

Errores

Julio 2019

Traducido por Leonardo Collado-Torres

@feligernon

lcolladotor@gmail.com

lcolladotor.github.io

Desarrollado por Charlotte Wickham para rstudio::conf(2019)

@cvwickham

cwickham@gmail.com

cwick.co.nz

Adapted from *Tidy Tools* by Hadley Wickham



Mostrando errores

como un **autor** de funciones

Cambia al proyecto:

[hadcol-test]

Incluido en los materiales del curso

Como recordatorio, **puedes obtener los materiales con:**
`usethis::use_course("ComunidadBioinfo/cdsb2019")`

Motivación: protección contra malos valores de entrada

```
# 0 Ctrl/Cmd + Shift + L
```

```
devtools::load_all()
```

```
df <- data.frame(x = 1, y = 2)
```

```
add_col(df, name = "z", value = 3, where = 0)
```



```
# Error in `[.default'(x, lhs) :
```

```
# only 0's may be mixed with negative subscripts
```

Encontrando donde ocurrió el error

```
> add_col(df, name = "z", value = 3, where = 0)
```

```
Error in `[.default`(x, lhs) :  
  only 0's may be mixed with negative  
subscripts
```

 Show Traceback
 Rerun with Debug

```
6. NextMethod("[")  
5. `[.data.frame`(x, lhs)  
4. x[lhs]  
3. cbind(x[lhs], y, x[-lhs]) at insert_into.R#8  
2. insert_into(x, df, where = where) at add_col.R#7  
1. add_col(df, name = "z", value = 3, where = 0)
```

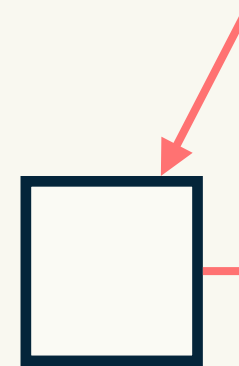
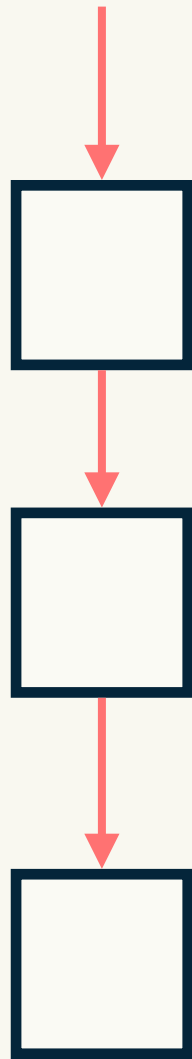
Si no estás en RStudio
traceback()

Falla rápido

Para código robusto, falla temprano

Mal valor de entrada

Mal valor de entrada



Error útil

Error no informativo

Checa los valores de entrada en insert_into()

```
df1 <- data.frame(a = 3, b = 4, c = 5)
```

```
df2 <- data.frame(X = 1, Y = 2)
```

```
# Necesitamos que estos casos regresen errores
```

```
insert_into(df1, df2, where = 0)
```

```
insert_into(df1, df2, where = NA)
```

```
insert_into(df1, df2, where = 1:10)
```

```
insert_into(df1, df2, where = "a")
```


Podríamos agregar esto a insert_into directamente

```
insert_into <- function(x, y, where = 1) {  
  if (!is.numeric(where) || length(where) != 1) {  
    stop("`where` no es un número", call. = FALSE)  
  } else if (where == 0 || is.na(where)) {  
    stop("`where` debería ser 0 o NA", call. = FALSE)  
  } else if (where == 1) {  
    cbind(y, x)  
  } else if (where > ncol(x)) {  
    cbind(x, y)  
  } else {  
    lhs <- 1:(where - 1)  
    cbind(x[lhs], y, x[-lhs])  
  }  
}
```

Pero confunde
el propósito de
insert_into()

