# Package 'magrittr'

July 22, 2025

**Type** Package

**Title** A Forward-Pipe Operator for R

**Version** 2.0.3

**Description** Provides a mechanism for chaining commands with a new forward-pipe operator, %>%. This operator will forward a value, or the result of an expression, into the next function call/expression. There is flexible support for the type of right-hand side expressions. For more information, see package vignette. To quote Rene Magritte, ``Ceci n'est pas un pipe."

**License** MIT + file LICENSE

**URL** https://magrittr.tidyverse.org, https://github.com/tidyverse/magrittr

**BugReports** https://github.com/tidyverse/magrittr/issues

**Depends** R (>= 3.4.0)

**Suggests** covr, knitr, rlang, rmarkdown, testthat

**VignetteBuilder** knitr

**ByteCompile** Yes

**Config/Needs/website** tidyverse/tidytemplate

**Encoding** UTF-8

**RoxygenNote** 7.1.2

**NeedsCompilation** yes

**Author** Stefan Milton Bache [aut, cph] (Original author and creator of magrittr),
Hadley Wickham [aut],
Lionel Henry [cre],
RStudio [cph, fnd]

**Maintainer** Lionel Henry <lionel@rstudio.com>

**Repository** CRAN

**Date/Publication** 2022-03-30 07:30:09 UTC

# Contents



---

`debug_fseq` *Debugging function for functional sequences.*

---

## Description

This is a utility function for marking functions in a functional sequence for debbuging.

## Usage

```
debug_fseq(fseq, ...)

undebug_fseq(fseq)
```

## Arguments

| | |
|---|---|
| `fseq` | a functional sequence. |
| `...` | indices of functions to debug. |

## Value

`invisible(NULL)`.

## debug_pipe *Debugging function for magrittr pipelines.*

### Description

This function is a wrapper around `browser`, which makes it easier to debug at certain places in a magrittr pipe chain.

### Usage

```
debug_pipe(x)
```

### Arguments

| | |
|---|---|
| x | a value |

### Value

x

## extract *Aliases*

### Description

magrittr provides a series of aliases which can be more pleasant to use when composing chains using the `%>%` operator.

### Details

Currently implemented aliases are

```
extract                 `[`
 extract2               `[[`
 inset                  `[<-`
 inset2                 `[[<-`
 use_series             `$`
 add                    `+`
 subtract               `-`
 multiply_by            `*`
 raise_to_power         `^`
 multiply_by_matrix     `%*%`
 divide_by              `/`
 divide_by_int          `%/%`
 mod                    `%%`
```

| | |
|---|---|
| is_in | `%in%` |
| and | `&` |
| or | `\|` |
| equals | `==` |
| is_greater_than | `>` |
| is_weakly_greater_than | `>=` |
| is_less_than | `<` |
| is_weakly_less_than | `<=` |
| not (`n'est pas`) | `¡` |
| set_colnames | `colnames<-` |
| set_rownames | `rownames<-` |
| set_names | `names<-` |
| set_class | `class<-` |
| set_attributes | `attributes<-` |
| set_attr | `attr<-` |

## Examples

```
 iris %>%
   extract(, 1:4) %>%
   head

good.times <-
  Sys.Date() %>%
  as.POSIXct %>%
  seq(by = "15 mins", length.out = 100) %>%
  data.frame(timestamp = .)

good.times$quarter <-
  good.times %>%
  use_series(timestamp) %>%
  format("%M") %>%
  as.numeric %>%
  divide_by_int(15) %>%
  add(1)
```

---

`faq-pipe-gender` *FAQ: What is the gender of the pipe?*

---

## Description

In Magritte's original quote "Ceci n'est pas une pipe," the word "pipe" is feminine. However the magrittr package quotes it as "Ceci n'est pas un pipe," with a masculine "pipe." This lighthearted misappropriation is intentional. Whereas the object represented in Magritte's painting (a pipe that you can smoke) is feminine in the French language, a computer pipe (which is an Anglicism in French) is masculine.

## freduce *Apply a list of functions sequentially*

### Description

This function applies the first function to `value`, then the next function to the result of the previous function call, etc.

### Usage

```
freduce(value, function_list)
```

### Arguments

| | |
|---|---|
| `value` | initial value. |
| `function_list` | a list of functions. |

### Value

The result after applying each function in turn.

## functions *Extract the function list from a functional sequence.*

### Description

This can be used to extract the list of functions inside a functional sequence created with a chain like `. %>% foo %>% bar`.

### Usage

```
functions(fseq)
```

### Arguments

| | |
|---|---|
| `fseq` | A functional sequence ala magrittr. |

### Value

a list of functions

pipe-eager *Eager pipe*

## Description

Whereas %>% is lazy and only evaluates the piped expressions when needed, %!>% is eager and evaluates the piped input at each step. This produces more intuitive behaviour when functions are called for their side effects, such as displaying a message.

