Package ‘stringr’

May 10, 2018

Title Simple, Consistent Wrappers for Common String Operations

Version 1.3.1

Description A consistent, simple and easy to use set of wrappers around the fantastic ‘stringi’ package. All function and argument names (and positions) are consistent, all functions deal with "NA"s and zero length vectors in the same way, and the output from one function is easy to feed into the input of another.

License GPL-2 | file LICENSE


BugReports https://github.com/tidyverse/stringr/issues

Depends R (>= 3.1)

Imports glue (>= 1.2.0), magrittr, stringi (>= 1.1.7)

Suggests covr, htmltools, htmlwidgets, knitr, rmarkdown, testthat

VignetteBuilder knitr

LazyData true

RoxygenNote 6.0.1

NeedsCompilation no

Author Hadley Wickham [aut, cre, cph], RStudio [cph, fnd]

Maintainer Hadley Wickham <hadley@rstudio.com>

Repository CRAN

Date/Publication 2018-05-10 21:32:15 UTC

R topics documented:

  case ................................................................. 2
  invert_match ...................................................... 3
  modifiers .......................................................... 4
  stringr-data ...................................................... 5
  str_c ............................................................. 6
case

Convert case of a string.

Description

Convert case of a string.

Usage

\[
\begin{align*}
\text{str_to_upper} & (\text{string}, \text{locale} = "en") \\
\text{str_to_lower} & (\text{string}, \text{locale} = "en") \\
\text{str_to_title} & (\text{string}, \text{locale} = "en")
\end{align*}
\]

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>String to modify</td>
</tr>
<tr>
<td>locale</td>
<td>Locale to use for translations. Defaults to &quot;en&quot; (English) to ensure consistent default ordering across platforms.</td>
</tr>
</tbody>
</table>
**invert_match**

*Switch location of matches to location of non-matches.*

---

**Examples**

```r
dog <- "The quick brown dog"
str_to_upper(dog)
str_to_lower(dog)
str_to_title(dog)

# Locale matters!
str_to_upper("i") # English
str_to_upper("i", "tr") # Turkish
```

---

**Description**

Invert a matrix of match locations to match the opposite of what was previously matched.

**Usage**

```r
invert_match(loc)
```

**Arguments**

`loc`  
matrix of match locations, as from `str_locate_all()`

**Value**

numeric match giving locations of non-matches

**Examples**

```r
numbers <- "1 and 2 and 4 and 456"
num_loc <- str_locate_all(numbers, "[0-9]+")[[1]]
str_sub(numbers, num_loc[, "start"], num_loc[, "end"])

text_loc <- invert_match(num_loc)
str_sub(numbers, text_loc[, "start"], text_loc[, "end"])
```
modifiers

Control matching behaviour with modifier functions.

Description

fixed Compare literal bytes in the string. This is very fast, but not usually what you want for non-ASCII character sets.

coll Compare strings respecting standard collation rules.

regex The default. Uses ICU regular expressions.

boundary Match boundaries between things.

Usage

fixed(pattern, ignore_case = FALSE)

coll(pattern, ignore_case = FALSE, locale = "en", ...)

regex(pattern, ignore_case = FALSE, multiline = FALSE, comments = FALSE, dotall = FALSE, ...)

boundary(type = c("character", "line_break", "sentence", "word"),
          skip_word_none = NA, ...)

Arguments

pattern Pattern to modify behaviour.
ignore_case Should case differences be ignored in the match?
locale Locale to use for comparisons. See stringi::stri_locale_list() for all possible options. Defaults to "en" (English) to ensure that the default collation is consistent across platforms.
... Other less frequently used arguments passed on to stringi::stri_opts_collator(),
         stringi::stri_opts_regex(), or stringi::stri_opts_brkiter() multiline If TRUE, $ and ^ match the beginning and end of each line. If FALSE, the default, only match the start and end of the input.
comments If TRUE, white space and comments beginning with # are ignored. Escape literal spaces with \.
dotall If TRUE, . will also match line terminators.
type Boundary type to detect.
character Every character is a boundary.
line_break Boundaries are places where it is acceptable to have a line break in the current locale.
sentence The beginnings and ends of sentences are boundaries, using intelligent rules to avoid counting abbreviations (details).
The beginnings and ends of words are boundaries.

`skip_word_none` Ignore "words" that don't contain any characters or numbers - i.e. punctuation. Default NA will skip such "words" only when splitting on word boundaries.

