Curso Básico de Linux

Expresiones Regulares

Contenido

Qué son las expresiones regulares

Las expresiones regulares (ER) son una forma de describir cadenas de caracteres. Una *expresión regular* es un patrón capaz de reconocer o filtrar cadenas de caracteres según ciertos criterios. El uso de comodines `*` para indicar cadenas de caracteres cualesquiera o `?` para indicar un caracter único son ejemplos de uso de expresiones regulares. Así, el patrón `aba*` reconoce cadenas como abaco, abajo, abatimiento, abalorio, aba-23; el patrón `do?` reconoce cadenas como doy dos, dot, don, do$; el patrón `aba*-txt` describe el conjunto de cadenas de caracteres que comienzan con `aba`, contienen cualquier otro grupo de caracteres, luego un guión, y finalmente la cadena `txt`.

Los patrones construidos como expresiones regulares permiten reconocer cadenas de caracteres de estructura compleja; esto las hace muy útiles para realizar búsquedas o sustituciones en textos. Las expresiones regulares son reconocidas por muchos lenguajes de programación, editores y otras herramientas. El editor `vi` y sus variantes `vim` y `gvim` las aceptan. También son aceptadas para búsquedas en textos dentro del comando `less`, con `/` para buscar hacia adelante y `?` para buscar hacia atrás.

Las expresiones regulares se construyen como las expresiones aritméticas, usando operadores para combinar expresiones más pequeñas. Analizaremos esos operadores y las reglas de construcción de expresiones regulares, atendiendo siempre al conjunto de cadenas que representa cada patrón.

Metacaracteres

La construcción de expresiones regulares depende de la asignación de significado especial a algunos caracteres. En el patrón `aba*-txt` el caracter `*` no vale por sí mismo, como el caracter asterisco, sino que indica un "conjunto de caracteres cualesquiera". Asimismo, el caracter `?` no se interpreta como el signo de interrogación sino que representa "un caracter cualquiera y uno solo". Estos caracteres a los que se asigna significado especial se denominan *metacaracteres*.

El conjunto de metacaracteres para expresiones regulares es el siguiente:

```
\ ^ $ . [ ] { } | ( ) * + ?
```

Estos caracteres, en una expresión regular, son interpretados en su significado especial y no como los caracteres que normalmente representan. Una búsqueda que implique alguno de estos caracteres obligará a "escaparlo" de la interpretación. Esto puede hacerse anteponiéndole el símbolo `\`, como se hace para evitar la interpretación por el shell de los metacaracteres propios del shell. En una expresión regular, el caracter `?` representa "un caracter cualquiera"; si queremos representar el caracter tal cual, como signo de interrogación, debemos escribir `\?`. Los caracteres anteriores, para ser interpretados como tales y no como metacaracteres, deben escribirse anteponiensdo `\`.

Versiones de sintaxis

Existen varias versiones de sintaxis para expresiones regulares. Las más sencillas son las expresiones regulares básicas (BRE, Basic Regular Expressions). Una sintaxis ampliada son las llamadas expresiones regulares extendidas (ERE, Extended Regular Expressions). Funcionalidades adicionales aparecen implementadas en las llamadas expresiones regulares compatibles con Perl (PCRE, Perl Compatible Regular Expressiones). Perl es un lenguaje de programación especialmente potente para el manejo de cadenas de caracteres. En Linux, comandos como `grep` para filtrar líneas de texto según un patrón soportan las expresiones regulares básicas y extendidas sin diferenciación. En muchos sistemas Linux también están disponibles la expresiones regulares compatibles con Perl.

Expresiones regulares básicas

Una expresión regular (ER) determina un conjunto de cadenas de caracteres. Un miembro de este conjunto de cadenas se dice que aparea, equipara o satisface la expresión regular.

ERs de un solo caracter

Las expresiones regulares se componen de expresiones regulares elementales que aparean con un único caracter:

Expresión	**aparea con**
`c`	caracter ordinario c
`.`	(punto) aparea con un caracter cualquiera excepto nueva línea
`[abc]`	ER de un caracter que aparea con uno de a, b, c
`[^abc]`	ER de un caracter que no sea uno de a, b, c
`[0-9]` `[a-z]` `[A-Z]`	ERs de un caracter que aparean con cualquier caracter en el intervalo indicado El signo `-` indica un intervalo de caracteres consecutivos

`\e`	ER que aparea con alguno de estos caracteres (en lugar de la e):
	`. * [ \` cuando no están dentro de `[ ]`
	`^` al principio de la ER, o al principio dentro de `[ ]`
	`$` al final de una ER
	`/` usado para delimitar una ER

Los paréntesis rectos `[]` delimitan listas de caracteres individuales. Muchos metacaracteres pierden su significado si están dentro de estas listas: los caracteres especiales `.` `*` `[` `\` valen por sí dentro de `[]`. Para incluir un caracter `]` en una lista, colocarlo al principio; para incluir un `^` colocarlo en cualquier lugar menos al principio; para incluir un `-` colocarlo al final.

Dentro de los conjuntos de caracteres individuales, se reconocen las siguientes categorías:

`[:alnum:]`	caracteres alfanuméricos
`[:alpha:]`	caracteres alfabéticos
`[:cntrl:]`	caracteres de control
`[:digit:]`	dígitos
`[:graph:]`	caracteres gráficos
`[:lower:]`	minúsculas
`[:print:]`	caracteres imprimibles
`[:punct:]`	signos de puntuación
`[:space:]`	espacios
`[:upper:]`	mayúsculas
`[:xdigit:]`	dígitos hexadecimales

En estos nombres de categorías, los paréntesis rectos forman parte del nombre de la categoría, no pueden ser omitidos.

