Package 'filesstrings'

August 9, 2022

Type Package

Title Handy File and String Manipulation

Version 3.2.3

Maintainer Rory Nolan <rorynoolan@gmail.com>

Description This started out as a package for file and string manipulation. Since then, the 'fs' and 'strex' packages emerged, offering functionality previously given by this package (but it's done better in these new ones). Those packages have hence almost pushed 'filesstrings' into extinction. However, it still has a small number of unique, handy file manipulation functions which can be seen in the vignette. One example is a function to remove spaces from all file names in a directory.

License GPL-3

URL https://github.com/rorynolan/filesstrings

BugReports https://github.com/rorynolan/filesstrings/issues

Depends R (>= 3.5), stringr (>= 1.4)

Imports checkmate (>= 1.9.3), magrittr (>= 1.5), purrr (>= 0.3.0), rlang (>= 0.3.3), strex (>= 1.4.3), stringi (>= 1.7.8), withr (>= 2.1.0)

Suggests covr, dplyr, knitr, rmarkdown, spelling, testthat (>= 2.1)

VignetteBuilder knitr

Encoding UTF-8

Language en-US

RoxygenNote 7.2.1

NeedsCompilation no

Author Rory Nolan [aut, cre, cph] (<https://orcid.org/0000-0002-5239-4043>), Sergi Padilla-Parra [ths] (<https://orcid.org/0000-0002-8010-9481>)

Repository CRAN

Date/Publication 2022-08-09 17:20:02 UTC

R topics documented:

`all_equal` *An alternative version of* `base::all.equal()`.

Description

This function will return TRUE whenever base::all.equal() would return TRUE, however it will also return TRUE in some other cases:

- If a is given and b is not, TRUE will be returned if all of the elements of a are the same.
- If a is a scalar and b is a vector or array, TRUE will be returned if every element in b is equal to a.
- If a is a vector or array and b is a scalar, TRUE will be returned if every element in a is equal to b.

This function ignores names and attributes (except for dim).

When this function does not return TRUE, it returns FALSE (unless it errors). This is unlike base::all.equal().

Usage

```
all_equal(a, b = NULL)
```

Arguments

a	A vector, array or list.
b	Either NULL or a vector, array or list of length either 1 or length(a).

Value

TRUE if "equality of all" is satisfied (as detailed in 'Description' above) and FALSE otherwise.

Note

- This behaviour is totally different from base::all.equal().
- There's also dplyr::all_equal(), which is different again. To avoid confusion, always use the full filesstrings::all_equal() and never library(filesstrings) followed by just all_equal().

Examples

```
all_equal(1, rep(1, 3))
all_equal(2, 1:3)
all_equal(1:4, 1:4)
all_equal(1:4, c(1, 2, 3, 3))
all_equal(rep(1, 10))
all_equal(c(1, 88))
all_equal(1:2)
all_equal(list(1:2))
all_equal(1:4, matrix(1:4, nrow = 2)) # note that this gives TRUE
```

before_last_dot *Get the part of a string before the last period.*

Description

See strex::str_before_last_dot().

Usage

```
before_last_dot(string)

str_before_last_dot(string)
```

Arguments

string	A character vector.

can_be_numeric *Check if a string could be considered as numeric.*

Description

See strex::str_can_be_numeric().

Usage

```
can_be_numeric(string)

str_can_be_numeric(string)
```

Arguments

string	A character vector.

create_dir *Create directories if they don't already exist*

Description

Given the names of (potential) directories, create the ones that do not already exist.

Usage

```
create_dir(...)
```

Arguments

...	The names of the directories, specified via relative or absolute paths. Duplicates are ignored.

Value

Invisibly, a vector with a TRUE for each time a directory was actually created and a FALSE otherwise. This vector is named with the paths of the directories that were passed to the function.

Examples

```
## Not run:
create_dir(c("mydir", "yourdir"))
remove_dir(c("mydir", "yourdir"))

## End(Not run)
```

currency *Get the currencies of numbers within a string.*

Description

See strex::str_extract_currencies().

