Package ‘styler’

January 15, 2023

Type Package

Title Non-Invasive Pretty Printing of R Code

Version 1.9.0

Description Pretty-prints R code without changing the user's formatting intent.

License MIT + file LICENSE

URL `https://github.com/r-lib/styler`, `https://styler.r-lib.org`

BugReports `https://github.com/r-lib/styler/issues`

Depends R (>= 3.5.0)

Imports cli (>= 3.1.1), magrittr (>= 2.0.0), purrr (>= 0.2.3), R.cache (>= 0.15.0), rlang (>= 0.1.1), rprojroot (>= 1.1), tools, vctrs (>= 0.4.1), withr (>= 2.3.0),

Suggests data.tree (>= 0.1.6), digest, dplyr, here, knitr, prettycode, rmarkdown, roxygen2, rstudioapi (>= 0.7), tibble (>= 1.4.2), testthat (>= 3.0.0)

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.2.3

Config/testthat/edition 3

Config/testthat/parallel true

Collate 'addins.R' 'communicate.R' 'compat-dplyr.R' 'compat-tidyr.R' 'detect-alignment-utils.R' 'detect-alignment.R' 'environments.R' 'expr-is.R' 'indent.R' 'initialize.R' 'io.R' 'nest.R' 'nested-to-tree.R' 'parse.R' 'reindent.R' 'token-define.R' 'relevel.R' 'roxygen-examples-add-remove.R' 'roxygen-examples-find.R' 'roxygen-examples-parse.R' 'roxygen-examples.R' 'rules-indention.R' 'rules-line-breaks.R' 'rules-spaces.R' 'rules-tokens.R' 'serialize.R' 'set-assert-args.R' 'style-guides.R' 'styler-package.R' 'stylerignore.R' 'testing-mocks.R' 'testing-public-api.R' 'ui-caching.R' 'testing.R' 'token-create.R' 'transform-block.R'

'transform-code.R' 'transform-files.R' 'ui-styling.R'
'unindent.R' 'utils-cache.R' 'utils-files.R'
'utils-navigate-nest.R' 'utils-strings.R' 'utils.R'
'vertical.R' 'visit.R' 'zzz.R'

NeedsCompilation no

Author Kirill Müller [aut] (<https://orcid.org/0000-0002-1416-3412>),
Lorenz Walthert [cre, aut],
Indrajeet Patil [ctb] (<https://orcid.org/0000-0003-1995-6531>,
@patilindrajeets)

Maintainer Lorenz Walthert <lorenz.walthert@icloud.com>

Repository CRAN

Date/Publication 2023-01-15 13:10:02 UTC

R topics documented:

styler-package *Non-invasive pretty printing of R code*

Description

styler allows you to format .R, .Rmd, .Rmarkdown and/or .qmd, .Rnw files, R packages, or entire R source trees according to a style guide. The following functions can be used for styling:

- style_text() to style a character vector.
- style_file() to style a single file.
- style_dir() to style all files in a directory.
- style_pkg() to style the source files of an R package.
- styler_addins (RStudio Addins) to style either selected code or the active file.

Author(s)

Maintainer: Lorenz Walthert <lorenz.walthert@icloud.com>

Authors:

- Kirill Müller <kirill@cynkra.com> (ORCID)

Other contributors:

- Indrajeet Patil <patilindrajeet.science@gmail.com> (ORCID) (@patilindrajeets) [contributor]

See Also

Useful links:

- https://github.com/r-lib/styler
- https://styler.r-lib.org
- Report bugs at https://github.com/r-lib/styler/issues

Examples

```
style_text("call( 1)")
style_text("1    + 1", strict = FALSE)
style_text("a%>%b", scope = "spaces")
style_text("a%>%b; a", scope = "line_breaks")
style_text("a%>%b; a", scope = "tokens")
```

`cache_activate` *Activate or deactivate the styler cache*

Description

Helper functions to control the behavior of caching. Simple wrappers around `base::options()`.

Usage

```
cache_activate(cache_name = NULL, verbose = !getOption("styler.quiet", FALSE))

cache_deactivate(verbose = !getOption("styler.quiet", FALSE))
```

Arguments

`cache_name`	The name of the styler cache to use. If `NULL`, the option "styler.cache_name" is considered which defaults to the version of styler used.
`verbose`	Whether or not to print an informative message about what the function is doing.

See Also

Other cache managers: `cache_clear()`, `cache_info()`, `caching`

`cache_clear` *Clear the cache*

Description

Clears the cache that stores which files are already styled. You won't be able to undo this. Note that the file corresponding to the cache (a folder on your file system) won't be deleted, but it will be empty after calling `cache_clear`.

Usage

```
cache_clear(cache_name = NULL, ask = TRUE)
```

Arguments

`cache_name`	The name of the styler cache to use. If `NULL`, the option "styler.cache_name" is considered which defaults to the version of styler used.
`ask`	Whether or not to interactively ask the user again.

Details

Each version of styler has its own cache by default, because styling is potentially different with different versions of styler.

