name of the output report

clear_cache *Clear the lintr cache*

Description

Clear the lintr cache

Usage

```
clear_cache(file = NULL, path = NULL)
```

Arguments

file	filename whose cache to clear. If you pass NULL, it will delete all of the caches.
path	directory to store caches. Reads option 'lintr.cache_directory' as the default.

Value

0 for success, 1 for failure, invisibly.

default_linters *Default linters*

Description

List of default linters for lint. Use with_defaults to customize it.

Usage

```
default_linters
```

Format

An object of class list of length 22.

default_settings *Default lintr settings*

Description

Default lintr settings

Usage

```
default_settings
```

Format

An object of class `list` of length 9.

See Also

`read_settings`, `default_linters`

exclude *Exclude lines or files from linting*

Description

Exclude lines or files from linting

Usage

```
exclude(lints, exclusions = settings$exclusions, ...)
```

Arguments

`lints`	that need to be filtered.
`exclusions`	manually specified exclusions
`...`	additional arguments passed to `parse_exclusions`

Details

Exclusions can be specified in three different ways.

1. single line in the source file. default: `# nolint`
2. line range in the source file. default: `# nolint start`, `# nolint end`
3. exclusions parameter, a named list of the files and lines to exclude, or just the filenames if you want to exclude the entire file.

`expect_lint` *Lint expectation*

Description

This is an expectation function to test that the lints produced by `lint` satisfy a number of checks.

Usage

```
expect_lint(content, checks, ..., file = NULL)
```

Arguments

`content`	a character vector for the file content to be linted, each vector element representing a line of text.
`checks`	checks to be performed: **NULL** check that no lints are returned. **single string or regex object** check that the single lint returned has a matching message. **named list** check that the single lint returned has fields that match. Accepted fields are the same as those taken by `Lint`. **list of named lists** for each of the multiple lints returned, check that it matches the checks in the corresponding named list (as described in the point above). Named vectors are also accepted instead of named lists, but this is a compatibility feature that is not recommended for new code.
`...`	arguments passed to `lint`, e.g. the linters or cache to use.
`file`	if not `NULL`, read content from the specified file rather than from `content`.

Value

`NULL`, invisibly.

Examples

```
# no expected lint
expect_lint("a", NULL, trailing_blank_lines_linter)

# one expected lint
expect_lint("a\n", "superfluous", trailing_blank_lines_linter)
expect_lint("a\n", list(message="superfluous", line_number=2), trailing_blank_lines_linter)

# several expected lints
expect_lint("a\n\n", list("superfluous", "superfluous"), trailing_blank_lines_linter)
expect_lint(
  "a\n\n",
  list(list(message="superfluous", line_number=2), list(message="superfluous", line_number=3)),
  trailing_blank_lines_linter)
```

expect_lint_free *Test that the package is lint free*

Description

This function is a thin wrapper around lint_package that simply tests there are no lints in the package. It can be used to ensure that your tests fail if the package contains lints.

Usage

```
expect_lint_free(...)
```

Arguments

`...`	arguments passed to `lint_package`

get_source_expressions *Parsed sourced file from a filename*

Description

This object is given as input to each linter

Usage

```
get_source_expressions(filename)
```

Arguments

`filename`	the file to be parsed.

ids_with_token *Get parsed IDs by token*

Description

Gets the source IDs (row indices) corresponding to given token.

Usage

```
ids_with_token(source_file, value, fun = `==`)

with_id(source_file, id)
```

Arguments

source_file	A list of source expressions, the result of a call to ‘get_source_expressions()‘, for the desired filename.
value	Character. String correspondin to the token to search for. For example: • "SYMBOL" • "FUNCTION" • "EQ_FORMALS" • "$" • "("
fun	For additionaly flexibility, a function to search for in the ‘token‘ column of ‘parsed_content‘. Typically ‘==‘ or ‘%in%‘.
id	Integer. The index corresponding to the desired row of ‘parsed_content‘.

Value

‘ids_with_token‘: The indices of the ‘parsed_content‘ data frame entry of the list of source expressions. Indices correspond to the *rows* where ‘fun‘ evaluates to ‘TRUE‘ for the ‘value‘ in the *token* column.

