Expresiones regulares | ProgPLN

Víctor Peinado v.peinado@filol.ucm.es

24 de octubre de 2014

Referencias

- *Ayuda sobre expressiones regulares de Mozilla* [1]
- Wikipedia: Expresión regular [2]
- regular-expressions.info [3]

[1] *Ayuda sobre expressiones regulares de Mozilla* `https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global_Objects/RegExp#Special_characters_in_regular_expressions`

[2] Wikipedia: Expresión regular `http://es.wikipedia.org/wiki/Expresi%C3%B3n_regular`

[3] `http://www.regular-expressions.info`

Expresiones regulares

Las expresiones regulares (*regular expressions* o *regexes* en inglés) son el procedimiento más sencillo y básico que tenemos a nuestro alcance para procesar texto. Se trata de un un lenguaje formal que nos permite especificar cadenas de texto. A pesar de su sencillez, son un mecanismo muy potente para encontrar y sustituir patrones al procesar ficheros de texto.

Para crear una expresión regular debemos utilizar una sintaxis específica, es decir, caracteres especiales y reglas de construcción. Muchas herramientas de UNIX (p. ej., `grep` y todas sus variantes (`egrep`, `fgrep`) aceptan patrones de búsqueda basados en expresiones regulares, así como muchos editores de texto (`vim`, `emacs`, `notepad++`, `sublimetext`, etc.).

Para nuestras pruebas iniciales, vamos a jugar con un par de herramientas *online*:

- RegExr [4]
- RegexPal [5]
- Refiddle [6]
- Rubular [7]

[4] `http://www.regexr.com`

[5] `http://regexpal.com`.

[6] `http://refiddle.com`

[7] `http://rubular.com`

Expresiones regulares sencillas

La expresión regular más sencilla consiste en indicar la cadena exacta que queremos encontrar.

Así, la regex `r` hace *matching* con cualquier *r* que aparezca en el texto.

La regex `the` hace *matching* con cualquier palabra que contenga la subcadena *the* como en en *the*, *there*, *aesthetic*, *farther*, *smoothed*.

La regex `mente` hace *matching* con cualquier ocurrencia de palabra que contenga esa cadena tal cual, en palabras como *mente*, *mentes*, *obviamente*, *últimamente*, *mentecato*, *Armenteros*, etc.

La regex `he visto una vaca` hace *matching* con cualquier secuencia de caracteres que contenga dicha subcadena

Disyunción

En los ejemplos anteriores hemos visto que las expresiones regulares distinguen entre mayúsculas y minúsculas. De hecho, en el ejemplo con 'mente no hacíamos *matching* con otras variantes de la misma palabra escritas en mayúsculas como *Mente* o *MENTE*.

¿Cómo podemos tenerlas en cuenta? O ¿cómo podemos buscar solo ocurrencias de la palabra mente, ya esté escrita en mayúsculas o minúsculas, y de ninguna palabra más?

Podemos indicar **disyunción** entre varios caracteres encerrándolos entre corchetes `[]`.

Una regex como `[Rr]` hará *matching* con una *R* o con una *r*, es decir, cualquier erre que encuentre, ya sea mayúscula o minúscula.

- `[Mm]ente` hace *matching* con cualquier secuencia *Mente* o *mente*.
- `[Aa]mig[ao]` hace *matching* con *Amigo, amigo, Amiga, amiga, Amigos, amigos*.
- `[aeiou]` hace *matching* con cualquier vocal minúscula.
- `[12345]` hace *matching* con cualquier vocal dígito entre 1 y 5.

Cuando queremos indicar disyunción entre varias opciones, a veces es conveniente utilizar rangos de caracteres, en lugar de indicarlas explícitamente. Los rangos se especifican con guiones y van encerrados también entre corchetes. Algunos rangos muy útiles ya están predefinidos, por ejemplo.

- `[a-z]` equivale a *cualquier letra minúscula* `[abcdefghijklmnopqrstuvwxyz]`.
- `[A-Z]` equivale a *cualquier letra mayúscula* `[ABCDEFGHIJKLMNOPQRSTUVWXYZ]`.
- `[a-zA-Z]` o `[A-z]` equivale a *cualquier carácter alfabético*, ya sea en minúsculas o en mayúsculas.
- `[0-9]` equivale a *cualquier dígito* `[0123456789]`.

Estos rangos que acabamos de ver no funcionan con caracteres que no estén en el alfabeto inglés y fallan con diacríticos y otros símbolos.

Cuando trabajamos sobre textos en lenguas diferentes debemos asegurarnos de estar considerando todos los símbolos propios del idioma. Para el caso del español, deberíamos incluir las eñes y las vocales con tilde, p. ej.: `[A-ZÑÁÉÍÓÚÜa-zñáéíóúü]`.

Negación de la disyunción

En ocasiones, es conveniente poder negar estas disyunciones y expresar que queremos hacer *matching* con cualquier símbolo que no

coincida con alguno de los expresados en la regex.

Esta negación la indicamos incluyendo un acento circunflejo `[^]` dentro de los corchetes.

El acento circunflejo solo indica negación cuando aparece como primer elemento de una disyunción encerrada entre corchetes. En otros contextos, como veremos más adelante, tiene otros significados.

- `[^aeiou]` equivale a *cualquier cosa que no sea una letra minúscula*.
- `[^0-9]` equivale a *cualquier cosa que no sea un dígito*.

