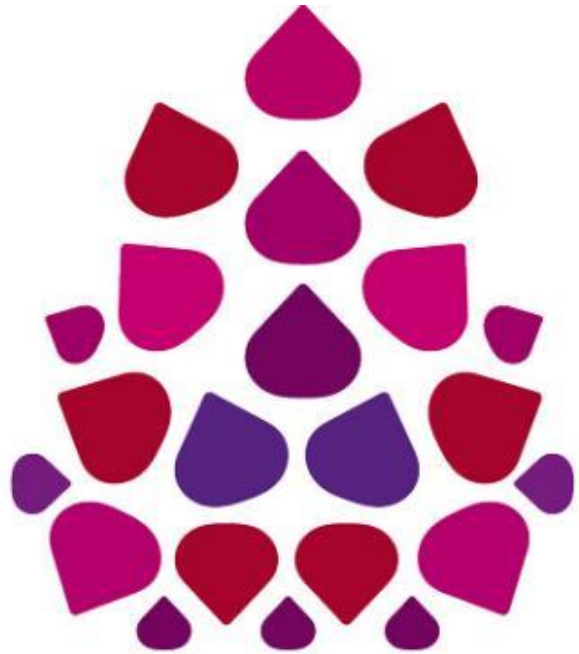
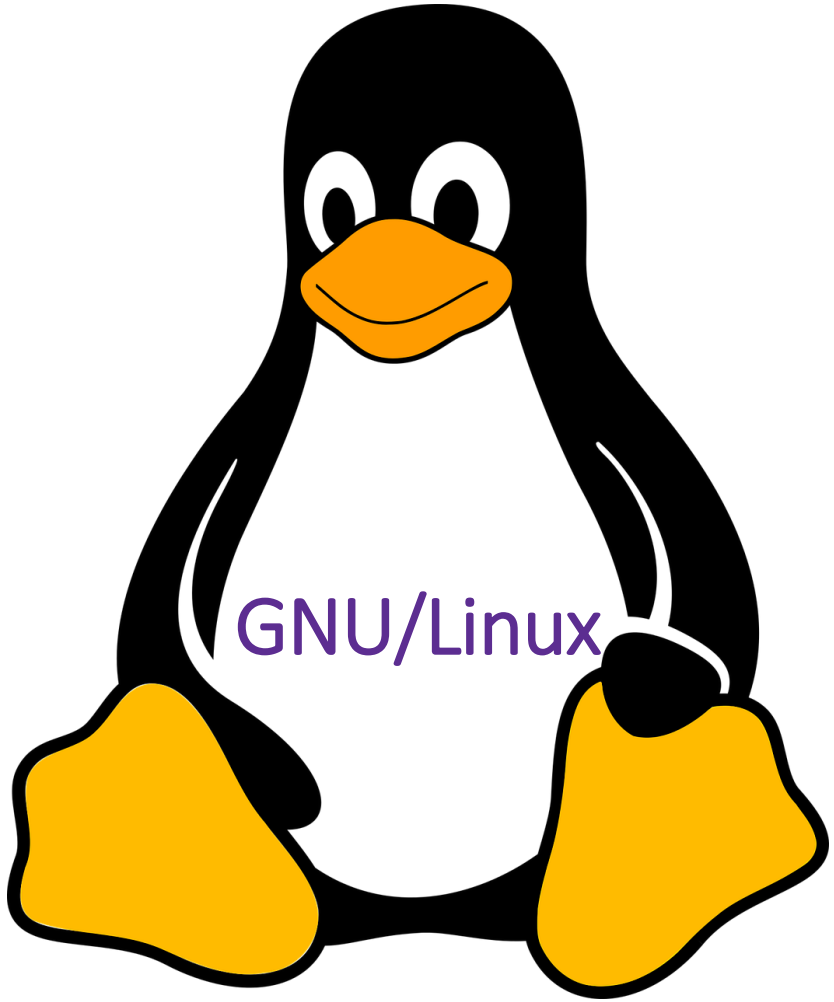


fundación esplai

ciudadanía comprometida


T Systems
Let’s power
higher performance


GNU/Linux

ESSENTIAL 3

Expresiones Regulares RegExp

Expresiones regulares. ¿Qué es?

Una expresión regular es una **secuencia de caracteres que define un patrón de búsqueda**, dicho técnicamente, una forma de representar los lenguajes regulares (finitos o infinitos) y se construye utilizando caracteres del alfabeto sobre el cual se define el lenguaje.

Expresiones regulares. ¿Origen?