‘with_id‘: A data frame corresponding to the row(s) specified in ‘id‘.

Functions

- with_id: Return the row of the ‘parsed_content‘ entry of the ‘get_source_expressions()‘ object. Typically used in conjunction with ‘ids_with_token‘ to iterate over rows containing desired tokens.

Lint *Create a* Lint *object*

Description

Create a Lint object

Usage

```
Lint(filename, line_number = 1L, column_number = 1L,
  type = c("style", "warning", "error"), message = "", line = "",
  ranges = NULL, linter = "")
```

Arguments

filename	path to the source file that was linted.
line_number	line number where the lint occurred.
column_number	column number where the lint occurred.
type	type of lint.
message	message used to describe the lint error
line	code source where the lint occurred
ranges	a list of ranges on the line that should be emphasized.
linter	name of linter that created the Lint object.

lintr *Lintr*

Description

Checks adherence to a given style, syntax errors and possible semantic issues. Supports on the fly checking of R code edited with Emacs, Vim and Sublime Text.

See Also

lint, lint_package, lint_dir, linters

lintr-deprecated *Deprecated functions*

Description

Functions that have been deprecated and replaced by newer ones. They will be removed in an upcoming version of **lintr** and should thus not be used anymore.

Usage

```
absolute_paths_linter(source_file)

trailing_semicolons_linter(source_file)

snake_case_linter(source_file)

multiple_dots_linter(source_file)
```

Arguments

source_file	returned by get_source_expressions

Functions

- absolute_paths_linter: checks that no absolute paths are used.
- trailing_semicolons_linter: check there are no trailing semicolons.
- snake_case_linter: check that objects are not in snake_case.
- multiple_dots_linter: check that objects do not have.multiple.dots.

lint_dir *Lint a directory*

Description

Apply one or more linters to all of the R files in a directory

Usage

```
lint_dir(path = ".", relative_path = TRUE, ..., exclusions = NULL,
  pattern = rex::rex(".", one_of("Rr"), end), parse_settings = TRUE)
```

Arguments

path	the path to the base directory, by default, it will be searched in the parent directories of the current directory.
relative_path	if TRUE, file paths are printed using their path relative to the base directory. If FALSE, use the full absolute path.
...	additional arguments passed to lint, e.g. cache or linters.
exclusions	exclusions for exclude, relative to the package path.
pattern	pattern for files, by default it will take files with .R or .r extension.
parse_settings	whether to try and parse the settings

Value

A list of lint objects.

Examples

```
## Not run:
  lint_dir()
  lint_dir(
    linters = list(semicolon_terminator_linter())
    cache = TRUE,
    exclusions = list("inst/doc/creating_linters.R" = 1, "inst/example/bad.R")
  )

## End(Not run)
```

lint_file *Lint a file*

Description

Apply one or more linters to a file and return the lints found.

Usage

```
lint(filename, linters = NULL, cache = FALSE, ...,
  parse_settings = TRUE)
```

Arguments

filename	the given filename to lint.
linters	a named list of linter functions to apply see linters for a full list of default and available linters.
cache	given a logical, toggle caching of lint results. If passed a character string, store the cache in this directory.
...	additional arguments passed to exclude.
parse_settings	whether to try and parse the settings

Value

A list of lint objects.

---

lint_package *Lint a package*

Description

Apply one or more linters to all of the R files in a package.

Usage

```
lint_package(path = ".", relative_path = TRUE, ...,
  exclusions = list("R/RppExports.R"))
```

Arguments

path	the path to the base directory of the package, if NULL, it will be searched in the parent directories of the current directory.
relative_path	if TRUE, file paths are printed using their path relative to the base directory. If FALSE, use the full absolute path.
...	additional arguments passed to lint, e.g. cache or linters.
exclusions	exclusions for exclude, relative to the package path.

Value

A list of lint objects.