Es mejor tener una función que sea responsable de esto

```
insert_into <- function(x, y, where = 1) {  
  where <- check_where(where)  
  if (where == 1) {  
    cbind(y, x)  
  } else if (where > ncol(x)) {  
    cbind(x, y)  
  } else {  
    lhs <- 1:(where - 1)  
    cbind(x[lhs], y, x[-lhs])  
  }  
}
```

Agrega protección contra valores de entrada incorrectos

Desarrollo motivado
por pruebas



1. Decide que debería pasar con valores de entrada incorrectos
2. Escribe pruebas en `check_where()` que reflejen #1
3. Escribe `check_where()`
4. Actualiza `insert_into()` para usar `check_where()`

Estructura de un mensaje de error

1. **Planteamiento del problema**

(usa debe o no puede)

must
can't

2. **Ubicación del error** (de ser posible)

3. **Pista**

(si es común)

Tu turno

```
# Escribe el mensaje de error que piensas  
# que cada de estas líneas debería generar  
check_where(where = 0)  
check_where(where = NA)  
check_where(where = 1:10)  
check_where(where = "a")
```

Mis resultados

```
check_where(0)
```

```
#> Error: `where` must not be  
zero or a missing value.
```

```
check_where(NA)
```

```
#> Error: `where` must not be  
zero or a missing value.
```

```
check_where(1:10)
```

```
#> Error: `where` must be a  
length one numeric vector.
```

```
check_where("a")
```

```
#> Error: `where` must be a  
length one numeric vector.
```

Estilo

- Encapsula nombres de variables con `...`, y de texto con ‘...’
- Escribe oraciones

Usa `expect_error()` para probar los errores

Esta prueba pasará si ocurre un error

```
expect_error(  
  check_where("a")  
)
```

Esta prueba pasará si el mensaje de error coincide

```
expect_error(  
  check_where("a"),  
  "not a number"  
)
```



Una expresión regular

Tu turno

Escribe pruebas para `check_where()` para asegurarte que solo acepte valores de entrada correctos.

(¿Dónde deberían vivir las pruebas? ¿Cuántas pruebas necesitas? ¿Cuántas expectativas?)

```
check_where(0)
```

```
#> Error: `where` must not be zero or a missing value.
```

```
check_where(NA)
```

```
#> Error: `where` must not be zero or a missing value.
```

```
check_where(1:10)
```

```
#> Error: `where` must be a length one numeric vector.
```

```
check_where("a")
```

```
#> Error: `where` must be a length one numeric vector.
```


Mis pruebas

```
# Yo creo que las pruebas deberían vivir en tests/testthat/test-  
insert_into.R
```

```
test_that("where debe ser un valor válido", {  
  expect_error(check_where("a"), "length one numeric vector")  
  expect_error(check_where(1:10), "length one numeric vector")  
  
  expect_error(check_where(0), "not be zero or missing")  
  expect_error(check_where(NA_real_), "not be zero or missing")  
})
```

Señala un error con stop()

```
f <- function(){  
  stop("Este es un mensaje de error.",  
    call. = FALSE)  
}
```

No incluyas la llamada en
el mensaje de error

```
f()  
# Error : Este es un mensaje de error.
```

Checa valores de entrada combinando con if()

```
# Un patrón general
f <- function(x){
  if (!is.numeric(x)) {
    stop("`x` must be numeric",
         call. = FALSE)
  }
  x
}
f("a")
```

Tu turno

Escribe `check_where()`. Debería regresar un error si el valor de entrada es incorrecto. Sugiero que lo pongas en el mismo archivo que `insert_into()`.

```
check_where(0)
```

```
check_where(NA)
```

```
check_where(1:10)
```

```
check_where("a")
```

