Package 'filesstrings'

June 29, 2018

Type Package

Title Handy File and String Manipulation

Version 2.5.0

Maintainer Rory Nolan <rorynoolan@gmail.com>

Description Convenient functions for moving files, deleting directories, and a variety of string operations that facilitate manipulating files and extracting information from strings.

License GPL-3

Encoding UTF-8

LazyData true

Imports tibble, Rcpp, magrittr, tools, ore, matrixStats, purrr, checkmate, rlang

RoxygenNote 6.0.1

Suggests testthat, covr, knitr, rmarkdown, dplyr

LinkingTo Rcpp

Depends stringr

SystemRequirements C++11

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

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

VignetteBuilder knitr

NeedsCompilation yes

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 2018-06-29 15:38:55 UTC

R topics documented:

`all_equal` *A more flexible version of all.equal for vectors.*

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.

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

This is usually used to get the part of a file name that doesn't include the file extension. It is vectorized over `string`. If there is no period in `string`, the input is returned.

Usage

```
before_last_dot(string)
```

Arguments

`string`	A character vector.

Value

A character vector.

Examples

```
before_last_dot(c("spreadsheet1.csv", "doc2.doc", ".R"))
```

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

Description

After padding is removed, could the input string be considered to be numeric, i.e. could it be coerced to numeric. This function is vectorized over its one argument.

Usage

```
can_be_numeric(string)
```

Arguments

`string`	A character vector.

Value

A character vector. `TRUE` if the argument can be considered to be numeric or `FALSE` otherwise.

Examples

```
can_be_numeric("3")
can_be_numeric("5 ")
can_be_numeric(c("1a", "abc"))
```

`count_matches` *Count the number of the matches of a pattern in a string.*

Description

Vectorized over `string` and `pattern`.

Usage

```
count_matches(string, pattern)
```

Arguments

`string`	A character vector.
`pattern`	A character vector. Pattern(s) specified like the pattern(s) in the stringr package (e.g. look at `stringr::str_locate()`). If this has length >1 its length must be the same as that of `string`.

Value

An integer vector giving the number of matches in each string.

Examples

```
count_matches(c("abacad", "xyz"), "a")
count_matches("2.1.0.13", ".")
count_matches("2.1.0.13", stringr::coll("."))
```

`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

The currency of a number is defined as the character coming before the number in the string. If nothing comes before (i.e. if the number is the first thing in the string), the currency is the empty string, similarly the currency can be a space, comma or any manner of thing.

- get_currency takes a string and returns the currency of the first number therein. It is vectorized over string.
- get_currencies takes a string and returns the currencies of all of the numbers within that string. It is not vectorized.

Usage

```
get_currencies(string)

get_currency(strings)
```

Arguments

string	A string.
strings	A character vector.

Details

These functions do not allow for leading decimal points.

Value

- `get_currency` returns a character vector.
- `get_currencies` returns a data frame with one column for the currency symbol and one for the amount.

Examples

```
get_currencies("35.00 $1.14 abc5 $3.8 77")
get_currency(c("ab3 13", "$1"))
```

`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)
```

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_numbers *Extract numbers (or non-numbers) from a string.*

Description

extract_numbers extracts the numbers (or non-numbers) from a string where decimals are optionally allowed. extract_non_numerics extracts the bits of the string that aren't extracted by extract_numbers. nth_number is a convenient wrapper for extract_numbers, allowing you to choose which number you want. Similarly nth_non_numeric. Please view the examples at the bottom of this page to ensure that you understand how these functions work, and their limitations. These functions are vectorized over string.

Usage

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

extract_non_numerics(string, decimals = FALSE, leading_decimals = FALSE,
  negs = FALSE)

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

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

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

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

first_non_numeric(string, decimals = FALSE, leading_decimals = FALSE,
  negs = FALSE)

last_non_numeric(string, decimals = FALSE, leading_decimals = FALSE,
  negs = FALSE)
```

Arguments

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

`n`
: The index of the number (or non-numeric) that you seek. Negative indexing is allowed i.e. `n = 1` (the default) will give you the first number (or non-numeric) whereas `n = -1` will give you the last number (or non-numeric), `n = -2` will give you the second last number and so on.

Details

If any part of a string contains an ambiguous number (e.g. `1.2.3` would be ambiguous if `decimals = TRUE` (but not otherwise)), the value returned for that string will be `NA`. Note that these functions do not know about scientific notation (e.g. `1e6` for 1000000).

- `first_number(...)` is just `nth_number(..., n = 1)`.
- `last_number(...)` is just `nth_number(..., n = -1)`.
- `first_non_numeric(...)` is just `nth_non_numeric(..., n = 1)`.
- `last_non_numeric(...)` is just `nth_non_numeric(..., n = -1)`.

