Package 'styler'

February 23, 2020

Type Package

Title Non-Invasive Pretty Printing of R Code

Version 1.3.2

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

License GPL-3

URL https://github.com/r-lib/styler

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

Imports backports (>= 1.1.0), cli (>= 1.1.0), magrittr (>= 1.0.1), purrr (>= 0.2.3), R.cache (>= 0.14.0), rematch2 (>= 2.0.1), rlang (>= 0.1.1), rprojroot (>= 1.1), tibble (>= 1.4.2), tools, withr (>= 1.0.0), xfun (>= 0.1)

Suggests data.tree (>= 0.1.6), digest, dplyr, here, knitr, prettycode, rmarkdown, rstudioapi (>= 0.7), testthat (>= 2.1.0)

VignetteBuilder knitr

Encoding UTF-8

LazyData true

RoxygenNote 7.0.2

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-line-break.R' 'rules-other.R' 'rules-replacement.R' 'rules-spacing.R' 'serialize.R' 'set-assert-args.R' 'style-guides.R' 'styler.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],
Lorenz Walthert [cre, aut]

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

Repository CRAN

Date/Publication 2020-02-23 05:50:02 UTC

R topics documented:

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

Description

styler allows you to format .R files, 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 .R file.
- `style_dir()` to style all .R 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 <krlmlr+r@mailbox.org>

See Also

Useful links:

- https://github.com/r-lib/styler
- 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 = TRUE)

cache_deactivate(verbose = 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.
verbose	Whether or not to print an informative message about what the function is doing.

See Also

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

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 it's own cache by default, because styling is potentially different with different versions of styler.

See Also

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

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 innode 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 *Remember the past to be quicker in the future*

Description

Caching makes styler faster on repeated styling. It does not cache input but output code. That means if you style code that already complies to a style guide and you have previously styled that code, it will be quicker. Code is cached by expression and the cache is shared across all APIs (e.g. style_text() and Addin.

Setup

styler by default uses caching, via the {R.cache} package. You will be asked you to let it create a permanent cache on your file system that styler will use in case it is not set that up already for another tool that uses {R.cache}. We encourage users to let {R.cache} create a permanent directory for caching, because otherwise, the cache is lost at restart of R.

Non-interactive use

Note that if you have never authorized {R.cache} to create the cache in a permanent directory, it will build the cache in a temporary directory. To create a permanent cache, just open an interactive R session and type cache_info(). You can see under Location: if a permanent directory is used and if not, {R.cache} will ask you to create one the first time you use {R.cache} in an R session.

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.

Manage the cache

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

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. The former can be done with R.cache::setCacheRootPath("/path/to/cach
the latter can often be set in config files of CI/CD tools, e.g. see the the Travis documentation on caching.

`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.

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
)
```

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.

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 = tibble::lst(set_line_break_before_curly_opening)
  )
}
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
)
```

---

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 = 0, 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())
```

stylerignore *Turn off styling for parts of the code*

Description

Using stylerignore markers, you can temporarily turn off styler. See a few illustrative examples below.

Details

Styling is on 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. Note that inline statements cannot contain other comments apart from the marker, i.e. a line like `1 # comment # styler: off` won't be ignored.

To use something else as start and stop markers, set the R options `styler.ignore_start` and `styler.ignore_stop` using `options()`. If you want these settings to persist over multiple R sessions, consider setting them in your R profile, e.g. with `usethis::edit_rprofile()`.

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)
```

`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 environment variable `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 `styler::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
Sys.setenv(save_after_styling = TRUE)
# only style with scope = "spaces" when using the Addin
options(
  styler.addins_style_transformer = "styler::tidyverse_style(scope = 'spaces')"
)

## End(Not run)
```

style_dir *Prettify arbitrary R code*

Description

Performs various substitutions in all `.R`, `.Rmd` 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"),
  recursive = TRUE,
  exclude_files = NULL,
  exclude_dirs = c("packrat", "renv"),
  include_roxygen_examples = TRUE
)
```

Arguments

`path`	Path to a directory with files to transform.
`...`	Arguments passed on to the `style` function.
`style`	A function that creates a style guide to use, by default `tidyverse_style()` (without the parentheses). 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", "rnw".
`recursive`	A logical value indicating whether or not files in subdirectories 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.
`include_roxygen_examples`	Whether or not to style code in roxygen examples.