Usage

```
str_extract_currencies(string)

extract_currencies(string)

str_nth_currency(string, n)

nth_currency(string, n)
```

```
str_first_currency(string)

first_currency(string)

str_last_currency(string)

last_currency(string)
```

Arguments

string	A character vector.
n	A vector of integerish values. Must be either length 1 or have length equal to the length of string. Negative indices count from the back: while n = 1 and n = 2 correspond to first and second, n = -1 and n = -2 correspond to last and second-last. n = 0 will return NA.

---

extend_char_vec *Pad a character vector with empty strings.*

Description

Extend a character vector by appending empty strings at the end.

Usage

```
extend_char_vec(char_vec, extend_by = NA, length_out = NA)

str_extend_char_vec(char_vec, extend_by = NA, length_out = NA)
```

Arguments

char_vec	A character vector. The thing you wish to expand.
extend_by	A non-negative integer. By how much do you wish to extend the vector?
length_out	A positive integer. How long do you want the output vector to be?

Value

A character vector.

Examples

```
extend_char_vec(1:5, extend_by = 2)
extend_char_vec(c("a", "b"), length_out = 10)
```

`extract_non_numerics` *Extract non-numbers from a string.*

Description

See `strex::str_extract_non_numerics()`

Usage

```
extract_non_numerics(
  string,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE
)

str_extract_non_numerics(
  string,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE
)

nth_non_numeric(
  string,
  n,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE
)

str_nth_non_numeric(
  string,
  n,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE
)
```

```
first_non_numeric(
  string,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE
)

str_first_non_numeric(
  string,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE
)

last_non_numeric(
  string,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE
)

str_last_non_numeric(
  string,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE
)
```

Arguments

string	A string.
decimals	Do you want to include the possibility of decimal numbers (TRUE) or not (FALSE, the default).
leading_decimals	Do you want to allow a leading decimal point to be the start of a number?
negs	Do you want to allow negative numbers? Note that double negatives are not handled here (see the examples).
sci	Make the search aware of scientific notation e.g. 2e3 is the same as 2000.
commas	Allow comma separators in numbers (i.e. interpret 1,100 as a single number (one thousand one hundred) rather than two numbers (one and one hundred)).

n A vector of integerish values. Must be either length 1 or have length equal to the length of `string`. Negative indices count from the back: while `n = 1` and `n = 2` correspond to first and second, `n = -1` and `n = -2` correspond to last and second-last. `n = 0` will return `NA`.

`extract_numbers` *Extract numbers from a string.*

Description

See `strex::str_extract_numbers()`.

Usage

```
extract_numbers(
  string,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

str_extract_numbers(
  string,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

nth_number(
  string,
  n,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

str_nth_number(
  string,
```

```
  n,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

first_number(
  string,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

str_first_number(
  string,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

last_number(
  string,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

str_last_number(
  string,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)
```

Arguments

`string`	A string.
`decimals`	Do you want to include the possibility of decimal numbers (`TRUE`) or not (`FALSE`, the default).
`leading_decimals`	Do you want to allow a leading decimal point to be the start of a number?
`negs`	Do you want to allow negative numbers? Note that double negatives are not handled here (see the examples).
`sci`	Make the search aware of scientific notation e.g. 2e3 is the same as 2000.
`commas`	Allow comma separators in numbers (i.e. interpret 1,100 as a single number (one thousand one hundred) rather than two numbers (one and one hundred)).
`leave_as_string`	Do you want to return the number as a string (`TRUE`) or as numeric (`FALSE`, the default)?
`n`	A vector of integerish values. Must be either length 1 or have length equal to the length of `string`. Negative indices count from the back: while `n = 1` and `n = 2` correspond to first and second, `n = -1` and `n = -2` correspond to last and second-last. `n = 0` will return `NA`.

`filesstrings` `filesstrings`*: handy file and string manipulation*

Description

This started out as a package for file and string manipulation. Since then, the `fs` file manipulation package and the `strex` string manipulation package emerged, offering functionality previously given by this package (but slightly better). Those packages have hence almost pushed 'filesstrings' into extinction. However, it still has a small number of unique, handy file manipulation functions which can be seen in the vignette.. One example is a function to remove spaces from all file names in a directory.