**See Also**

`str_wrap()` for breaking text to form paragraphs

`stringi::stringi-search-boundaries` for more detail on the various boundaries

**Examples**

```r
pattern <- "a.b"
strings <- c("abb", "a.b")
str_detect(strings, pattern)
str_detect(strings, fixed(pattern))
str_detect(strings, coll(pattern))

# coll() is useful for locale-aware case-insensitive matching
i <- c("I", "\u0130", "i")
i
str_detect(i, fixed("i", TRUE))
str_detect(i, coll("i", TRUE))
str_detect(i, coll("i", TRUE, locale = "tr"))

# Word boundaries
words <- c("These are some words.")
str_count(words, boundary("word"))
str_split(words, " ")[1]
str_split(words, boundary("word"))[1]

# Regular expression variations
str_extract_all("The Cat in the Hat", "[a-z]+")
str_extract_all("The Cat in the Hat", regex("[a-z]+", TRUE))

str_extract_all("a\nb\nc", ".")
str_extract_all("a\nb\nc", regex("\.", multiline = TRUE))

str_extract_all("a\nb\nc", "a")
str_extract_all("a\nb\nc", regex("a", dotall = TRUE))
```

**Description**

`fruit` and `word` come from the `rcorpora` package written by Gabor Csardi; the data was collected by Darius Kazemi and made available at [https://github.com/dariusk/corpora](https://github.com/dariusk/corpora). `sentences` is a collection of "Harvard sentences" used for standardised testing of voice.
str_c

Join multiple strings into a single string.

Usage

str_c(..., sep = "", collapse = NULL)

Arguments

... One or more character vectors. Zero length arguments are removed. Short arguments are recycled to the length of the longest. Like most other R functions, missing values are "infectious": whenever a missing value is combined with another string the result will always be missing. Use str_replace_na() to convert NA to "NA"

sep String to insert between input vectors.

collapse Optional string used to combine input vectors into single string.

Description

To understand how str_c works, you need to imagine that you are building up a matrix of strings. Each input argument forms a column, and is expanded to the length of the longest argument, using the usual recycling rules. The sep string is inserted between each column. If collapse is NULL each row is collapsed into a single string. If non-NULL that string is inserted at the end of each row, and the entire matrix collapsed to a single string.

Examples

length(sentences)
sentences[1:5]

length(fruit)
fruit[1:5]

length(words)
words[1:5]
str_conv

Specify the encoding of a string.

Description

This is a convenient way to override the current encoding of a string.

Usage

str_conv(string, encoding)

Arguments

string String to re-encode.
encoding Name of encoding. See stringi::stri_enc_list() for a complete list.

Examples

# Example from encoding?stringi::stringi
x <- rawToChar(as.raw(177))
x
str_conv(x, "ISO-8859-2") # Polish "a with ogonek"
str_conv(x, "ISO-8859-1") # Plus-minus
**str_count**  
*Count the number of matches in a string.*

**Description**

Vectorised over `string` and `pattern`.

**Usage**

```r
str_count(string, pattern = "")
```

**Arguments**

- `string`  
  Input vector. Either a character vector, or something coercible to one.

- `pattern`  
  Pattern to look for.

  The default interpretation is a regular expression, as described in `stringi::stringi-search-regex`. Control options with `regex()`.  
  Match a fixed string (i.e. by comparing only bytes), using `fixed()`. This is fast, but approximate. Generally, for matching human text, you’ll want `coll()` which respects character matching rules for the specified locale.
  
  Match character, word, line and sentence boundaries with `boundary()`. An empty pattern, "", is equivalent to `boundary("character")`.

**Value**

An integer vector.

**See Also**

- `stringi::stri_count()` which this function wraps.
- `str_locate()`/`str_locate_all()` to locate position of matches

**Examples**

```r
fruit <- c("apple", "banana", "pear", "pineapple")
str_count(fruit, "a")
str_count(fruit, "p")
str_count(fruit, "e")
str_count(fruit, c("a", "b", "p", "p"))

str_count(c("a.", "...", ".a.a"), ".")
str_count(c("a.", "...", ".a.a"), fixed("."))
```
str_detect

Detect the presence or absence of a pattern in a string.

Description

Vectorised over string and pattern.

Usage

str_detect(string, pattern)

Arguments

string

Input vector. Either a character vector, or something coercible to one.

pattern

Pattern to look for.

The default interpretation is a regular expression, as described in stringi::stringi-search-regex. Control options with regex().

Match a fixed string (i.e. by comparing only bytes), using fixed(). This is fast, but approximate. Generally, for matching human text, you’ll want coll() which respects character matching rules for the specified locale.

Match character, word, line and sentence boundaries with boundary(). An empty pattern, "", is equivalent to boundary("character").

Value

A logical vector.