El concepto de expresión regular surgió en la **década de los 50** cuando el matemático americano **Stephen Cole Kleene** quién lo formalizó dentro de los campos de la ciencia computacional teórica y la teoría de lenguaje formal.

Expresiones regulares. ¿Objetivo?

Su objetivo principal dentro de la informática es la de **encontrar un conjunto de caracteres** (cadena de caracteres) que **cumplen el patrón descrito**, así como ser capaz de reemplazarlo por otra cadena.

El motor de búsqueda interpreta ese patrón y busca todas las coincidencias para destacarlas, extraerlas o reemplazarlas.

Expresiones regulares. ¿dónde se usa?

Esta herramienta (expresiones regulares) son muy usadas:

- dentro de los **procesadores de lenguajes** (e.g. compiladores): para el analisis léxico;
- en cualquier editor de texto profesional, por ejemplo, **Notepad++** (Multiplataforma), VI (GNU-Linux/Unix), **Emacs** (Multiplataforma), **LibreOffice** (Multiplataforma), etc. Paquete Office, WordPad y Block de notas de Microsoft no lo permiten (bueno, permiten una manera similar o propia de hacer expresiones regulares);
- en GNU-Linux, editores de flujos como **sed**, la utilidad de línea de órdenes como **grep** o el lenguaje de procesado de texto AWK son increíblemente versátiles, gracias a la capacidad de usar entrada y salida estándar y enlazar su ejecución con otras órdenes cualquiera mediante pipes (**tuberías**);

Expresiones regulares. ¿dónde se usa?

Esta herramienta (expresiones regulares) son muy usadas:

- en el **tratamiento de logs**, gracias a su disponibilidad en línea de comandos, es uno de sus máximos exponentes tanto para sysadmin, analistas de seguridad y quizás algo menos usado para desarrolladores;

- dentro de la seguridad, existen sistemas como los SIM(Security Information Management), SEM(Security Event Management) o SIEM(Security Information and Event Management) que basan su funcionamiento de gestión y normalización de logs mediante expresiones regulares;

- en **validaciones y extracción automatizada de información web** (el mundo de las arañas o crawlers y scraping web) usan o se ayudan de las expresiones regulares;

- búsqueda de **fechas con 20 mil formatos**.

Expresiones regulares. Ventajas

- La **curva de aprendizaje** es quizás un poco **lioso** en un inicio, pero se hace muy sencillo después, ya que la sintaxis es la que es. Son unos cuantos conceptos y una lista de términos que aprenderse.

- Aprender puede ser muy práctico y sencillo: con un pequeño **cheat-sheet** y una herramienta online para probar expresiones regulares te ayudaran a aprender rápidamente.

- Permite abstraerte de ejemplos y buscar patrones que engloben lo que se desea.

Expresiones regulares. Ventajas

- **Simplificar la carga de tareas repetitivas** como Copy-Paste y además permite meter complejidad, es decir, no cambiar una palabra por otra, si no estructuras sintácticas (e.g. **editar/refactorizar gran cantidad de líneas de código**)

- **Validar entradas de datos** es muy cómoda y fácil con expresiones regulares, aún así siempre con mucho cuidado de controlar todos los casos que engloba el patrón.

- Importantísimo: es una herramienta que puedes usar casi como buscador universal, por la integración con casi todo (Salvo con los productos de Microsoft comentados anteriormente).

- Puede llegar a ser muy potente y veloz.

Expresiones regulares. Inconvenientes

- Hay **muchas maneras de lograr una búsqueda**, pero hay que jugar con la balanza tiempo-complejidad-optimización.

- Cuesta entender expresiones regulares, sobre todo al principio.

- Es muy sencillo acudir a **generalizaciones** ineficientes de buscar cualquier carácter (.) y el motor irá comprobando el carácter con todas las opciones (alfanuméricos,…). Cuidado dónde y cómo las uséis, ya que algo ineficiente multiplicado por muchos usuarios usando esa función significa DDoS.

- En páginas webs y lenguajes de etiquetas (HTML, XML, …) muchas veces es muchísimo más sencillo y óptimo jugar con expresiones XPath para delimitar dentro del contenido de las etiquetas o extraer valores de las propiedades.

