Package ‘stringr’

February 15, 2013

Maintainer Hadley Wickham <h.wickham@gmail.com>
License GPL-2
Title Make it easier to work with strings.
Type Package
Author Hadley Wickham <h.wickham@gmail.com>
Description stringr is a set of simple wrappers that make R’s string functions more consistent, simpler and easier to use. It does this by ensuring that: function and argument names (and positions) are consistent, all functions deal with NA’s and zero length character appropriately, and the output data structures from each function matches the input data structures of other functions.
Version 0.6.2
Depends R (>= 2.14)
Suggests testthat (>= 0.3)
Collate ‘c.r’ ‘checks.r’ ‘count.r’ ‘detect.r’ ‘dup.r’ ‘extract.r’ ‘length.r’ ‘locate.r’ ‘match.r’ ‘modifiers.r’ ‘padtrim.r’ ‘replace.r’ ‘split.r’ ‘sub.r’ ‘vectorise.r’ ‘word.r’ ‘wrap.r’ ‘utils.r’
Repository CRAN
Date/Publication 2012-12-06 08:39:59
NeedsCompilation no

R topics documented:

fixed .................................................. 2
ignore.case .......................................... 3
invert_match .......................................... 3
perl .................................................... 4
fixed

Match fixed characters, not regular expression.

Description

This function specifies that a pattern is a fixed string, rather than a regular expression. This can yield substantial speed ups, if regular expression matching is not needed.

Usage

fixed(string)

Arguments

string string to match exactly as is

See Also

Other modifiers: ignore.case, perl
Examples

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

Description

This function specifies that a pattern should ignore the case of matches.

Usage

```r
ignore.case(string)
```

Arguments

- `string`: pattern for which to ignore case

See Also

Other modifiers: `fixed`, `perl`

Examples

```r
pattern <- "a.b"
strings <- c("ABB", "aaB", "aab")
str_detect(strings, pattern)
str_detect(strings, ignore.case(pattern))
```

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

```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"])
```

---

perl

Use perl regular expressions.

Description

This function specifies that a pattern should use the Perl regular expression engine, rather than the default POSIX 1003.2 extended regular expressions

Usage

```r
perl(string)
```

Arguments

- **string**: pattern to match with Perl regexps

See Also

Other modifiers: **fixed, ignore.case**

Examples

```r
pattern <- "(?x)a.b"
strings <- c("abb", "a.b")
## Not run: str_detect(strings, pattern)
str_detect(strings, perl(pattern))
```
**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 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.

**Usage**

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

**Arguments**

- `...` one or more character vectors. Zero length arguments are removed
- `sep` string to insert between input vectors
- `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` which this function wraps

**Examples**

```r
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 = ", ")
```
str_count  

Count the number of matches in a string.

Description

Vectorised over string and pattern, shorter is recycled to same length as longest.

Usage

str_count(string, pattern)

Arguments

string  
input vector. This must be an atomic vector, and will be coerced to a character vector

pattern  
pattern to look for, as defined by a POSIX regular expression. See the “Extended Regular Expressions” section of regex for details. See fixed, ignore.case and perl for how to use other types of matching: fixed, case insensitive and perl-compatible.

Value

integer vector

See Also

regexpr which this function wraps
str_locate/str_locate_all to locate position of matches

Examples

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_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. This must be an atomic vector, and will be coerced to a character vector

pattern pattern to look for, as defined by a POSIX regular expression. See the “Extended Regular Expressions” section of regex for details. See fixed, ignore.case and perl for how to use other types of matching: fixed, case insensitive and perl-compatible.

Value

boolean vector

See Also

grepl which this function wraps

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**

character vector

**Examples**

```r
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 first piece of a string that matches a pattern.**

**Description**

Vectorised over `string`. `pattern` should be a single pattern, i.e. a character vector of length one.

**Usage**

```
str_extract(string, pattern)
```

**Arguments**

- `string`: input vector. This must be an atomic vector, and will be coerced to a character vector
- `pattern`: pattern to look for, as defined by a POSIX regular expression. See the “Extended Regular Expressions” section of `regex` for details. See `fixed`, `ignore.case` and `perl` for how to use other types of matching: fixed, case insensitive and perl-compatible.
Value

character vector.

See Also

str_extract_all to extract all matches

Examples

shopping_list <- c("apples x4", "flour", "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")
### str_length

**The length of a string (in characters).**

**Description**

The length of a string (in characters).

**Usage**

```r
str_length(string)
```

**Arguments**

- `string` input vector. This must be an atomic vector, and will be coerced to a character vector.

**Value**

numeric vector giving number of characters in each element of the character vector. Missing string have missing length.

**See Also**

- `nchar` which this function wraps

**Examples**

```r
str_length(letters)
str_length(c("i", "like", "programming", NA))
```

---

### str_locate

**Locate the position of the first occurrence of a pattern in a string.**

**Description**

Vectorised over `string` and `pattern`, shorter is recycled to same length as longest.

**Usage**