See Also

stringi::stri_detect() which this function wraps, str_subset() for a convenient wrapper around x[str_detect(x, pattern)]

Examples

fruit <- c("apple", "banana", "pear", "pineapple")
str_detect(fruit, "a")
str_detect(fruit, "^a")
str_detect(fruit, "a$")
str_detect(fruit, "b")
str_detect(fruit, "[aeiou]")

# Also vectorised over pattern
str_detect("aecfg", letters)
str_dup

Duplicate and concatenate strings within a character vector.

Description

Vectorised over string and times.

Usage

str_dup(string, times)

Arguments

string Input character vector.
times Number of times to duplicate each string.

Value

A character vector.

Examples

fruit <- c("apple", "pear", "banana")
str_dup(fruit, 2)
str_dup(fruit, 1:3)
str_c("ba", str_dup("na", 0:5))

str_extract

Extract matching patterns from a string.

Description

Vectorised over string and pattern.

Usage

str_extract(string, pattern)

str_extract_all(string, pattern, simplify = FALSE)
str_extract

Arguments

string Input vector. Either a character vector, or something coercible to one.

pattern Pattern to look for.

The default interpretation is a regular expression, as described in stringi::stringi-search-regex. Control options with regex().

Match a fixed string (i.e. by comparing only bytes), using fixed(). This is fast, but approximate. Generally, for matching human text, you’ll want coll() which respects character matching rules for the specified locale.

Match character, word, line and sentence boundaries with boundary(). An empty pattern, "", is equivalent to boundary("character").

simplify If FALSE, the default, returns a list of character vectors. If TRUE returns a character matrix.

Value

A character vector.

See Also

str_match() to extract matched groups; stringi::stri_extract() for the underlying implementation.

Examples

shopping_list <- c("apples x4", "bag of flour", "bag of sugar", "milk x2")
str_extract(shopping_list, "\\d")
str_extract(shopping_list, "[a-z]+")
str_extract(shopping_list, "[a-z]{1,4}"")
str_extract(shopping_list, "\\b[a-z]{1,4}\\b")

# Extract all matches
str_extract_all(shopping_list, "[a-z]+")
str_extract_all(shopping_list, "\\b[a-z]+\\b")
str_extract_all(shopping_list, "\\d")

# Simplify results into character matrix
str_extract_all(shopping_list, "\\b[a-z]+\\b", simplify = TRUE)
str_extract_all(shopping_list, "\\d", simplify = TRUE)

# Extract all words
str_extract_all("This is, suprisingly, a sentence.", boundary("word"))
**str_flatten**

*Flatten a string*

**Description**

Flatten a string

**Usage**

```
str_flatten(string, collapse = "")
```

**Arguments**

- **string**: Character to flatten
- **collapse**: String to insert between each piece

**Value**

A character vector of length 1

**Examples**

```
str_flatten(letters)
str_flatten(letters, "-")
```

---

**str_glue**

*Format and interpolate a string with glue*

**Description**

These functions are wrappers around `glue::glue()` and `glue::glue_data()`, which provide a powerful and elegant syntax for interpolating strings. These wrappers provide a small set of the full options. Use the functions directly from glue for more control.

**Usage**

```
str_glue(..., .sep = "", .envir = parent.frame())
str_glue_data(.x, ..., .sep = "", .envir = parent.frame(), .na = "NA")
```
Arguments

... [expressions]
Expressions string(s) to format, multiple inputs are concatenated together before formatting.

.sep [character(1): `'”’`]
Separator used to separate elements.

.envir [environment: parent.frame()]
Environment to evaluate each expression in. Expressions are evaluated from left to right. If `.x` is an environment, the expressions are evaluated in that environment and `.envir` is ignored.

.x [listish]
An environment, list or data frame used to lookup values.

.na [character(1): ‘NA’]
Value to replace NA values with. If NULL missing values are propagated, that is an NA result will cause NA output. Otherwise the value is replaced by the value of `.na`.

Examples

```r
name <- "Fred"
age <- 50
anniversary <- as.Date("1991-10-12")
str_glue(
  "My name is {name}, ",
  "my age next year is {age + 1}, ",
  "and my anniversary is {format(anniversary, "%A, %B %d, %Y")}."
)

# single braces can be inserted by doubling them
str_glue("My name is {name}, not {{name}}.")

# You can also used named arguments
str_glue(
  "My name is {name}, ",
  "and my age next year is {age + 1}.", 
  name = "Joe",
  age = 40
)

# `str_glue_data()` is useful in data pipelines
mtcars %>% str_glue_data("{rownames(.)} has {hp} hp")
```

---

**str_length**

The length of a string.
str_locate

Description

Locate the position of patterns in a string.

Usage

str_locate(string)

Arguments

string Input vector. Either a character vector, or something coercible to one.

