UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Universidad Técnica Federico Santa María

Tarea Certamen 1

"Expresiones Regulares en UNIX"

Introducción a la Informática Teórica

Cristián Maureira - 2673030-9
cmaureir@inf.utfsm.cl

Valparaíso, Septiembre 2008.

1. Introducción

Una expresión regular, a menudo llamada también patrón, es una expresión que describe un conjunto de cadenas sin enumerar sus elementos.

Además, normalmente representan otro grupo de caracteres mayor, de tal forma que podemos comparar el patrón con otro conjunto de caracteres para ver las coincidencias.

La mayoría de las formalizaciones proporcionan los siguientes constructores: una expresión regular es una forma de representar a los lenguajes regulares (finitos o infinitos) y se construye utilizando caracteres del alfabeto sobre el cual se define el lenguaje.

Específicamente, las expresiones regulares se construyen utilizando los operadores unión, concatenación y clausura de Kleene.

Las expresiones regulares en UNIX (RE's) están recogidas en el estándar POSIX 1003.2.

2. Simbología y Patrones

2.1. El Punto "."

El punto es interpretado por el motor de búsqueda como cualquier otro carácter excepto los caracteres que representan un salto de línea, a menos que se le especifique esto al motor de Expresiones Regulares.

El punto se utiliza de la siguiente forma:
Si se le dice al motor de RegEx que busque "g.t" en la cadena "el gato de piedra en la gótica puerta de getisboro goot" el motor de búsqueda encontrará "gat", "gót" y por último "get".

Nótese que el motor de búsqueda no encuentra "goot"; esto es porque el punto representa un solo carácter y únicamente uno.

Aunque el punto es muy útil para encontrar caracteres que no conocemos, es necesario recordar que corresponde a cualquier carácter y que muchas veces esto no es lo que se requiere.

Es muy diferente buscar cualquier carácter que buscar cualquier carácter alfanumérico o cualquier dígito o cualquier no-dígito o cualquier no-alfanumérico.

2.2. Los corchetes "[ ]"

La función de los corchetes en el lenguaje de las expresiones regulares es representar "clases de caracteres", o sea, agrupar caracteres en grupos o clases.

Son útiles cuando es necesario buscar uno de un grupo de caracteres.

Dentro de los corchetes es posible utilizar el guión "-" para especificar rangos de caracteres. Adicionalmente, los meta-caracteres pierden su significado y se convierten en literales cuando se encuentran dentro de los corchetes.

Por ejemplo para buscar cualquier carácter que representa un dígito o un punto podemos utilizar la expresión regular "[\$d.]".

Los corchetes nos permiten también encontrar palabras aún si están escritas de forma

errónea, por ejemplo, la expresión regular "expresi[oó]n" permite encontrar en un texto la palabra "expresión" aunque se haya escrito con o sin tilde.

2.3. La barra "|"

Sirve para indicar una de varias opciones. Por ejemplo, la expresión regular "$a \mid e$" encontrará cualquier "a" o "e" dentro del texto. La expresión regular "este|oeste|norte|sur" permitirá encontrar cualquiera de los nombres de los puntos cardinales. La barra se utiliza comúnmente en conjunto con otros caracteres especiales.

2.4. El signo peso "$"

Representa el final de la cadena de caracteres o el final de la línea, si se utiliza el modo multi-línea. No representa un carácter en especial sino una posición. Si se utiliza la expresión regular "\.$" el motor encontrará todos los lugares donde un punto finalice la línea, lo que es útil para avanzar entre párrafos.

2.5. El acento circunflejo "∧"

Este carácter tiene una doble funcionalidad, que difiere cuando se utiliza individualmente y cuando se utiliza en conjunto con otros caracteres especiales.

En primer lugar su funcionalidad como carácter individual: el carácter "∧" representa el inicio de la cadena (de la misma forma que el signo peso "$" representa el final de la cadena). Por tanto, si se utiliza la expresión regular "∧[a-z]" el motor encontrará todos los párrafos que den inicio con una letra minúscula. Cuando se utiliza en conjunto con los corchetes de la siguiente forma "[∧\$w$]" permite encontrar cualquier carácter que NO se encuentre dentro del grupo indicado. La expresión indicada permite encontrar, por ejemplo, cualquier carácter que no sea alfanumérico o un espacio, es decir, busca todos los símbolos de puntuación y demás caracteres especiales.

La utilización en conjunto de los caracteres especiales "∧" y "$" permite realizar validaciones en forma sencilla.

Por ejemplo "∧\d" permite asegurar que la cadena a verificar representa un único dígito inicial.

2.6. Los paréntesis "()"

De forma similar que los corchetes, los paréntesis sirven para agrupar caracteres, sin embargo existen varias diferencias fundamentales entre los grupos establecidos por medio de corchetes y los grupos establecidos por paréntesis:

- Los caracteres especiales conservan su significado dentro de los paréntesis.