Expresiones regulares. Ejemplos

Ejemplo 1: Extraer todos los apellidos de los usuarios que empiecen por A. Siendo cada registro del formato: “Lucía Pérez Rocasolano;Madrid;22/01/1989”:

Opción 1: Bastante específica, pero no cumple con las tildes, valora fechas inverosímiles y no es eficiente.

```
[A-Za-z]+\ ([A-Za-z]+\ [A-Za-z]+)\;[A-Za-z]+\;[0–9][0–9]\/[0–9][0–9]\/[0–9][0–9][0–9[0–9]
```

Expresiones regulares. Ejemplos

Explicación ejemplo 1
[A-Za-z]+ →Uno o más caracteres mayúsculas y minúsculas = Una palabra

\ →significa que escapa el siguiente carácter, es decir, en \ , representa el espacio y en \; representa el carácter ;

[0–9] → Una cifra del 0 al 9

( ) → Significa una agrupación: Se puede usar para extraer el contenido que hay entre los paréntesis tras encontrar una coincidencia del patrón entero.

```
[A-Za-z]+\ ([A-Za-z]+\ [A-Za-z]+)\;[A-Za-z]+\;[0–9][0–9]\/[0–9][0–9]\/[0–9][0–9][0–9[0–9]
```

Expresiones regulares. Ejemplos

Opción 2: Equivalente pero con otra terminología**.**

```
\w+\ (\w+\ \w+)\;\w+\;\d{2}\/\d{2}\/\d{4}
```

\w → representa una letra mayúscula y minúsculas
\d → representa un digito de 0 a 9
{num} → representa el número de veces que se repite el término precedente.

Opción 3: Muy eficiente, pero no tiene problemas con las tildes, no es específica y mantiene el problema con las fechas.

```
\S+\s\S+\s\S\;\S\;\d{2}\/\d{2}\/\d{4}
```

\s → representa un espacio o tabulador
\S → representa lo opuesto de \s, es decir, cualquier carácter que no sea un espacio (espacio o tabulador)

El uso del término \S permite hacer que cada carácter se compruebe únicamente con los valores de espacio y tabulador, por consiguiente entran todos los alfanuméricos y caracteres especiales.

Expresiones regulares. Ejemplos

Ejemplo 2: Extrae cualquier fecha con el formato Standard ISO 8601: “2016–08–21T12:34:48+01:00Z”
Opción: Extrae fechas únicamente con ese formato, pero no delimita fechas con un mes mayor que 12 y días mayores al 31.
.

```
(\d{4}\-\d{2}\-\d{2}T\d{2}\:\d{2}\:\d{2}\+\d{2}\:\d{2}Z)
```

La complejidad está en dedicarle más tiempo a especificar los valores permitidos.
Si le quitas opciones la regex quedará más larga, pero más rápida de procesar.

Para terminar, os dejo una lista, más que suficiente, para aprender respecto al mundo de las expresiones regulares y ayudarse en este camino.

Expresiones regulares. Ejemplos

Ejemplo 2: Extrae cualquier fecha con el formato Standard ISO 8601: “2016–08–21T12:34:48+01:00Z”
Opción: Extrae fechas únicamente con ese formato, pero no delimita fechas con un mes mayor que 12 y días mayores al 31.
.

```
(\d{4}\-\d{2}\-\d{2}T\d{2}\:\d{2}\:\d{2}\+\d{2}\:\d{2}Z)
```

La complejidad está en dedicarle más tiempo a especificar los valores permitidos.
Si le quitas opciones la regex quedará más larga, pero más rápida de procesar.

Para terminar, os dejo una lista, más que suficiente, para aprender respecto al mundo de las expresiones regulares y ayudarse en este camino.

Expresiones regulares.

- **CARACTERES ESPECIALES**
  - **^** (acento circunflejo): Inicio de linea
  - **$**: Fin de línea
  - **.** :cualquier carácter. Si queremos quitar el significado especial tendremos que poner la contrabarra \ delante
- **CORCHETES []**
  - Se usan para indicar que en una posición determinada pueden aparecer un conjunto determinado de caracteres. Por ejemplo **[aeiou]** significa donde se encuentre puede haber cualquier vocal minúscula.
  - Ejemplo: c[aei]s[ao] => casa 👍 cese 👎 caso 👍 ceso 👍 casi 👎

Expresiones regulares.

- **CLASES: representan un conjunto predefinido de caracteres**

```
[:alnum:] : Las letras y Dígitos
[:alpha:] : Letras
[:blank:] : Espacios en Blanco
[:cntrl:] : Caracteres de control
[:space:] : Los Espacios en Blanco verticales y horizontales
[:graph:] : Caracteres imprimibles, sin incluir el Espacio en Blanco
[:print:] : Caracteres imprimibles, incluyendo el Espacio en Blanco
[:digit:] : Dígitos
[:lower:] : Letras minúsculas.
[:upper:] : Letras mayúsculas.
[:punct:] : Signos de puntuación.
```

Expresiones regulares.