Value

A numeric vector giving number of characters (code points) in each element of the character vector. Missing string have missing length.

See Also

stringi::stri_length() which this function wraps.

Examples

str_locate(letters)
str_locate(NA)
str_locate(factor("abc"))
str_locate(c("i", "like", "programming", NA))

# Two ways of representing a u with an umlaut
u1 <- "\u00fc"
u2 <- stringi::stri_trans_nfd(u1)
# The print the same:
u1
u2
# But have a different length
str_locate(u1)
str_locate(u2)
# Even though they have the same number of characters
str_count(u1)
str_count(u2)
Usage

\texttt{str\_locate(string, pattern)}

\texttt{str\_locate\_all(string, pattern)}

Arguments

\textbf{string} \hspace{1cm} Input vector. Either a character vector, or something coercible to one.

\textbf{pattern} \hspace{1cm} Pattern to look for.

The default interpretation is a regular expression, as described in \texttt{stringi::stringi-search-regex}. Control options with \texttt{regex().}

Match a fixed string (i.e. by comparing only bytes), using \texttt{fixed().} This is fast, but approximate. Generally, for matching human text, you’ll want \texttt{coll()} which respects character matching rules for the specified locale.

Match character, word, line and sentence boundaries with \texttt{boundary().} An empty pattern, "", is equivalent to \texttt{boundary(“character”).}

Value

For \texttt{str\_locate}, an integer matrix. First column gives start position of match, and second column gives end position. For \texttt{str\_locate\_all} a list of integer matrices.

See Also

\texttt{str\_extract()} for a convenient way of extracting matches, \texttt{stringi::stri\_locate()} for the underlying implementation.

Examples

fruit <- c("apple", "banana", "pear", "pineapple")
str_locate(fruit, ")
str_locate(fruit, "a")
str_locate(fruit, "e")
str_locate(fruit, c("a", "b", "p", "p"))

str_locate_all(fruit, "a")
str_locate_all(fruit, "e")
str_locate_all(fruit, c("a", "b", "p", "p"))

# Find location of every character
str_locate_all(fruit, "")
str_match

Extract matched groups from a string.

Description

Vectorised over string and pattern.

Usage

str_match(string, pattern)
str_match_all(string, pattern)

Arguments

string Input vector. Either a character vector, or something coercible to one.
pattern Pattern to look for, as defined by an ICU regular expression. See string::stringi
search-regex for more details.

Value

For str_match, a character matrix. First column is the complete match, followed by one column for each capture group. For str_match_all, a list of character matrices.

See Also

str_extract() to extract the complete match, string::stri_match() for the underlying implementation.

Examples

strings <- c("219 733 8965", "329-293-8753 ", "banana", "595 794 7569",
"387 287 6718", "apple", "233.398.9187 ", "482 952 3315",
"239 923 8115 and 842 566 4692", "Work: 579-499-7527", "$1000",
"Home: 543.355.3679")
phone <- "([2-9][0-9]*[2])([- .][0-9]{3})[- .][0-9]{4}"

str_extract(strings, phone)
str_match(strings, phone)

# Extract/match all
str_extract_all(strings, phone)
str_match_all(strings, phone)

x <- c("<a> <b>", "<a> <>", "<a>", ",
str_match(x, "<.?> <(.>?)>")
str_match_all(x, "<(.?>)>")

str_extract(x, "<.?>")
str_extract_all(x, "<.?>")
**str_order**

Order or sort a character vector.

**Description**

Order or sort a character vector.

**Usage**

```r
str_order(x, decreasing = FALSE, na_last = TRUE, locale = "en",
numeric = FALSE, ...)
```

```r
str_sort(x, decreasing = FALSE, na_last = TRUE, locale = "en",
numeric = FALSE, ...)
```

**Arguments**

- `x` A character vector to sort.
- `decreasing` A boolean. If FALSE, the default, sorts from lowest to highest; if TRUE sorts from highest to lowest.
- `na_last` Where should NA go? TRUE at the end, FALSE at the beginning, NA dropped.
- `locale` In which locale should the sorting occur? Defaults to the English. This ensures that code behaves the same way across platforms.
- `numeric` If TRUE, will sort digits numerically, instead of as strings.
- `...` Other options used to control sorting order. Passed on to `stringi::stri_opts_collator()`.

**See Also**

`stringi::stri_order()` for the underlying implementation.

**Examples**

```r
str_order(letters)
str_sort(letters)

str_order(letters, locale = "haw")
str_sort(letters, locale = "haw")

x <- c("100a10", "100a5", "2b", "2a")
str_sort(x)
str_sort(x, numeric = TRUE)
```
str_pad

Pad a string.