```r
str_locate(string, pattern)
```
str_locate_all

Arguments

string input vector. This must be an atomic vector, and will be coerced to a character vector

pattern pattern to look for, as defined by a POSIX regular expression. See the “Extended Regular Expressions” section of regex for details. See fixed, ignore.case and perl for how to use other types of matching: fixed, case insensitive and perl-compatible.

Value

integer matrix. First column gives start position of match, and second column gives end position.

See Also

regexpr which this function wraps

str_extract for a convenient way of extracting matches str_locate_all to locate position of all matches

Examples

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

str_locate_all Locate the position of all occurrences of a pattern in a string.

Description

Vectorised over string and pattern, shorter is recycled to same length as longest.

Usage

str_locate_all(string, pattern)

Arguments

string input vector. This must be an atomic vector, and will be coerced to a character vector

pattern pattern to look for, as defined by a POSIX regular expression. See the “Extended Regular Expressions” section of regex for details. See fixed, ignore.case and perl for how to use other types of matching: fixed, case insensitive and perl-compatible.
Details

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

Value

list of integer matrices. First column gives start position of match, and second column gives end position.

See Also

regexp which this function wraps
str_extract for a convenient way of extracting matches
str_locate to locate position of first match

Examples

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

Examples

phone <- "([2-9][0-9]{2})[- .]([0-9]{3})[- .]([0-9]{4})"
str_extract_all(strings, phone)
str_match_all(strings, phone)

str_match_all

Extract all matched groups from a string.

Description

Vectorised over string. pattern should be a single pattern, i.e. a character vector of length one.

Usage

str_match_all(string, pattern)

Arguments

pattern  
pattern to look for, as defined by a POSIX regular expression. Pattern should contain groups, defined by (). See the “Extended Regular Expressions” section of regex for details.

string  
input vector. This must be an atomic vector, and will be coerced to a character vector

Value

list of character matrices, as given by str_match

Examples

phone <- "([2-9][0-9]{2})[- .]([0-9]{3})[- .]([0-9]{4})"
str_extract_all(strings, phone)
str_match_all(strings, phone)
str_pad  

Pad a string.

Description

Vectorised over string. All other inputs should be of length 1.

Usage

str_pad(string, width, side = "left", pad = " ")

Arguments

- **string**: input character vector
- **width**: pad strings to this minimum width
- **side**: side on which padding character is added (left, right or both)
- **pad**: single padding character (default is a space)

Value

character vector

See Also

str_trim to remove whitespace

Examples

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

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

Replace first occurrence of a matched pattern in a string.

Description

Vectorised over string, pattern and replacement. Shorter arguments will be expanded to length of longest.

Usage

str_replace(string, pattern, replacement)

Arguments

replacement replacement string. References of the form \1, \2 will be replaced with the contents of the respective matched group (created by \()\) within the pattern.

string input vector. This must be an atomic vector, and will be coerced to a character vector

pattern pattern to look for, as defined by a POSIX regular expression. See the “Extended Regular Expressions” section of regex for details. See fixed, ignore.case and perl for how to use other types of matching: fixed, case insensitive and perl-compatible.

Value

character vector.

See Also

sub which this function wraps, str_replace_all to replace all matches

Examples

fruits <- c("one apple", "two pears", "three bananas")
str_replace(fruits, "[aeiou]", ")n")
str_replace_all(fruits, "[aeiou]", ")n")

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

Replace all occurrences of a matched pattern in a string.

Description

Vectorised over string, pattern and replacement. Shorter arguments will be expanded to length of longest.

Usage

`str_replace_all(string, pattern, replacement)`

Arguments

- `replacement`: replacement string. References of the form \1, \2 will be replaced with the contents of the respective matched group (created by (\)) within the pattern.
- `string`: input vector. This must be an atomic vector, and will be coerced to a character vector.
- `pattern`: pattern to look for, as defined by a POSIX regular expression. See the “Extended Regular Expressions” section of `regex` for details. See `fixed`, `ignore.case` and `perl` for how to use other types of matching: fixed, case insensitive and perl-compatible.

Value

character vector.

See Also

- `gsub` which this function wraps, `str_replace` to replace a single match.

Examples

