Package 'stringr'

February 18, 2017

Version 1.2.0
Title Simple, Consistent Wrappers for Common String Operations
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
Depends R (>= 2.14)
Imports stringi (>= 0.4.1), magrittr
Suggests testthat, knitr, htmltools, htmlwidgets, rmarkdown, covr
VignetteBuilder knitr
URL http://stringr.tidyverse.org,https://github.com/tidyverse/stringr
BugReports https://github.com/tidyverse/stringr/issues
RoxygenNote 6.0.1
LazyData true
NeedsCompilation no
Author Hadley Wickham [aut, cre, cph], RStudio [cph]
Maintainer Hadley Wickham <hadley@rstudio.com></hadley@rstudio.com>
Repository CRAN
Date/Publication 2017-02-18 21:23:06
R topics documented:
case invert_match modifiers stringr-data str_c

2 case

Index		28
	word	27
	str_wrap	
	str_view	
	str_trunc	
	str_trim	24
	str_subset	22
	str_sub	21
	str_split	20
	str_replace_na	19
	str_replace	
	str_pad	17
	str_order	
	str_match	
	str_locate	
	str_length	
	str_interp	
	str_extract	
	str_dup	
	str_detect	
	str_count	8
	str_conv	/

case

Convert case of a string.

Description

Convert case of a string.

Usage

```
str_to_upper(string, locale = "en")
str_to_lower(string, locale = "en")
str_to_title(string, locale = "en")
```

Arguments

string String to modify

locale Locale to use for translations. Defaults to "en" (English) to ensure consistent default ordering across platforms.

invert_match 3

Examples

```
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</pre>
```

invert_match

Switch location of matches to location of non-matches.

Description

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

Usage

```
invert_match(loc)
```

Arguments

loc

matrix of match locations, as from str_locate_all

Value

numeric match giving locations of non-matches

```
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"])</pre>
```

4 modifiers

mod	÷	£;	_	
moa	1	Т1	e	rs

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 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 $stri_opts_collator, stri_opts_regex, or stri_opts_brkiter$
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.
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.

stringr-data 5

Examples

```
pattern <- "a.b"
strings <- c("abb", "a.b")</pre>
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")
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.")</pre>
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))
```

stringr-data

Sample character vectors for practicing string manipulations.

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. sentences is a collection of "Harvard sentences" used for standardised testing of voice.

Usage

sentences fruit words

Format

A character vector.

6 str_c

Examples

```
length(sentences)
sentences[1:5]
length(fruit)
fruit[1:5]
length(words)
words[1:5]
```

str_c

Join multiple strings into a 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 recyling 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.

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"

String to insert between input vectors. sep

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

Value

If collapse = NULL (the default) a character vector with length equal to the longest input string. If collapse is non-NULL, a character vector of length 1.

See Also

paste for equivalent base R functionality, and stri_join which this function wraps

str_conv 7

Examples

```
str_c("Letter: ", letters)
str_c("Letter", letters, sep = ": ")
str_c(letters, " is for", "...")
str_c(letters[-26], " comes before ", letters[-1])
str_c(letters, collapse = "")
str_c(letters, collapse = ", ")

# Missing inputs give missing outputs
str_c(c("a", NA, "b"), "-d")
# Use str_replace_NA to display literal NAs:
str_c(str_replace_na(c("a", NA, "b")), "-d")
```

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 stri_enc_list for a complete list.
```

```
# 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</pre>
```

8 str_count

str_count

Count the number of matches in a string.

Description

Vectorised over string and pattern.

Usage

```
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-search-regex. Control options with regex().

Match a fixed string (i.e. by comparing only bytes), using fixed(x). This is fast, but approximate. Generally, for matching human text, you'll want coll(x) 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

```
stri_count which this function wraps.
str_locate/str_locate_all to locate position of matches
```

```
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("."))</pre>
```

str_detect 9

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-search-

regex. Control options with regex().

Match a fixed string (i.e. by comparing only bytes), using fixed(x). This is fast, but approximate. Generally, for matching human text, you'll want coll(x)

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

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

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

# Also vectorised over pattern
str_detect("aecfg", letters)</pre>
```

10 str_extract

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))</pre>
```

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_interp 11

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-search-

regex. Control options with regex().

Match a fixed string (i.e. by comparing only bytes), using fixed(x). This is fast, but approximate. Generally, for matching human text, you'll want coll(x)

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 char-

acter matrix.

Value

A character vector.

