Package 'incase'

May 8, 2026

Type Package

Title Pipe-Friendly Vector Replacement with Case Statements

Version 0.4.0

Description Offers a pipe-friendly alternative to the 'dplyr' functions case_when() and if_else(), as well as a number of user-friendly simplifications for common use cases. These functions accept a vector as an optional first argument, allowing conditional statements to be built using the 'magrittr' dot operator. The functions also coerce all outputs to the same type, meaning you no longer have to worry about using specific typed variants of NA or explicitly declaring integer outputs, and evaluate outputs somewhat lazily, so you don't waste time on long operations that won't be used.

License MIT + file LICENSE

URL https://pkg.rossellhayes.com/incase/, https://github.com/rossellhayes/incase

BugReports https://github.com/rossellhayes/incase/issues

Imports backports, cli, glue, lifecycle, magrittr, plu, rlang

Suggests dplyr, testthat (>= 3.0.0), tibble

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation no

Author Alexander Rossell Hayes [aut, cre, cph] (ORCID: <https://orcid.org/0000-0001-9412-0457>), Patrice Kiener [ctb] (Contributed example for fn_case(), ORCID: <https://orcid.org/0000-0002-0505-9920>)

Maintainer Alexander Rossell Hayes <alexander@rossellhayes.com>

Repository CRAN

Date/Publication 2025-07-24 04:30:07 UTC

Contents

`fn_case` *Case statements applying a function to all inputs*

Description

Case statements applying a function to all inputs

Usage

```
fn_case(
  x,
  fn,
  ...,
  .preserve = FALSE,
  .default = NA,
  .exhaustive = FALSE,
  preserve = deprecated(),
  default = deprecated()
)
```

Arguments

`x`	A vector
`fn`	A function to apply to the left-hand side of each formula in `...` Either a quoted or unquoted function name, an anonymous `function`, or a purrr-style formula. The function should take two inputs, the first being `x` and the second being the left-hand side of the formula. The function should return a logical vector, either of length 1 or the same length as `x`.
`...`	<`dynamic-dots`> A sequence of two-sided formulas or named arguments. • **Formulas**: Elements of `x` that return `TRUE` when passed to `fn` with the left hand side (LHS) of each formula will be replaced with the value in the right hand side (RHS). The LHS must evaluate to a logical vector when passed to `fn` with `x`. The RHS must be of length 1 or the same length as all other RHS.

- **Named arguments**: named arguments are passed as additional arguments to the function `fn`.

`.preserve`, `preserve`
: If `TRUE`, unmatched elements of `x` will be returned unmodified. (The elements may have their type coerced to be compatible with replacement values.) If `FALSE`, unmatched elements of `x` will be replaced with `.default`. Defaults to `FALSE`.

`.default`, `default`
: If `.preserve` is `FALSE`, a value to replace unmatched elements of `x`. Defaults to `NA`.

`.exhaustive`
: If `TRUE`, unmatched elements of `x` will result in an error. This can be useful to ensure you aren't accidentally forgetting to recode any values. Defaults to `FALSE`.

  Note that if `.preserve` is `TRUE`, `.exhaustive` will never have any effect.

Value

A vector of length 1 or n, matching the length of the logical input or output vectors. Inconsistent lengths will generate an error.

