Expresiones regulares básicas en R

Cheat Sheet

Clases de caracteres

[[:digit:]] o \\d	Dígitos; `[0-9]`
\\D	No dígitos; `[^0-9]`
[[:lower:]]	Minúsculas; `[a-z]`
[[:upper:]]	Mayúsculas; `[A-Z]`
[[:alpha:]]	Alfabéticos; `[A-Za-z]`
[[:alnum:]]	Alfanuméricos; `[A-Za-z0-9]`
\\w	De palabra; `[A-Za-z0-9_]`
\\W	Caracteres que no son de palabra
[[:xdigit:]]	Dígitos hexadec.; `[0-9A-Fa-f]`
[[:blank:]]	Espacio y tabulación
[[:space:]] o \\s	Espacio, tab, tab vert., salto de línea o de página, retorno
\\S	No espacio; `[^[:space:]]`
[[:punct:]]	Signos de puntuación; `!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~`
[[:graph:]]	Caracteres gráficos; `[[:alnum:][:punct:]]`
[[:print:]]	Caracteres imprimibles; `[[:alnum:][:punct:] ]`
[[:cntrl:]]	Caracteres de control; `\n, \r` etc.

Metacaracteres especiales

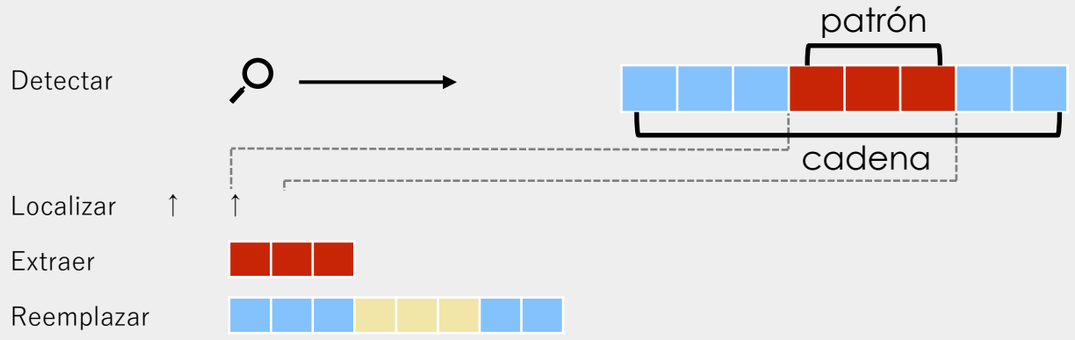
\n	Nueva línea
\r	Retorno de carro
\t	Tabulación
\v	Tabulación vertical
\f	Salto de página

Lookarounds y condicionales*

(?=)	Lookahead (requiere perl = TRUE), p. ej. `(?=yx)`: posición seguida de `'yx'`
(?!)	Lookahead negativo (perl = TRUE); posición NO seguida del patrón
(?<=)	Lookbehind (perl = TRUE), p. ej. `(?<=yx)`: posición precedida de `'yx'`
(?<!)	Lookbehind negativo (perl = TRUE); posición NO precedida del patrón
(?(if)then)	Si-entonces (perl = TRUE); en el if usa lookaheads, carácter opcional, etc.
(?(if)then|else)	Si-entonces-si no (perl = TRUE)

*ver, p. ej. http://www.regular-expressions.info/lookaround.html
http://www.regular-expressions.info/conditional.html

Funciones para buscar patrones




```
> string <- c("Hiphopopotamus", "Rhymenoceros", "time for bottomless lyrics")
> pattern <- "t.m"
```

Detectar patrones

grep(pattern, string)
```
[1] 1 3
```

grep(pattern, string, value = TRUE)
```
[1] "Hiphopopotamus"
[2] "time for bottomless lyrics"
```

grepl(pattern, string)
```
[1]  TRUE FALSE  TRUE
```

stringr::str_detect(string, pattern)
```
[1]  TRUE FALSE  TRUE
```

Localizar patrones

regexpr(pattern, string)
inicio y longitud de la primera coincidencia

gregexpr(pattern, string)
inicio y longitud de todas las coincidencias

stringr::**str_locate**(string, pattern)
inicio y fin de la primera coincidencia

stringr::**str_locate_all**(string, pattern)
inicio y fin de todas las coincidencias