Description

Vectorised over string, width and pad.

Usage

str_pad(string, width, side = c("left", "right", "both"), pad = " ")

Arguments

  string     A character vector.
  width      Minimum width of padded strings.
  side       Side on which padding character is added (left, right or both).
  pad        Single padding character (default is a space).

Value

A character vector.

See Also

str_trim() to remove whitespace; str_trunc() to decrease the maximum width of a string.

Examples

rbind(
  str_pad("hadley", 30, "left"),
  str_pad("hadley", 30, "right"),
  str_pad("hadley", 30, "both")
)

# All arguments are vectorised except side
str_pad(c("a", "abc", "abcdef"), 10)
str_pad("a", c(5, 10, 20))
str_pad("a", 10, pad = c("-", ",", " "))

# Longer strings are returned unchanged
str_pad("hadley", 3)


**str_remove**

Remove matched patterns in a string.

**Description**

Alias for `str_replace(string, pattern, ")").

**Usage**

```r
str_remove(string, pattern)
str_remove_all(string, pattern)
```

**Arguments**

- `string`  
  Input vector. Either a character vector, or something coercible to one.
- `pattern`  
  Pattern to look for.
  The default interpretation is a regular expression, as described in `stringr::stringi-search-regex`. Control options with `regex()`.  
  Match a fixed string (i.e. by comparing only bytes), using `fixed()`. This is fast, but approximate. Generally, for matching human text, you'll want `coll()` which respects character matching rules for the specified locale. 
  Match character, word, line and sentence boundaries with `boundary()`. An empty pattern, "", is equivalent to `boundary("character")`.

**Value**

A character vector.

**See Also**

`str_replace()` for the underlying implementation.

**Examples**

```r
fruits <- c("one apple", "two pears", "three bananas")
str_remove(fruits, "[aeiou]" batch)
str_remove_all(fruits, "[aeiou]"
```
str_replace  
Replace matched patterns in a string.

Description
Vectorised over string, pattern and replacement.

Usage
str_replace(string, pattern, replacement)
str_replace_all(string, pattern, replacement)

Arguments
string  
Input vector. Either a character vector, or something coercible to one.
pattern  
Pattern to look for.
The default interpretation is a regular expression, as described in stringi::stringi-search-regex. Control options with regex().
Match a fixed string (i.e. by comparing only bytes), using fixed(). This is fast, but approximate. Generally, for matching human text, you’ll want coll() which respects character matching rules for the specified locale.
replacement  
A character vector of replacements. Should be either length one, or the same length as string or pattern. References of the form |1, |R, etc will be replaced with the contents of the respective matched group (created by ()).
To perform multiple replacements in each element of string, pass a named vector (c(pattern1 = replacement1)) to str_replace_all. Alternatively, pass a function to replacement: it will be called once for each match and its return value will be used to replace the match.
To replace the complete string with NA, use replacement = NA_character_.

Value
A character vector.

See Also
str_replace_na() to turn missing values into "NA"; stri_replace() for the underlying implementation.

Examples
fruits <- c("one apple", "two pears", "three bananas")
str_replace(fruits, "[aeiou]", "-")
str_replace_all(fruits, "[aeiou]", "--")
str_replace_all(fruits, "[aeiou]", toupper)
str_replace_all(fruits, "b", NA_character_)
str_replace_na

str_replace(fruits, "([aeiou])", "")
str_replace(fruits, "([aeiou])", "\\1\\1")
str_replace(fruits, "[aeiou]", c("1", "2", "3"))
str_replace(fruits, c("a", "e", "i"), ")")

# If you want to apply multiple patterns and replacements to the same
# string, pass a named vector to pattern.
fruits %>%
  str_c(collapse = "---") %>%
  str_replace_all(c("one" = "1", "two" = "2", "three" = "3"))

# Use a function for more sophisticated replacement. This example
# replaces colour names with their hex values.
colours <- str_c("\b", colors(), "\b", collapse="|")
col2hex <- function(col) {
  rgb <- col2rgb(col)
  rgb(rgb["red", ], rgb["green", ], rgb["blue", ], max = 255)
}
x <- c(
  "Roses are red, violets are blue",
  "My favourite colour is green"
)
str_replace_all(x, colours, col2hex)

---

str_replace_na Turn NA into "NA"

Description

Turn NA into "NA"

Usage

str_replace_na(string, replacement = "NA")

Arguments

string Input vector. Either a character vector, or something coercible to one.
replacement A single string.

Examples

str_replace_na(c(NA, "abc", "def"))
\textbf{str_split} \hspace{1cm} \textit{Split up a string into pieces.}

\textbf{Description}  
Vectorised over string and pattern.