See Also

`fn_case_fct()` to return a factor and `fn_case_list()` to return a list

`fn_switch_case()`, which applies a function to each formula's LHS, but not `x`

`switch_case()`, a simpler alternative for exact matching

`grep_case()`, a simpler alternative for regex pattern matching

`in_case()`, a pipeable alternative to `dplyr::case_when()`

Examples

```
# Replicate switch_case()
parties <- sample(c("d", "r", "i", "g", "l"), 20, replace = TRUE)

fn_case(
  parties,
  fn = `%in%`,
  "d" ~ "Democrat",
  "r" ~ "Republican",
  "i" ~ "Independent",
  "g" ~ "Green",
  "l" ~ "Libertarian"
)

# Replicate grep_case()
countries <- c(
  "France", "Ostdeutschland", "Westdeutschland", "Nederland",
  "Belgie (Vlaanderen)", "Belgique (Wallonie)", "Luxembourg", "Italia"
)
```

```
fn_case(
  countries,
  fn = function(x, pattern, ...) {grepl(pattern, x, ...)},
  "Deutschland" ~ "Germany",
  "Belgi(qu)?e" ~ "Belgium",
  "Nederland"   ~ "Netherlands",
  "Italia"      ~ "Italy",
  .preserve     = TRUE,
  ignore.case   = TRUE
)

fn_case(
  countries,
  fn = ~ grepl(.y, .x),
  "Deutschland" ~ "Germany",
  "Belgi(qu)?e" ~ "Belgium",
  "Nederland"   ~ "Netherlands",
  "Italia"      ~ "Italy",
  .preserve     = TRUE,
  ignore.case   = TRUE
)

# Recode values in a range
time    <- runif(10, 1, 12)
hours   <- time %/% 1
minutes <- time %% 1 * 60

hours <- hours %>%
  if_case(minutes > 32.5, (. + 1) %% 12, .) %>%
  switch_case(0 ~ 12, .preserve = TRUE)

minutes %>%
  fn_case(
    fn = ~ abs(.x - .y) <= 2.5,
    0  ~ "o'clock",
    60 ~ "o'clock",
    30 ~ "half past",
    15 ~ "quarter past",
    45 ~ "quarter to",
    5  ~ "five past",
    10 ~ "ten past",
    20 ~ "twenty past",
    25 ~ "twenty-five past",
    55 ~ "five to",
    50 ~ "ten to",
    40 ~ "twenty to",
    35 ~ "twenty-five to"
  ) %>%
  switch_case(
    "o'clock" ~ paste(hours, .),
    .default  = paste(., hours)
  )
```

```
# Replicate vctrs::vec_ptype_abbr() (used for tibble column labels)
# Based on a contribution by Patrice Kiener
in_herits <- function(x) {
  fn_case(
    x,
    fn = inherits,
    "factor"     ~ "fct",
    "character"  ~ "chr",
    "numeric"    ~ "dbl",
    "integer"    ~ "int",
    "logical"    ~ "lgl",
    "complex"    ~ "cpl",
    "raw"        ~ "raw",
    "matrix"     ~ "mat",
    "array"      ~ "arr",
    "data.frame" ~ "df",
    "list"       ~ "lst",
    "function"   ~ "fn",
    .default     = class(x)[[1]]
  )
}

in_herits(1:3)
in_herits(letters[1:3])
in_herits(fn_case)
```

---

grep_case *Switch-style recoding of values with string pattern matching*

Description

Switch-style recoding of values with string pattern matching

Usage

```
grep_case(
  x,
  ...,
  .preserve = FALSE,
  .default = NA,
  .exhaustive = FALSE,
  preserve = deprecated(),
  default = deprecated()
)
```

Arguments

x	A vector
...	<dynamic-dots> A sequence of two-sided formulas or named arguments.

- **Formulas**: Elements of `x` that match the regex pattern on the left hand side (LHS) of formulas will be replaced with the value in the right hand side (RHS). The LHS must evaluate to a character string. The RHS must be of length one. `NULL` inputs are ignored.
- **Named arguments**: named arguments are passed to `grepl()`.

`.preserve`, `preserve`
: If `TRUE`, unmatched elements of `x` will be returned unmodified. (The elements may have their type coerced to be compatible with replacement values.) If `FALSE`, unmatched elements of `x` will be replaced with `.default`. Defaults to `FALSE`.

`.default`, `default`
: If `.preserve` is `FALSE`, a value to replace unmatched elements of `x`. Defaults to `NA`.

`.exhaustive`
: If `TRUE`, unmatched elements of `x` will result in an error. This can be useful to ensure you aren't accidentally forgetting to recode any values. Defaults to `FALSE`.

  Note that if `.preserve` is `TRUE`, `.exhaustive` will never have any effect.

Value

A vector of the same length as `x`.

See Also

`grep_case_fct()` to return a factor and `grep_case_list()` to return a list

`fn_case()`, to apply a function other than `grepl()` to each case

`switch_case()` to recode values with exact matching

`in_case()`, a pipeable alternative to `dplyr::case_when()`

`switch()` and `grepl()`, which inspired this function

Examples

```
words <- c("caterpillar", "dogwood", "catastrophe", "dogma")

grep_case(
  words,
  "cat" ~ "feline",
  "dog" ~ "canine"
)

caps_words <- c("caterpillar", "dogwood", "Catastrophe", "DOGMA")

grep_case(
  caps_words,
  "cat" ~ "feline",
  "dog" ~ "canine",
  ignore.case = TRUE
)
```

```
countries <- c(
  "France", "Ostdeutschland", "Westdeutschland", "Nederland",
  "Belgie (Vlaanderen)", "Belgique (Wallonie)", "Luxembourg", "Italia"
)

grep_case(
  countries,
  "Deutschland" ~ "Germany",
  "Belgi(qu)?e" ~ "Belgium",
  "Nederland"   ~ "Netherlands",
  "Italia"      ~ "Italy",
  .preserve     = TRUE,
  ignore.case   = TRUE
)
```