See Also

Other cache managers: `cache_activate()`, `cache_info()`, `caching`

`cache_info` *Show information about the styler cache*

Description

Gives information about the cache. Note that the size consumed by the cache will always be displayed as zero because all the cache does is creating an empty file of size 0 bytes for every cached expression. The inode is excluded from this displayed size but negligible.

Usage

```
cache_info(cache_name = NULL, format = "both")
```

Arguments

`cache_name`	The name of the cache for which to show details. If `NULL`, the active cache is used. If none is active the cache corresponding to the installed styler version is used.
`format`	Either "lucid" for a summary emitted with `base::cat()`, "tabular" for a tabular summary from `base::file.info()` or "both" for both.

See Also

Other cache managers: `cache_activate()`, `cache_clear()`, `caching`

`caching` *Remember the past to be quicker in the future*

Description

Caching makes styler faster on repeated styling and is shared across all APIs (e.g. `style_text()` and Addin). That means if you style code that already complies to a style guide and you have previously styled that code, it will be quicker.

Configuring the cache

To comply with the CRAN policy, {styler} will by default clean up cache files that are older than 6 days. This implies that you loose the benefit of the cache for the files not styled in the last 6 days.

If you want to avoid this, i.e., if you want the cache to last longer, you can use the R option `styler.cache_root` to opt for an indefinitely long-lived cache by setting it to `options(styler.cache_root = "styler-perm")`.

If you are happy with the cache being cleared after 6 days, you can confirm the default and silence this message by setting it instead to `options(styler.cache_root = "styler")`.

You can make this change in your `.Rprofile` using `usethis::edit_r_profile()`.

Manage the cache

See `cache_info()`,`cache_activate()` or `cache_clear()` for utilities to manage the cache. You can deactivate it altogether with `cache_deactivate()`. Since we leverage `{R.cache}` to manage the cache, you can also use any `{R.cache}` functionality to manipulate it.

In some cases, you want to use a non-standard cache location. In that situation, you can set the path to the cache with the R option `R.cache.rootPath` or the environment variable `R_CACHE_ROOTPATH` to an existent path before you call the styler API.

Invalidation

The cache is specific to a version of styler by default, because different versions potentially format code differently. This means after upgrading styler or a style guide you use, the cache will be re-built.

Mechanism and size

The cache works by storing hashed output code as a whole and by expression, which is why it takes zero space on disk (the cache is a directory with empty files which have the hash of output code as name).

The cache literally takes zero space on your disk, only the inode, and you can always manually clean up with `cache_clear()` or just go to the directory where the cache lives (find it with `cache_info()`) and manually delete files.

Using a cache for styler in CI/CD

If you want to set up caching in a CI/CD pipeline, we suggest to set the `{R.cache}` root path to a directory for which you have the cache enabled. This can often be set in config files of CI/CD tools, e.g. see the Travis documentation on caching.

See Also

Other cache managers: `cache_activate()`, `cache_clear()`, `cache_info()`

`compute_parse_data_nested`

Obtain a nested parse table from a character vector

Description

Parses `text` to a flat parse table and subsequently changes its representation into a nested parse table with `nest_parse_data()`.

Usage

```
compute_parse_data_nested(
  text,
  transformers = tidyverse_style(),
  more_specs = NULL
)
```

Arguments

`text`	The text to parse.
`transformers`	Passed to `cache_make_key()` to generate a key.
`more_specs`	Passed to `cache_make_key()` to generate a key.

Value

A nested parse table. See `tokenize()` for details on the columns of the parse table.

Examples

```
code <- "
ab     <- 1L # some comment
abcdef <- 2L
"
writeLines(code)
compute_parse_data_nested(code)
```

`create_style_guide` *Create a style guide*

Description

This is a helper function to create a style guide, which is technically speaking a named list of groups of transformer functions where each transformer function corresponds to one styling rule. The output of this function can be used as an argument for `style` in top-level functions like `style_text()` and friends. Note that for caching to work properly, unquote all inputs to the transformer function if possible with rlang's `!!`, otherwise, they will be passed as references (generic variable names) instead of literals and `styler:::is_cached()` won't pick up changes. See how it's done in `tidyverse_style()` with `indent_by` and other arguments.

Usage

```
create_style_guide(
  initialize = default_style_guide_attributes,
  line_break = NULL,
  space = NULL,
  token = NULL,
  indention = NULL,
  use_raw_indention = FALSE,
  reindention = tidyverse_reindention(),
  style_guide_name = NULL,
  style_guide_version = NULL,
  more_specs_style_guide = NULL,
  transformers_drop = specify_transformers_drop(),
  indent_character = " "
)
```