Value

For `extract_numbers` and `extract_non_numerics`, a list of numeric or character vectors, one list element for each element of `string`. For `nth_number` and `nth_non_numeric`, a vector the same length as `string` (as in `length(string)`, not `nchar(string)`).

Examples

```
extract_numbers(c("abc123abc456", "abc1.23abc456"))
extract_numbers(c("abc1.23abc456", "abc1..23abc456"), decimals = TRUE)
extract_numbers("abc1..23abc456", decimals = TRUE)
extract_numbers("abc1..23abc456", decimals = TRUE, leading_decimals = TRUE)
extract_numbers("abc1..23abc456", decimals = TRUE, leading_decimals = TRUE,
                leave_as_string = TRUE)
extract_numbers("-123abc456")
extract_numbers("-123abc456", negs = TRUE)
extract_numbers("--123abc456", negs = TRUE)
extract_non_numerics("abc123abc456")
extract_non_numerics("abc1.23abc456")
extract_non_numerics("abc1.23abc456", decimals = TRUE)
extract_non_numerics("abc1..23abc456", decimals = TRUE)
extract_non_numerics("abc1..23abc456", decimals = TRUE,
leading_decimals = TRUE)
extract_non_numerics(c("-123abc456", "ab1c"))
extract_non_numerics("-123abc456", negs = TRUE)
extract_non_numerics("--123abc456", negs = TRUE)
extract_numbers(c(rep("abc1.2.3", 2), "a1b2.2.3", "e5r6"), decimals = TRUE)
extract_numbers("ab.1.2", decimals = TRUE, leading_decimals = TRUE)
nth_number("abc1.23abc456", 2)
nth_number("abc1.23abc456", 2, decimals = TRUE)
```

```
nth_number("-123abc456", -2, negs = TRUE)
extract_non_numerics("--123abc456", negs = TRUE)
nth_non_numeric("--123abc456", 1)
nth_non_numeric("--123abc456", -2)
```

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

Description

Convenient functions for moving files, deleting directories, and a variety of string operations that facilitate manipulating files and extracting information from strings.

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.

`filesstrings-defunct` *Defunct functions*

Description

These functions have been made defunct, mostly because the naming style of the package has been changed. Some have been removed because they were not well-done.

Arguments

`...`	Defunct function arguments.

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

Description

Say you want to ensure a name is fit to be the name of a csv file. Then, if the input doesn't end with ".csv", this function will tack ".csv" onto the end of it. This is vectorized over the first argument.

Usage

```
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)?

Value

A string: the file name in your intended form.

Examples

```
give_ext(c("abc", "abc.csv"), "csv")
give_ext("abc.csv", "pdf")
give_ext("abc.csv", "pdf", replace = TRUE)
```

---

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

Give the positions of (,), [,], {, } within a string.

Usage

```
locate_braces(string)
```

Arguments

string	A character vector

Value

A list of data frames, one for each member of the string character vector. Each data frame has a "position" and "brace" column which give the positions and types of braces in the given string.

Examples

```
locate_braces(c("a{](kkj)})", "ab(]c{}"))
```

match_arg *Argument Matching*

Description

Match arg against a series of candidate choices where NULL means take the first one. arg *matches* an element of choices if arg is a prefix of that element.

Usage

```
match_arg(arg, choices, 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? Default no.
several_ok	Allow arg to have length greater than one to match several arguments at once? Default no.
ignore_case	Ignore case while matching. Default no. If this is TRUE, the returned value is the matched element of choices (with its original casing).

Details

ERRORs are thrown when a match is not made and where the match is ambiguous. However, sometimes ambiguities are inevitable. Consider the case where `choices = c("ab", "abc")`, then there's no way to choose `"ab"` because `"ab"` is a prefix for `"ab"` and `"abc"`. If this is the case, you need to provide a full match, i.e. using `arg = "ab"` will get you `"ab"` without an error, however `arg = "a"` will throw an ambiguity error.

This function inspired by `RSAGA::match.arg.ext()`. Its behaviour is almost identical (the difference is that `RSAGA::match.arg.ext(..., ignore.case = TRUE)` guarantees that the function returns strings in all lower case, but that is not so with `filesstrings::match_arg(..., ignore_case = TRUE)`) but RSAGA is a heavy package to depend upon so `filesstrings::match_arg()` might be handy for package developers.

