

Manipulación de cadenas con stringr : : GUÍA RÁPIDA



El paquete **stringr** proporciona un conjunto de herramientas internamente consistentes para trabajar con cadenas de caracteres, por ej. secuencias de caracteres delimitados por comillas.

Detectar Coincidencias

str_detect(cadena, patrón) Detecta el patrón coincidente en una cadena.
`str_detect(fruta, "a")`

str_which(cadena, patrón) Encuentra los índices de las cadenas que contienen un patrón coincidente.
`str_which(fruta, "a")`

str_count(cadena, patrón) Cuenta el número de coincidencias en una cadena.
`str_count(fruta, "a")`

str_locate(cadena, patrón) Localiza las posiciones en las que el patrón coincide con la cadena. También **str_locate_all**.
`str_locate(fruta, "a")`

Subconjunto de cadenas

str_sub(cadena, start = 1L, end = -1L) Extrae subcadenas de un vector de caracteres.
`str_sub(fruta, 1, 3); str_sub(fruta, -2)`

str_subset(cadena, patrón) Devuelve sólo las cadenas que contienen un patrón coincidente.
`str_subset(fruta, "b")`

str_extract(cadena, patrón) Devuelve el primer patrón encontrado que coincide en cada cadena, como un vector. También **str_extract_all** para devolver cada patrón coincidente.
`str_extract(fruta, "[aeiou]")`

str_match(cadena, patrón) Devuelve el primer patrón encontrado que coincide en cada cadena, como una matriz, con una columna para cada una () agrupado por patrón. También **str_match_all**.
`str_match(sentences, "(el|la) ([^]+)")`

Gestionar Longitudes

str_length(cadena) Los largos de las cadenas (por ej. número de puntos de código, suele ser igual al número de caracteres). `str_length(fruta)`

str_pad(cadena, largo, side = c("left", "right", "both"), pad = " ") Extiende cadenas a un largo constante. `str_pad(fruta, 17)`

str_trunc(cadena, ancho, side = c("right", "left", "center"), ellipsis = "...") Trunca el ancho de una cadena, eliminando el contenido sobrante.
`str_trunc(fruta, 3)`

str_trim(cadena, side = c("both", "left", "right")) Elimina los espacios en blanco desde el inicio y/o al final de una cadena. `str_trim(fruta)`

Transformar Cadenas

str_sub() <- valor. Reemplaza subcadenas identificadas con `str_sub()` y se asignan al resultado.
`str_sub(fruta, 1, 3) <- "str"`

str_replace(cadena, patrón, reemplazo) Reemplaza el primer patrón coincidente en cada cadena. `str_replace(fruta, "a", "-")`

str_replace_all(cadena, patrón, replacement) Reemplaza todos los patrones coincidentes en cada cadena.
`str_replace_all(fruta, "a", "-")`

str_to_lower(cadena, locale = "en")¹ Convierte cadenas a minúscula.
`str_to_lower(oraciones)`

str_to_upper(string, locale = "en")¹ Convierte cadenas a mayúsculas.
`str_to_upper(oraciones)`

str_to_title(string, locale = "en")¹ Convierte cadenas a título. `str_to_title(oraciones)`

Juntar y Separar

str_c(..., sep = "", collapse = NULL) Une múltiples cadenas en una. `str_c(letters, LETTERS)`

str_c(..., sep = "", collapse = NULL) Colapsa un vector de cadenas en una sola cadena.
`str_c(letters, collapse = "")`

str_dup(cadena, veces) Repite cadenas varias veces. `str_dup(fruta, times = 2)`

str_split_fixed(cadena, patrón, n) Divide un vector de cadenas en una matriz de subcadenas (dividiendo en las ocurrencias de cada patrón de coincidencia). También **str_split** para devolver una lista de subcadenas.
`str_split_fixed(fruta, " ", n=2)`

str_glue(..., .sep = "", .envir = parent.frame()) Crea una cadena a partir de cadenas y {expresiones} para evaluar. `str_glue("Pi is {pi}")`

str_glue_data(.x, ..., .sep = "", .envir = parent.frame(), .na = "NA") Usa un conjunto de datos, lista, o entorno para crear una cadena a partir de cadenas y {expresiones} para evaluar. `str_glue_data(mtcars, "{rownames(mtcars)} has {hp} hp")`

Ordenar Cadenas

str_order(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)¹ Devuelve el vector de índices que ordena un vector de caracteres. `x[str_order(x)]`

str_sort(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)¹ Ordena un vector de caracteres.
`str_sort(x)`