Arguments

`initialize`	The bare name of a function that initializes various variables on each level of nesting.
`line_break`	A list of transformer functions that manipulate line_break information.
`space`	A list of transformer functions that manipulate spacing information.
`token`	A list of transformer functions that manipulate token text.
`indention`	A list of transformer functions that manipulate indention.
`use_raw_indention`	Boolean indicating whether or not the raw indention should be used.
`reindention`	A list of parameters for regex re-indention, most conveniently constructed using `specify_reindention()`.
`style_guide_name`	The name of the style guide. Used as a meta attribute inside the created style guide, for example for caching. By convention, this is the style guide qualified by the package namespace plus the location of the style guide, separated by `@`. For example, `"styler::tidyverse_style@https://github.com/r-lib"`.
`style_guide_version`	The version of the style guide. Used as a meta attribute inside the created style guide, for example for caching. This should correspond to the version of the R package that exports the style guide.
`more_specs_style_guide`	Named vector (coercible to character) with all arguments passed to the style guide and used for cache invalidation. You can easily capture them in your style guide function declaration with `as.list(environment())` (compare source code of `tidyverse_style()`).
`transformers_drop`	A list specifying under which conditions transformer functions can be dropped since they have no effect on the code to format, most easily constructed with `specify_transformers_drop()`. This is argument experimental and may change in future releases without prior notification. It was mainly introduced to improve speed. Listing transformers here that occur almost always in code does not make sense because the process of excluding them also takes some time.
`indent_character`	The character that is used for indention. We strongly advise for using spaces as indention characters.

Examples

```
set_line_break_before_curly_opening <- function(pd_flat) {
  op <- pd_flat$token %in% "'{'"
  pd_flat$lag_newlines[op] <- 1L
  pd_flat
}
set_line_break_before_curly_opening_style <- function() {
  create_style_guide(
    line_break = list(set_line_break_before_curly_opening),
    style_guide_name = "some-style-guide",
    style_guide_version = "some-version"
  )
}
style_text(
  "a <- function(x) { x }",
  style = set_line_break_before_curly_opening_style
)
```

`math_token_spacing` *Specify spacing around math tokens*

Description

Helper function to create the input for the argument `math_token_spacing` in `tidyverse_style()`.

Usage

```
specify_math_token_spacing(zero = "'^'", one = c("'+'", "'-'", "'*'", "'/'"))

tidyverse_math_token_spacing()
```

Arguments

`zero`	Character vector of tokens that should be surrounded with zero spaces.
`one`	Character vector with tokens that should be surrounded by at least one space (depending on `strict = TRUE` in the styling functions `style_text()` and friends). See 'Examples'.

Functions

- `specify_math_token_spacing()`: Allows to fully specify the math token spacing.
- `tidyverse_math_token_spacing()`: Simple forwarder to `specify_math_token_spacing` with spacing around math tokens according to the tidyverse style guide.

Examples

```
style_text(
  "1+1   -3",
  math_token_spacing = specify_math_token_spacing(zero = "'+'"),
  strict = FALSE
)
style_text(
  "1+1   -3",
  math_token_spacing = specify_math_token_spacing(zero = "'+'"),
  strict = TRUE
)
style_text(
  "1+1   -3",
  math_token_spacing = tidyverse_math_token_spacing(),
  strict = FALSE
)
style_text(
  "1+1   -3",
  math_token_spacing = tidyverse_math_token_spacing(),
  strict = TRUE
)
```

`next_non_comment` *Find the index of the next or previous non-comment in a parse table.*

Description

Find the index of the next or previous non-comment in a parse table.

Usage

```
next_non_comment(pd, pos)

previous_non_comment(pd, pos)
```

Arguments

pd	A parse table.
pos	The position of the token to start the search from.

See Also

Other third-party style guide helpers: `pd_is`, `scope_normalize()`

Examples

```
code <- "a <- # hi \n x %>% b()"
writeLines(code)
pd <- compute_parse_data_nested(code)
child <- pd$child[[1]]
previous_non_comment(child, 4L)
next_non_comment(child, 2L)
```

---

pd_is *What is a parse table representing?*

Description

Check whether a parse table corresponds to a certain expression.

Usage

```
is_curly_expr(pd)

is_for_expr(pd)

is_conditional_expr(pd)

is_while_expr(pd)

is_function_call(pd)

is_function_declaration(pd)

is_comment(pd)

is_tilde_expr(pd, tilde_pos = c(1L, 2L))

is_asymmetric_tilde_expr(pd)

is_symmetric_tilde_expr(pd)
```

Arguments

`pd`	A parse table.
`tilde_pos`	Integer vector indicating row-indices that should be checked for tilde. See 'Details'.

Details

A tilde is on the top row in the parse table if it is an asymmetric tilde expression (like `~column`), in the second row if it is a symmetric tilde expression (like `a~b`).

Functions

- `is_curly_expr()`: Checks whether pd contains an expression wrapped in curly brackets.
- `is_for_expr()`: Checks whether pd contains a `for` loop.
- `is_conditional_expr()`: Checks whether pd contains is a conditional expression.
- `is_while_expr()`: Checks whether pd contains a `while` loop.
- `is_function_call()`: Checks whether pd is a function call.
- `is_function_declaration()`: Checks whether pd is a function declaration.
- `is_comment()`: Checks for every token whether or not it is a comment.
- `is_tilde_expr()`: Checks whether pd contains a tilde.
- `is_asymmetric_tilde_expr()`: If pd contains a tilde, checks whether it is asymmetrical.
- `is_symmetric_tilde_expr()`: If pd contains a tilde, checks whether it is symmetrical.