\textbf{Usage}  
\begin{verbatim}
str_split(string, pattern, n = Inf, simplify = FALSE)
str_split_fixed(string, pattern, n)
\end{verbatim}

\textbf{Arguments}  
\begin{itemize}
  \item \texttt{string}  
    Input vector. Either a character vector, or something coercible to one.
  \item \texttt{pattern}  
    Pattern to look for.
    \begin{itemize}
      \item The default interpretation is a regular expression, as described in \texttt{stringi::stringi-search-regex}. Control options with \texttt{regex()}.
      \item Match a fixed string (i.e. by comparing only bytes), using \texttt{fixed()}. This is fast, but approximate. Generally, for matching human text, you'll want \texttt{coll()} which respects character matching rules for the specified locale.
      \item Match character, word, line and sentence boundaries with \texttt{boundary()}. An empty pattern, ",", is equivalent to \texttt{boundary("character")}.
    \end{itemize}
  \item \texttt{n}  
    number of pieces to return. Default (Inf) uses all possible split positions.
    \begin{itemize}
      \item For \texttt{str_split_fixed}, if \texttt{n} is greater than the number of pieces, the result will be padded with empty strings.
    \end{itemize}
  \item \texttt{simplify}  
    If \texttt{FALSE}, the default, returns a list of character vectors. If \texttt{TRUE} returns a character matrix.
\end{itemize}

\textbf{Value}  
For \texttt{str_split_fixed}, a character matrix with \texttt{n} columns. For \texttt{str_split}, a list of character vectors.

\textbf{See Also}  
\texttt{stri_split()} for the underlying implementation.

\textbf{Examples}  
\begin{verbatim}
fruits <- c(
  "apples and oranges and pears and bananas",
  "pineapples and mangos and guavas"
)
\end{verbatim}
str_sub

```
str_split(fruits, " and ")
str_split(fruits, " and ", simplify = TRUE)

# Specify n to restrict the number of possible matches
str_split(fruits, " and ", n = 3)
str_split(fruits, " and ", n = 2)
# If n greater than number of pieces, no padding occurs
str_split(fruits, " and ", n = 5)

# Use fixed to return a character matrix
str_split_fixed(fruits, " and ", 3)
str_split_fixed(fruits, " and ", 4)
```

---

**str_sub**

*Extract and replace substrings from a character vector.*

**Description**

`str_sub` will recycle all arguments to be the same length as the longest argument. If any arguments are of length 0, the output will be a zero length character vector.

**Usage**

```
str_sub(string, start = 1L, end = -1L)
str_sub(string, start = 1L, end = -1L, omit_na = FALSE) <- value
```

**Arguments**

- **string**: input character vector.
- **start, end**: Two integer vectors. `start` gives the position of the first character (defaults to first), `end` gives the position of the last (defaults to last character). Alternatively, pass a two-column matrix to `start`. Negative values count backwards from the last character.
- **omit_na**: Single logical value. If TRUE, missing values in any of the arguments provided will result in an unchanged input.
- **value**: replacement string

**Details**

Substrings are inclusive - they include the characters at both start and end positions. `str_sub(string, 1, -1)` will return the complete substring, from the first character to the last.

**Value**

A character vector of substring from `start` to `end` (inclusive). Will be length of longest input argument.
See Also

The underlying implementation in `stringi::stri_sub()`

Examples

```r
hw <- "Hadley Wickham"

str_sub(hw, 1, 6)
str_sub(hw, end = 6)
str_sub(hw, 8, 14)
str_sub(hw, 8)
str_sub(hw, c(1, 8), c(6, 14))

# Negative indices
str_sub(hw, -1)
str_sub(hw, -7)
str_sub(hw, end = -7)

# Alternatively, you can pass in a two column matrix, as in the
# output from str_locate_all
pos <- str_locate_all(hw, "[aeio]"[[1]]
str_sub(hw, pos)
str_sub(hw, pos[, 1], pos[, 2])

# Vectorisation
str_sub(hw, seq_len(str_length(hw)))
str_sub(hw, end = seq_len(str_length(hw)))

# Replacement form
x <- "BCDEFG"
str_sub(x, 1, 1) <- "A"; x
str_sub(x, -1, -1) <- "K"; x
str_sub(x, -2, -2) <- "GHJI"; x
str_sub(x, 2, -2) <- ""; x

# If you want to keep the original if some argument is NA,
# use omit_na = TRUE
x1 <- x2 <- x3 <- x4 <- "AAA"
str_sub(x1, 1, NA) <- "B"
str_sub(x2, 1, 2) <- NA
str_sub(x3, 1, NA, omit_na = TRUE) <- "B"
str_sub(x4, 1, 2, omit_na = TRUE) <- NA
x1; x2; x3; x4
```