- **CARACTERES ESPECIALES: ^, $, .**
- **CORCHETES [] y CLASES [:digit:],[:upper:]**, etc..
- **CORCHETES**
  - **Exclusión** [^]: Se usa para indicar que en una posición puede encontrarse cualquier carácter **EXCEPTO** los que se encuentran entre corchetes.
  - Ejemplo: `c[^aei]s[^ao]` => casa 👎 cese 👎 cose 👍 cusi 👍 coso 👎
  - **Rangos** [-]: Se usa para indicar todos los valores intermedios entre un inicio y un final. Tienen que ser datos con una ordenación conocida, por ejemplo números o letras:
  - Ejemplo: `c[a-d]s[0-5]` sería igual a `c[abcd]s[012345]`

Las expresiones se pueden mezclar, por ejemplo: `[3-8[:upper:]mty]` en esta posición se admiten números del 3 al 8 o cualquier mayúscula o las minúsculas m, t o y.

Expresiones regulares.

- **REPETICIONES (Regex Extendidas)**

X*	El asterisco concuerda con cero o más repeticiones de la expresión regular que le precede (X)
X?	El carácter interrogación concuerda con cero o una aparición de la expresión regular que le precede (X)
X+	El signo más concuerda con una o más repeticiones de la expresión regular que le precede (X)
X{n}	Concuerda con n repeticiones exactas de X
X{n,}	Concuerda con n o más repeticiones de X
X{,n}	Concuerda con cero o a lo sumo n repeticiones de X
X{n,m}	Concuerda con al menos n repeticiones de X, o como mucho m repeticiones.

Expresiones regulares. Los más utilizados

?. Representa un único carácter. Así por si ejecutas **ls /dev/sda?** te listará todos los archivos que sean igual a /dev/sda más un único caracter, que puede ser cualquier letra o número. Así en mi caso, ha listado mis unidades, de la /dev/sda1 a la /dev/sda7.

*. Este comodín representa desde nada, hasta cualquier cantidad de caracteres y dígitos. Así, en el ejemplo que he indicado anteriormente, si ejecutados **ls /dev/sd*** te listará /dev/sda, /dev/sda1 a /dev/sda7 y si tienes otra unidad, /dev/sdb, /dev/sdb1,…

[]. En este caso, este comodín representa un rango, ya sea de caracteres o de números. Siguiendo con el ejemplo anterior, podemos listar **ls /dev/sda[1-5]**. Ya te puedes hacer una idea de lo que va a listar.

Expresiones regulares. Los más utilizados

{}. Esto en realidad es mas un conjunto de comodines separados por comas. Igual que en casos anteriores, podrías ejecutar la siguiente orden **ls {sda*,sdb*}**.
Cuidado con dejar espacios alrededor de la coma, por que te dará un error.

[!]. El funcionamiento de este comodín es similar a [], salvo que representa justo lo contrario. Es decir se trata de listar todo aquello que que no esté en ese rango. Por ejemplo, **ls /dev/sda[!1-5]** te mostrará /dev/sda6 y /dev/sda7. Como te imaginas, esto es en mi caso, en tu equipo seguramente el resultado será otro.

****. Esta es la secuencia de escape y que tienes que tener siempre muy presente. Con este carácter puedes mostrar otros caracteres. Así, si quieres crear el directorio esta casa y ejecutas mkdir esta casa, te creará dos directorios, esta y casa. Para hacer lo que quieres, tienes diferentes alternativas, o bien **mkdir "esta casa"** o bien **mkdir esta\ casa**. En esta segunda orden utilizamos \ para escapar el carácter espacio.

Expresiones regulares. Los más utilizados

Las expresiones regulares tienen un potencial increíble, pero a diferencia de los comodines necesitan una mayor esfuerzo por tu parte. Por una lado necesitan de una curva de aprendizaje mas o menos pronunciada, y por otro lado, su aplicación no es tan intuitiva como en el caso de los comodines.

A continuación encontrarás algunos de los elementos con los que construir expresiones regulares. Sin embargo, al contrario que con los comodines, dejaré los ejemplos para el final.

Expresiones regulares. Los más utilizados

. Representa un solo carácter, es el equivalente a la ? de los comodines.

\. Se utiliza para escapar caracteres, al igual que en el caso de los comodines.

+. El elemento precedente puede aparecer 1 o mas veces.

