Last week, a new version of {attempt} was published on CRAN. This  
version includes some improvements in the current code base, and the  
addition of new functions.

You can get it with our old friend install.packages

install.packages("attempt")

**News in version 0.3.0**

library(attempt)

packageVersion("attempt")

## [1] '0.3.0'

Newcomers in this version:

* The is\_try\_error() function, that tests if an object is of class  
  “try-error”.

x <- attempt(log(letters), silent = TRUE)

is\_try\_error(x)

## [1] TRUE

* The on\_error() function, that behaves as on.exit() except it  
  happens only when there is an error in the function, and can for  
  example be used to write logs to a file.

y <- function(x){

on\_error(~ print("ouch")) # or on\_error(~ write("error", "error.logs"))

log(x)

}

y(12)

y("a")

[1] 2.484907

Error in log(x) : non-numeric argument to mathematical function

[1] "ouch"

* discretly(), for removing warnings and message.

y <- function(x){

warning("ouch\n")

message("bla")

x \* 10

}

y(10)

## Warning in y(10): ouch

## bla

## [1] 100

discrete\_y <- discretly(y)

discrete\_y(10)

## [1] 100

**About {attempt}**

{attempt} is a package that provides a series of tools for defensive  
programming. It’s mainly designed for (package) developers, as it  
provides easier ways to handle exceptions and to manipulate functions  
for messages, warnings and errors. It’s also a very lightweight package,  
as it only depends on {rlang}, which itself has no dependencies.

For example, let’s say you have a function that needs at least one  
argument to be not-null. You would be tempted to write something like  
this:

this <- function(a = NULL, b = NULL, c = NULL){

if (all(is.null(a), is.null(b), is.null(c))){

stop("a, b and c can't be all NULL")

}

list(a, b, c)

}

this()

## Error in this(): a, b and c can't be all NULL

# Or

this <- function(a = NULL, b = NULL, c = NULL){

if (all(vapply(c(a, b, c), is.null, logical(1)))){

stop("a, b and c can't be all NULL")

}

list(a, b, c)

}

this()

## Error in this(): a, b and c can't be all NULL

this(a = 1)

## [[1]]

## [1] 1

##

## [[2]]

## NULL

##

## [[3]]

## NULL

With {attempt}, you can refactor your code this way:

library(attempt)

this <- function(a = NULL, b = NULL, c = NULL){

stop\_if\_all(c(a, b, c), is.null, "a, b and c can't be all NULL")

list(a, b, c)

}

this()

## Error: a, b and c can't be all NULL

# Mappers can also be used

this <- function(a){

stop\_if(a, ~ .x < 5, "a must be over 5")

}

this(1)

## Error: a must be over 5

To handle all cases, there is a series of function combining stop\_if /  
warn\_if / message\_if & all, any and none. See ?stop\_if for a  
list of all functions.

The idea being of course that if you have a series of tests, this can  
drastically reduce the amount of code at the beginning of the function,  
and make it more readable on the long run. So you can refactor this:

this <- function(a = NULL, b = NULL, c = NULL){

if (all(is.null(a), is.null(b), is.null(c))){

stop("a, b and c can't be all NULL\n")

}

if (any(vapply(c(a, b, c), function(x) x < 5, logical(1)))){

warning("using input below 5 is not recommended\n")

}

if (!any(c(a == 13, b == 13, c == 13))){

message("No input equal to 13\n")

}

# Do things

}

this()

## Error in this(): a, b and c can't be all NULL

this(a = 3)

## Warning in this(a = 3): using input below 5 is not recommended

## No input equal to 13

this(a = 10)

## No input equal to 13

To this:

this <- function(a = NULL, b = NULL, c = NULL){

stop\_if\_all(c(a, b, c), is.null, "a, b and c can't be all NULL")

warn\_if\_any(c(a, b, c), ~ .x < 5, "using input below 5 is not recommended")

message\_if\_none(c(a, b, c), ~ .x == 13, "No input equal to 13")

# Do things

}

The attempt() function, along with try\_catch, are friendlier version  
of try() and tryCatch(). They behave exactly like the base  
functions, but provide an easier interface. For example, if you need to  
try something and throw a message if it fails, with base, you’ll do:

x <- try(log("a"), silent = TRUE)

if (class(x)[1] == "try-error"){

stop("There was an error in your code")

}

## Error in eval(expr, envir, enclos): There was an error in your code

attempt() provides a concise way to send a message on error:

attempt(log("a"), "There was an error in your code")

## Error: There was an error in your code

Adverbs, finally, transform functions behavior:

* silently returns nothing unless an error occurs

silent\_log <- silently(log)

silent\_log(1)

silent\_log("a")

## Error in .f(...) : non-numeric argument to mathematical function

* surely will wrap the function in an attempt call:

sure\_log <- surely(log)

sure\_log(1)

## [1] 0

sure\_log("a")

## Error: non-numeric argument to mathematical function

* The with\_message() / with\_warning() & without\_message() /  
  without\_warning() / discretly() functions add or remove messages  
  and  
  warnings:

matrix(1:3, ncol = 2)

## Warning in matrix(1:3, ncol = 2): data length [3] is not a sub-multiple or

## multiple of the number of rows [2]

## [,1] [,2]

## [1,] 1 3

## [2,] 2 1

no\_warning\_matrix <- without\_warning(matrix)

no\_warning\_matrix(1:3, ncol = 2)

## [,1] [,2]

## [1,] 1 3

## [2,] 2 1