

# Package ‘edm1’

July 11, 2024

**Title** Set of functions bringing new feature to common functions

**Version** 2.0.0.0

**Description** Provides the ability to perform split operation with multiple elements in input and multiple split pattern, a sub operation with multiple replacor patterns for multiple replaced patterns..., match by another element from another vector...

**License** GPL (==3)

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.1

**Imports** stringr,  
stringi

## Contents

better_match . . . . .	1
better_split . . . . .	2
better_split_any . . . . .	3
better_sub . . . . .	4
better_sub_mult . . . . .	5
better_unique . . . . .	6
grep_all . . . . .	7
grep_all2 . . . . .	7
gsub_mult . . . . .	8
match_by . . . . .	9
sub_mult . . . . .	9

<b>Index</b>	<b>11</b>
--------------	-----------

---

better_match	<i>better_match</i>
--------------	---------------------

---

## Description

Allow to get the nth element matched in a vector

**Usage**

```
better_match(inpt_v = c(), ptrn, until = 1, nvr_here = NA)
```

**Arguments**

inpt_v	is the input vector
ptrn	is the pattern to be matched
until	is the maximum number of matched pattern outputed
nvr_here	is a value you are sure is not present in inpt_v

**Examples**

```
print(better_match(inpt_v=c(1:12, 3, 4, 33, 3), ptrn=3, until=1))
#[1] 3

print(better_match(inpt_v=c(1:12, 3, 4, 33, 3), ptrn=3, until=5))
#[1] 3 13 16

print(better_match(inpt_v=c(1:12, 3, 4, 33, 3), ptrn=c(3, 4), until=5))
[1] 3 13 16 4 14

print(better_match(inpt_v=c(1:12, 3, 4, 33, 3), ptrn=c(3, 4), until=c(1, 5)))
[1] 3 4 14
```

---

better_split	<i>better_split</i>
--------------	---------------------

---

**Description**

Allows to split a string by multiple split, returns a vector and not a list.

**Usage**

```
better_split(inpt, split_v = c())
```

**Arguments**

inpt	is the input character
split_v	is the vector containing the splits

**Examples**

```
print(better_split(inpt = "o-u_i", split_v = c("-")))
```

```
[1] "o"    "u_i"
```

```
print(better_split(inpt = "o-u_i", split_v = c("-", "_")))
```

```
[1] "o" "u" "i"
```

---

better_split_any	<i>better_split_any</i>
------------------	-------------------------

---

**Description**

Allows to split a string by multiple split regardless of their length, returns a vector and not a list. Contrary to better\_split, this functions keep the delimiters in the output.

**Usage**

```
better_split_any(inpt, split_v = c())
```

**Arguments**

inpt	is the input character
split_v	is the vector containing the splits

**Examples**

```
print(better_split_any(inpt = "o-u_i", split_v = c("-")))
```

```
[1] "o"  "-" "u_i"
```

```
print(better_split_any(inpt = "o-u_i", split_v = c("-", "_")))
```

```
[1] "o" "-" "u"  "_" "i"
```

```
print(better_split_any(inpt = "--o--_/m/m/___opo-/m/-u_i_--", split_v = c("--", "_", "/"
```

```
[1] "--"    "o"      "--"      "_"      "/"      "m"      "/"      "m"      "/"
```

```
[10] "_"      "_"      "-opo-"  "/"      "m"      "/"      "-u"     "_"      "i-"
```

```
[19] "_"      "--"
```

better\_sub

*better\_sub***Description**

Allow to perform a sub operation to a given number of matched patterns, see examples

**Usage**

```
better_sub(inpt_v = c(), pattern, replacement, until_v = c())
```

**Arguments**

<code>inpt_v</code>	is a vector containing all the elements that contains expressions to be substituted
<code>pattern</code>	is the expression that will be substituted
<code>replacement</code>	is the expression that will substitute pattern
<code>until_v</code>	is a vector containing, for each element of <code>inpt_v</code> , the number of pattern that will be substituted