***.** Es similar al anterior, pero en este caso, el elemento puede aparecer **cero** o mas veces.

^ ó \A. Dependiendo de su ubicación tiene un significado u otro. Así si lo encuentras al principio se
corresponde con que lo que buscamos debe comenzar igual. Si lo encontramos entre corchetes es una negación, como veremos mas adelante.

Expresiones regulares. Los más utilizados

$ ó \Z. Es el opuesto al anterior, en tanto en cuanto, cuando lo encuentras al final indica que lo que buscamos tiene que acabar igual.

[]. Especifica un rango, ya sea separado por comas como [a,b,c] o bien [a-c].

|. Representa un **o lógico**.

[^]. Como he indicado anteriormente, esta expresión se utiliza para negar rangos.

{}. Indica el número de repeticiones del caracter precedente. Así a{3} es equivalente a aaa. Pero además a{3,} representa 3 ó mas a. En el caso de a{1,5} se refiere a entre 1 y 5.

Expresiones regulares. Los más utilizados

Además de estas que son muy parecidas a los comodines, en el caso de las expresiones regulares tenemos mas, como son las siguientes,

\s. Se corresponde con un espacio en blanco.

\S. El contrario del anterior, es decir, cualquier carácter que no sea un espacio en blanco.

\d. Equivale a un dígito.

\D. Cualquier carácter que no sea un dígito.

\w. Se corresponde a cualquier carácter que se pueda utilizar en una palabra. Es equivalente a [a-zA-Z0-9_].

\W. El opuesto al anterior, es decir, es equivalente a [^a-zA-Z0-9_].

Expresiones regulares. Los más utilizados

Por supuesto, también tiene en cuenta los *caracteres escapados*, que comenté anteriormente, y entre los que cabe citar,

\n. Se corresponde con una línea nueva.

\r. Para retorno de carro.

\t. Representa una tabulación.

\0. Se utiliza para un carácter nulo

() cuyo objetivo es el de capturar todo lo que está en su interior.

Expresiones regulares. Algunas RegExp útiles

Indicarte antes de comentarte sobre estos ejemplos de expresiones regulares que, normalmente van delimitados por ***//.*** Es decir, la expresión regular viene a ser /expresión-regular/.

Caracteres alfabéticos. Estos patrones son los más sencillos de utilizar, sin embargo es un buen comienzo.

```
/^[a-zA-Z]*$/
```

Caracteres en minúsculas. Igual que en el caso anterior, pero esta vez en minúculas.

```
/^([a-z]*$
```

Números. La tercera opción pero solo para dígitos. Sin embargo, a partir de ahora, la cosa se va a ir complicando.

```
/^[0-9]*$/
```

Expresiones regulares. Algunas RegExp útiles

Nombre de usuario. . En este caso, el nombre de usuario debe tener una longitud mínima de tres caracteres y máxima de 16. Solo puede estar compuesto por letras minúsculas, números, además de _ y -.

```
/^[a-z0-9_-]{3,16}$/
```

Podrías incluir también las mayúsculas de la siguiente forma, .

```
/^[a-zA-Z0-9_-]{3,16}$/
```

Contraseña. Esto es muy similar al anterior, pero, por un lado podemos considerar cambiar la longitud mínima y máxima de la contraseña, y por otro lado incluir algunos otros caracteres. Hemos aumentado la longitud mínima a 8 y la máxima a 20 y además hemos incorporado algunos caracteres mas o menos extraños como pueden ser $!¡@?¿=;:.

```
[a-zA-Z0-9_-$!¡@?¿=;:]{8,20}$/
```

Expresiones regulares. Algunas RegExp útiles

Correo electrónico. Este patrón ya tiene un poco mas d e dificultad.
Se divide en tres grupos diferenciados, tipo antes@despues.fin.
Los dos primeros admiten letras, números, además de _, - y .
Mientras que el último solo admite letras y el .
pero además con una longitud mínima de 2 y máxima de 6.

```
/^([a-z0-9_\.-]+)@([a-z0-9_\.-])\.([a-z\.]{2,6})$/
```

Dirección web. Otro patrón realmente interesante,

```
/^(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]*)*\/?$/
```

Código postal.

```
[0-9]{5}(\-?[0-9]{4})?$
```

Expresiones regulares. Algunas RegExp útiles

Para direcciones IP. Con esto nos aseguramos que cada uno de los grupos es número comprendido entre 0 y 255.

```
/^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$
```