Examples

```
## Not run:
  lint_package()

  lint_package(
    linters = with_defaults(semicolon_linter = semicolon_terminator_linter())
    cache = TRUE,
    exclusions = list("inst/doc/creating_linters.R" = 1, "inst/example/bad.R")
  )

## End(Not run)
```

parse_exclusions *read a source file and parse all the excluded lines from it*

Description

read a source file and parse all the excluded lines from it

Usage

```
parse_exclusions(file, exclude = settings$exclude,
  exclude_start = settings$exclude_start,
  exclude_end = settings$exclude_end)
```

Arguments

file	R source file
exclude	regular expression used to mark lines to exclude
exclude_start	regular expression used to mark the start of an excluded range
exclude_end	regular expression used to mark the end of an excluded range

read_settings *Read lintr settings*

Description

Lintr searches for settings for a given source file in the following order.

1. options defined as `linter.setting`.
2. `linter_file` in the same directory
3. `linter_file` in the project directory
4. `linter_file` in the user home directory
5. `default_settings`

Usage

```
read_settings(filename)
```

Arguments

filename	source file to be linted

Details

The default linter_file name is `.lintr` but it can be changed with option `lintr.linter_file`. This file is a dcf file, see `read.dcf` for details.

T_and_F_symbol_linter *linters*

Description

Available linters

Usage

```
T_and_F_symbol_linter(source_file)

assignment_linter(source_file)

closed_curly_linter(allow_single_line = FALSE)

commas_linter(source_file)

commented_code_linter(source_file)

todo_comment_linter(todo = c("todo", "fixme"))

cyclocomp_linter(complexity_limit = 25)

object_name_linter(styles = "snake_case")

object_length_linter(length = 30L)

camel_case_linter(source_file)

equals_na_linter(source_file)

extraction_operator_linter(source_file)

function_left_parentheses_linter(source_file)

implicit_integer_linter(source_file)

infix_spaces_linter(source_file)

line_length_linter(length)

no_tab_linter(source_file)

object_usage_linter(source_file)

open_curly_linter(allow_single_line = FALSE)

paren_brace_linter(source_file)

absolute_path_linter(lax = TRUE)
```

```
nonportable_path_linter(lax = TRUE)

pipe_continuation_linter(source_file)

semicolon_terminator_linter(semicolon = c("compound", "trailing"))

seq_linter(source_file)

single_quotes_linter(source_file)

spaces_inside_linter(source_file)

spaces_left_parentheses_linter(source_file)

trailing_blank_lines_linter(source_file)

trailing_whitespace_linter(source_file)

undesirable_function_linter(fun = default_undesirable_functions)

undesirable_operator_linter(op = default_undesirable_operators)

unneeded_concatenation_linter(source_file)
```

Arguments

source_file	returned by get_source_expressions
allow_single_line	if true allow a open and closed curly pair on the same line.
todo	Vector of strings that identify TODO comments.
complexity_limit	expressions with a cyclomatic complexity higher than this are linted, defaults to 25. See cyclocomp.
styles	A subset of 'CamelCase', 'camelCase', 'snake_case', 'dotted.case', 'lowercase', 'UPPERCASE'. A name should match at least one of these styles.
length	the length cutoff to use for the given linter.
lax	Less stringent linting, leading to fewer false positives.
semicolon	A character vector defining which semicolons to report: **compound** Semicolons that separate two statements on the same line. **trailing** Semicolons following the last statement on the line.
fun	Named character vector, where the names are the names of the undesirable functions, and the values are the text for the alternative function to use (or NA).
op	Named character vector, where the names are the names of the undesirable operators, and the values are the text for the alternative operator to use (or NA).