**Examples**

```
print(better_sub(inpt_v = c("yes NAME, i will call NAME and NAME",
                           "yes NAME, i will call NAME and NAME"),
               pattern = "NAME",
               replacement = "Kevin",
               until = c(2)))

[1] "yes Kevin, i will call Kevin and NAME"
[2] "yes Kevin, i will call Kevin and NAME"

print(better_sub(inpt_v = c("yes NAME, i will call NAME and NAME",
                           "yes NAME, i will call NAME and NAME"),
               pattern = "NAME",
               replacement = "Kevin",
               until = c(2, 3)))

[1] "yes Kevin, i will call Kevin and NAME"
[2] "yes Kevin, i will call Kevin and Kevin"

print(better_sub(inpt_v = c("yes NAME, i will call NAME and NAME",
                           "yes NAME, i will call NAME and NAME"),
               pattern = "NAME",
               replacement = "Kevin",
               until = c("max", 3)))

[1] "yes Kevin, i will call Kevin and Kevin"
[2] "yes Kevin, i will call Kevin and Kevin"
```

---

better_sub_mult	<i>better_sub_mult</i>
-----------------	------------------------

---

## Description

Allow to perform a sub\_mult operation to a given number of matched patterns, see examples

## Usage

```
better_sub_mult (
  inpt_v = c(),
  pattern_v = c(),
  replacement_v = c(),
  until_v = c()
)
```

## Arguments

inpt_v	is a vector containing all the elements that contains expressions to be substituted
pattern_v	is a vector containing all the patterns to be substituted in any elements of inpt_v
replacement_v	is a vector containing the expression that are going to substitute those provided by pattern_v
until_v	is a vector containing, for each element of inpt_v, the number of pattern that will be substituted

## Examples

```
print(better_sub_mult(inpt_v = c("yes NAME, i will call NAME and NAME2",
                                "yes NAME, i will call NAME and NAME2, especially NAME2"),
  pattern_v = c("NAME", "NAME2"),
  replacement_v = c("Kevin", "Paul"),
  until = c(1, 3)))

[1] "yes Kevin, i will call NAME and Paul"
[2] "yes Kevin, i will call NAME and Paul, especially Paul"

print(better_sub_mult(inpt_v = c("yes NAME, i will call NAME and NAME2",
                                "yes NAME, i will call NAME and NAME2, especially NAME2"),
  pattern_v = c("NAME", "NAME2"),
  replacement_v = c("Kevin", "Paul"),
  until = c("max", 3)))

[1] "yes Kevin, i will call Kevin and Kevin2"
[2] "yes Kevin, i will call Kevin and Kevin2, especially Kevin2"
```

---

better_unique	<i>better_unique</i>
---------------	----------------------

---

## Description

Returns the element that are not unique from the input vector

## Usage

```
better_unique(inpt_v, occu = ">-1-")
```

## Arguments

inpt_v	is the input vector containing the elements
occu	is a parameter that specifies the occurrence of the elements that must be returned, defaults to ">-1-" it means that the function will return all the elements that are present more than one time in inpt_v. The syntax is the following "comparaison_type-actual_value-". The comparaison type may be "==" or ">" or "<". Occu can also be a vector containing all the occurrence that must have the elements to be returned.

## Examples

```
print(better_unique(inpt_v=c("oui", "oui", "non", "non", "peut", "peut1", "non")))
#[1] "oui" "non"

print(better_unique(inpt_v=c("oui", "oui", "non", "non", "peut", "peut1", "non"), occu=">-1-"))
#[1] "oui"

print(better_unique(inpt_v=c("oui", "oui", "non", "non", "peut", "peut1", "non"), occu=">-2-"))
#[1] "non"

print(better_unique(inpt_v=c("oui", "oui", "non", "non", "peut", "peut1", "non"), occu=c("non", "peut", "peut1")))
#[1] "non" "peut" "peut1"

print(better_unique(inpt_v = c("a", "b", "c", "c"), occu = "=="-1-))
[1] "a" "b"

print(better_unique(inpt_v = c("a", "b", "c", "c"), occu = "<-2-"))
[1] "a" "b"
```

---

```
grep_all
```

```
grep_all
```

---

### Description

Allow to perform a grep function on multiple input elements

### Usage

```
grep_all(inpt_v, pattern_v)
```

### Arguments

`inpt_v` is the input vectors to grep elements from  
`pattern_v` is a vector containing the patterns to grep

### Examples

```
print(grep_all(inpt_v = c(1:14, "z", 1:7, "z", "a", "z"),
               pattern_v = c("z", "4")))

[1] 15 23 25 4 14 19

print(grep_all(inpt_v = c(1:14, "z", 1:7, "z", "a", "z"),
               pattern_v = c("z", "^4$")))

[1] 15 23 25 4 19

print(grep_all(inpt_v = c(1:14, "z", 1:7, "z", "a", "z"),
               pattern_v = c("z")))

[1] 15 23 25
```