References

Rory Nolan and Sergi Padilla-Parra (2017). filesstrings: An R package for file and string manipulation. The Journal of Open Source Software, 2(14). doi:10.21105/joss.00260.

give_ext *Ensure a file name has the intended extension.*

Description

See strex::str_give_ext().

Usage

```
give_ext(string, ext, replace = FALSE)

str_give_ext(string, ext, replace = FALSE)
```

Arguments

string	The intended file name.
ext	The intended file extension (with or without the ".").
replace	If the file has an extension already, replace it (or append the new extension name)?

group_close *Group together close adjacent elements of a vector.*

Description

Given a strictly increasing vector (each element is bigger than the last), group together stretches of the vector where *adjacent* elements are separated by at most some specified distance. Hence, each element in each group has at least one other element in that group that is *close* to it. See the examples.

Usage

```
group_close(vec_ascending, max_gap = 1)
```

Arguments

vec_ascending	A strictly increasing numeric vector.
max_gap	The biggest allowable gap between adjacent elements for them to be considered part of the same *group*.

Value

A where each element is one group, as a numeric vector.

Examples

```
group_close(1:10, 1)
group_close(1:10, 0.5)
group_close(c(1, 2, 4, 10, 11, 14, 20, 25, 27), 3)
```

---

locate_braces *Locate the braces in a string.*

Description

See `strex::str_locate_braces()`.

Usage

```
locate_braces(string)

str_locate_braces(string)
```

Arguments

string	A character vector

---

match_arg *Argument Matching*

Description

See `strex::match_arg()`.

Usage

```
match_arg(
  arg,
  choices = NULL,
  index = FALSE,
  several_ok = FALSE,
  ignore_case = FALSE
)

str_match_arg(
  arg,
  choices = NULL,
  index = FALSE,
  several_ok = FALSE,
  ignore_case = FALSE
)
```

Arguments

arg	A character vector (of length one unless several_ok = TRUE).
choices	A character vector of candidate values.
index	Return the index of the match rather than the match itself?
several_ok	Allow arg to have length greater than one to match several arguments at once?
ignore_case	Ignore case while matching. If this is TRUE, the returned value is the matched element of choices (with its original casing).

---

move_files *Move files around.*

Description

Move specified files into specified directories

Usage

```
move_files(files, destinations, overwrite = FALSE)

file.move(files, destinations, overwrite = FALSE)
```

Arguments

files	A character vector of files to move (relative or absolute paths).
destinations	A character vector of the destination directories into which to move the files.
overwrite	Allow overwriting of files? Default no.

Details

If there are n files, there must be either 1 or n directories. If there is one directory, then all n files are moved there. If there are n directories, then each file is put into its respective directory. This function also works to move directories.

If you try to move files to a directory that doesn't exist, the directory is first created and then the files are put inside.

Value

Invisibly, a logical vector with a TRUE for each time the operation succeeded and a FALSE for every fail.

Examples

```
## Not run:
dir.create("dir")
files <- c("1litres_1.txt", "1litres_30.txt", "3litres_5.txt")
file.create(files)
file.move(files, "dir")

## End(Not run)
```

`nice_file_nums` *Make file numbers comply with alphabetical order*

Description

If files are numbered, their numbers may not *comply* with alphabetical order, i.e. "file2.ext" comes after "file10.ext" in alphabetical order. This function renames the files in the specified directory such that they comply with alphabetical order, so here "file2.ext" would be renamed to "file02.ext".

Usage

```
nice_file_nums(dir = ".", pattern = NA)
```

Arguments

`dir`	Path (relative or absolute) to the directory in which to do the renaming (default is current working directory).
`pattern`	A regular expression. If specified, files to be renamed are restricted to ones matching this pattern (in their name).

Details

It works on file names with more than one number in them e.g. "file01part3.ext" (a file with 2 numbers). All the file names that it works on must have the same number of numbers, and the non-number bits must be the same. One can limit the renaming to files matching a certain pattern. This function wraps `nice_nums()`, which does the string operations, but not the renaming. To see examples of how this function works, see the examples in that function's documentation.