Pista para empezar en la próxima diapositiva

Pista: empezando

```
# Empieza con un esqueleto en R/insert_into.R
```

```
check_where <- function(x){
```

```
}
```

```
# Asegúrate de que hayas copiado las pruebas en
```

```
# tests/testthat/test-insert_into.R
```

```
# Revisa que obtengas cuatro fayas con
```

```
devtools::test()
```

```
# Edita check_where() hasta que pasen las pruebas
```

Mi respuesta

```
check_where <- function(x) {  
  if (length(x) != 1 || !is.numeric(x)) {  
    stop("`where` must be a length one numeric vector.",  
        call. = FALSE)  
  }  
  
  x <- as.integer(x)  
  
  if (x == 0 || is.na(x)) {  
    stop("`where` must not be zero or missing",  
        call. = FALSE)  
  }  
  x  
}
```

Otras condiciones

Errores `stop()`

No hay forma de continuar, la ejecución debe parar.

Alertas `warning()`

Señala que algo pasó mal, pero que el código se pudo recuperar y continuar. No lo uses frecuentemente, ¿sería mejor un error?

Mensajes `message()`

Solo proveen información.

usa `cat()` cuando el objetivo principal es imprimir

Manejando errores

como un **usuario** de funciones

Iteración: ¿qué pasa si hay un error?

```
library(purrr)
```

```
input <- list(1:10, sqrt(4), 5, "n")
```

```
map(input, log)
```

```
# Error in .Primitive("log")(x, base) :  
#   non-numeric argument to mathematical  
#   function
```

No hay resultados.

Ni idea de cuál fue el problema.

Principio:

Convierte los efectos
secundarios en datos

¿Qué es lo que hace safely()?

```
library(purrr)
```

```
input <- list(1:10, sqrt(4), 5, "n")
```

```
# Esto nunca va a fallar
```

```
map(input, safely(log))
```

```
# ¿Qué es lo que regresa cuando es exitosa la  
función?
```

```
# ¿Qué regresa cuando falla la función?
```