See Also

Other third-party style guide helpers: `next_non_comment()`, `scope_normalize()`

Examples

```
code <- "if (TRUE) { 1 }"
pd <- compute_parse_data_nested(code)
is_curly_expr(pd)
child_of_child <- pd$child[[1]]$child[[5]]
is_curly_expr(child_of_child)

code <- "for (i in 1:5) print(1:i)"
pd <- compute_parse_data_nested(code)
is_for_expr(pd)
is_for_expr(pd$child[[1]])

code <- "if (TRUE) x <- 1 else x <- 0"
pd <- compute_parse_data_nested(code)
is_conditional_expr(pd)
is_conditional_expr(pd$child[[1]])

code <- "x <- list(1:3)"
pd <- compute_parse_data_nested(code)
is_function_call(pd)
child_of_child <- pd$child[[1]]$child[[3]]
is_function_call(child_of_child)

code <- "foo <- function() NULL"
pd <- compute_parse_data_nested(code)
is_function_declaration(pd)
child_of_child <- pd$child[[1]]$child[[3]]
is_function_declaration(child_of_child)

code <- "x <- 1 # TODO: check value"
pd <- compute_parse_data_nested(code)
```

```
is_comment(pd)

code <- "lm(wt ~ mpg, mtcars)"
pd <- compute_parse_data_nested(code)
is_tilde_expr(pd$child[[1]]$child[[3]])
is_symmetric_tilde_expr(pd$child[[1]]$child[[3]])
is_asymmetric_tilde_expr(pd$child[[1]]$child[[3]])
```

`print.vertical` *Print styled code*

Description

Print styled code

Usage

```
## S3 method for class 'vertical'
print(
  x,
  ...,
  colored = getOption("styler.colored_print.vertical"),
  style = prettycode::default_style()
)
```

Arguments

`x`	A character vector, one element corresponds to one line of code.
`...`	Not currently used.
`colored`	Whether or not the output should be colored with `prettycode::highlight()`.
`style`	Passed to `prettycode::highlight()`.

`reindention` *Specify what is re-indented how*

Description

This function returns a list that can be used as an input for the argument `reindention` of the function `tidyverse_style()`. It features sensible defaults, so the user can specify deviations from them conveniently without the need of setting all arguments explicitly.

Usage

```
specify_reindention(regex_pattern = NULL, indention = 0L, comments_only = TRUE)

tidyverse_reindention()
```

Arguments

`regex_pattern`	Character vector with regular expression patterns that are to be re-indented with spaces, `NULL` if no reindention needed.
`indention`	The indention tokens should have if they match `regex_pattern`.
`comments_only`	Whether the `regex_reindention_pattern` should only be matched against comments or against all tokens. Mainly added for performance.

Functions

- `specify_reindention()`: Allows to specify which tokens are reindented and how.
- `tidyverse_reindention()`: Simple forwarder to `specify_reindention` with reindention according to the tidyverse style guide.

Examples

```
style_text("a <- xyz", reindention = specify_reindention(
  regex_pattern = "xyz", indention = 4, comments_only = FALSE
))
style_text("a <- xyz", reindention = tidyverse_reindention())
```

`scope_normalize` *Convert the styling scope to its lower-level representation*

Description

If `scope` is of class `character` and of length one, the value of the argument and all less-invasive levels are included too (e.g. styling tokens includes styling spaces). If `scope` is of class `AsIs`, every level to be included has to be declared individually. See compare `tidyverse_style()` for the possible levels and their order.

Usage

```
scope_normalize(scope, name = substitute(scope))
```

Arguments

`scope`	A character vector of length one or a vector of class `AsIs`.
`name`	The name of the character vector to be displayed if the construction of the factor fails.

See Also

Other third-party style guide helpers: `next_non_comment()`, `pd_is`

Examples

```
scope_normalize(I("tokens"))
scope_normalize(I(c("indention", "tokens")))
```

specify_transformers_drop

Specify which tokens must be absent for a transformer to be dropped

Description

`{styler}` can remove transformer functions safely removed from the list of transformers to be applied on every *nest* with `transformers_drop()` if the tokens that trigger a manipulation of the parse data are absent in the text to style. `specify_transformers_drop()` helps you specify these conditions.

Usage

```
specify_transformers_drop(
  spaces = NULL,
  indention = NULL,
  line_breaks = NULL,
  tokens = NULL
)
```

Arguments

`spaces, indention, line_breaks, tokens`	Each a list (or `NULL`) where the name of each element is the concerning transformer, the value is an unnamed vector with tokens that match the rule. See 'Examples'.

Details

Note that the negative formulation (must be absent in order to be dropped) means that when you add a new rule and you forget to add a rule for when to drop it, it will not be dropped. If we required to specify the complement (which tokens must be present for the transformer to be kept), the transformer would be silently removed, which is less save.