See Also

str_match to extract matched groups; 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}\\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, "\\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"))</pre>
```

str_interp

String interpolation.

12 str_interp

Description

String interpolation is a useful way of specifying a character string which depends on values in a certain environment. It allows for string creation which is easier to read and write when compared to using e.g. paste or sprintf. The (template) string can include expression placeholders of the form \${expression} or \$[format]{expression}, where expressions are valid R expressions that can be evaluated in the given environment, and format is a format specification valid for use with sprintf.

Usage

```
str_interp(string, env = parent.frame())
```

Arguments

string A template character string. This function is not vectorised: a character vector

will be collapsed into a single string.

env The environment in which to evaluate the expressions.

Value

An interpolated character string.

Author(s)

Stefan Milton Bache

```
# Using values from the environment, and some formats
user_name <- "smbache"
amount <- 6.656
account <- 1337
str_interp("User ${user_name} (account $[08d]{account}) has $$[.2f]{amount}.")
# Nested brace pairs work inside expressions too, and any braces can be
# placed outside the expressions.
str_interp("Works with } nested { braces too: $[.2f]{{{2 + 2}*{amount}}}}")
# Values can also come from a list
str_interp(
  "One value, ${value1}, and then another, ${value2*2}.",
 list(value1 = 10, value2 = 20)
# Or a data frame
str_interp(
  "Values are $[.2f]{max(Sepal.Width)} and $[.2f]{min(Sepal.Width)}.",
 iris
)
```

str_length 13

```
# Use a vector when the string is long:
max_char <- 80
str_interp(c(
   "This particular line is so long that it is hard to write ",
   "without breaking the ${max_char}-char barrier!"
))</pre>
```

str_length

The length of a string.

Description

Technically this returns the number of "code points", in a string. One code point usually corresponds to one character, but not always. For example, an u with a umlaut might be represented as a single character or as the combination a u and an umlaut.

Usage

```
str_length(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

stri_length which this function wraps.

```
str_length(letters)
str_length(NA)
str_length(factor("abc"))
str_length(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_length(u1)
str_length(u2)</pre>
```

14 str_locate

```
# Even though they have the same number of characters
str_count(u1)
str_count(u2)
```

str_locate

Locate the position of patterns in a string.

Description

Vectorised over string and pattern. If the match is of length 0, (e.g. from a special match like \$) end will be one character less than start.

Usage

```
str_locate(string, pattern)
str_locate_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 stringi-search-

regex. Control options with regex().

Match a fixed string (i.e. by comparing only bytes), using fixed(x). This is fast, but approximate. Generally, for matching human text, you'll want coll(x)

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

For str_locate, an integer matrix. First column gives start postion of match, and second column gives end position. For str_locate_all a list of integer matrices.

See Also

str_extract for a convenient way of extracting matches, stri_locate for the underlying implementation.

```
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"))</pre>
```

str_match 15

```
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 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, stri_match for the underlying implementation.

```
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)

# Extract/match all
str_extract_all(strings, phone)
str_match_all(strings, phone)</pre>
```

str_order

```
x <- c("<a> <b>", "<a> <>", "<a>", "", NA)
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

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

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 stri_opts_collator.

See Also

stri_order for the underlying implementation.

```
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)</pre>
```

str_pad 17

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.

```
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)
```

18 str_replace

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-searchregex. Control options with regex().

> Match a fixed string (i.e. by comparing only bytes), using fixed(x). This is fast, but approximate. Generally, for matching human text, you'll want coll(x) 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").

replacement

A character vector of replacements. Should be either length one, or the same length as string or pattern. References of the form \1, \2, 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.

str_replace_na 19

Examples

```
fruits <- c("one apple", "two pears", "three bananas")</pre>
str_replace(fruits, "[aeiou]", "-")
str_replace_all(fruits, "[aeiou]", "-")
str_replace_all(fruits, "[aeiou]", toupper)
str_replace_all(fruits, "b", NA_character_)
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"), "-")
fruits <- c("one apple", "two pears", "three bananas")</pre>
str_replace(fruits, "[aeiou]", "-")
str_replace_all(fruits, "[aeiou]", "-")
str_replace_all(fruits, "([aeiou])", "")
str_replace_all(fruits, "([aeiou])", "\\1\\1")
str_replace_all(fruits, "[aeiou]", c("1", "2", "3"))
str\_replace\_all(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="|")</pre>
col2hex <- function(col) {</pre>
  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")
```

20 str_split

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"))
```

str_split

Split up a string into pieces.

Description

Vectorised over string and pattern.

Usage

```
str_split(string, pattern, n = Inf, simplify = FALSE)
str_split_fixed(string, pattern, n)
```

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-search-

regex. Control options with regex().

Match a fixed string (i.e. by comparing only bytes), using fixed(x). This is fast, but approximate. Generally, for matching human text, you'll want coll(x)

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").

n number of pieces to return. Default (Inf) uses all possible split positions.

For str_split_fixed, if n is greater than the number of pieces, the result will

be padded with empty strings.

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

acter matrix.

Value

For str_split_fixed, a character matrix with n columns. For str_split, a list of character vectors.