`if_case` *Pipe-friendly vectorized if*

Description

Compared to `dplyr::if_else()`, this function is easier to use with a pipe. A vector piped into this function will be quietly ignored. This allows magrittr dots to be used in arguments without requiring workarounds like wrapping the function in braces.

Usage

```
if_case(condition, true, false, missing = NA, ...)
```

Arguments

`condition`	Logical vector
`true`, `false`, `missing`	Values to use for `TRUE`, `FALSE`, and `NA` values of `condition`. They must be either the same length as `condition`, or length 1.
`...`	Values passed to `...` produce an error. This facilitates the quiet ignoring of a piped vector.

Details

This function is also less strict than `dplyr::if_else()`. If `true`, `false`, and `missing` are different types, they are silently coerced to a common type.

Value

Where `condition` is `TRUE`, the matching value from `true`; where it's `FALSE`, the matching value from `false`; and where it's `NA`, the matching value from `missing`.

See Also

in_case(), a pipeable alternative to dplyr::case_when()

switch_case(), a reimplementation of switch()

dplyr::if_else(), from which this function is derived

Examples

```
x <- c(1, 2, 5, NA)

# if_case() produces the same output as dplyr::if_else()
if_case(x > 3, "high", "low", "missing")
dplyr::if_else(x > 3, "high", "low", "missing")

# if_case() does not throw an error if arguments are not of the same type
if_case(x > 3, "high", "low", NA)
try(dplyr::if_else(x > 3, "high", "low", NA))

# if_case() can accept a piped input without an error or requiring braces
x %>% if_case(. > 3, "high", "low", "missing")
try(x %>% dplyr::if_else(. > 3, "high", "low", "missing"))
x %>% {dplyr::if_else(. > 3, "high", "low", "missing")}

# You can also pipe a conditional test like dplyr::if_else()
{x > 3} %>% if_case("high", "low", "missing")
{x > 3} %>% dplyr::if_else("high", "low", "missing")
```

---

in_case *A pipe-friendly general vectorized if*

Description

This function allows you to vectorize multiple if_else() statements. If no cases match, NA is returned. This function derived from dplyr::case_when(). Unlike dplyr::case_when(), in_case() supports piping elegantly and attempts to handle inconsistent types (see examples).

Usage

```
in_case(
  ...,
  .preserve = FALSE,
  .default = NA,
  preserve = deprecated(),
  default = deprecated()
)
```

Arguments

`...`	<dynamic-dots> A sequence of two-sided formulas. The left hand side (LHS) determines which values match this case. The right hand side (RHS) provides the replacement value. The LHS must evaluate to a logical vector. Both LHS and RHS may have the same length of either 1 or `n`. The value of `n` must be consistent across all cases. The case of `n == 0` is treated as a variant of `n != 1`. `NULL` inputs are ignored.
`.preserve`	If `TRUE`, unmatched elements of the input will be returned unmodified. (The elements may have their type coerced to be compatible with replacement values.) If `FALSE`, unmatched elements of the input will be replaced with `.default`. Defaults to `FALSE`.
`.default`	If `.preserve` is `FALSE`, a value to replace unmatched elements of the input. Defaults to `NA`.
`preserve, default`	**[Deprecated]** Deprecated in favor of `.preserve` and `.default`

Value

A vector of length 1 or n, matching the length of the logical input or output vectors. Inconsistent lengths will generate an error.