Dividir una cadena con un patrón

strsplit(string, pattern) o stringr::**str_split(string, pattern)**

Extraer patrones

regmatches(string, regexpr(pattern, string))
primera coincidencia `[1] "tam" "tim"`

regmatches(string, gregexpr(pattern, string))
todas las coincidencias, como lista
`[[1]] "tam" [[2]] character(0) [[3]] "tim" "tom"`

stringr::**str_extract**(string, pattern)
primera coincidencia `[1] "tam" NA "tim"`

stringr::**str_extract_all**(string, pattern)
todas las coincidencias, como lista

stringr::**str_extract_all**(string, pattern, simplify = TRUE)
todas las coincidencias, como matriz

stringr::**str_match**(string, pattern)
primera coincidencia + cada grupo de captura

stringr::**str_match_all**(string, pattern)
todas las coincidencias + cada grupo de captura

Reemplazar patrones

sub(pattern, replacement, string)
reemplaza la primera coincidencia

gsub(pattern, replacement, string)
reemplaza todas las coincidencias

stringr::**str_replace**(string, pattern, replacement)
reemplaza la primera coincidencia

stringr::**str_replace_all**(string, pattern, replacement)
reemplaza todas las coincidencias

Clases de caracteres y grupos

.	Cualquier carácter excepto \n
|	O, p. ej. `(a|b)`
[…]	Caracteres permitidos, p. ej. `[abc]`
[a-z]	Define rangos de caracteres
[^…]	Lista de caracteres excluidos
(…)	Grupo; ref. inversa con `\\N` (N entero)

Anclas

^	Inicio de la cadena
$	Fin de la cadena
\\b	Cadena vacía en el borde de una palabra
\\B	NO es el borde de una palabra
\\<	Inicio de una palabra
\\>	Fin de una palabra

Cuantificadores

*	Coincide al menos 0 veces
+	Coincide al menos 1 vez
?	Coincide 0 o 1 vez; cadena opcional
{n}	Coincide exactamente n veces
{n,}	Coincide al menos n veces
{n,m}	Coincide entre n y m veces

Modos generales

Por defecto, R usa expresiones regulares *extendidas*. Puedes cambiar a *PCRE* con `perl = TRUE` en base; stringr usa el motor ICU, ajustable con `regex()`.

Toda función admite búsquedas literales con `fixed = TRUE` en base o envolviendo los patrones con `fixed()` en stringr.

Para no distinguir mayúsculas de minúsculas en las funciones base, usa `ignore.case = TRUE`.

Escapar caracteres

Los metacaracteres (`. * +` etc.) pueden usarse como caracteres literales si se escapan, con `\\` o encerrándolos en `\\Q...\\E` (con perl = TRUE).

Mayúsculas y minúsculas

Las regex ignoran mayúsculas/minúsculas con `(?i)`. En referencias inversas, las cadenas pueden convertirse a minúsculas o mayúsculas con `\\L` o `\\U` (p. ej. `\\L\\1`). Requiere `perl = TRUE`.

Coincidencia codiciosa

Por defecto, el asterisco * es codicioso: coincide siempre con la cadena más larga posible. Puede usarse en modo perezoso con `?`, es decir, `*?`.

El modo codicioso se desactiva con `(?U)`. Esto invierte la sintaxis: `(?U)a*` es perezoso y `(?U)a*?` es codicioso.

Nota

Las expresiones regulares pueden crearse cómodamente con, p. ej., los paquetes rex o rebus.

CC BY Ian Kopacka • ian.kopacka@ages.at • Traducido por Anthony Romero-Cerdán • anthonyromero.netlify.app