Examples

```
choices <- c("Apples", "Pears", "Bananas", "Oranges")
match_arg(NULL, choices)
match_arg("A", choices)
match_arg("B", choices, index = TRUE)
match_arg(c("a", "b"), choices, several_ok = TRUE, ignore_case = TRUE)
match_arg(c("b", "a"), choices, ignore_case = TRUE, index = TRUE,
          several_ok = TRUE)
```

---

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 TRUEs) 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

If strings are numbered, their numbers may not *comply* with alphabetical order, i.e. "abc2" comes after "abc10" in alphabetical order. We might (for whatever reason) wish to change them such that they come in the order *that we would like*. This function alters the strings such that they comply with alphabetical order, so here "abc2" would be renamed to "abc02". It works on file names with more than one number in them e.g. "abc01def3" (a string 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.

Usage

```
nice_nums(strings)
```

Arguments

strings	A vector of strings.

Examples

```
strings <- paste0("abc", 1:12)
strings
nice_nums(strings)

nice_nums(c("abc9def55", "abc10def7"))
nice_nums(c("01abc9def55", "5abc10def777", "99abc4def4"))
nice_nums(1:10)

## Not run:
nice_nums(c("abc9def55", "abc10xyz7"))
## End(Not run)
```

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

Description

Given a string, a pattern and natural numbers `n` and `m`, find the `n`th number after the `m`th occurrence of the pattern.

Usage

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

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

nth_number_after_last(string, pattern, n, decimals = FALSE,
  leading_decimals = FALSE, negs = FALSE, leave_as_string = FALSE)

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

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

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

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

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

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

Arguments

`string`	A character vector.
`pattern`	A character vector. Pattern(s) specified like the pattern(s) in the stringr package (e.g. look at `stringr::str_locate()`). If this has length >1 its length must be the same as that of `string`.
`n, m`	Natural numbers.
`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).

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

Value

A numeric vector.

Examples

```
string <- c("abc1abc2abc3abc4abc5abc6abc7abc8abc9",
            "abc1def2ghi3abc4def5ghi6abc7def8ghi9")
nth_number_after_mth(string, "abc", 1, 3)
nth_number_after_mth(string, "abc", 2, 3)
nth_number_after_first(string, "abc", 2)
nth_number_after_last(string, "abc", -1)
first_number_after_mth(string, "abc", 2)
last_number_after_mth(string, "abc", 1)
first_number_after_first(string, "abc")
first_number_after_last(string, "abc")
last_number_after_first(string, "abc")
last_number_after_last(string, "abc")
```

---

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)
```

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

If any parts of a string are quoted (between quotation marks), remove those parts of the string, including the quotes. Run the examples and you'll know exactly how this function works.

Usage

```
remove_quoted(string)
```

Arguments

string	A character vector.

Value

A character vector.

Examples

```
string <- "\"abc\"67a\'dk\'f"
cat(string)
remove_quoted(string)
```

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

If a string contains a given pattern duplicated back-to-back a number of times, remove that duplication, leaving the pattern appearing once in that position (works if the pattern is duplicated in different parts of a string, removing all instances of duplication). This is vectorized over string and pattern.

Usage

```
singleize(string, pattern)
```

Arguments

`string`	A character vector. The string(s) to be purged of duplicates.
`pattern`	A character vector. Pattern(s) specified like the pattern(s) in the stringr package (e.g. look at `stringr::str_locate()`). If this has length >1 its length must be the same as that of `string`.

Value

The string with the duplicates fixed.

Examples

```
singleize("abc//def", "/")
singleize("abababcabab", "ab")
singleize(c("abab", "cdcd"), "cd")
singleize(c("abab", "cdcd"), c("ab", "cd"))
```

`str_after_nth` *Text before or after nth occurrence of pattern.*

Description

Extract the part of a string which is before or after the *n*th occurrence of a specified pattern, vectorized over the string. `n` can be negatively indexed. See 'Arguments'.

Usage

```
str_after_nth(strings, pattern, n)

str_after_first(strings, pattern)

str_after_last(strings, pattern)

str_before_nth(strings, pattern, n)

str_before_first(strings, pattern)

str_before_last(strings, pattern)
```

Arguments

`strings`	A character vector.
`pattern`	A character vector. Pattern(s) specified like the pattern(s) in the stringr package (e.g. look at `stringr::str_locate()`). If this has length >1 its length must be the same as that of `string`.
`n`	A natural number to identify the nth occurrence (defaults to first (`n = 1`)). This can be negatively indexed, so if you wish to select the *last* occurrence, you need `n = -1`, for the second-last, you need `n = -2` and so on.

Details

- `str_after_first(...)` is just `str_after_nth(..., n = 1)`.
- `str_after_last(...)` is just `str_after_nth(..., n = -1)`.
- `str_before_first(...)` is just `str_before_nth(..., n = 1)`.
- `str_before_last(...)` is just `str_before_nth(..., n = -1)`.

Value

A character vector of the desired strings.

Examples