See Also

`in_case_fct()` to return a factor and `in_case_list()` to return a list

`switch_case()` a simpler alternative for when each case involves `==` or `%in%`

`fn_case()`, a simpler alternative for when each case uses the same function

`if_case()`, a pipeable alternative to `dplyr::if_else()`

`dplyr::case_when()`, from which this function is derived

Examples

```
# Non-piped statements are handled the same as dplyr::case_when()
x <- 1:30
in_case(
  x %% 15 == 0 ~ "fizz buzz",
  x %%  3 == 0 ~ "fizz",
  x %%  5 == 0 ~ "buzz",
  TRUE         ~ x
)

# A vector can be directly piped into in_case() without error
1:30 %>%
  in_case(
    . %% 15 == 0 ~ "fizz buzz",
    . %%  3 == 0 ~ "fizz",
```

```
    . %%  5 == 0 ~ "buzz",
    TRUE         ~ .
  )

# in_case() silently converts types
1:30 %>%
  in_case(
    . %% 15 == 0 ~ 35,
    . %%  3 == 0 ~ 5,
    . %%  5 == 0 ~ 7,
    TRUE         ~ NA
  )

x <- 1:30
try(
  dplyr::case_when(
    x %% 15 == 0 ~ 35,
    x %%  3 == 0 ~ 5,
    x %%  5 == 0 ~ 7,
    TRUE         ~ NA
  )
)

# .default and .preserve make it easier to handle unmatched values
1:30 %>%
  in_case(
    . %% 15 == 0 ~ "fizz buzz",
    . %%  3 == 0 ~ "fizz",
    . %%  5 == 0 ~ "buzz",
    .default     = "pass"
  )

1:30 %>%
  in_case(
    . %% 15 == 0 ~ "fizz buzz",
    . %%  3 == 0 ~ "fizz",
    . %%  5 == 0 ~ "buzz",
    .preserve    = TRUE
  )
```

`in_case_fct` *Case statements returning a factor*

Description

These functions are equivalent to `in_case()`, `switch_case()`, `grep_case()`, `fn_case()`, and `fn_switch_case()` but return factors with their levels determined by the order of their case statements.

Usage

```
in_case_fct(
  ...,
  .preserve = FALSE,
  .default = NA,
  .ordered = FALSE,
  preserve = deprecated(),
  default = deprecated(),
  ordered = deprecated()
)

switch_case_fct(
  x,
  ...,
  .preserve = FALSE,
  .default = NA,
  .ordered = FALSE,
  .exhaustive = FALSE,
  preserve = deprecated(),
  default = deprecated(),
  ordered = deprecated()
)

grep_case_fct(
  x,
  ...,
  .preserve = FALSE,
  .default = NA,
  .ordered = FALSE,
  .exhaustive = FALSE,
  preserve = deprecated(),
  default = deprecated(),
  ordered = deprecated()
)

fn_case_fct(
  x,
  fn,
  ...,
  .preserve = FALSE,
  .default = NA,
  .ordered = FALSE,
  .exhaustive = FALSE,
  preserve = deprecated(),
  default = deprecated(),
  ordered = deprecated()
)
```

```
fn_switch_case_fct(
  x,
  fn,
  ...,
  .preserve = FALSE,
  .default = NA,
  .ordered = FALSE,
  .exhaustive = FALSE,
  preserve = deprecated(),
  default = deprecated(),
  ordered = deprecated()
)
```

Arguments

`...`	<dynamic-dots> A sequence of two-sided formulas or named arguments. • **Formulas**: Elements of `x` that return `TRUE` when passed to `fn` with the left hand side (LHS) of each formula will be replaced with the value in the right hand side (RHS). The LHS must evaluate to a logical vector when passed to `fn` with `x`. The RHS must be of length 1 or the same length as all other RHS. • **Named arguments**: named arguments are passed as additional arguments to the function `fn`.
`.preserve`	If `TRUE`, unmatched elements of `x` will be returned unmodified. (The elements may have their type coerced to be compatible with replacement values.) If `FALSE`, unmatched elements of `x` will be replaced with `.default`. Defaults to `FALSE`.
`.default`	If `.preserve` is `FALSE`, a value to replace unmatched elements of `x`. Defaults to `NA`.
`.ordered`	A logical. If `TRUE`, returns an ordered factor. If `FALSE`, returns an unordered factor.
`preserve, default, ordered`	**[Deprecated]** Deprecated in favor of `.preserve`, `.default`, and `.ordered`
`x`	A vector
`.exhaustive`	If `TRUE`, unmatched elements of `x` will result in an error. This can be useful to ensure you aren't accidentally forgetting to recode any values. Defaults to `FALSE`. Note that if `.preserve` is `TRUE`, `.exhaustive` will never have any effect.
`fn`	A function to apply to the left-hand side of each formula in `...` Either a quoted or unquoted function name, an anonymous `function`, or a purrr-style formula. The function should take two inputs, the first being `x` and the second being the left-hand side of the formula. The function should return a logical vector, either of length 1 or the same length as `x`.