---

**str_subset**  
*Keep strings matching a pattern, or find positions.*

---

Description

`str_subset()` is a wrapper around `x[str_detect(x, pattern)]`, and is equivalent to `grep(pattern, x, value = TRUE)`.  
`str_which()` is a wrapper around `which(str_detect(x, pattern))`, and is equivalent to `grep(pattern, x).`
str_subset

Usage

str_subset(string, pattern)
str_which(string, pattern)

Arguments

string  Input vector. Either a character vector, or something coercible to one.
pattern Pattern to look for.

The default interpretation is a regular expression, as described in stringi::stringi-search-regex. Control options with regex().

Match a fixed string (i.e. by comparing only bytes), using fixed(). This is fast, but approximate. Generally, for matching human text, you’ll want coll() which respects character matching rules for the specified locale.

Match character, word, line and sentence boundaries with boundary(). An empty pattern, "", is equivalent to boundary("character").

Details

Vectorised over string and pattern

Value

A character vector.

See Also

grep() with argument value = TRUE, stringi::stri_subset() for the underlying implementation.

Examples

fruit <- c("apple", "banana", "pear", "pineapple")
str_subset(fruit, "a")
str_which(fruit, "a")

str_subset(fruit, "^a")
str_subset(fruit, "a$")
str_subset(fruit, "b")
str_subset(fruit, "[aeiou]")

# Missings never match
str_subset(c("a", NA, "b"), ".")
str_which(c("a", NA, "b"), ".")
Str Trim

Description

Str_trunc() removes whitespace from start and end of string; str_squish() also reduces repeated whitespace inside a string.

Usage

str_trunc(string, side = c("both", "left", "right"))
str_squish(string)

Arguments

string A character vector.
side Side on which to remove whitespace (left, right or both).

Value

A character vector.

See Also

str_pad() to add whitespace

Examples

str_trunc(" String with trailing and leading white space")
str_trunc("\n\nString with trailing and leading white space\n\n")
str_squish(" String with trailing, middle, and leading white space")
str_squish("\n\nString with excess, trailing and leading white space\n\n")

Str Trunc

Truncate a character string.

Description

Truncate a character string.

Usage

str_trunc(string, width, side = c("right", "left", "center"),
ellipsis = "."
str_view

Arguments

- **string**: A character vector.
- **width**: Maximum width of string.
- **side, ellipsis**: Location and content of ellipsis that indicates content has been removed.

See Also

str_pad() to increase the minimum width of a string.

Examples

```r
x <- "This string is moderately long"
rbind(
  str_trunc(x, 20, "right"),
  str_trunc(x, 20, "left"),
  str_trunc(x, 20, "center")
)
```

### Description

str_view shows the first match; str_view_all shows all the matches.

### Usage

```r
str_view(string, pattern, match = NA)
str_view_all(string, pattern, match = NA)
```

### Arguments

- **string**: Input vector. Either a character vector, or something coercible to one.
- **pattern**: Pattern to look for.
  The default interpretation is a regular expression, as described in stringi::stringi-search-regex. Control options with regex().
  Match a fixed string (i.e. by comparing only bytes), using fixed(). This is fast, but approximate. Generally, for matching human text, you’ll want coll() which respects character matching rules for the specified locale.
  Match character, word, line and sentence boundaries with boundary(). An empty pattern, "", is equivalent to boundary("character”).
- **match**: If TRUE, shows only strings that match the pattern. If FALSE, shows only the strings that don’t match the pattern. Otherwise (the default, NA) displays both matches and non-matches.
Examples

```r
grep("[aeiou]", c("abc", "def", "fgh"))
grep("[^]", c("abc", "def", "fgh"))
grep("\..\", c("abc", "def", "fgh"))

# Show all matches with str_view_all
str_view_all(c("abc", "def", "fgh"), "d\|e")

# Use match to control what is shown
str_view(c("abc", "def", "fgh"), "d\|e")
str_view(c("abc", "def", "fgh"), "d\|e", match = TRUE)
str_view(c("abc", "def", "fgh"), "d\|e", match = FALSE)
```

---

**str_wrap**

Wrap strings into nicely formatted paragraphs.

Description

This is a wrapper around `stringi::stri_wrap()` which implements the Knuth-Plass paragraph wrapping algorithm.