Funciones auxiliares

str_conv(cadena, encoding) Sobreescribe el tipo de codificación de una cadena.
`str_conv(fruta, "ISO-8859-1")`

str_view(cadena, patrón, match = NA) Vista en HTML de la primera coincidencia de una expresión regular en cada cadena.
`str_view(fruta, "[aeiou]")`

str_view_all(cadena, patrón, match = NA) Vista en HTML de todas las coincidencias de la expresión regular. `str_view_all(fruta, "[aeiou]")`

str_wrap(cadena, width = 80, indent = 0, exdent = 0) Envuelve cadenas en párrafos formateados adecuadamente.
`str_wrap(sentences, 20)`

¹ Ver bit.ly/ISO639-1 para una lista completa de locales.

Necesitas Saber

Los argumentos de los patrones en stringr son interpretados como expresiones regulares después de cada carácter que ha sido segmentado

En R, se escriben expresiones regulares como cadenas, secuencias de caracteres rodeados de comillas dobles (") o simples (').

Algunos caracteres no se pueden representar directamente como una cadena en R. Estos son representados por **caracteres especiales**, secuencias de caracteres que tienen un significado específico., e.g.

Especial Character	Representa
\\	\
\"	"
\n	nueva línea

Ejecuta `?""` para ver una lista completa

Por esto, cuando aparece \ en una expresión regular, se tiene que escribir como \\ en la cadena que representa la expresión regular.

Usa `writeLines()` para ver como R ve tu cadena después de que todos los caracteres especiales se han parseado.

```
writeLines("\\ ")
# \
```

```
writeLines("\\ is a backslash")
# \ is a backslash
```

INTERPRETACIÓN

Los patrones en stringr son interpretados como *regexs* (expresiones regulares). Para cambiar este argumento predeterminado, envuelva el patrón con una de estas opciones:

regex(pattern, ignore_case = FALSE, multiline = FALSE, comments = FALSE, dotall = FALSE, ...) Modifica una *regex* para ignorar casos, coincide fin de líneas como también fin de cadenas, permite comentarios de R dentro de las *regex*, y/o tienen .coincide cualquier cosa incluyendo \n.
`str_detect("I", regex("i", TRUE))`

fixed() Coincide raw bits pero ignorará algunos caracteres que se pueden representar de múltiples formas (rápido). `str_detect("u0130", fixed("i"))`

coll() Coincide raw bits y usará patrones específicos de los parámetros locales para reconocer caracteres que pueden ser representados en múltiples formas (lento).
`str_detect("u0130", coll("i", TRUE, locale = "tr"))`

boundary() Coincide límites entre caracteres, separadores de líneas, sentencias o palabras.
`str_split(sentencias, boundary("word"))`

Expresiones Regulares

Expresiones regulares, o *regexps*, es un lenguaje conciso para describir patrones en cadenas.

MATCH CHARACTERS

cadena (escribe esto)	regex (para decir esto)	coincidencias (que coincide con esto)	ejemplo
\\	\\	a (etc.)	<code>see("a")</code> abc ABC 123
\\!	\\!	.	<code>see("\\.")</code> abc ABC 123
\\!	\\!	!	<code>see("\\!")</code> abc ABC 123
\\?	\\?	?	<code>see("\\?")</code> abc ABC 123
\\	\\		<code>see("\\ ")</code> abc ABC 123
\\(\\((<code>see("\\(")</code> abc ABC 123
\\)	\\))	<code>see("\\)")</code> abc ABC 123
\\{	\\{	{	<code>see("\\{")</code> abc ABC 123
\\}	\\}	}	<code>see("\\}")</code> abc ABC 123
\\n	\\n	nueva línea (retorno)	<code>see("\\n")</code> abc ABC 123
\\t	\\t	tab	<code>see("\\t")</code> abc ABC 123
\\s	\\s	cualquier espacio en blanco (NS para no-blancos)	<code>see("\\s")</code> abc ABC 123
\\d	\\d	cualquier dígito (ND para no-dígitos)	<code>see("\\d")</code> abc ABC 123
\\w	\\w	cualquier carácter (AW para no-caracteres)	<code>see("\\w")</code> abc ABC 123
\\b	\\b	bordes de palabras	<code>see("\\b")</code> abc ABC 123
	<code>[:digit:]</code>	dígitos	<code>see("[:digit:]")</code> abc ABC 123
	<code>[:alpha:]</code>	letras	<code>see("[:alpha:]")</code> abc ABC 123
	<code>[:lower:]</code>	letras minúsculas	<code>see("[:lower:]")</code> abc ABC 123
	<code>[:upper:]</code>	letras mayúsculas	<code>see("[:upper:]")</code> abc ABC 123
	<code>[:alnum:]</code>	letras y números	<code>see("[:alnum:]")</code> abc ABC 123
	<code>[:punct:]</code>	puntuación	<code>see("[:punct:]")</code> abc ABC 123
	<code>[:graph:]</code>	letras, números, y puntuación	<code>see("[:graph:]")</code> abc ABC 123
	<code>[:space:]</code>	caracteres espacio (por ej. \\s)	<code>see("[:space:]")</code> abc ABC 123
	<code>[:blank:]</code>	espacios y tab (pero no nueva línea)	<code>see("[:blank:]")</code> abc ABC 123
	.	cada carácter excepto una nueva línea	<code>see(".")</code> abc ABC 123