Warning

It is the responsibility of the developer to ensure expected behavior, in particular that:

- the name of the supplied dropping criteria matches the name of the transformer function.
- the dropping criteria (name + token) reflects correctly under which circumstances the transformer does not have an impact on styling and can therefore be safely removed without affecting the styling outcome.

You can use the unexported function `test_transformers_drop()` for some checks.

Examples

```
dropping <- specify_transformers_drop(
  spaces = c(remove_space_after_excl = "'!'")
)
style_guide <- create_style_guide(
  space = list(remove_space_after_excl = styler:::remove_space_after_excl),
  transformers_drop = dropping
)
# transformers_drop() will remove the transformer when the code does not
# contain an exclamation mark
style_guide_with_some_transformers_dropped <- styler:::transformers_drop(
  "x <- 3;2", style_guide
)
setdiff(
  names(style_guide$space),
  names(style_guide_with_some_transformers_dropped)
)
# note that dropping all transformers of a scope means that this scope
# has an empty named list for this scope
style_guide_with_some_transformers_dropped$space
# this is not the same as if this scope was never specified.
tidyverse_style(scope = "none")$space
# Hence, styler should check for length 0 to decide if a scope is present or
# not, not via `is.null()` and we can use the `is.null()` check to see if
# this scope was initially required by the user.
```

---

stylerignore *Turn off styling for parts of the code*

Description

Using stylerignore markers, you can temporarily turn off styler. Beware that for `styler > 1.2.0`, some alignment is detected by styler, making stylerignore redundant. See a few illustrative examples below.

Details

Styling is on for all lines by default when you run styler.

- To mark the start of a sequence where you want to turn styling off, use `# styler: off`.
- To mark the end of this sequence, put `# styler: on` in your code. After that line, styler will again format your code.
- To ignore an inline statement (i.e. just one line), place `# styler: off` at the end of the line. To use something else as start and stop markers, set the R options `styler.ignore_start` and `styler.ignore_stop` using `options()`. For styler version > 1.6.2, the option supports character vectors longer than one and the marker are not exactly matched, but using a regular expression, which means you can have multiple marker on one line, e.g. `# nolint start styler: off`.

Examples

```
# as long as the order of the markers is correct, the lines are ignored.
style_text(
  "
  1+1
  # styler: off
  1+1
  # styler: on
  1+1
  "
)

# if there is a stop marker before a start marker, styler won't be able
# to figure out which lines you want to ignore and won't ignore anything,
# issuing a warning.
## Not run:
style_text(
  "
  1+1
  # styler: off
  1+1
  # styler: off
  1+1
  "
)

## End(Not run)
# some alignment of code is detected, so you don't need to use stylerignore
style_text(
  "call(
    xyz =  3,
    x   = 11
  )"
)
```

`styler_addins` *Stylers for RStudio Addins*

Description

Helper functions for styling via RStudio Addins.

Addins

- Set style: Select the style transformers to use. For flexibility, the user input is passed to the `transformers` argument, not the `style` argument, so entering `styler::tidyverse_style(scope = "spaces")` in the Addin is equivalent to `styler::style_text("1+1", scope = "spaces")` and `styler::style_text("1+1", transformers = styler::tidyverse_style(scope = "spaces"))` if the text to style is `1+1`. The style transformers are memorized within an R session via the R option `styler.addins_style_transformer` so if you want it to persist over sessions, set the option `styler.addins_style_transformer` in your `.Rprofile`.

- Style active file: Styles the active file, by default with `tidyverse_style()` or the value of the option `styler.addins_style_transformer` if specified.
- Style selection: Same as *Style active file*, but styles the highlighted code instead of the whole file.

Auto-Save Option

By default, both of the RStudio Addins will apply styling to the (selected) file contents without saving changes. Automatic saving can be enabled by setting the R option `styler.save_after_styling` to `TRUE`. Consider setting this in your `.Rprofile` file if you want to persist this setting across multiple sessions. Untitled files will always need to be saved manually after styling.

Life cycle

The way of specifying the style in the Addin as well as the auto-save option (see below) are experimental. We are currently considering letting the user specify the defaults for other style APIs like `style_text()`, either via R options, config files or other ways as well. See r-lib/styler#319 for the current status of this.

See Also

Other stylers: `style_dir()`, `style_file()`, `style_pkg()`, `style_text()`

Examples

```
## Not run:
# save after styling when using the Addin
options(styler.save_after_styling = TRUE)
# only style with scope = "spaces" when using the Addin
val <- "styler::tidyverse_style(scope = 'spaces')"
options(
  styler.addins_style_transformer = val
)

## End(Not run)
```

`style_dir` *Prettify arbitrary R code*

Description

Performs various substitutions in all `.R`, `.Rmd`, `.Rmarkdown`, `qmd` and/or `.Rnw` files in a directory (by default only `.R` files are styled - see `filetype` argument). Carefully examine the results after running this function!