Note that you can also solve this by making your function strict. Call force() on the first argument in your function to force sequential evaluation, even with the lazy %>% pipe. See the examples section.

## Usage

```
lhs %!>% rhs
```

## Arguments

| | |
|---|---|
| lhs | A value or the magrittr placeholder. |
| rhs | A function call using the magrittr semantics. |

## Examples

```
f <- function(x) {
  message("foo")
  x
}
g <- function(x) {
  message("bar")
  x
}
h <- function(x) {
  message("baz")
  invisible(x)
}

# The following lazy pipe sequence is equivalent to `h(g(f()))`.
# Given R's lazy evaluation behaviour,`f()` and `g()` are lazily
# evaluated when `h()` is already running. This causes the messages
# to appear in reverse order:
NULL %>% f() %>% g() %>% h()

# Use the eager pipe to fix this:
NULL %!>% f() %!>% g() %!>% h()

# Or fix this by calling `force()` on the function arguments
f <- function(x) {
  force(x)
  message("foo")
```

```
  x
}
g <- function(x) {
  force(x)
  message("bar")
  x
}
h <- function(x) {
  force(x)
  message("baz")
  invisible(x)
}

# With strict functions, the arguments are evaluated sequentially
NULL %>% f() %>% g() %>% h()

# Instead of forcing, you can also check the type of your functions.
# Type-checking also has the effect of making your function lazy.
```

---

`print.fseq` *Print method for functional sequence.*

---

## Description

Print method for functional sequence.

## Usage

```
## S3 method for class 'fseq'
print(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | A functional sequence object |
| `...` | not used. |

## Value

x

## [[.fseq *Extract function(s) from a functional sequence.*

### Description

Functional sequences can be subset using single or double brackets. A single-bracket subset results in a new functional sequence, and a double-bracket subset results in a single function.

### Usage

```
## S3 method for class 'fseq'
x[[...]]

## S3 method for class 'fseq'
x[...]
```

### Arguments

| | |
|---|---|
| x | A functional sequence |
| ... | index/indices. For double brackets, the index must be of length 1. |

### Value

A function or functional sequence.

## %<>% *Assignment pipe*

### Description

Pipe an object forward into a function or call expression and update the `lhs` object with the resulting value.

### Usage

```
lhs %<>% rhs
```

### Arguments

| | |
|---|---|
| lhs | An object which serves both as the initial value and as target. |
| rhs | a function call using the magrittr semantics. |

### Details

The assignment pipe, `%<>%`, is used to update a value by first piping it into one or more `rhs` expressions, and then assigning the result. For example, `some_object %<>% foo %>% bar` is equivalent to `some_object <- some_object %>% foo %>% bar`. It must be the first pipe-operator in a chain, but otherwise it works like `%>%`.

### See Also

`%>%`, `%T>%`, `%$%`

### Examples

```
iris$Sepal.Length %<>% sqrt

x <- rnorm(100)

x %<>% abs %>% sort

is_weekend <- function(day)
{
   # day could be e.g. character a valid representation
   day %<>% as.Date

   result <- day %>% format("%u") %>% as.numeric %>% is_greater_than(5)

   if (result)
     message(day %>% paste("is a weekend!"))
   else
     message(day %>% paste("is not a weekend!"))

   invisible(result)
}
```

---

## %$% *Exposition pipe*

---

### Description

Expose the names in `lhs` to the `rhs` expression. This is useful when functions do not have a built-in data argument.

### Usage

```
lhs %$% rhs
```

### Arguments

| | |
|---|---|
| `lhs` | A list, environment, or a data.frame. |
| `rhs` | An expression where the names in lhs is available. |

## Details

Some functions, e.g. `lm` and `aggregate`, have a data argument, which allows the direct use of names inside the data as part of the call. This operator exposes the contents of the left-hand side object to the expression on the right to give a similar benefit, see the examples.

## See Also

%>%, %<>%, %T>%

## Examples

```
iris %>%
  subset(Sepal.Length > mean(Sepal.Length)) %$%
  cor(Sepal.Length, Sepal.Width)

data.frame(z = rnorm(100)) %$%
  ts.plot(z)
```

---

# %>% *Pipe*

---

## Description

Pipe an object forward into a function or call expression.

## Usage