- Los grupos establecidos con paréntesis establecen una "etiqueta" o "punto de referencia" para el motor de búsqueda que puede ser utilizada posteriormente como se denota más adelante.
- Utilizados en conjunto con la barra "|" permite hacer búsquedas opcionales. Por ejemplo la expresión regular "al (este|oeste|norte|sur) de" permite buscar textos que den indicaciones por medio de puntos cardinales, mientras que la expresión regular "este|oeste|norte|sur" encontraría "este" en la palabra "esteban", no pudiendo cumplir con este propósito.
- Utilizado en conjunto con otros caracteres especiales que se detallan posteriormente, ofrece funcionalidad adicional.

2.7. El signo de interrogación "?"

El signo de pregunta tiene varias funciones dentro del lenguaje de las expresiones regulares. La primera de ellas es especificar que una parte de la búsqueda es opcional. Por ejemplo, la expresión regular "ob?scuridad" permite encontrar tanto "oscuridad" como "obscuridad". En conjunto con los parentesis redondos permite especificar que un conjunto mayor de caracteres es opcional; por ejemplo "Nov(\.|iembre|ember)?" permite encontrar tanto "Nov" como "Nov.", "Noviembre" y "November".

2.8. Las llaves "{}"

Comúnmente las llaves son caracteres literales cuando se utilizan por separado en una expresión regular. Para que adquieran su función de metacaracteres es necesario que encierren uno o varios números separados por coma y que estén colocados a la derecha de otra expresión regular de la siguiente forma: "$\backslash d\{2\}$" Esta expresión le dice al motor de búsqueda que encuentre dos dígitos contiguos. Utilizando esta fórmula podríamos convertir el ejemplo "$\wedge\backslash d\backslash d/\backslash d\backslash d/\backslash d\backslash d\backslash d\backslash d\$$" que servía para validar un formato de fecha en "$\wedge\backslash d\{2\}/\backslash d\{2\}/\backslash d\{4\}\$$" para una mayor claridad en la lectura de la expresión.

2.9. El asterisco "*"

El asterisco sirve para encontrar algo que se encuentra repetido 0 o más veces.

Por ejemplo, utilizando la expresión "$[a-zA-Z]\backslash d*$" será posible encontrar tanto "H" como "H1", "H01", "H100" y "H1000", es decir, una letra seguida de un número indefinido de dígitos.

Es necesario tener cuidado con el comportamiento del asterisco, ya que este por defecto trata de encontrar la mayor cantidad posible de caracteres que correspondan con el patrón que se busca.

2.10. El signo de suma "+"

Se utiliza para encontrar una cadena que se encuentre repetida 1 o más veces.

A diferencia del asterisco, la expresión "$[a-zA-Z]\backslash d+$" encontrará "H1" pero no encontrará "H". También es posible utilizar este metacaracter en conjunto con el signo de pregunta para limitar hasta donde se efectúa la repetición.

3. Abreviaciones y Rangos

3.1. Abreviaciones

$\backslash t$ **:** Representa un tabulador.

$\backslash r$ **:** Representa el "retorno de carro" o "regreso al inicio" o sea el lugar en que la línea vuelve a iniciar.

$\backslash n$ **:** Representa la "nueva línea" el carácter por medio del cual una línea da inicio.

$\backslash a$ **:** Representa una "campana" o "beep" que se produce al imprimir este carácter.

$\backslash e$ **:** Representa la tecla "Esc" o "Escape"

$\backslash f$ **:** Representa un salto de página

$\backslash v$ **:** Representa un tabulador vertical

$\backslash x$ **:** Se utiliza para representar caracteres ASCII o ANSI si conoce su código.

$\backslash u$ **:** Se utiliza para representar caracteres Unicode si se conoce su código.

$\backslash d$ **:** Representa un dígito del 0 al 9.

$\backslash w$ **:** Representa cualquier carácter alfanumérico.

$\backslash s$ **:** Representa un espacio en blanco.

$\backslash D$ **:** Representa cualquier carácter que no sea un dígito del 0 al 9.

$\backslash W$ **:** Representa cualquier carácter no alfanumérico.

$\backslash S$ **:** Representa cualquier carácter que no sea un espacio en blanco.

$\backslash A$ **:** Representa el inicio de la cadena. No un carácter sino una posición.

$\backslash Z$ **:** Representa el final de la cadena. No un carácter sino una posición.

$\backslash b$ **:** Marca el inicio y el final de una palabra.

$\backslash B$ **:** Marca la posición entre dos caracteres alfanuméricos o dos no-alfanuméricos.

3.2. Conjuntos (grupos)

[**a-z**] Letras minúsculas

[**A-Z**] Letras mayúsculas

[**0-9**] Números

[**',?!¡;:\.?**] Caracteres de puntuación

- La barra invertida hace que no se consideren como comando ni en punto ni el interrogante

[**A-Za-z**] Letras del alfabeto

[**A-Za-z0-9**] Todos los caracteres alfanuméricos habituales (sin los de puntuación)

[∧ **a-z**] El símbolo ∧ es el de negación. Esto es decir *todo menos* las letras minúsculas.

