Preparación para el examen LPI 101

Tema 103.7

Buscando cadenas de texto usando expresiones regulares

Créditos y licencia de uso

Coordinación:
Manuel Guillán (xLekOx) lpi@xlekox.org
Oscar Casal (ocs) oscar@glug.es

Traducción:
Juan Maria Gil (Smooth) yo@juanmaria.com

Maquetación:
Manuel Guillán (xLekOx) lpi@xlekox.org
Kiefer Von Jammo (Kiefer) kiefer@khrooon.net

Versión 1.0 (10-08-2004 16:00)

ÍNDICE

Índice de contenido

Tema 103.7

Introducción

En éste capítulo se verá como usar expresiones regulares para usar y filtrar ficheros y cadenas de texto. Se crearán construcciones simples con varios elementos de notación así como herramientas para las expresiones regulares que permitirán realizar búsquedas en el sistema de ficheros o en contenidos de ficheros.

Los comandos que se verán en este tema son:

- grep
- regexp (Expresiones regulares)
- sed

En este capítulo no se harán ejercicios, ya que hay suficientes ejemplos para trabajar con el tema y crear o modificar los existentes para hacer pruebas.

Este tema tiene un peso (importancia) de 3 de cara al examen final de la certificación LPI 101. El total de la suma de pesos de todos los temas es de 106.

Utilizando las Expresiones Regulares

Anteriormente se describió el "filename globbing" con comodines, lo que nos permite listar o encontrar ficheros con elementos comunes (p.e., nombres de fichero o extensiones) con una sola expresión. Los "file globs" utilizan caracteres especiales como *, que tienen un significado especial en el contexto de la línea de comandos. La shell bash interpreta varios comodines, los suficientes como para manejar el problema relativamente simple del "filename globbing". Otros problemas no son tan simples, y extender el concepto de "glob" para cualquier forma genérica de texto (ficheros, streams de texto, variables de cadenas de programas, etc.) puede ofrecer un amplio abanico de posibilidades. Esto se consigue con la utilización de las Expresiones Regulares.

Dos utilidades importantes para los exámenes LPIC Nivel 1 que utilizan estas expresiones regulares son *grep* y *sed*. Estas utilidades son prácticas para las búsquedas de texto. Hay muchas otras herramientas que hacen uso de las expresiones regulares incluyendo awk, los lenguajes Perl y Python y otras utilidades pero, de momento, no es necesario preocuparse de ellas para los exámenes LPIC de Nivel 1.

Utilizando grep

Hace mucho tiempo, cuando la idea de las expresiones regulares comenzaba a calar, el editor de líneas ed incluía un comando para visualizar las líneas del fichero en edición que coincidiesen con una expresión regular determinada. El comando era:

g/expresión regular/p

Esto es, "de forma global, imprime la línea actual cuando se encuentre una coincidencia con la expresión regular", o de forma más simple y en inglés, "global Expresión Regular print." Esta función era tan práctica que pronto se convirtió en una utilidad independiente a la que se le llamó, apropiadamente, *grep*. Más adelante se expandió la gramática de las expresiones regulares de grep en un nuevo comando llamado *egrep* (por "extended grep"). Se pueden encontrar ambos comandos en cualquier sistema GNU/Linux actual, existiendo ligeras diferencias en la forma de interpretar las expresiones regulares. Para la preparación del examen 101 se usará *grep* que también puede utilizar las expresiones regulares "extendidas" si se incluye la opción -E.

Sintaxis

grep [opciones] regexp [ficheros]

Descripción

Busca en ficheros o en la entrada estándar líneas que contengan alguna coincidencia con la expresión regular regexp. Por defecto, solo se listarán las líneas coincidentes. Cuando se especifiquen varios ficheros en la búsqueda, grep incluirá el nombre del fichero como prefijo en las líneas listadas.
En la tabla 7-1 se pueden observar algunas opciones del comando.

Tabla 7-1 Opciones utilizadas frecuentemente

Opción	***Uso***
-c	Muestra solo un contador de las líneas coincidentes pero no las líneas en si mismas.
-h	Muestra las líneas coincidentes pero no los nombres de ficheros en búsquedas de múltiples ficheros.
-i	Ignora las mayúsculas y minúsculas de tal forma que abc coincidiría tanto con abc como con ABC.
-n	Muestra las líneas coincidentes precedidas por su número de línea. Cuando se utiliza con varios ficheros se incluye tanto el nombre del fichero como el número de línea.
-v	Imprime todas las líneas que no coinciden con *regexp*. Esta es una opción útil e importante, a veces se quiere utilizar expresiones regulares no solo para seleccionar información, sino para eliminar información. La utilización de -v invierte la salida de ésta forma.

Ejemplos:
Como las expresiones regulares pueden contener tanto metacaracteres como literales, podemos utilizar grep con una expresión regular totalmente literal.
Por ejemplo, para encontrar todas las líneas en fichero1 que contengan tanto “Linux” como “linux” se podría utilizar *grep* de la siguiente manera:

```
$ grep -i linux fichero1
```

En este ejemplo, la expresión regular es, simplemente, "linux.". La L mayúscula en "Linux" es coincidente gracias a la opción -i (ignorar mayúsculas/minúsculas). Ésto está bien para expresiones literales comunes. No obstante, en situaciones en las cuales regexp incluya metacaracteres que también sean caracteres especiales de la shell (como $ o *), regexp debe estar entrecomillada para evitar la expansión de la shell y poder pasar estos metacaracteres a grep.