Usage

```r
str_wrap(string, width = 80, indent = 0, exdent = 0)
```

Arguments

- **string**: character vector of strings to reformat.
- **width**: positive integer giving target line width in characters. A width less than or equal to 1 will put each word on its own line.
- **indent**: non-negative integer giving indentation of first line in each paragraph
- **exdent**: non-negative integer giving indentation of following lines in each paragraph

Value

A character vector of re-wrapped strings.

Examples

```r
thanks_path <- file.path(R.home("doc"), "THANKS")
thanks <- str_c(readLines(thanks_path), collapse = "\n")
thanks <- word(thanks, 1, 3, fixed("\n\n"))
cat(str_wrap(thanks, "\n"))
cat(str_wrap(thanks, width = 40), "\n")
cat(str_wrap(thanks, width = 60, indent = 2), "\n")
cat(str_wrap(thanks, width = 60, exdent = 2), "\n")
cat(str_wrap(thanks, width = 80, exdent = 2), "\n")
```
word

Extract words from a sentence.

Description

Extract words from a sentence.

Usage

word(string, start = 1L, end = start, sep = fixed(" "))

Arguments

string      input character vector.
start       integer vector giving position of first word to extract. Defaults to first word. If negative, counts backwards from last character.
end         integer vector giving position of last word to extract. Defaults to first word. If negative, counts backwards from last character.
sep         separator between words. Defaults to single space.

Value

character vector of words from start to end (inclusive). Will be length of longest input argument.

Examples

sentences <- c("Jane saw a cat", "Jane sat down")
word(sentences, 1)
word(sentences, 2)
word(sentences, -1)
word(sentences, 2, -1)

# Also vectorised over start and end
word(sentences[1], 1:3, -1)
word(sentences[1], 1, 1:4)

# Can define words by other separators
str <- 'abc.def.123.4568.999'
word(str, 1, sep = fixed('..'))
word(str, 2, sep = fixed('..'))
Index

*Topic datasets
  stringr-data, 5
  boundary(modifiers), 4
  boundary(), 8, 9, 11, 15, 19, 22, 25, 27
  case, 2
  coll(modifiers), 4
  coll(), 8, 9, 11, 15, 19, 20, 22, 25, 27
  fixed(modifiers), 4
  fixed(), 8, 9, 11, 15, 19, 20, 22, 25, 27
  fruit(stringr-data), 5
  glue::glue(), 12
  glue::glue_data(), 12
  grep(), 25
  invert_match, 3
  modifiers, 4
  paste(), 7
  regex(modifiers), 4
  regex(), 8, 9, 11, 15, 19, 20, 22, 25, 27
  sentences(stringr-data), 5
  str_c, 6
  str_conv, 7
  str_count, 8
  str_detect, 9
  str_dup, 10
  str_extract, 10
  str_extract(), 15, 16
  str_extract_all(str_extract), 10
  str_flatten, 12
  str_glue, 12
  str_glue_data(str_glue), 12
  str_length, 13
  str_locate, 14
  str_locate_all(str_locate), 14
  str_locate_all(), 3, 8
  str_match, 16
  str_match(), 11
  str_match_all(str_match), 16
  str_order, 17
  str_pad, 18
  str_pad(), 26, 27
  str_remove, 19
  str_remove_all(str_remove), 19
  str_replace, 20
  str_replace(), 19
  str_replace_all(str_replace), 20
  str_replace_na, 21
  str_replace_na(), 6, 20
  str_sort(str_order), 17
  str_split, 22
  str_split_fixed(str_split), 22
  str_squish(str_trim), 26
  str_sub, 23
  str_sub<-(str_sub), 23
  str_subset, 24
  str_subset(), 9
  str_to_lower(case), 2
  str_to_title(case), 2
  str_to_upper(case), 2
  str_trim, 26
  str_trim(), 18
  str_trunc, 26
  str_trunc(), 18
  str_view, 27
  str_view_all(str_view), 27
  str_which(str_subset), 24
  str_wrap, 28
  str_wrap(), 5
  stri_replace(), 20
  stri_split(), 22
  stringi::stri_count(), 8
stringi::stri_detect(), 9
stringi::stri_enc_list(), 7
stringi::stri_extract(), 11
stringi::stri_join(), 7
stringi::stri_length(), 14
stringi::stri_locale_list(), 4
stringi::stri_locate(), 15
stringi::stri_match(), 16
stringi::stri_opts_brkiter(), 4
stringi::stri_opts_collator(), 4, 17
stringi::stri_opts_regex(), 4
stringi::stri_order(), 17
stringi::stri_sub(), 24
stringi::stri_subset(), 25
stringi::stri_wrap(), 28
stringi::stringiMsearchMregex, 8, 9, 11, 15, 16, 19, 20, 22, 25, 27
stringr-data, 5
word, 29
words(stringr-data), 5