Functions

- T_and_F_symbol_linter: Avoid the symbols T and F (for TRUE and FALSE).
- assignment_linter: checks that '<-' is always used for assignment

- `closed_curly_linter`: check that closed curly braces should always be on their own line unless they follow an else.
- `commas_linter`: check that all commas are followed by spaces, but do not have spaces before them.
- `commented_code_linter`: Check that there is no commented code outside roxygen blocks
- `todo_comment_linter`: Check that the source contains no TODO comments (case-insensitive).
- `cyclocomp_linter`: Check for overly complicated expressions. See `cyclocomp`.
- `object_name_linter`: Check that object names conform to a naming style.
- `object_length_linter`: check that object names are not too long.
- `camel_case_linter`: check that objects are not in camelCase.
- `equals_na_linter`: that checks for x == NA
- `extraction_operator_linter`: Check that the '[[' operator is used when extracting a single element from an object, not '[' (subsetting) nor '$' (interactive use).
- `function_left_parentheses_linter`: check that all left parentheses in a function call do not have spaces before them.
- `implicit_integer_linter`: Check that integers are explicitly typed using the form `1L` instead of `1`.
- `infix_spaces_linter`: check that all infix operators have spaces around them.
- `line_length_linter`: check the line length of both comments and code is less than length.
- `no_tab_linter`: check that only spaces are used for indentation, not tabs.
- `object_usage_linter`: checks that closures have the proper usage using `checkUsage`. Note this runs `eval` on the code, so do not use with untrusted code.
- `open_curly_linter`: check that opening curly braces are never on their own line and are always followed by a newline.
- `paren_brace_linter`: check that there is a space between right parenthesis and an opening curly brace.
- `absolute_path_linter`: Check that no absolute paths are used (e.g. "/var", "C:\System", "~/docs").
- `nonportable_path_linter`: Check that file.path() is used to construct safe and portable paths.
- `pipe_continuation_linter`: Check that each step in a pipeline is on a new line, or the entire pipe fits on one line.
- `semicolon_terminator_linter`: Check that no semicolons terminate statements.
- `seq_linter`: check for `1:length(...)`, `1:nrow(...)`, `1:ncol(...)`, `1:NROW(...)` and `1:NCOL(...)` expressions. These often cause bugs when the right hand side is zero. It is safer to use `seq_len` or `seq_along` instead.
- `single_quotes_linter`: checks that only single quotes are used to delimit string constants.
- `spaces_inside_linter`: check that parentheses and square brackets do not have spaces directly inside them.
- `spaces_left_parentheses_linter`: check that all left parentheses have a space before them unless they are in a function call.
- `trailing_blank_lines_linter`: check there are no trailing blank lines.
- `trailing_whitespace_linter`: check there are no trailing whitespace characters.

- undesirable_function_linter: Report the use of undesirable functions, e.g. return, options, or sapply and suggest an alternative.
- undesirable_operator_linter: Report the use of undesirable operators, e.g. `:::` or `<<-` and suggest an alternative.
- unneeded_concatenation_linter: Check that the c function is not used without arguments nor with a single constant.

`with_defaults` *Modify lintr defaults*

Description

Make a new list based on **lintr**'s default linters, undesirable operators or functions. The result of this function is meant to be passed to the 'linters' argument of 'lint()', or put in your configuration file.

Usage

```
with_defaults(..., default = default_linters)
```

Arguments

...	arguments of elements to change. If unnamed, the argument is named. If the named argument already exists in "default", it is replaced by the new element. If it does not exist, it is added. If the value is NULL, the element is removed.
default	list of elements to modify.

Value

A modified list of elements.

Examples

```
# When using interatively you will usuaully pass the result onto `lint` or `lint_package()`
## Not run:
lint("foo.R", linters = with_defaults(line_length_linter = line_length_linter(120)))

## End(Not run)
# the default linter list with a different line length cutoff
my_linters <- with_defaults(line_length_linter = line_length_linter(120))

# omit the argument name if you are just using different arguments
my_linters <- with_defaults(default = my_linters,
                            object_name_linter("camelCase"))

# remove assignment checks (with NULL), add absolute path checks
my_linters <- with_defaults(default = my_linters,
                            assignment_linter = NULL,
                            absolute_path_linter)

# custom list of undesirable functions:
#    remove sapply (using NULL)
```

```
#    add cat (with a accompanying message),
#    add print (unnamed, i.e. with no accompanying message)
#    add return (as taken from all_undesirable_functions)
my_undesirable_functions <- with_defaults(default = default_undesirable_functions,
  sapply=NULL, "cat"="No cat allowed", "print", all_undesirable_functions[["return"]])
```

Index