[∧ **0-9**] Todo menos los números.

etc

4. Herramientas para trabajar con ER

4.1. GREP

Definición

grep es una utilidad de la línea de comandos escrita originalmente para ser usada con el sistema operativo Unix. Usualmente, *grep* toma una expresión regular de la línea de comandos, lee la entrada estándar o una lista de archivos, e imprime las líneas que contengan coincidencias para la expresión regular.

Su nombre deriva de un comando en el editor de texto *ed* que tiene la siguiente forma:

```
g/re/p
global/regular expression/print
```

nota: *se recomienda al lector que posea un sistema operativo, basado en UNIX, obtener una documentación más robusta acerca de "grep", mediante el "man", que su sistema operativo posee (man grep)*

Ejemplos

Acá tenemos algunos ejemplos que intentarán familiarizar al lector, con el *útil*, o mas bien, indispensable comando de *UNIX* como lo es *grep*.

1. *grep 'hello' archivo*
   Desplegar todas las líneas del archivo que poseen la palabra 'hello'

2. *grep -v 'hello' archivo* Desplegar todas las líneas del archivo que NO poseen la palabra 'hello'

3. *grep -i 'hello.*world' menu.h main.c*
   Éste comando listará todas las lineas en los archivos *menu.h main.c* que contienen el string *hello* seguido por el string *world*; ésto se debe porque $.*$ encuentra 0 o mas caracteres entre líneas. (La opción "-i" provoca que grep ignore mayúsculas y minúsculas, ya las vea todas por igual)

4. *grep -l 'main' *.c*
   Imprime o lista, sólo el nombre de los archivos (que son de la forma *nombre.c*)en que se produce el "matching", es decir que poseen la palabra "main", en su interior.

5. *grep -r 'hello' /home/user*
   Busca dentro de directorios, es decir, recursivamente, la palabra "hello" dentro de los distintos archivos de cada directorio.

6. *grep -w 'hello' **
   Buscar una palabra "completa", no una parte dentro de una cadena más grande

7. *grep 'hello\ >' **
   Buscar las cadenas de caracteres que terminan con 'hello'.

8. *grep -C 2 'hello' **
   Buscamos la palabra 'hello', pero al momento de desplegar dicha información por salida estándar, mostramos 2 lineas de contexto, alrededor de la línea que afirma el *match*

9. *grep -L 'hello'*
   Imprime o lista, los nombres de todos los archivos que no contienen una línea con la palabra 'hello'

10. *grep -w -e '\(.\)\(.\).\2\1' file*
    Buscamos un palíndromo de 4 caracteres (ej. radar, civic)

Una característica fundamental de *grep* y *UNIX*, es la capacidad de poder utilizar *pipes* mediante comandos, lo cual nos permite "mezclar" distintas operaciones.

Un ejemplo simple podría ser, desplegar todos archivos PDF en un directorio determinado:

```
[cmaureir@nexus~]$ ls | grep -i "\.pdf$"
```

Extrapolando ésta funcionalidad, nos daremos cuenta que la cantidad de aplicaciones *útiles* para *grep* en *UNIX* son casi infinitas.

4.2. Otros sabores

Además las Expresiones Regulares, puede ser utilizadas con otros lenguajes u aplicaciones en UNIX, como lo son:

- AWK
- SED
- Bash (que utiliza grep, sed, awk, etc...)
- etc

Ejemplos breves:

- *AWK:*
  Busqueda:

  ```
  awk '/expresion/{ print }' archivo
  ```

- *SED:*
  Busqueda:

  ```
  sed [opciones] "patrón/acción" archivo
  ```

  Sustituciones:

  ```
  sed [opciones] "s/patrón1/patrón2/X" archivo
  ```

  (donde X puede ser g, reemplazar en todo; n, reemplazar n veces)

5. Bibliografía

- Una web interesante para probar expresiones regulares
  `http://www.metriplica.com/4_4_herramientas.asp`
- Expresiones Regulares útiles
  `http://www.mis-algoritmos.com/ejemplos/regex.php`
- Ejemplos Varios
  `http://www.zvon.org/other/PerlTutorial/Output_spa/contents.html`
- Tutorial Expresiones Regulares
  `http://www.bulma.net/impresion.phtml?nIdNoticia=770`
- UNIX scripting `http://plutarco.disca.upv.es/~jcperez/AdminUnix/tema4/tema4.html`
- grep en Wikipedia
  `http://es.wikipedia.org/wiki/Grep`
- ER en Wikipedia
  `http://es.wikipedia.org/wiki/Expresion_regular`
- Regular Expression Cheat Sheet (muy util)
  `http://www.addedbytes.com/cheat-sheets/download/regular-expressions-cheat-sheet-v1.pdf`