Value

A factor vector of length 1 or n, matching the length of the logical input or output vectors. Levels are determined by the order of inputs to `...` and `.default`. Inconsistent lengths will generate an error.

The position of the `.default` argument is taken into account when setting factor levels in `*_case_fct()` functions. For example, if the `.default` argument is given before any case statements, the default value will be the first level of the factor; if the `.default` argument is positioned in between two case statements, the default value will be ordered in between the value of the two case statements.

See Also

`in_case()`, `switch_case()`, `grep_case()`, `fn_case()`, and `fn_case_fct()` on which these functions are based.

Examples

```
1:10 %>%
  in_case_fct(
    . %% 2 == 0 ~ "even",
    . %% 2 == 1 ~ "odd"
  )

switch_case_fct(
  c("a", "b", "c"),
  "c" ~ "cantaloupe",
  "b" ~ "banana",
  "a" ~ "apple"
)

switch_case_fct(
  c("a", "b", "c", "d"),
  "c" ~ "cantaloupe",
  "b" ~ "banana",
  "a" ~ "apple"
)

switch_case_fct(
  c("a", "b", "c", "d"),
  "c" ~ "cantaloupe",
  "b" ~ "banana",
  "a" ~ "apple",
  .preserve = TRUE
)

switch_case_fct(
  c("a", "b", "c", "d"),
  "c" ~ "cantaloupe",
  "b" ~ "banana",
  "a" ~ "apple",
  .default = "other"
)
```

```
switch_case_fct(
  c("a", "b", "c", "d"),
  .default = "other",
  "c" ~ "cantaloupe",
  "b" ~ "banana",
  "a" ~ "apple"
)

switch_case_fct(
  c("a", "b", "c", "d"),
  "c" ~ "cantaloupe",
  "b" ~ "banana",
  .default = "other",
  "a" ~ "apple"
)

grep_case_fct(
  c("caterpillar", "dogwood", "catastrophe", "dogma"),
  "cat" ~ "feline",
  "dog" ~ "canine"
)

fn_case_fct(
  c("a", "b", "c"),
  `%in%`,
  "c" ~ "cantaloupe",
  "b" ~ "banana",
  "a" ~ "apple"
)
```

`in_case_list` *Case statements returning a list*

Description

These functions are equivalent to `in_case()`, `switch_case()`, `grep_case()`, `fn_case()`, and `fn_switch_case()` but return lists.

Usage

```
in_case_list(
  ...,
  .preserve = FALSE,
  .default = NA,
  preserve = deprecated(),
  default = deprecated()
)
```

```
switch_case_list(
  x,
  ...,
  .preserve = FALSE,
  .default = NA,
  .exhaustive = FALSE,
  preserve = deprecated(),
  default = deprecated()
)

grep_case_list(
  x,
  ...,
  .preserve = FALSE,
  .default = NA,
  .exhaustive = FALSE,
  preserve = deprecated(),
  default = deprecated()
)

fn_case_list(
  x,
  fn,
  ...,
  .preserve = FALSE,
  .default = NA,
  .exhaustive = FALSE,
  preserve = deprecated(),
  default = deprecated()
)

fn_switch_case_list(
  x,
  fn,
  ...,
  .preserve = FALSE,
  .default = NA,
  .exhaustive = FALSE,
  preserve = deprecated(),
  default = deprecated()
)
```

Arguments

`...` <dynamic-dots> A sequence of two-sided formulas or named arguments.

- **Formulas**: Elements of `x` that return `TRUE` when passed to `fn` with the left hand side (LHS) of each formula will be replaced with the value in the right hand side (RHS). The LHS must evaluate to a logical vector when passed

to fn with x. The RHS must be of length 1 or the same length as all other RHS.

- **Named arguments**: named arguments are passed as additional arguments to the function fn.

.preserve, preserve
: If TRUE, unmatched elements of x will be returned unmodified. (The elements may have their type coerced to be compatible with replacement values.) If FALSE, unmatched elements of x will be replaced with .default. Defaults to FALSE.

.default, default
: If .preserve is FALSE, a value to replace unmatched elements of x. Defaults to NA.

x
: A vector

.exhaustive
: If TRUE, unmatched elements of x will result in an error. This can be useful to ensure you aren't accidentally forgetting to recode any values. Defaults to FALSE.