Las categorías también pueden escribirse en forma explícita. Por ejemplo, `[[:alnum:]]` significa `[0-9A-Za-z]`. Sin embargo, esta última expresión explícita depende de la secuencia de codificación ASCII, en tanto la expresión con categorías es portable, no pierde su significado cualquiera sea la codificación de caracteres del sistema (ASCII, UTF-8, etc.).

Construcción de Expresiones Regulares

Una expresión regular se construye con uno o más operadores que indican, cada uno, el caracter a buscar. Los operadores más comunes y aceptados son los siguientes:

Operador	**Significado**
`c`	un caracter no especial concuerda consigo mismo
`\c`	elimina significado especial de un caracter c; el \ escapa el significado especial
`^`	indica ubicado al comienzo de la línea (cadena nula al principio de línea)
`$`	indica ubicado al final de la línea (cadena nula al final de línea)
`.`	(punto) un caracter individual cualquiera
`[...]`	uno cualquiera de los caracteres incluidos; acepta listas del tipo a-z, 0-9, A-Z
`[^...]`	un caracter distinto de los indicados; acepta intervalos del tipo a-z, 0-9, A-Z
`r*`	0, 1 o más ocurrencias de la ER `r` (repetición)
`r1r2`	la ER `r1` seguida de la ER `r2` (concatenación)

Ejemplos de Expresiones Regulares Básicas

Las expresiones regulares se aprenden mejor con los ejemplos y el uso.

Expresión	**aparea con**

`a.b`	axb aab abb aSb a#b ...
`a..b`	axxb aaab abbb a4$b ...
`[abc]`	a b c (cadenas de un caracter)
`[aA]`	a A (cadenas de un caracter)
`[aA][bB]`	ab Ab aB AB (cadenas de dos caracteres)
`[0123456789]`	uno de 0 1 2 3 4 5 6 7 8 9
`[0-9]`	uno de 0 1 2 3 4 5 6 7 8 9
`[A-Za-z]`	uno de A B C ... Z a b c ... z
`[0-9][0-9][0-9]`	tres dígitos 000 001 .. 009 010 .. 019 100 .. 999
`[0-9]*`	cadena_vacía 0 1 9 00 99 123 456 999 9999 ...
`[0-9][0-9]*`	0 1 9 00 99 123 456 999 9999 99999 99999999 ...
`[0-9][0-9]+`	00 01 29 474 99 123 456 999 9999 99999 99999999 ...
`^.*$`	cualquier línea completa
`^$`	una línea vacía

En el editor `vi`, las expresiones regulares permiten realizar búsquedas tales como:

```
/^Desde
```

busca líneas que empiecen con la cadena `Desde`.

```
/final$
```

busca líneas que termine con la cadena `final`.

```
/\$25
```

busca líneas que contengan `$25`; `\` escapa el $.

Expresiones Regulares Extendidas

Las expresiones regulares extendidas agregan a las expresiones regulares básicas algunos operadores que permiten construcciones más complejas:

Operador	**Significado**
`r+`	1 o más ocurrencias de la ER r
`r?`	0 o una ocurrencia de la ER r, y no más
`r{n}`	n ocurrencias de la ER r
`r{n,}`	n o más ocurrencias de la ER r
`r{,m}`	0 o a lo sumo m ocurrencias de la ER r
`r{n,m}`	n o más ocurrencias de la ER r, pero a lo sumo m
`r1|r2`	la ER r1 o la ER r2 (alternativa)
`(r)`	ER anidada
`"r"`	evita que los caracteres de la ER r sean interpretados por el shell

La repetición tiene precedencia sobre la concatenación; la concatenación tiene precedencia sobre la alternativa. Una expresión puede encerrarse entre paréntesis para ser evaluada primero.

Ejemplos de Expresiones Regulares Extendidas

Expresión	aparea con
`[0-9]+`	`0 1 9 00 99 123 456 999 9999 99999 99999999 ..`
`[0-9]?`	`cadena_vacía 0 1 2 .. 9`
`^a|b`	`a b`
`(ab)*`	`cadena_vacía ab abab ababab ...`
`^[0-9]?b`	`b 0b 1b 2b .. 9b`
`([0-9]+ab)*`	`cadena_vacía 1234ab 9ab9ab9ab 9876543210ab 99ab99ab ...`

Expresiones regulares en Linux

En Linux no se distinguen expresiones regulares básicas de extendidas; los comandos aceptan todas las expresiones regulares. En ese caso, como siempre se están usando extendidas, los metacaracteres

```
?  +  {  |  (  )
```

deben ser escapados cuando se quieren usar como caracteres normales, escribiendo

```
\?  \+  \{  \|  \(  \)
```

El comando `grep`

Para experimentar con expresiones regulares, es útil el comando `grep` (Global Regular Expression and Print). Este comando busca un patrón expresado como una expresión regular dentro de uno o varios archivos de texto o de su entrada estándar, extrayendo las líneas donde se encuentra el patrón indicado. Su sintaxis es

```
grep [opciones] patrón [archivo] ...
```

donde el patrón a buscar es una expresión regular.

```
grep bash /etc/passwd
cat /etc/passwd | grep bash
```

en ambos casos, se obtienen las líneas del archivo `/etc/passwd` donde figura la palabra `bash`, y solo ésas.

Referencias, lecturas complementarias

- Shotts, William. *The Linux Command Line*. Third Internet Edition, 2016. Chapter 19, Regular Expressions. Disponible online: https://razaoinfo.dl.sourceforge.net/project/linuxcommand/TLCL/16.07/TLCL-16.07.pdf
- Páginas man: `regex(7)`, `grep`.
- Wikipedia: Regular expression.