Value

Invisibly returns a data frame that indicates for each file considered for styling whether or not it was actually changed.

Warning

This function overwrites files (if styling results in a change of the code to be formatted). 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 the style guide has `scope < "tokens"`, 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 comments. styler compares error if the AST has changed through styling.

Note that with `scope = "tokens"` 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

```
## Not run:
style_dir(file_type = "r")

## End(Not run)
```

`style_file` *Style* `.R`, `.Rmd` *or* `.Rnw` *files*

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
)
```

Arguments

path	A character vector with paths to files to style.
...	Arguments passed on to the style function.
style	A function that creates a style guide to use, by default tidyverse_style() (without the parentheses). 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.

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.

Warning

This function overwrites files (if styling results in a change of the code to be formatted). 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 the style guide has scope < "tokens", 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 comments. styler compares error if the AST has changed through styling.

Note that with scope = "tokens" 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

```
# the following is identical but the former is more convenient:
file <- tempfile("styler", fileext = ".R")
xfun::write_utf8("1++1", file)
style_file(file, style = tidyverse_style, strict = TRUE)
style_file(file, transformers = tidyverse_style(strict = TRUE))
xfun::read_utf8(file)
unlink(file)
```

---

style_pkg *Prettify R source code*

Description

Performs various substitutions in all .R files in a package (code and tests). One can also (optionally) style .Rmd and/or .Rnw files (vignettes and readme) by changing the filetype argument. Carefully examine the results after running this function!

Usage

```
style_pkg(
  pkg = ".",
  ...,
  style = tidyverse_style,
  transformers = style(...),
  filetype = c("R", "Rprofile"),
  exclude_files = "R/RcppExports.R",
  exclude_dirs = c("packrat", "renv"),
  include_roxygen_examples = TRUE
)
```

Arguments

pkg	Path to a (subdirectory of an) R package.
...	Arguments passed on to the style function.
style	A function that creates a style guide to use, by default tidyverse_style() (without the parentheses). 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", "rnw".

`exclude_files` Character vector with paths to files that should be excluded from styling.

`exclude_dirs` Character vector with directories to exclude. 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.

Warning

This function overwrites files (if styling results in a change of the code to be formatted). 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 the style guide has `scope < "tokens"`, 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 comments. styler compares error if the AST has changed through styling.

Note that with `scope = "tokens"` 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.

See Also

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

Examples

```
## Not run:

style_pkg(style = tidyverse_style, strict = TRUE)
style_pkg(
  scope = "line_breaks",
  math_token_spacing = specify_math_token_spacing(zero = "'+'")
)

## End(Not run)
```

`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
)
```

Arguments

`text`	A character vector with text to style.
`...`	Arguments passed on to the `style` function.
`style`	A function that creates a style guide to use, by default `tidyverse_style()` (without the parentheses). 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.

See Also

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

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")
# the following is identical but the former is more convenient:
style_text("a<-3++1", style = tidyverse_style, strict = TRUE)
style_text("a<-3++1", transformers = tidyverse_style(strict = TRUE))
```

---

tidyverse_style *The tidyverse style*

Description

Style code according to the tidyverse style guide.

Usage

```
tidyverse_style(
  scope = "tokens",
  strict = TRUE,
  indent_by = 2,
  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 "token" (most invasive). See 'Details'. This argument is a vector of length one.
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 options for `scope` are available.

- "none": Performs no transformation at all.
- "spaces": Manipulates spacing between token on the same line.
- "indention": In addition to "spaces", this option also manipulates the indention level.
- "line_breaks": In addition to "indention", this option also manipulates line breaks.
- "tokens": In addition to "line_breaks", this option also manipulates tokens.

As it becomes clear from this description, more invasive operations can only be performed if all less invasive operations are performed too.

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 "<-"
```

Index