Etiquetas html. Esta es muy interesante y seguro que la utilizarás en mas de una ocasión. Se trata de un patrón para etiquetas html.

```
/^<([a-z]+)([^<]+)*(?:>(.*)<\/\1>|\s+\/>)$/
```

Tarjetas de crédito. Como ves las cosas se van complicando, pero las posibilidades son realmente espectaculares. Puedes hacer casi cualquier cosa que te puedas imaginar, tan solo es cuestión de estudiarlo con el suficiente detalle. .

```
/^(?:4[0-9]{12}(?:[0-9]{3})?|5[1-5][0-9]{14}|6011[0-9]{12}|622((12[6-9]|1[3-9][0-9])|([2-8][0-9][0-9])|(9(([0-1][0-9])|(2[0-5]))))[0-9]{10}|64[4-9][0-9]{13}|65[0-9]{14}|3(?:0[0-5]|[68][0-9])[0-9]{11}|3[47][0-9]{13})*$/
```

Expresiones regulares. Algunas RegExp útiles

Fechas. Al fin y al cabo, se pueden establecer expresiones regulares que nos ayuden en todas nuestras taresm y simjplifiquen el trabajo.

```
'#^((19|20)?[0-9]{2}[- /.](0?[1-9]|1[012])[- /.](0?[1-9]|[12][0-9]|3[01]))*$#
```

Imágenes. /[^\s]+(?=\.(jpg|gif|png))\.\2)

```
/[^\s]+(?=\.(jpg|gif|png))\.\2)
```

Expresiones regulares. Resumen

El Caracter `.`

El caracter `.` representa...

Cualquier caracter o símbolo disponible. Si dices `ab.ve`, estás diciendo algo que comienza con `ab` y termina con `ve`

Puedes usar el `.` tantas veces como quieras; la expresión regular reemplazará el `.` con cualquier carácter tantas veces como aparezca el `.`

Usa el contenedor de la derecha para jugar con otras sucesiones simples de caracteres.

Expresiones regulares. Resumen

El Caracter Rango `[ ]`

El caracter `[ ]` representa...

Un grupo de posibles caracteres. A veces, nos gustaría ser un poco más específicos... aquí es donde los **rangos** son útiles. Especificamos un rango de caracteres encerrándolos entre corchetes ([]).

También puede usar el `[]` para alinear números o letras con un guión intermedio. El guión representa un rango de números o caracteres. Por ejemplo:

- `[0-9]` representa cualquier número entre 0 y 9.
- `[a-z]` representa cualquier letra en minúscula
- `[A-Z]` Representar cualquier letra en mayúsculas.

También puedes combinar rangos de caracteres de esta forma:

- Cualquier letra en mayúsculas o minúsculas: `[a-zA-Z]`
- Números del 1 al 5 y también del 9: `[1-59]`
- Números del 1 al 5, letras de la a la f y también la X mayúscula: `[1-5a-fX]`

Expresiones regulares. Resumen

El caracter `^` (simbolo de intercalación): Negación o comienzo de un término

Si colocamos `^` al comienzo de un [rango]:

Estamos negando el rango. Por ejemplo:

- Todos los términos que comienzan con `li` y terminan con `e` pero no tienen `i` o `v` en el interior: `li [^ v] e`

Si colocamos `^` al comienzo de una expresión regular:

- Estamos diciendo que solo queremos probar el Regex desde el principio del string (no se evaluarán los substrings - partes más pequeñas de la cadena):
- Un string que comienza con http: `^http`

Expresiones regulares. Resumen

Atajos para los dígitos `\d` y Palabras `\w`

Si lo prefieres, puedes usar estos accesos directos en tus expresiones regulares:

Operador	Descripción
\w	Coincide con cualquier caracter de palabra (igual a `[a-zA-Z0-9_]`)
\W	Coincide con cualquier otra cosa que no sea una letra, dígito o subrayado
\d	Coincide con cualquier dígito decimal. Equivalente a `[0-9]`
\D	Coincide con cualquier cosa que no sea un dígito decimal

Expresiones regulares. Resumen

Agrupar o encerrar expresiones regulares con `()`

Siempre hablamos de "divide y vencerás", ¿verdad? Bueno, tu mejor amigo para eso será el operador de paréntesis `()`. Ahora podemos agrupar cualquier patrón como lo hacemos en matemáticas.

Ahora que podemos agrupar, podemos multiplicar (repetir) nuestros patrones, negarlos, etc.