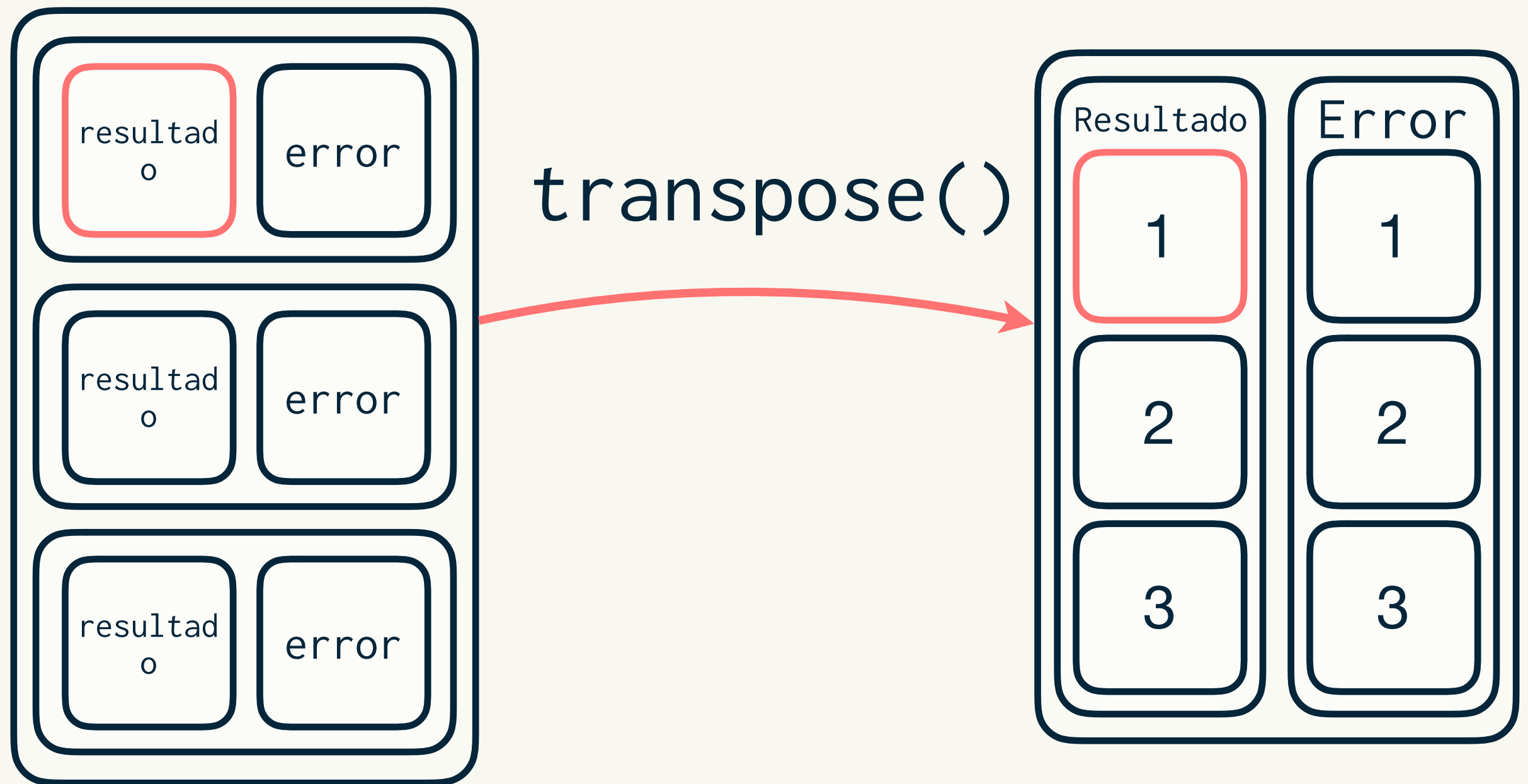
Un ejemplo más útil

```
urls <- c(
  "https://google.com",
  "https://en.wikipedia.org",
  "asdfasdskfjlda"
)

# Falla
contents <- urls %>%
  map(readLines, warn = FALSE)

# Siempre funciona
contents <- urls %>%
  map(safely(readLines), warn = FALSE)
str(contents)
```

Pero `map()` + `safely()` regresa un objeto raro



`x[[1]][["result"]]` \longrightarrow `x[["result"]][1]`

Tu turno

Aplica `transpose()` a los contenidos de "Un ejemplo más útil" y luego:

1. Haz un vector lógico que es `TRUE` si la descarga funcionó. (`map_lgl()`)
2. Enumera los URLs fallidos
3. Extrae exitosamente el texto extraído

Patrón común con safely()

```
contents <- urls %>%  
  map(safely(readLines)) %>%  
  transpose()
```

```
ok <- map_lgl(contents$error, is.null)
```

```
# Esto es subóptimo:
```

```
ok <- !map_lgl(contents$result, is.null)
```

```
urls[!ok]
```

```
contents$result[ok]
```

Operadores funcionales

una o más función(es) como valor de entrada, una función como valor de salida

<code>safely()</code> <code>possibly()</code> <code>quietly()</code>	convierten efectos secundarios en datos
<code>partial()</code>	fijan unos argumentos
<code>lift()</code>	cambian como los argumentos son proveídos
<code>memoise::memoise()</code>	agrega memoria

Piensa en **adverbios**: alteran el comportamiento de una función

Manejar errores dentro de tus funciones

`try()`

`tryCatch()`

Para capturar y manejar errores en formas personalizadas.

Checa: <https://adv-r.hadley.nz/conditions.html#handling-conditions>

Adapted from *Tidy Tools* by Hadley Wickham

This work is licensed as

Creative Commons
Attribution-ShareAlike 4.0
International

To view a copy of this license, visit
[https://creativecommons.org/
licenses/by-sa/4.0/](https://creativecommons.org/licenses/by-sa/4.0/)