¹ Muchas funciones de R base requieren que las clases se envuelvan en un segundo juego de [], e.g. `[:digit:]`



[:space:]
← nueva línea

[:blank:]
espacio
tab

[:graph:]

[:punct:]
.,:;?!\\|/\' = * + - ^
_ ~ " ' [] { } () < > @ # \$

[:alnum:]

[:digit:]
0 1 2 3 4 5 6 7 8 9

[:alpha:]

[:lower:]	[:upper:]
a b c d e f	A B C D E F
g h i j k l	G H I J K L
m n o p q r	M N O P Q R
s t u v w x	S T U V W X
z	Z

ALTERNATIVAS

`alt <- function(rx) str_view_all("abcde", rx)`

regex	coincidencias	ejemplo
<code>ab d</code>	o	<code>alt("ab d")</code> abcde
<code>[abe]</code>	una de	<code>alt("[abe]")</code> abcde
<code>^[abe]</code>	excepto	<code>alt("^[abe]")</code> abcde
<code>[a-c]</code>	rango	<code>alt("[a-c]")</code> abcde

ANCLAS

`anchor <- function(rx) str_view_all("aaa", rx)`

regex	coincidencias	ejemplo
<code>^a</code>	comienzo cadena	<code>anchor("^a")</code> aaa
<code>a\$</code>	fin de cadena	<code>anchor("a\$")</code> aaa

MIRAR ALREDEDOR

`look <- function(rx) str_view_all("bacad", rx)`

regex	coincidencias	ejemplo
<code>a(=?c)</code>	seguido por	<code>look("a(=?c)")</code> bacad
<code>a(!?c)</code>	no seguido por	<code>look("a(!?c)")</code> bacad
<code>(?<=b)a</code>	precedido por	<code>look("(?<=b)a")</code> bacad
<code>(?<!b)a</code>	no precedido por	<code>look("(?<!b)a")</code> bacad

CUANTIFICADORES

`cuant <- function(rx) str_view_all(".a.aa.aaa", rx)`

regex	coincidencias	ejemplo
<code>a?</code>	ceros o uno	<code>quant("a?")</code> .a.aa.aaa
<code>a*</code>	ceros o más	<code>quant("a*")</code> .a.aa.aaa
<code>a+</code>	uno o más	<code>quant("a+")</code> .a.aa.aaa
<code>a{n}</code>	exactamente n	<code>quant("a{2}")</code> .a.aa.aaa
<code>a{n,}</code>	n o más	<code>quant("a{2,}")</code> .a.aa.aaa
<code>a{n,m}</code>	entre n y m	<code>quant("a{2,4}")</code> .a.aa.aaa

GRUPOS

`ref <- function(rx) str_view_all("abbaab", rx)`

Usa paréntesis para fijar el precedente (orden de evaluación) y crea grupos

regex	coincidencia	ejemplo
<code>(ab d)e</code>	fija precedencia	<code>alt("(ab d)e")</code> abcde

Usar un número escapado para referirse y duplicar un grupo de paréntesis que ocurre antes en un patrón. Referirse a cada grupo por su orden de aparición

Cadena (escribe esto)	regex (para decir esto)	coincidencia (que coincide con esto)	ejemplo (el resultado es el mismo que <code>ref("abba")</code>)
<code>\\1</code>	<code>\\1</code> (etc.)	<code>first () group, etc.</code>	<code>ref("(a)(b)\\2\\1")</code> abbaab