---

```
grep_all2
```

```
grep_all2
```

---

### Description

Performs the `grep_all` function with another algorithm, potentially faster

### Usage

```
grep_all2(inpt_v, pattern_v)
```

### Arguments

`inpt_v` is the input vectors to grep elements from  
`pattern_v` is a vector containing the patterns to grep

**Examples**

```
print(grep_all2(inpt_v = c(1:14, "z", 1:7, "z", "a", "z"),
               pattern_v = c("z", "4")))

[1] 15 23 25  4 14 19

print(grep_all2(inpt_v = c(1:14, "z", 1:7, "z", "a", "z"),
               pattern_v = c("z", "^4$")))

[1] 15 23 25  4 19

print(grep_all2(inpt_v = c(1:14, "z", 1:7, "z", "a", "z"),
               pattern_v = c("z")))

[1] 15 23 25
```

---

 gsub\_mult

*gsub\_mult*


---

**Description**

Performs a gsub operation with n patterns and replacements.

**Usage**

```
gsub_mult(inpt_v, pattern_v = c(), replacement_v = c())
```

**Arguments**

`inpt_v` is a vector containing all the elements that contains expressions to be substituted

`pattern_v` is a vector containing all the patterns to be substituted in any elements of `inpt_v`

`replacement_v` is a vector containing the expression that are going to substitute those provided by `pattern_v`

**Examples**

```
print(gsub_mult(inpt_v = c("X and Y programming languages are great", "More X, more X!"),
               pattern_v = c("X", "Y", "Z"),
               replacement_v = c("C", "R", "GO")))

[1] "C and R programming languages are great"
[2] "More C, more C!"
```



---

match_by	<i>match_by</i>
----------	-----------------

---

### Description

Allow to match elements by ids, see examples.

### Usage

```
match_by(to_match_v = c(), inpt_v = c(), inpt_ids = c())
```

### Arguments

to_match_v	is the vector containing all the elements to match
inpt_v	is the input vector containong all the elements that could contains the elements to match. Each elements is linked to an element from inpt_ids at any given index, see examples. So inpt_v and inpt_ids must be the same size
inpt_ids	is the vector containing all the ids for the elements in inpt_v. An element is linked to the id x is both are at the same index. So inpt_v and inpt_ids must be the same size

### Examples

```
print(match_by(to_match_v = c("a"), inpt_v = c("a", "z", "a", "p", "p", "e", "e", "a"),
              inpt_ids = c(1, 1, 1, 2, 2, 3, 3, 3)))

[1] 1 8

print(match_by(to_match_v = c("a"), inpt_v = c("a", "z", "a", "a", "p", "e", "e", "a"),
              inpt_ids = c(1, 1, 1, 2, 2, 3, 3, 3)))

[1] 1 4 8

print(match_by(to_match_v = c("a", "e"), inpt_v = c("a", "z", "a", "a", "p", "e", "e", "a"),
              inpt_ids = c(1, 1, 1, 2, 2, 3, 3, 3)))

[1] 1 4 8 6
```

---

sub_mult	<i>sub_mult</i>
----------	-----------------

---

### Description

Performs a sub operation with n patterns and replacements.

### Usage

```
sub_mult(inpt_v, pattern_v = c(), replacement_v = c())
```

**Arguments**

`inpt_v` is a vector containing all the elements that contains expressions to be substituted  
`pattern_v` is a vector containing all the patterns to be substituted in any elements of `inpt_v`  
`replacement_v` is a vector containing the expression that are going to substitute those provided by `pattern_v`

**Examples**

```
print(sub_mult(inpt_v = c("X and Y programming languages are great", "More X, more X!"),
              pattern_v = c("X", "Y", "Z"),
              replacement_v = c("C", "R", "GO")))
```

[1] "C and R programming languages are great"  
[2] "More C, more X!"

# Index

`better_match`, [1](#)  
`better_split`, [2](#)  
`better_split_any`, [3](#)  
`better_sub`, [4](#)  
`better_sub_mult`, [5](#)  
`better_unique`, [6](#)  
  
`grep_all`, [7](#)  
`grep_all2`, [7](#)  
`gsub_mult`, [8](#)  
  
`match_by`, [9](#)  
  
`sub_mult`, [9](#)