Usage

```
style_dir(
  path = ".",
  ...,
  style = tidyverse_style,
  transformers = style(...),
  filetype = c("R", "Rprofile", "Rmd", "Rmarkdown", "Rnw", "Qmd"),
  recursive = TRUE,
  exclude_files = NULL,
  exclude_dirs = c("packrat", "renv"),
  include_roxygen_examples = TRUE,
  base_indention = 0L,
  dry = "off"
)
```

Arguments

`path`	Path to a directory with files to transform.
`...`	Arguments passed on to the `style` function, see `tidyverse_style()` for the default argument.
`style`	A function that creates a style guide to use, by default `tidyverse_style`. Not used further except to construct the argument `transformers`. See `style_guides()` for details.
`transformers`	A set of transformer functions. This argument is most conveniently constructed via the `style` argument and `...`. See 'Examples'.
`filetype`	Vector of file extensions indicating which file types should be styled. Case is ignored, and the `.` is optional, e.g. `c(".R", ".Rmd")`, or `c("r", "rmd")`. Supported values (after standardization) are: "r", "rprofile", "rmd", "rmarkdown", "rnw". Rmarkdown is treated as Rmd.
`recursive`	A logical value indicating whether or not files in sub directories of `path` should be styled as well.
`exclude_files`	Character vector with paths to files that should be excluded from styling.
`exclude_dirs`	Character vector with directories to exclude (recursively).
`include_roxygen_examples`	Whether or not to style code in roxygen examples.
`base_indention`	Integer scalar indicating by how many spaces the whole output text should be indented. Note that this is not the same as splitting by line and add a `base_indention` spaces before the code in the case multi-line strings are present. See 'Examples'.
`dry`	To indicate whether styler should run in *dry* mode, i.e. refrain from writing back to files .`"on"` and `"fail"` both don't write back, the latter returns an error if the input code is not identical to the result of styling. "off", the default, writes back if the input and output of styling are not identical.

Value

Invisibly returns a data frame that indicates for each file considered for styling whether or not it was actually changed (or would be changed when `dry` is not "off").

Warning

This function overwrites files (if styling results in a change of the code to be formatted and `dry = "off"`). It is strongly suggested to only style files that are under version control or to create a backup copy.

We suggest to first style with `scope < "tokens"` and inspect and commit changes, because these changes are guaranteed to leave the abstract syntax tree (AST) unchanged. See section 'Round trip validation' for details.

Then, we suggest to style with `scope = "tokens"` (if desired) and carefully inspect the changes to make sure the AST is not changed in an unexpected way that invalidates code.

Round trip validation

The following section describes when and how styling is guaranteed to yield correct code.

If tokens are not in the styling scope (as specified with the `scope` argument), no tokens are changed and the abstract syntax tree (AST) should not change. Hence, it is possible to validate the styling by comparing whether the parsed expression before and after styling have the same AST. This comparison omits roxygen code examples and comments. styler throws an error if the AST has changed through styling.

Note that if tokens are to be styled, such a comparison is not conducted because the AST might well change and such a change is intended. There is no way styler can validate styling, that is why we inform the user to carefully inspect the changes.

See section 'Warning' for a good strategy to apply styling safely.

See Also

Other stylers: `style_file()`, `style_pkg()`, `style_text()`, `styler_addins`

Examples

```
style_dir("path/to/dir", filetype = c("rmd", ".R"))

# the following is identical (because of ... and defaults)
# but the first is most convenient:
style_dir(strict = TRUE)
style_dir(style = tidyverse_style, strict = TRUE)
style_dir(transformers = tidyverse_style(strict = TRUE))
```

`style_file` *Style files with R source code*

Description

Performs various substitutions in the files specified. Carefully examine the results after running this function!

Usage

```
style_file(
  path,
  ...,
  style = tidyverse_style,
  transformers = style(...),
  include_roxygen_examples = TRUE,
  base_indention = 0L,
  dry = "off"
)
```

Arguments

`path`	A character vector with paths to files to style. Supported extensions: `.R`, `.Rmd`, `.Rmarkdown`, `.qmd` and `.Rnw`.
`...`	Arguments passed on to the `style` function, see `tidyverse_style()` for the default argument.
`style`	A function that creates a style guide to use, by default `tidyverse_style`. Not used further except to construct the argument `transformers`. See `style_guides()` for details.
`transformers`	A set of transformer functions. This argument is most conveniently constructed via the `style` argument and `...`. See 'Examples'.
`include_roxygen_examples`	Whether or not to style code in roxygen examples.
`base_indention`	Integer scalar indicating by how many spaces the whole output text should be indented. Note that this is not the same as splitting by line and add a `base_indention` spaces before the code in the case multi-line strings are present. See 'Examples'.
`dry`	To indicate whether styler should run in *dry* mode, i.e. refrain from writing back to files ."on" and "fail" both don't write back, the latter returns an error if the input code is not identical to the result of styling. "off", the default, writes back if the input and output of styling are not identical.

Encoding

UTF-8 encoding is assumed. Please convert your code to UTF-8 if necessary before applying styler.

Value

Invisibly returns a data frame that indicates for each file considered for styling whether or not it was actually changed (or would be changed when `dry` is not "off").

Warning

This function overwrites files (if styling results in a change of the code to be formatted and `dry = "off"`). It is strongly suggested to only style files that are under version control or to create a backup copy.