Value

A logical vector with a `TRUE` for each successful rename (should be all `TRUE`s) and a `FALSE` otherwise.

Examples

```
## Not run:
dir.create("NiceFileNums_test")
setwd("NiceFileNums_test")
files <- c("1litres_1.txt", "1litres_30.txt", "3litres_5.txt")
file.create(files)
nice_file_nums()
nice_file_nums(pattern = "\\.txt$")
setwd("..")
dir.remove("NiceFileNums_test")

## End(Not run)
```

---

nice_nums *Make string numbers comply with alphabetical order*

Description

See `strex::str_alphord_nums()`.

Usage

```
nice_nums(string)

str_nice_nums(string)

str_alphord_nums(string)

alphord_nums(string)
```

Arguments

`string`	A character vector.

---

nth_number_after_mth *Find the* nth *number after the* mth *occurrence of a pattern.*

Description

See `strex::str_nth_number_after_mth()`.

Usage

```
nth_number_after_mth(
  string,
  pattern,
  n,
  m,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

str_nth_number_after_mth(
  string,
  pattern,
  n,
  m,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

nth_number_after_first(
  string,
  pattern,
  n,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

nth_number_after_last(
  string,
  pattern,
  n,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
```

```
  leave_as_string = FALSE
)

first_number_after_mth(
  string,
  pattern,
  m,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

last_number_after_mth(
  string,
  pattern,
  m,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

first_number_after_first(
  string,
  pattern,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

first_number_after_last(
  string,
  pattern,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)
```

```
last_number_after_first(
  string,
  pattern,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

last_number_after_last(
  string,
  pattern,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

str_nth_number_after_first(
  string,
  pattern,
  n,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

str_nth_number_after_last(
  string,
  pattern,
  n,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

str_first_number_after_mth(
```

```
  string,
  pattern,
  m,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

str_last_number_after_mth(
  string,
  pattern,
  m,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

str_first_number_after_first(
  string,
  pattern,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

str_first_number_after_last(
  string,
  pattern,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

str_last_number_after_first(
  string,
  pattern,
```

```
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

str_last_number_after_last(
  string,
  pattern,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)
```

Arguments

`string`	A character vector.
`pattern`	The pattern to look for. The default interpretation is a regular expression, as described in stringi::about_search_regex. To match a without regular expression (i.e. as a human would), use coll(). For details see `stringr::regex()`.
`n, m`	Vectors of integerish values. Must be either length 1 or have length equal to the length of `string`. Negative indices count from the back: while `1` and `2` correspond to first and second, `-1` and `-2` correspond to last and second-last. `0` will return `NA`.
`decimals`	Do you want to include the possibility of decimal numbers (`TRUE`) or not (`FALSE`, the default).
`leading_decimals`	Do you want to allow a leading decimal point to be the start of a number?
`negs`	Do you want to allow negative numbers? Note that double negatives are not handled here (see the examples).
`sci`	Make the search aware of scientific notation e.g. 2e3 is the same as 2000.
`commas`	Allow comma separators in numbers (i.e. interpret 1,100 as a single number (one thousand one hundred) rather than two numbers (one and one hundred)).
`leave_as_string`	Do you want to return the number as a string (`TRUE`) or as numeric (`FALSE`, the default)?

nth_number_before_mth *Find the* nth *number before the* mth *occurrence of a pattern.*

Description

See strex::str_nth_number_before_mth().

