

# Package: retry (via r-universe)

August 21, 2024

**Type** Package

**Title** Repeated Evaluation

**Version** 0.1.1

**Date** 2024-01-22

**Description** Provide simple mechanism to repeatedly evaluate an expression until either it succeeds or timeout exceeded. It is useful in situations that random failures could happen.

**License** MIT + file LICENSE

**URL** <https://github.com/randy3k/retry>

**Encoding** UTF-8

**RoxygenNote** 7.3.0

**Suggests** testthat (>= 2.1.0), covr

**Imports** rlang, later

**Repository** <https://randy3k.r-universe.dev>

**RemoteUrl** <https://github.com/randy3k/retry>

**RemoteRef** HEAD

**RemoteSha** 59e5a444c56bd0482a426d5670bb3a7d28449557

## Contents

retry-package . . . . .	2
retry . . . . .	2
wait_until . . . . .	4

<b>Index</b>	<b>5</b>
--------------	----------

---

retry-package            *retry: Repeated Evaluation*

---

### Description

Provide simple mechanism to repeatedly evaluate an expression until either it succeeds or timeout exceeded. It is useful in situations that random failures could happen.

### Author(s)

**Maintainer:** Randy Lai <randy.cs.lai@gmail.com>

### See Also

Useful links:

- <https://github.com/randy3k/retry>

---

retry                    *Repeatedly evaluate an expression*

---

### Description

Repeatedly evaluate an expression until a condition is met or timeout is exceeded.

### Usage

```
retry(  
  expr,  
  upon = "error",  
  when = NULL,  
  until = NULL,  
  envir = parent.frame(),  
  silent = FALSE,  
  timeout = Inf,  
  max_tries = Inf,  
  interval = 0.1,  
  later_run_now = FALSE  
)
```

## Arguments

<code>expr</code>	an expression to be evaluated, supports quasiquotation.
<code>upon</code>	a vector of condition classes. The expression will be evaluated again after the delay if a condition is thrown. See the <code>classes</code> parameter of <code>rlang::catch_cnd</code> .
<code>when</code>	regular expression pattern that matches the message of the condition. It is used to decide if we need to evaluate <code>expr</code> .
<code>until</code>	a function of two arguments. This function is used to check if we need to evaluate <code>expr</code> . The first argument is the result of <code>expr</code> and the second argument is the condition thrown when <code>expr</code> was evaluated. It could be also a one sided formula that is later converted to a function using <code>rlang::as_function</code> .
<code>envir</code>	the environment in which the expression is to be evaluated.
<code>silent</code>	suppress messages and warnings
<code>timeout</code>	raise an error if this amount of time in seconds has passed.
<code>max_tries</code>	maximum number of attempts
<code>interval</code>	delay between retries.
<code>later_run_now</code>	execute <code>later::run_now()</code> periodically when <code>later</code> is loaded?

## Examples

```
retry(10, until = ~TRUE) # returns immediately

f <- function(x) {
  if (runif(1) < 0.9) {
    stop("random error")
  }
  x + 1
}
# keep retrying when there is a random error
retry(f(1), when = "random error")
# keep retrying until a condition is met
retry(f(1), until = function(val, cnd) val == 2)
# or using one sided formula
retry(f(1), until = ~ . == 2)

try({
  # it doesn't capture the error of "a" + 1
  retry(f("a"), when = "random error")
})

try({
  # an error is raised after 1 second
  retry(stop("foo"), when = "foo", timeout = 1)
})

try({
  # timeout also works for indefinite R code
  retry(while(TRUE) {}, until = ~FALSE, timeout = 1)
})
```

---

`wait_until`*Wait until a condition is met*

---

**Description**

Block the current runtime until the expression returns TRUE.

**Usage**

```
wait_until(  
  expr,  
  envir = parent.frame(),  
  timeout = Inf,  
  interval = 0.1,  
  later_run_now = TRUE  
)
```

**Arguments**

<code>expr</code>	an expression to check, supports quasiquotation.
<code>envir</code>	the environment in which the expression is to be evaluated.
<code>timeout</code>	raise an error if this amount of time in second has passed.
<code>interval</code>	delay between retries.
<code>later_run_now</code>	execute <code>later::run_now()</code> periodically later is loaded?

**Examples**

```
s <- Sys.time()  
system.time(wait_until(Sys.time() - s > 1))  
  
z <- 0  
later::later(function() z <<- 1, 1)  
wait_until(z == 1)  
z == 1
```

# Index

`retry`, [2](#)  
`retry-package`, [2](#)  
`wait_until`, [4](#)