We suggest to first style with `scope < "tokens"` and inspect and commit changes, because these changes are guaranteed to leave the abstract syntax tree (AST) unchanged. See section 'Round trip validation' for details.

Then, we suggest to style with `scope = "tokens"` (if desired) and carefully inspect the changes to make sure the AST is not changed in an unexpected way that invalidates code.

Round trip validation

The following section describes when and how styling is guaranteed to yield correct code.

If tokens are not in the styling scope (as specified with the `scope` argument), no tokens are changed and the abstract syntax tree (AST) should not change. Hence, it is possible to validate the styling by comparing whether the parsed expression before and after styling have the same AST. This comparison omits roxygen code examples and comments. styler throws an error if the AST has changed through styling.

Note that if tokens are to be styled, such a comparison is not conducted because the AST might well change and such a change is intended. There is no way styler can validate styling, that is why we inform the user to carefully inspect the changes.

See section 'Warning' for a good strategy to apply styling safely.

See Also

Other stylers: `style_dir()`, `style_pkg()`, `style_text()`, `styler_addins`

Examples

```
file <- tempfile("styler", fileext = ".R")
writeLines("1++1", file)

# the following is identical (because of ... and defaults),
# but the first is most convenient:
style_file(file, strict = TRUE)
style_file(file, style = tidyverse_style, strict = TRUE)
style_file(file, transformers = tidyverse_style(strict = TRUE))

# only style indention and less invasive  levels (i.e. spaces)
style_file(file, scope = "indention", strict = TRUE)
# name levels explicitly to not style less invasive levels
style_file(file, scope = I(c("tokens", "spaces")), strict = TRUE)

readLines(file)
unlink(file)
```

`style_pkg` *Prettify R source code*

Description

Performs various substitutions in all `.R` files in a package (code and tests), `.Rmd`, `.Rmarkdown` and/or `.qmd`, `.Rnw` files (vignettes and readme). Carefully examine the results after running this function!

Usage

```
style_pkg(
  pkg = ".",
  ...,
  style = tidyverse_style,
  transformers = style(...),
  filetype = c("R", "Rprofile", "Rmd", "Rmarkdown", "Rnw", "Qmd"),
  exclude_files = c("R/RcppExports.R", "R/cpp11.R"),
  exclude_dirs = c("packrat", "renv"),
  include_roxygen_examples = TRUE,
  base_indention = 0L,
  dry = "off"
)
```

Arguments

`pkg`	Path to a (subdirectory of an) R package.
`...`	Arguments passed on to the `style` function, see `tidyverse_style()` for the default argument.
`style`	A function that creates a style guide to use, by default `tidyverse_style`. Not used further except to construct the argument `transformers`. See `style_guides()` for details.
`transformers`	A set of transformer functions. This argument is most conveniently constructed via the `style` argument and `...`. See 'Examples'.
`filetype`	Vector of file extensions indicating which file types should be styled. Case is ignored, and the `.` is optional, e.g. `c(".R", ".Rmd")`, or `c("r", "rmd")`. Supported values (after standardization) are: "r", "rprofile", "rmd", "rmarkdown", "rnw". Rmarkdown is treated as Rmd.
`exclude_files`	Character vector with paths to files that should be excluded from styling.
`exclude_dirs`	Character vector with directories to exclude (recursively). Note that the default values were set for consistency with `style_dir()` and as these directories are anyways not styled.
`include_roxygen_examples`	Whether or not to style code in roxygen examples.
`base_indention`	Integer scalar indicating by how many spaces the whole output text should be indented. Note that this is not the same as splitting by line and add a `base_indention` spaces before the code in the case multi-line strings are present. See 'Examples'.
`dry`	To indicate whether styler should run in *dry* mode, i.e. refrain from writing back to files .`"on"` and `"fail"` both don't write back, the latter returns an error if the input code is not identical to the result of styling. "off", the default, writes back if the input and output of styling are not identical.

Warning

This function overwrites files (if styling results in a change of the code to be formatted and `dry = "off"`). It is strongly suggested to only style files that are under version control or to create a backup copy.

We suggest to first style with `scope < "tokens"` and inspect and commit changes, because these changes are guaranteed to leave the abstract syntax tree (AST) unchanged. See section 'Round trip validation' for details.

Then, we suggest to style with `scope = "tokens"` (if desired) and carefully inspect the changes to make sure the AST is not changed in an unexpected way that invalidates code.

Round trip validation

The following section describes when and how styling is guaranteed to yield correct code.

If tokens are not in the styling scope (as specified with the `scope` argument), no tokens are changed and the abstract syntax tree (AST) should not change. Hence, it is possible to validate the styling by comparing whether the parsed expression before and after styling have the same AST. This comparison omits roxygen code examples and comments. styler throws an error if the AST has changed through styling.

Note that if tokens are to be styled, such a comparison is not conducted because the AST might well change and such a change is intended. There is no way styler can validate styling, that is why we inform the user to carefully inspect the changes.

See section 'Warning' for a good strategy to apply styling safely.