Usage

```
nth_number_before_mth(
  string,
  pattern,
  n,
  m,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

str_nth_number_before_mth(
  string,
  pattern,
  n,
  m,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

nth_number_before_first(
  string,
  pattern,
  n,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)
```

```
nth_number_before_last(
  string,
  pattern,
  n,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

first_number_before_mth(
  string,
  pattern,
  m,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

last_number_before_mth(
  string,
  pattern,
  m,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

first_number_before_first(
  string,
  pattern,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

first_number_before_last(
```

```
  string,
  pattern,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

last_number_before_first(
  string,
  pattern,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

last_number_before_last(
  string,
  pattern,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

str_nth_number_before_first(
  string,
  pattern,
  n,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

str_nth_number_before_last(
  string,
  pattern,
  n,
```

```
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

str_first_number_before_mth(
  string,
  pattern,
  m,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

str_last_number_before_mth(
  string,
  pattern,
  m,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

str_first_number_before_first(
  string,
  pattern,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

str_first_number_before_last(
  string,
  pattern,
  decimals = FALSE,
  leading_decimals = decimals,
```

```
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

str_last_number_before_first(
  string,
  pattern,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)

str_last_number_before_last(
  string,
  pattern,
  decimals = FALSE,
  leading_decimals = decimals,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE,
  leave_as_string = FALSE
)
```

Arguments

string	A character vector.
pattern	The pattern to look for. The default interpretation is a regular expression, as described in stringi::about_search_regex. To match a without regular expression (i.e. as a human would), use coll(). For details see stringr::regex().
n, m	Vectors of integerish values. Must be either length 1 or have length equal to the length of string. Negative indices count from the back: while 1 and 2 correspond to first and second, -1 and -2 correspond to last and second-last. 0 will return NA.
decimals	Do you want to include the possibility of decimal numbers (TRUE) or not (FALSE, the default).
leading_decimals	Do you want to allow a leading decimal point to be the start of a number?
negs	Do you want to allow negative numbers? Note that double negatives are not handled here (see the examples).
sci	Make the search aware of scientific notation e.g. 2e3 is the same as 2000.

commas — Allow comma separators in numbers (i.e. interpret 1,100 as a single number (one thousand one hundred) rather than two numbers (one and one hundred)).

leave_as_string — Do you want to return the number as a string (TRUE) or as numeric (FALSE, the default)?

`put_in_pos` *Put specified strings in specified positions in an otherwise empty character vector.*

Description

Create a character vector with a set of strings at specified positions in that character vector, with the rest of it taken up by empty strings.

Usage

```
put_in_pos(strings, positions)

str_put_in_pos(strings, positions)
```

Arguments

strings	A character vector of the strings to put in positions (coerced by as.character if not character already).
positions	The indices of the character vector to be occupied by the elements of strings. Must be the same length as strings or of length 1.

Value

A character vector.

Examples

```
put_in_pos(1:3, c(1, 8, 9))
put_in_pos(c("Apple", "Orange", "County"), c(5, 7, 8))
put_in_pos(1:2, 5)
```

remove_dir *Remove directories*

Description

Delete directories and all of their contents.

Usage

```
remove_dir(...)

dir.remove(...)
```

Arguments

...	The names of the directories, specified via relative or absolute paths.

Value

Invisibly, a logical vector with TRUE for each success and FALSE for failures.

Examples

```
## Not run:
sapply(c("mydir1", "mydir2"), dir.create)
remove_dir(c("mydir1", "mydir2"))

## End(Not run)
```

remove_filename_spaces *Remove spaces in file names*

Description

Remove spaces in file names in a specified directory, replacing them with whatever you want, default nothing.

Usage

```
remove_filename_spaces(dir = ".", pattern = "", replacement = "")
```

Arguments

dir	The directory in which to perform the operation.
pattern	A regular expression. If specified, only files matching this pattern will be treated.
replacement	What do you want to replace the spaces with? This defaults to nothing, another sensible choice would be an underscore.

Value

A logical vector indicating which operation succeeded for each of the files attempted. Using a missing value for a file or path name will always be regarded as a failure.

Examples

```
## Not run:
dir.create("RemoveFileNameSpaces_test")
setwd("RemoveFileNameSpaces_test")
files <- c("1litres 1.txt", "1litres 30.txt", "3litres 5.txt")
file.create(files)
remove_filename_spaces()
list.files()
setwd("..")
dir.remove("RemoveFileNameSpaces_test")

## End(Not run)
```

---

remove_quoted *Remove the quoted parts of a string.*

Description

See `strex::str_remove_quoted()`.