```
string <- "ab..cd..de..fg..h"
str_after_nth(string, "\\.\\.", 3)
str_before_nth(string, "e", 1)
str_before_nth(string, "\\.", -3)
str_before_nth(string, ".", -3)
str_before_nth(rep(string, 2), fixed("."), -3)
```

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

Description

If the element does not exist, this function returns the empty string.

Usage

```
str_elem(string, index)
```

Arguments

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

Value

A one-character string.

Examples

```
str_elem(c("abcd", "xyz"), 3)
str_elem("abcd", -2)
```

`str_nth_instance_indices`
Get the indices of the nth instance of a pattern.

Description

The nth instance of an pattern will cover a series of character indices. These functions tell you which indices those are.

Usage

```
str_nth_instance_indices(string, pattern, n)

str_first_instance_indices(string, pattern)

str_last_instance_indices(string, pattern)
```

Arguments

string	A character vector. These functions are vectorized over this argument.
pattern	A character vector. Pattern(s) specified like the pattern(s) in the stringr package (e.g. look at `stringr::str_locate()`). If this has length >1 its length must be the same as that of `string`.
n	Then n for the nth instance of the pattern.

Details

- `str_first_instance_indices(...)` is just `str_nth_instance_indices(..., n = 1)`.
- `str_last_instance_indices(...)` is just `str_nth_instance_indices(..., n = -1)`.

Value

A two-column matrix. The ith row of this matrix gives the start and end indices of the nth instance of `pattern` in the ith element of `string`.

Examples

```
str_nth_instance_indices(c("abcdabcxyz", "abcabc"), "abc", 2)
```

---

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

Description

Extract characters - specified by their indices - from a string and paste them together

Usage

```
str_paste_elems(string, indices)
```

Arguments

string	A string.
indices	A numeric vector of positive integers detailing the indices of the characters of `string` that we wish to paste together.

Value

A string.

Examples

```
str_paste_elems("abcdef", c(2, 5:6))
```

`str_split_by_nums` *Split a string by its numeric characters.*

Description

Break a string wherever you go from a numeric character to a non-numeric or vice-versa.

Usage

```
str_split_by_nums(string, decimals = FALSE, leading_decimals = FALSE,
  negs = 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).

Examples

```
str_split_by_nums(c("abc123def456.789gh", "a1b2c344"))
str_split_by_nums("abc123def456.789gh", decimals = TRUE)
str_split_by_nums("22")
```

str_split_camel_case *Split a string based on CamelCase*

Description

Vectorized over `string`.

Usage

```
str_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)?

Value

A list of character vectors, one list element for each element of `string`.

References

Adapted from Ramnath Vaidyanathan's answer at http://stackoverflow.com/questions/8406974/splitting-camelcase-in-r.

Examples

```
str_split_camel_case(c("RoryNolan", "NaomiFlagg", "DepartmentOfSillyHats"))
```

str_to_vec *Convert a string to a vector of characters*

Description

Go from a string to a vector whose ith element is the ith character in the string.

Usage

```
str_to_vec(string)
```

Arguments

`string`	A string.

Value

A character vector.

Examples

```
str_to_vec("abcdef")
```

`str_with_patterns` *Which strings match the patterns?*

Description

Given a character vector of strings and one of patterns (in regular expression), which of the strings match all (or any) of the patterns.

Usage

```
str_with_patterns(strings, patterns, ignore_case = FALSE, any = FALSE)
```

Arguments

`strings`	A character vector.
`patterns`	Regular expressions.
`ignore_case`	Do we want to ignore case when matching patterns?
`any`	Set this to `TRUE` if you want to see which strings match *any* of the patterns and not *all* (all is the default).

Details

For huge character vectors, this can be quite slow and you're better off implementing your own solution with `stringr::str_detect()`.

Value

A character vector of strings matching the patterns.

Examples

```
str_with_patterns(c("abc", "bcd", "cde"), c("b", "c"))
str_with_patterns(c("abc", "bcd", "cde"), c("b", "c"), any = TRUE)
str_with_patterns(toupper(c("abc", "bcd", "cde")), c("b", "c"), any = TRUE)
str_with_patterns(toupper(c("abc", "bcd", "cde")), c("b", "c"), any = TRUE,
                  ignore_case = TRUE)
```

trim_anything *Trim something other than whitespace*

Description

The `stringi` and `stringr` packages let you trim whitespace, but what if you want to trim something else from either (or both) side(s) of a string? This function lets you select which pattern to trim and from which side(s).

Usage

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

Arguments

`string`	A string.
`pattern`	A string. The pattern to be trimmed (*not* interpreted as regular expression). So to trim a period, use `char = "."` and not `char = "\\."`).
`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"`).

Value

A string.

Examples

```
trim_anything("..abcd.", ".", "left")
trim_anything("-ghi--", "-")
trim_anything("-ghi--", "--")
```

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