```r
fruits <- c("one apple", "two pears", "three bananas")
str_replace(fruits, \[aeiou\], ");")
str_replace_all(fruits, \[aeiou\], ");")
str_replace_all(fruits, \((\[aeiou\])\), ");")
str_replace_all(fruits, \([aeiou]\), c("1", "2", "3");")
str_replace_all(fruits, c("a", "e", "i");")
```
str_split

Split up a string into a variable number of pieces.

Description

Vectorised over string. pattern should be a single pattern, i.e. a character vector of length one.

Usage

str_split(string, pattern, n = Inf)

Arguments

string input character vector
pattern pattern to split up by, as defined by a POSIX regular expression. See the “Ex-
tended Regular Expressions” section of regex for details. If NA, returns original
string. If "" splits into individual characters.
n maximum number of pieces to return. Default (Inf) uses all possible split posi-
tions.

Value

a list of character vectors.

See Also

str_split_fixed for fixed number of splits

Examples

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

# 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)
str_split_fixed

Split up a string into a fixed number of pieces.

Description

Vectorised over string. pattern should be a single pattern, i.e. a character vector of length one.

Usage

str_split_fixed(string, pattern, n)

Arguments

string input character vector
pattern pattern to split up by, as defined by a POSIX regular expression. See the “Extended Regular Expressions” section of regex for details. If NA, returns original string. If "" splits into individual characters.
n number of pieces to return. Default (Inf) uses all possible split positions. If n is greater than the number of pieces, the result will be padded with empty strings.

Value

character matrix with n columns.

See Also

str_split for variable number of splits

Examples

fruits <- c(
  "apples and oranges and pears and bananas",
  "pineapples and mangos and guavas"
)
str_split_fixed(fruits, " and ", 3)
str_split_fixed(fruits, " and ", 4)
Extract 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

```r
str_sub(string, start = 1L, end = -1L)
```

Arguments

- `string`: input character vector.
- `start`: integer vector giving position of first character in substring, defaults to first character. If negative, counts backwards from last character.
- `end`: integer vector giving position of last character in substring, defaults to last character. If negative, counts backwards from last character.

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

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

See Also

`substring` which this function wraps, and `link{str_sub_replace}` for the replacement version

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))
str_sub(hw, -1)
str_sub(hw, -7)
str_sub(hw, end = -7)
```
str_sub_replace

Replace substrings in a character vector. str_sub< will recycle all arguments to be the same length as the longest argument.

Description

Replace substrings in a character vector. str_sub< will recycle all arguments to be the same length as the longest argument.

Usage

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

Arguments

- **string**: input character vector.
- **start**: integer vector giving position of first character in substring, defaults to first character. If negative, counts backwards from last character.
- **end**: integer vector giving position of last character in substring, defaults to last character. If negative, counts backwards from last character.
- **value**: replacement string

Value

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

Examples

```r
x <- "BBCDEF"
str_sub(x, 1, 1) <- "A"; x
str_sub(x, -1, -1) <- "K"; x
str_sub(x, -2, -2) <- "GHIJ"; x
str_sub(x, 2, -2) <- ""; x
```
**str_trim**

Trim whitespace from start and end of string.

### Description

Trim whitespace from start and end of string.

### Usage

```r
str_trim(string, side = "both")
```

### Arguments

- **string**: input character vector
- **side**: side on which whitespace is removed (left, right or both)

### Value

character vector with leading and trailing whitespace removed

### See Also

- `str_pad` to add whitespace

### Examples

```r
str_trim(" String with trailing and leading white space\t")
str_trim("\n\nString with trailing and leading white space\n\n")
```

**str_wrap**

Wrap strings into nicely formatted paragraphs.

### Description

This is currently implemented as thin wrapper over `strwrap`, but is vectorised over stringr, and collapses output into single strings. See `strwrap` for more details.

### 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.
- **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 reformatted 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")
```

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

---

**word**

Extract words from a sentence.

Description

Extract words from a sentence.

Usage

```r
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

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

```r
# Also vectorised over start and end
word(text[,1], 1:3, -1)
word(text[,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 character
   fixed, 2
   ignore.case, 3
   perl, 4
   str_c, 5
   str_count, 6
   str_detect, 7
   str_dup, 8
   str_extract, 8
   str_extract_all, 9
   str_length, 10
   str_locate, 6
   str_locate_all, 11
   str_match, 12
   str_match_all, 13
   str_pad, 14
   str_replace, 15
   str_replace_all, 16
   str_split, 17
   str_split_fixed, 18
   str_sub, 19
   str_trim, 21
   fixed, 2, 3, 4, 6–9, 11, 15, 16
   grepl, 7
   gsub, 16
   ignore.case, 2, 3, 4, 6–9, 11, 15, 16
   invert_match, 3
   nchar, 10
   paste, 5
   perl, 2, 3, 4, 6–9, 11, 15, 16
   regex, 6–9, 11–13, 15–18
   regexpr, 6, 11, 12
   str_c, 5
   str_count, 6
   str_detect, 7
   str_dup, 8
   str_extract, 8, 9, 11, 12
   str_extract_all, 9, 9
   str_join (str_c), 5
   str_length, 10
   str_locate, 6, 10, 12
   str_locate_all, 3, 6, 11, 11
   str_match, 12, 13
   str_match_all, 13
   str_pad, 14, 21
   str_replace, 15, 16
   str_replace_all, 15, 16
   str_split, 17, 18
   str_split_fixed, 17, 18
   str_sub, 19
   str_sub (str_sub_replace), 20
   str_sub_replace, 20
   str_split, 14, 21
   str_wrap, 21
   strwrap, 21
   sub, 15
   substring, 19
   word, 22