Usage

```
remove_quoted(string)

str_remove_quoted(string)
```

Arguments

string	A character vector.

---

rename_with_nums *Replace file names with numbers*

Description

Rename the files in the directory, replacing file names with numbers only.

Usage

```
rename_with_nums(dir = ".", pattern = NULL)
```

Arguments

dir	The directory in which to rename the files (relative or absolute path). Defaults to current working directory.
pattern	A regular expression. If specified, only files with names matching this pattern will be treated.

Value

A logical vector with a TRUE for each successful renaming and a FALSE otherwise.

Examples

```
## Not run:
dir.create("RenameWithNums_test")
setwd("RenameWithNums_test")
files <- c("1litres 1.txt", "1litres 30.txt", "3litres 5.txt")
file.create(files)
rename_with_nums()
list.files()
setwd("..")
dir.remove("RenameWithNums_test")

## End(Not run)
```

---

singleize *Remove back-to-back duplicates of a pattern in a string.*

Description

See strex::str_singleize().

Usage

```
singleize(string, pattern)

str_singleize(string, pattern)
```

Arguments

string	A character vector.
pattern	The pattern to look for. The default interpretation is a regular expression, as described in stringi::about_search_regex. To match a without regular expression (i.e. as a human would), use coll(). For details see stringr::regex().

str_after_nth *Text after the* nth *occurrence of pattern.*

Description

See strex::str_after_nth().

Usage

```
str_after_nth(string, pattern, n)

after_nth(string, pattern, n)

str_after_first(string, pattern)

after_first(string, pattern)

str_after_last(string, pattern)

after_last(string, pattern)
```

Arguments

string	A character vector.
pattern	The pattern to look for. The default interpretation is a regular expression, as described in stringi::about_search_regex. To match a without regular expression (i.e. as a human would), use coll(). For details see stringr::regex().
n	A vector of integerish values. Must be either length 1 or have length equal to the length of string. Negative indices count from the back: while n = 1 and n = 2 correspond to first and second, n = -1 and n = -2 correspond to last and second-last. n = 0 will return NA.

str_before_nth *Text before the* nth *occurrence of pattern.*

Description

See strex::str_before_nth().

Usage

```
str_before_nth(string, pattern, n)

before_nth(string, pattern, n)

str_before_first(string, pattern)

before_first(string, pattern)

str_before_last(string, pattern)

before_last(string, pattern)
```

Arguments

string	A character vector.
pattern	The pattern to look for. The default interpretation is a regular expression, as described in stringi::about_search_regex. To match a without regular expression (i.e. as a human would), use coll(). For details see `stringr::regex()`.
n	A vector of integerish values. Must be either length 1 or have length equal to the length of `string`. Negative indices count from the back: while `n = 1` and `n = 2` correspond to first and second, `n = -1` and `n = -2` correspond to last and second-last. `n = 0` will return `NA`.

---

str_elem *Extract a single character from a string, using its index.*

Description

See `strex::str_elem()`.

Usage

```
str_elem(string, index)

elem(string, index)
```

Arguments

string	A character vector.
index	An integer. Negative indexing is allowed as in `stringr::str_sub()`.

str_elems *Extract several single elements from a string.*

Description

See strex::str_elems().

Usage

```
str_elems(string, indices, byrow = TRUE)

elems(string, indices, byrow = TRUE)
```

Arguments

string	A character vector.
indices	A vector of integerish values. Negative indexing is allowed as in stringr::str_sub().
byrow	Should the elements be organised in the matrix with one row per string (byrow = TRUE, the default) or one column per string (byrow = FALSE). See examples if you don't understand.

str_locate_nth *Get the indices of the nth instance of a pattern.*

Description

See strex::str_locate_nth().

Usage

```
str_locate_nth(string, pattern, n)

locate_nth(string, pattern, n)

str_locate_first(string, pattern)

locate_first(string, pattern)

str_locate_last(string, pattern)

locate_last(string, pattern)
```

Arguments

string	A character vector.
pattern	The pattern to look for. The default interpretation is a regular expression, as described in stringi::about_search_regex. To match a without regular expression (i.e. as a human would), use coll(). For details see stringr::regex().
n	A vector of integerish values. Must be either length 1 or have length equal to the length of string. Negative indices count from the back: while n = 1 and n = 2 correspond to first and second, n = -1 and n = -2 correspond to last and second-last. n = 0 will return NA.