```
lhs %>% rhs
```

## Arguments

| | |
|---|---|
| lhs | A value or the magrittr placeholder. |
| rhs | A function call using the magrittr semantics. |

## Details

**Using %>% with unary function calls:**
When functions require only one argument, `x %>% f` is equivalent to `f(x)` (not exactly equivalent; see technical note below.)

**Placing `lhs` as the first argument in `rhs` call:**
The default behavior of %>% when multiple arguments are required in the `rhs` call, is to place `lhs` as the first argument, i.e. `x %>% f(y)` is equivalent to `f(x, y)`.

**Placing `lhs` elsewhere in `rhs` call:**
Often you will want `lhs` to the `rhs` call at another position than the first. For this purpose you can use the dot (`.`) as placeholder. For example, `y %>% f(x, .)` is equivalent to `f(x, y)` and `z %>% f(x, y, arg = .)` is equivalent to `f(x, y, arg = z)`.

**Using the dot for secondary purposes:**

Often, some attribute or property of `lhs` is desired in the `rhs` call in addition to the value of `lhs` itself, e.g. the number of rows or columns. It is perfectly valid to use the dot placeholder several times in the `rhs` call, but by design the behavior is slightly different when using it inside nested function calls. In particular, if the placeholder is only used in a nested function call, `lhs` will also be placed as the first argument! The reason for this is that in most use-cases this produces the most readable code. For example, `iris %>% subset(1:nrow(.) %% 2 == 0)` is equivalent to `iris %>% subset(., 1:nrow(.) %% 2 == 0)` but slightly more compact. It is possible to overrule this behavior by enclosing the `rhs` in braces. For example, `1:10 %>% {c(min(.), max(.))}` is equivalent to `c(min(1:10), max(1:10))`.

**Using `%>%` with call- or function-producing `rhs`:**

It is possible to force evaluation of `rhs` before the piping of `lhs` takes place. This is useful when `rhs` produces the relevant call or function. To evaluate `rhs` first, enclose it in parentheses, i.e. `a %>% (function(x) x^2)`, and `1:10 %>% (call("sum"))`. Another example where this is relevant is for reference class methods which are accessed using the `$` operator, where one would do `x %>% (rc$f)`, and not `x %>% rc$f`.

**Using lambda expressions with `%>%`:**

Each `rhs` is essentially a one-expression body of a unary function. Therefore defining lambdas in magrittr is very natural, and as the definitions of regular functions: if more than a single expression is needed one encloses the body in a pair of braces, `{ rhs }`. However, note that within braces there are no "first-argument rule": it will be exactly like writing a unary function where the argument name is `"."` (the dot).

**Using the dot-place holder as `lhs`:**

When the dot is used as `lhs`, the result will be a functional sequence, i.e. a function which applies the entire chain of right-hand sides in turn to its input. See the examples.

## Technical notes

The magrittr pipe operators use non-standard evaluation. They capture their inputs and examines them to figure out how to proceed. First a function is produced from all of the individual right-hand side expressions, and then the result is obtained by applying this function to the left-hand side. For most purposes, one can disregard the subtle aspects of magrittr's evaluation, but some functions may capture their calling environment, and thus using the operators will not be exactly equivalent to the "standard call" without pipe-operators.

Another note is that special attention is advised when using non-magrittr operators in a pipe-chain (`+`, `-`, `$`, etc.), as operator precedence will impact how the chain is evaluated. In general it is advised to use the aliases provided by magrittr.

## See Also

`%<>%`, `%T>%`, `%$%`

## Examples

```
# Basic use:
iris %>% head
```

```
# Use with lhs as first argument
iris %>% head(10)

# Using the dot place-holder
"Ceci n'est pas une pipe" %>% gsub("une", "un", .)

# When dot is nested, lhs is still placed first:
sample(1:10) %>% paste0(LETTERS[.])

# This can be avoided:
rnorm(100) %>% {c(min(.), mean(.), max(.))} %>% floor

# Lambda expressions:
iris %>%
{
  size <- sample(1:10, size = 1)
  rbind(head(., size), tail(., size))
}

# renaming in lambdas:
iris %>%
{
  my_data <- .
  size <- sample(1:10, size = 1)
  rbind(head(my_data, size), tail(my_data, size))
}

# Building unary functions with %>%
trig_fest <- . %>% tan %>% cos %>% sin

1:10 %>% trig_fest
trig_fest(1:10)
```

---

`%T>%` *Tee pipe*

---

## Description

Pipe a value forward into a function- or call expression and return the original value instead of the result. This is useful when an expression is used for its side-effect, say plotting or printing.

## Usage

```
lhs %T>% rhs
```

## Arguments

| | |
|---|---|
| `lhs` | A value or the magrittr placeholder. |
| `rhs` | A function call using the magrittr semantics. |

## Details

The tee pipe works like %>%, except the return value is lhs itself, and not the result of rhs function/expression.

## See Also

%>%, %<>%, %$%

## Examples

```
rnorm(200) %>%
matrix(ncol = 2) %T>%
plot %>% # plot usually does not return anything.
colSums
```

# Index