  Note that if .preserve is TRUE, .exhaustive will never have any effect.

fn
: A function to apply to the left-hand side of each formula in ...

  Either a quoted or unquoted function name, an anonymous function, or a purrr-style formula.

  The function should take two inputs, the first being x and the second being the left-hand side of the formula. The function should return a logical vector, either of length 1 or the same length as x.

Details

This can be useful when returning a non-atomic value and/or when you want to create a list column inside a tibble.

Value

A list of length 1 or n, matching the length of the logical input vector.

See Also

in_case(), switch_case(), grep_case(), fn_case(), and fn_case_fct() on which these functions are based.

Examples

```
1:3 %>%
  in_case_list(
    . < 2    ~ mtcars,
    .default = letters
  )
```

`switch_case` *Switch-style recoding of values*

Description

Switch-style recoding of values

Usage

```
switch_case(
  x,
  ...,
  .preserve = FALSE,
  .default = NA,
  .exhaustive = FALSE,
  preserve = deprecated(),
  default = deprecated()
)

fn_switch_case(
  x,
  fn,
  ...,
  .preserve = FALSE,
  .default = NA,
  .exhaustive = FALSE,
  preserve = deprecated(),
  default = deprecated()
)
```

Arguments

`x`	A vector
`...`	<dynamic-dots> A sequence of two-sided formulas or named arguments. • **Formulas**: Elements of `x` that match the left hand side (LHS) of formulas will be replaced with the value in the right hand side (RHS). The LHS must evaluate to an atomic vector. The RHS must be of length one. `NULL` inputs are ignored. • **Named arguments**: for `fn_switch_case()`, named arguments are passed to the function `fn`. For `switch_case()`, named arguments will raise an error.
`.preserve`	If `TRUE`, unmatched elements of `x` will be returned unmodified. (The elements may have their type coerced to be compatible with replacement values.) If `FALSE`, unmatched elements of `x` will be replaced with `.default`. Defaults to `FALSE`.

`.default` If `.preserve` is `FALSE`, a value to replace unmatched elements of `x`. Defaults to `NA`.

`.exhaustive` If `TRUE`, unmatched elements of `x` will result in an error. This can be useful to ensure you aren't accidentally forgetting to recode any values. Defaults to `FALSE`.

Note that if `.preserve` is `TRUE`, `.exhaustive` will never have any effect.

`preserve, default`

[Deprecated] Deprecated in favor of `.preserve` and `.default`

`fn` A function to apply to the left-hand side of each formula in `...`

Value

A vector of the same length as `x`.

See Also

`switch_case_fct()` and `fn_switch_case_fct()` to return a factor and `switch_case_list()` and `fn_switch_case_list()` to return a list

`grep_case()` to recode values with string pattern matching

`fn_case()`, which applies a function to both `x` and each formula's LHS

`in_case()`, a pipeable alternative to `dplyr::case_when()`

`switch()` and `%in%`, which inspired this function

Examples

```
parties <- sample(c("d", "r", "i", "g", "l"), 20, replace = TRUE)

switch_case(
  parties,
  "d" ~ "Democrat",
  "r" ~ "Republican",
  "i" ~ "Independent",
  "g" ~ "Green",
  "l" ~ "Libertarian"
)

parties %>%
  switch_case(
    "d" ~ "Democrat",
    "r" ~ "Republican",
    "i" ~ "Independent",
    "g" ~ "Green",
    "l" ~ "Libertarian"
  )

parties %>%
  switch_case(
    "d" ~ "Democrat",
    "r" ~ "Republican",
```

```
    c("i", "g", "l") ~ "Other"
  )

parties %>%
  switch_case(
    "d" ~ "Democrat",
    "r" ~ "Republican",
    .default = "Other"
  )

parties %>%
  switch_case(
    "d" ~ "Democrat",
    "r" ~ "Republican",
    .preserve = FALSE
  )

parties %>%
  switch_case(
    "d" ~ "Democrat",
    "r" ~ "Republican",
    .preserve = TRUE
  )

data <- c(1, 4, 8, 12, 999, 6, 2, 888, 4, 6, 777)

fn_switch_case(
  data,
  function(x) paste(rep(x, 3), collapse = ""),
  7 ~ "Not asked",
  8 ~ "Refused",
  9 ~ "Missing",
  .preserve = TRUE
)
```

Index