Por ejemplo, este Regex acepta una o muchas repeticiones del string `ab` seguido de una letra `c` al final: `(ab) * c`

Expresiones regulares. Resumen

Agrupar o encerrar expresiones regulares con `()`

Siempre hablamos de "divide y vencerás", ¿verdad? Bueno, tu mejor amigo para eso será el operador de paréntesis `()`. Ahora podemos agrupar cualquier patrón como lo hacemos en matemáticas.

Ahora que podemos agrupar, podemos multiplicar (repetir) nuestros patrones, negarlos, etc.

Por ejemplo, este Regex acepta una o muchas repeticiones del string `ab` seguido de una letra `c` al final: `(ab) * c`

Expresiones regulares. Resumen

Agrupar o encerrar expresiones regulares con `()`

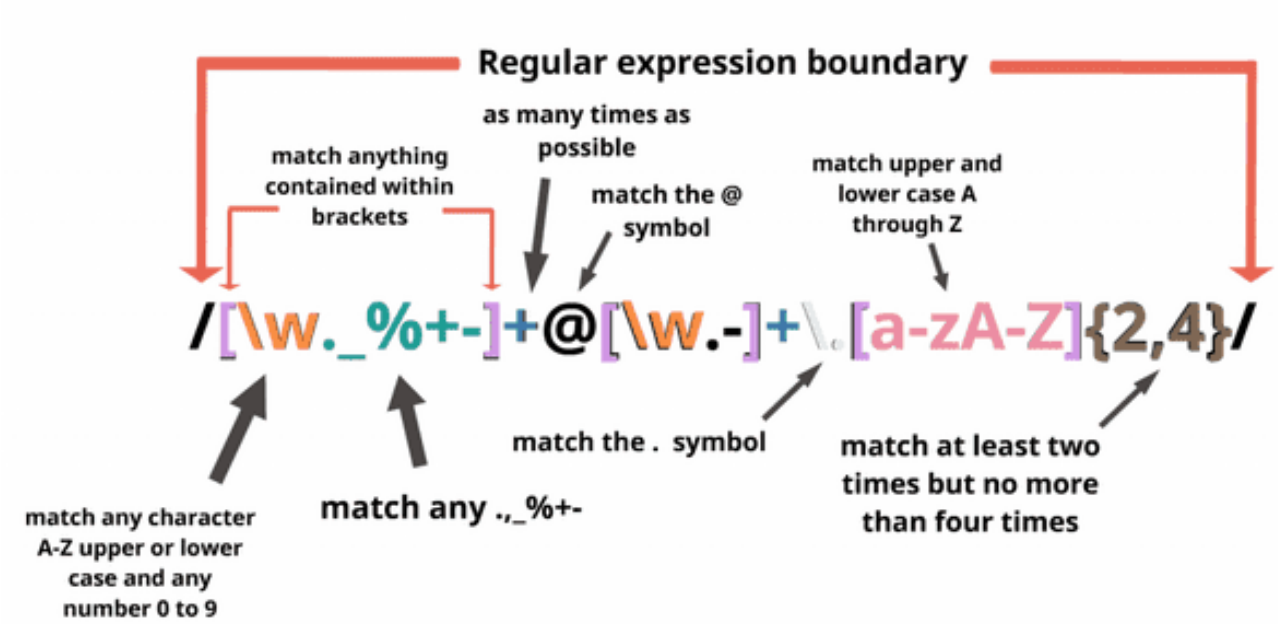
Siempre hablamos de "divide y vencerás", ¿verdad? Bueno, tu mejor amigo para eso será el operador de paréntesis `()`. Ahora podemos agrupar cualquier patrón como lo hacemos en matemáticas.

Ahora que podemos agrupar, podemos multiplicar (repetir) nuestros patrones, negarlos, etc.