Si la disyunción queremos expresarla entre dos cadenas de caracteres, podemos utilizar la tubería (*pipeline*). La regex `amigo|colega` hara *matching* con todas las ocurrencias de cualquiera de las dos palabras.

- `feo|bonito` hace *matching* con todas las apariciones de cualquier de las dos palabras.
- `a|b|c` es equivalente a `[abc]`.
- `[Aa]mig[oa]|[Cc]olega` hace *matching* con *Amigo, amigo, Amiga, amiga, Colega, colega*.

Metacaracteres

La sintaxis de las regex permite utilizan símbolos llamados **metacaracteres** que funcionan con significados muy concretos, p. ej.: `$`, `^`, `.`, `*`, `+`, `?`, `[`, `]` y `\`.

Si queremos buscar alguno de estos símbolos sin que sean interpretados como metacaracteres, tenemos que *escaparlos* anteponiéndoles una barra invertida `\`.

- `25\$` hará *matching* con *25$*.
- `2\.4` hará *matching* con *2.4*.
- `5 \+ 4` hará *matching* con *5 + 4* .
- `U\.S\.A\.` hará *matching* con *U.S.A* .

Veamos ahora el significado especial de los metacaracteres, es decir, los valores que adoptan **cuando no aparecen *escapados*** en la regex.

Cuantificación

Existe unos cuanto metacaracteres que nos permiten cuantificar las expresiones regulares, es decir, cuántas veces se repite un determinado patrón.

- `?`: el carácter anterior es opcional. La regex hará *matching* si el carácter aparece una vez o ninguna.
- `*`: el carácter anterior es opcional. La regex hará *matching* si el carácter aparece o o más veces.

- `+`: el carácter anterior es opcional. La regex hará *matching* si el carácter anterior aparece `1` o más veces.
- `{n}`: la regex hará *matching* si el carácter anterior aparece exactamente *n* veces.
- `{n,}`: la regex hará *matching* si el carácter anterior aparece como mínimo *n* veces.
- `{n,m}`: la regex hará *matching* si el carácter anterior aparece como mínimo *n* veces y como máximo *m* veces.

Algunos ejemplos de regex con cuantificación.

- `colou?r` hará *matching* con las palabras *color* y *colour*.
- `colou*r` hará *matching* con las cadenas *color*, *colour*, *colouur*....
- `colou+r` hará *matching* con las cadenas *colour*, *colouur*... .
- `baa+` hará *matching* con cadenas como *baa, baaa, baaaa, baaaaa*....
- `baa*` hará *matching* con cadenas como *ba, baa, baaa, baaaa, baaaaa*....
- `ba{2}` hará *matching* solo con la cadena *baa* .
- `ba{2,}` hará *matching* con las cadenas *baa, baaa, baaaa*....
- `ba{3,5}` hará *matching* solo con las cadenas *baaa, baaaa, baaaaa*.

Comodines

Otros metacaracteres importantes permiten especificar el inicio y final de la línea que estamos procesando:

- `^`: inicio de línea
- `$`: final de la línea
- `.`: cualquier carácter

No hay que confundirlo con la negación de una disyunción, p. ej. `[^A-Z]`.

Veamos algunos ejemplos:

- `^[A-Z]` significa *cualquier letra mayúscula que aparezca al inicio de la línea*.
- `\.$` hará *matching* con cualquier punto que aparezca al final de una línea.
- `^I'm` hará *matching* con cualquier secuencia *I'm* que aparezca al inicio de la línea.
- `ca.a` hará *matching* con cadenas como *casa*, *caja*, *cama*, *cala*, *caña*, *capa*, *cata*, *ca!a*... e incluso con *ca.a*.

Otros metacaracteres complejos

Por último, existen otros metacaracteres muy útiles que funcionan solo en el modo extendido de las expresiones regulares.

- `\s` significa *cualquier espacio en blanco o tabulador*.
- `\S` significa *cualquier carácter que no sea espacio en blanco o tabulador*.
- `\w` significa *cualquier carácter alfanumérico (letras y dígitos)*.
- `\W` significa *cualquier carácter que no sea alfanumérico*.
- `\d` significa *cualquier carácter numérico* (equivalente a `[0-9]`).
- `\D` significa *cualquier carácter que no sea numérico* (letras + espacios en blanco).
- `\b` sirve para indicar el *límite de una palabra*.

Estos metacaracteres se pueden combinar con los cuantificadores, de manera que:

- `\w+` significa *cualquier secuencia de caracteres alfanuméricos (letras y dígitos)*.
- `\S+` es equivalente, significa *cualquier secuencia de caracteres que no sean espacios*.
- `\d+` significa *cualquier secuencia de números*.

Quiero más

Excuse me, sir. Can I have some more? Sure, Oliver, here you go.

- *Introducing Regular Expressions* [8]
- *Masteringing Regular Expressions* [9]
- *Regular Expressions Cookbook* [10]

[8] *Introducing Regular Expressions*
`http://cdn.oreilly.com/oreilly/booksamplers/9781449392680_sampler.pdf`

[9] *Masteringing Regular Expressions*
`http://cdn.oreilly.com/oreilly/booksamplers/9780596528126_sampler.pdf`

[10] *Regular Expressions Cookbook*
`http://cdn.oreilly.com/oreilly/booksamplers/9781449319434_sampler.pdf`