Value

Invisibly returns a data frame that indicates for each file considered for styling whether or not it was actually changed (or would be changed when `dry` is not "off").

See Also

Other stylers: `style_dir()`, `style_file()`, `style_text()`, `styler_addins`

Examples

```
# the following is identical (because of ... and defaults)
# but the first is most convenient:
style_pkg(strict = TRUE)
style_pkg(style = tidyverse_style, strict = TRUE)
style_pkg(transformers = tidyverse_style(strict = TRUE))

# more options from `tidyverse_style()`
style_pkg(
  scope = "line_breaks",
  math_token_spacing = specify_math_token_spacing(zero = "'+'")
)

# don't write back and fail if input is not already styled
```

```
style_pkg("/path/to/pkg/", dry = "fail")
```

style_text *Style a string*

Description

Styles a character vector. Each element of the character vector corresponds to one line of code.

Usage

```
style_text(
  text,
  ...,
  style = tidyverse_style,
  transformers = style(...),
  include_roxygen_examples = TRUE,
  base_indention = 0L
)
```

Arguments

`text`	A character vector with text to style.
`...`	Arguments passed on to the `style` function, see `tidyverse_style()` for the default argument.
`style`	A function that creates a style guide to use, by default `tidyverse_style`. Not used further except to construct the argument `transformers`. See `style_guides()` for details.
`transformers`	A set of transformer functions. This argument is most conveniently constructed via the `style` argument and `...`. See 'Examples'.
`include_roxygen_examples`	Whether or not to style code in roxygen examples.
`base_indention`	Integer scalar indicating by how many spaces the whole output text should be indented. Note that this is not the same as splitting by line and add a `base_indention` spaces before the code in the case multi-line strings are present. See 'Examples'.

See Also

Other stylers: `style_dir()`, `style_file()`, `style_pkg()`, `styler_addins`

Examples

```
style_text("call( 1)")
style_text("1    + 1", strict = FALSE)

# the following is identical (because of ... and defaults)
# but the first is most convenient:
style_text("a<-3++1", strict = TRUE)
style_text("a<-3++1", style = tidyverse_style, strict = TRUE)
style_text("a<-3++1", transformers = tidyverse_style(strict = TRUE))

# more invasive scopes include less invasive scopes by default
style_text("a%>%b", scope = "spaces")
style_text("a%>%b; a", scope = "line_breaks")
style_text("a%>%b; a", scope = "tokens")

# opt out with I() to only style specific levels
style_text("a%>%b; a", scope = I("tokens"))
```

`tidyverse_style` *The tidyverse style*

Description

Style code according to the tidyverse style guide.

Usage

```
tidyverse_style(
  scope = "tokens",
  strict = TRUE,
  indent_by = 2L,
  start_comments_with_one_space = FALSE,
  reindention = tidyverse_reindention(),
  math_token_spacing = tidyverse_math_token_spacing()
)
```

Arguments

`scope`	The extent of manipulation. Can range from "none" (least invasive) to "tokens" (most invasive). See 'Details'. This argument is a string or a vector of class `AsIs`.
`strict`	A logical value indicating whether a set of strict or not so strict transformer functions should be returned. Compare the functions returned with or without `strict = TRUE`. For example, `strict = TRUE` means force *one* space e.g. after "," and *one* line break e.g. after a closing curly brace. `strict = FALSE` means to set spaces and line breaks to one if there is none and leave the code untouched otherwise. See 'Examples'.

`indent_by`	How many spaces of indention should be inserted after operators such as '('.
`start_comments_with_one_space`	Whether or not comments should start with only one space (see `start_comments_with_space()`).
`reindention`	A list of parameters for regex re-indention, most conveniently constructed using `specify_reindention()`.
`math_token_spacing`	A list of parameters that define spacing around math token, conveniently constructed using `specify_math_token_spacing()`.

Details

The following levels for `scope` are available:

- "none": Performs no transformation at all.
- "spaces": Manipulates spacing between token on the same line.
- "indention": Manipulates the indention, i.e. number of spaces at the beginning of each line.
- "line_breaks": Manipulates line breaks between tokens.
- "tokens": manipulates tokens.

`scope` can be specified in two ways:

- As a string: In this case all less invasive scope levels are implied, e.g. "line_breaks" includes "indention", "spaces". This is brief and what most users need.
- As vector of class `AsIs`: Each level has to be listed explicitly by wrapping one ore more levels of the scope in `I()`. This offers more granular control at the expense of more verbosity.

See 'Examples' for details.

Examples

```
style_text("call( 1)", style = tidyverse_style, scope = "spaces")
style_text("call( 1)", transformers = tidyverse_style(strict = TRUE))
style_text(c("ab <- 3", "a  <-3"), strict = FALSE) # keeps alignment of "<-"
style_text(c("ab <- 3", "a  <-3"), strict = TRUE) # drops alignment of "<-"

# styling line breaks only without spaces
style_text(c("ab <- 3", "a =3"), strict = TRUE, scope = I(c("line_breaks", "tokens")))
```

Index