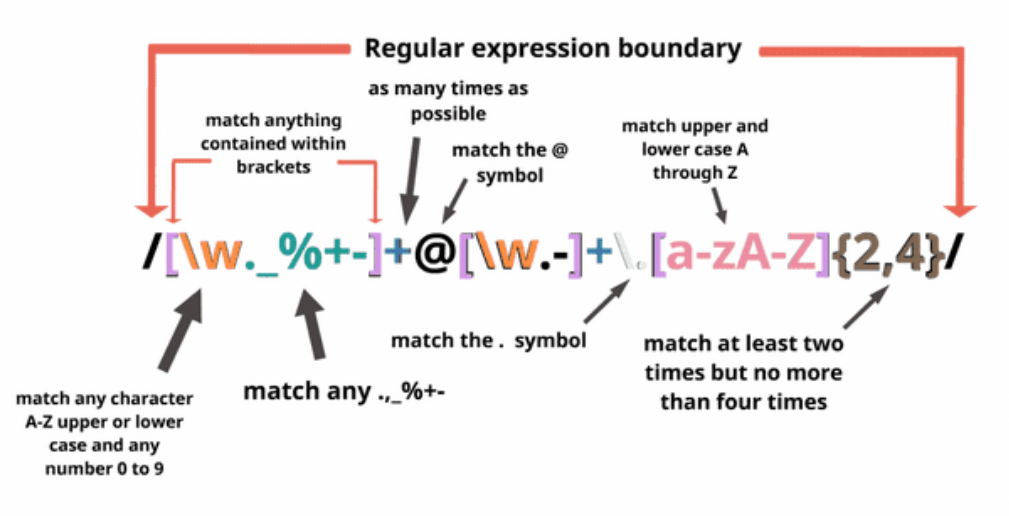
Por ejemplo, este Regex acepta una o muchas repeticiones del string `ab` seguido de una letra `c` al final: `(ab) * c`

Expresiones regulares. Correo electrónico

Expresión regular para: validar un correo electrónico



Expresiones regulares. Correo electrónico




```
[\w._%+-]+@[\w.-]+\.[a-zA-Z]{2,4}
```

Comenzamos diciéndole al analizador que encuentre el principio de la cadena (^).

Dentro del primer grupo, unimos una o más letras minúsculas, números, guiones bajos, puntos o guiones.

Hemos escapado del punto porque un punto no escapado significa cualquier caracter.

Directamente después de eso, debe haber un signo @.

El siguiente es el nombre de dominio, que debe ser: una o más letras minúsculas, números, guiones bajos, puntos o guiones. Luego otro punto (escapado), con la extensión de dos a seis letras o puntos. Tengo 2 a 6 debido a los TLD específicos del país (.ny.us o .co.uk).

Finalmente, queremos el final de la cadena ($).

Expresiones regulares. fechas

Fecha dd-mm-aaaa

```
^(?:3[01]|[12][0-9]|0?[1-9])([\-/.])(0?[1-9]|1[1-2])\1\d{4}$
```

Ejemplos:
31.12.3013
01/01/2013
5-3-2013
15.03.2013

Fecha mm-dd-aaaa

```
^(?:0?[1-9]|1[1-2])([\-/.])(3[01]|[12][0-9]|0?[1-9])\1\d{4}$
```

Ejemplos:
12/14/2013
03-05-2013
3-15-2013
3.5.2013

Fecha aaaa-mm-dd

```
^\d{4}([\-/.])(0?[1-9]|1[1-2])\1(3[01]|[12][0-9]|0?[1-9])$
```

Ejemplos:
2013-12-14
2013-07-08
2013-7-14
2013/11/8
2013.11.8

Expresiones regulares. horas

Sistema horario de 12 horas hh:mm

```
^(?:0?[1-9]|1[0-2]):[0-5][0-9]\s?(?:[aApP](\.?)[mM]\1)?$
```

3:14 03:14 12:27
1:05am 1:05a.m.
1:05PM 1:05P.M.

Sistema horario de 24 horas hh:mm

```
^([01]?[0-9]|2[0-3]):[0-5][0-9]$
```

1:34 01:34
12:27 15:09 0:05

Sistema horario de 24 horas hh:mm:ss

```
^([01]?[0-9]|2[0-3]):[0-5][0-9](:[0-5][0-9])?$
```

3:34 03:34 12:27
0:05:21 15:09:21
23:45:21

Expresiones regulares. Enlaces útiles

Enlaces útiles

regexr→Herramienta online para probar, depurar y aprender regex.
regex101→Herramienta online para probar y depurar regex en diferentes lenguajes.
regexper→Herramienta para visualizar graficamente cada elemento del patrón regex introducido.
regular-expressions.info→Web en inglés con información completa sobre regex.
eBook→Mastering Regular Expressions, 3rd Edition. Ed. O'REILLY.
regex cheat-sheet→Chuleta completa sobre regex.

Contacto

www.fundacionesplai.org

Facebook.com/FundacionEsplai

@fundacionEsplai

fundacion@fundacionesplai.org

SEDE MADRID Distrito LATINA / Barrio LUCERO C/ latina, 21, local 13 28047 – MADRID 911 681 686	CENTRE ESPLAI Calle Riu Anoia, 42-54 08820 - El Prat de Llobregat BARCELONA 934 747 474

T Systems Let's power higher performance

fundaciónesplai