See Also

```
stri_split for the underlying implementation.
```

str_sub 21

Examples

```
fruits <- c(
    "apples and oranges and pears and bananas",
    "pineapples and mangos and guavas"
)

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)</pre>
```

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) <- value</pre>
```

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.

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.

22 str_subset

Value

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

See Also

The underlying implementation in stri_sub

Examples

```
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 colum matrix, as in the
# output from str_locate_all
pos <- str_locate_all(hw, "[aeio]")[[1]]</pre>
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 <- "BBCDEF"
str_sub(x, 1, 1) < "A"; x
str_sub(x, -1, -1) < "K"; x
str\_sub(x, -2, -2) \leftarrow "GHIJ"; x
str_sub(x, 2, -2) < ""; x
```

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 23

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-search-

regex. Control options with regex().

Match a fixed string (i.e. by comparing only bytes), using fixed(x). This is fast, but approximate. Generally, for matching human text, you'll want coll(x)

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, stri_subset for the underlying implementation.

```
fruit <- c("apple", "banana", "pear", "pinapple")
str_subset(fruit, "a")
str_which(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"), ".")</pre>
```

24 str_trunc

str_trim

Trim whitespace from start and end of string.

Description

Trim whitespace from start and end of string.

Usage

```
str_trim(string, side = c("both", "left", "right"))
```

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\_trim(" String with trailing and leading white space\t") \\ str\_trim("\n\string with 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 = "...")
```

Arguments

string A character vector.

width Maximum width of string.

side, ellipsis Location and content of ellipsis that indicates content has been removed.

str_view 25

See Also

str_pad to increase the minimum width of a string.

Examples

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

str_view

View HTML rendering of regular expression match.

Description

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

Usage

```
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-search-regex. Control options with regex().

Match a fixed string (i.e. by comparing only bytes), using fixed(x). This is fast, but approximate. Generally, for matching human text, you'll want coll(x) 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.

26 str_wrap

Examples

```
str_view(c("abc", "def", "fgh"), "[aeiou]")
str_view(c("abc", "def", "fgh"), "^")
str_view(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 stri_wrap which implements the Knuth-Plass paragraph wrapping algorithm.

Usage

```
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.

```
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 = 0, exdent = 2), "\n")</pre>
```

word 27

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.

```
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('...'))</pre>
```

Index

*Topic datasets	str_match_all (str_match), 15
stringr-data,5	str_order, 16
houndamy 9 0 11 14 19 20 22 25	str_pad, 17, 24, 25
boundary, 8, 9, 11, 14, 18, 20, 23, 25	str_replace, 18
boundary (modifiers), 4	str_replace_all (str_replace), 18
case, 2	str_replace_na, 6, 18, 19
coll, 8, 9, 11, 14, 18, 20, 23, 25	str_sort (str_order), 16
coll (modifiers), 4	str_split, 20
GGII (modifier 5); 1	str_split_fixed(str_split), 20
fixed, 8, 9, 11, 14, 18, 20, 23, 25	str_sub, 21
fixed (modifiers), 4	str_sub<- (str_sub), 21
fruit (stringr-data), 5	str_subset, 9, 22
, , , , , , , , , , , , , , , , , , , ,	str_to_lower (case), 2
grep, 23	str_to_title (case), 2
	str_to_upper (case), 2
invert_match, 3	str_trim, <i>17</i> , 24
	str_trunc, <i>17</i> , 24
modifiers, 4	str_view,25
	str_view_all(str_view),25
paste, 6, 12	<pre>str_which(str_subset), 22</pre>
regex, 8, 9, 11, 14, 18, 20, 23, 25	str_wrap, 26
regex, 6, 9, 11, 14, 16, 20, 23, 23 regex (modifiers), 4	stri_count,8
regex (moutriers), 4	stri_detect,9
sentences (stringr-data), 5	stri_enc_list,7
sprintf, 12	stri_extract, <i>ll</i>
str_c, 6	stri_join,6
str_conv, 7	stri_length, <i>13</i>
str_count, 8	stri_locale_list,4
str_detect, 9	stri_locate, <i>14</i>
str_dup, 10	stri_match, <i>15</i>
str_extract, 10, 14, 15	stri_opts_brkiter,4
str_extract_all (str_extract), 10	stri_opts_collator, 4, 16
str_interp, 11	stri_opts_regex, 4
str_join(str_c), 6	stri_order, 16
str_length, 13	stri_replace, 18
str_locate, 8, 14	stri_split, 20
str_locate_all, 3, 8	stri_sub, 22
str_locate_all (str_locate), 14	stri_subset, 23
str_match, 11, 15	stri_wrap, 26
	- ~ p , - -

INDEX 29

```
\begin{array}{c} {\rm stringi-search-regex},\,8,\,9,\,11,\,14,\,15,\,18,\\ 20,\,23,\,25\\ {\rm stringr-data},\,5\\ \\ {\rm word},\,27\\ {\rm words}\,({\rm stringr-data}),\,5\\ \end{array}
```