---

str_paste_elems *Extract bits of a string and paste them together.*

Description

See strex::str_paste_elems().

Usage

```
str_paste_elems(string, indices, sep = "")

paste_elems(string, indices, sep = "")
```

Arguments

string	A character vector.
indices	A vector of integerish values. Negative indexing is allowed as in stringr::str_sub().
sep	A string. The separator for pasting string elements together.

---

str_split_by_nums *Split a string by its numeric characters.*

Description

See strex::str_split_by_numbers().

Usage

```
str_split_by_nums(
  string,
  decimals = FALSE,
  leading_decimals = FALSE,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE
)

split_by_nums(
  string,
  decimals = FALSE,
  leading_decimals = FALSE,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE
)

split_by_numbers(
  string,
  decimals = FALSE,
  leading_decimals = FALSE,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE
)

str_split_by_numbers(
  string,
  decimals = FALSE,
  leading_decimals = FALSE,
  negs = FALSE,
  sci = FALSE,
  commas = FALSE
)
```

Arguments

`string`	A string.
`decimals`	Do you want to include the possibility of decimal numbers (`TRUE`) or not (`FALSE`, the default).
`leading_decimals`	Do you want to allow a leading decimal point to be the start of a number?
`negs`	Do you want to allow negative numbers? Note that double negatives are not handled here (see the examples).
`sci`	Make the search aware of scientific notation e.g. 2e3 is the same as 2000.

commas	Allow comma separators in numbers (i.e. interpret 1,100 as a single number (one thousand one hundred) rather than two numbers (one and one hundred)).

---

str_split_camel_case *Split a string based on CamelCase*

Description

See strex::str_split_camel_case().

Usage

```
str_split_camel_case(string, lower = FALSE)

split_camel_case(string, lower = FALSE)
```

Arguments

string	A character vector.
lower	Do you want the output to be all lower case (or as is)?

---

str_to_vec *Convert a string to a vector of characters*

Description

See strex::str_to_vec().

Usage

```
str_to_vec(string)

to_vec(string)
```

Arguments

string	A character vector.

trim_anything *Trim something other than whitespace*

Description

See strex::str_trim_anything().

Usage

```
trim_anything(string, pattern, side = "both")

str_trim_anything(string, pattern, side = "both")
```

Arguments

string	A character vector.
pattern	The pattern to look for. The default interpretation is a regular expression, as described in stringi::about_search_regex. To match a without regular expression (i.e. as a human would), use coll(). For details see stringr::regex().
side	Which side do you want to trim from? "both" is the default, but you can also have just either "left" or "right" (or optionally the shortened "b", "l" and "r").

unitize_dirs *Put files with the same unit measurements into directories*

Description

Say you have a number of files with "5min" in their names, number with "10min" in the names, a number with "15min" in their names and so on, and you'd like to put them into directories named "5min", "10min", "15min" and so on. This function does this, but not just for the unit "min", for any unit.

Usage

```
unitize_dirs(unit, pattern = NULL, dir = ".")
```

Arguments

unit	The unit upon which to base the categorizing.
pattern	If set, only files with names matching this pattern will be treated.
dir	In which directory do you want to perform this action (defaults to current)?

Details

This function takes the number to be the last number (as defined in `nth_number()`) before the first occurrence of the unit name. There is the option to only treat files matching a certain pattern.

Value

Invisibly `TRUE` if the operation is successful, if not there will be an error.

Examples

```
## Not run:
dir.create("UnitDirs_test")
setwd("UnitDirs_test")
files <- c("1litres_1.txt", "1litres_3.txt", "3litres.txt", "5litres_1.txt")
file.create(files)
unitize_dirs("litres", "\\.txt")
setwd("..")
dir.remove("UnitDirs_test")

## End(Not run)
```

Index