Como un ejemplo simplista de ésto último, suponiendo que se tiene en el directorio local unos ficheros llamados abc, abc1, y abc2. Cuando se usa el comando *echo* de bash con la expresión comodín abc* se listarán todos los ficheros que comiencen por abc, como se muestra a continuación:

```
$ echo abc*
```

Ahora suponiendo que estos ficheros contienen líneas con las cadenas abc, abcc, abccc y similares y que se quiere emplear grep para encontrarlas. Se puede utilizar la expresión comodín abc* para que se expanda para todos los ficheros abc como se mostró con echo más arriba, y también se podría utilizar una expresión regular idéntica abc* para encontrar todas las líneas que contengan abc, abcc, abccc, etc. Si no se utiliza las comillas para evitar la expansión de la shell, el comando sería:

```
$ grep abc* abc*
```

Después de la expansión de la shell quedaría:

```
$ grep abc abc1 abc2 abc abc1 abc2  # ¡¡incorrecto!
```

¡Esto no es lo que se pretendía! grep buscará solo la expresión literal *abc,* porque es lo que

encuentra como primer argumento. Para evitar ésto, se encierra la expresión regular con comillas simples o dobles para protegerla de la expansión de la shell:

```
$ grep 'abc*' abc*
```

o:

```
$ grep "abc*" abc*
```

Después de la expansión, ambos ejemplos obtienen los mismos resultados:

```
$ grep abc* abc abc1 abc2
```

Ésto era lo que se pretendía. La expresión regular abc* se buscará en los tres ficheros abc, abc1, y abc2. Es una buena práctica habituarse a entrecomillar las expresiones regulares en la línea de comandos para evitar problemas similares, éstos no son, ni mucho menos, obvios ya que la expansión de la shell es invisible a no ser que se utilice el comando *echo*.

En el examen: Es usual la aparición de preguntas sobre la utilización de grep y sus opciones. Se debería estar familiarizado con lo que hace cada opción así como con el concepto de utilizar la salida de otros comandos como entrada de grep mediante pipes (|) para buscar coincidencias.

Utilizando sed

Anteriormente se presentó *sed*, el editor de streams, se comentó como *sed* utiliza direcciones para localizar el texto sobre el que operar. Entre los mecanismos de direccionamiento mencionados estaba la utilización de expresiones regulares delimitadas entre barras inclinadas.
La sintaxis que se vió de *sed* era:

```
sed [opciones] 'comando1' [ficheros]
sed [opciones] -e 'comando1' [-e 'comando2'] [ficheros]
sed [opciones] -f script [ficheros]
```

Descripción

Se observa que comando1 está escrito entre comillas simples, esto se ha hecho por las mismas razones que se hizo anteriormente con grep, el texto en comando1 debe ser protegido de la posible expansión y evaluación de la shell.
La parte de dirección de un comando sed puede contener expresiones regulares encerradas entre barras inclinadas. Por ejemplo, para mostrar el contenido de fichero1 exceptuando las líneas en blanco se podría utilizar la opción eliminar de sed (d) de la siguiente forma:

```
$ sed '/^$/ d' fichero1
```

En este caso, la expresión regular ^$ (^ indica principio de línea, $ indica final de línea; así por tanto, ^$ busca una línea en la que el principio y el final coinciden, es decir, una línea en blanco) coincide con las líneas en blanco y la opción d elimina esas líneas coincidentes de la salida de *sed*.

Entrecomillado

Como se mostró en los ejemplos de *grep* y *sed* es necesario entrecomillar los metacarácteres de las expresiones regulares si se quiere preservar sus funciones especiales. Si no se hace ésto, se pueden producir resultados inesperados cuando la shell interprete los metacaracteres como caracteres de file globbing. Hay tres maneras de entrecomillado que pueden utilizarse para preservar los caracteres especiales:

- \ (una barra invertida sin comillas)
  Si se pone una barra invertida antes de un carácter especial, éste no será interpretado por la shell, sino que pasará sin alteración hacia el comando que se está introduciendo.
  Por ejemplo, el metacaracter * podría utilizarse en una expresión regular de esta forma:

```
$ grep abc\* abc abc1 abc2
```

  Aquí se buscará en los ficheros *abc*, *abc1*, y *abc2* la expresión regular *abc**.

- *Comillas simples*
  Encerrar los metacarácteres con comillas simples también los protege de la interpretación de la shell. Se asume el valor literal de todos los caracteres entre comillas simples.

- *Comillas dobles*
  Encerrar los metacarácteres con comillas dobles tiene el mismo efecto que con comillas simples con la excepción de los caracteres $, ' (comilla simple), y \ (barra invertida). Tanto $ como ' mantienen sus significados especiales dentro de las comillas dobles. La barra invertida mantiene su significado especial cuando va seguida de $, ', otra barra invertida, o un carácter de nueva línea.

En general, las comillas simples son más seguras a la hora de preservar las expresiones regulares.

En el examen: se ha de prestar atención especial a los métodos de entrecomillado utilizados para preservar los carácteres especiales porque las distintas formas de entrecomillar no tienen por que producir los mismos resultados.

Expresiones Regulares

GNU/Linux ofrece muchas herramientas de procesamiento de texto para los administradores de sistemas. Muchas, como *sed* y los lenguajes *awk* y *Perl*, son capaces de editar automáticamente varios ficheros proporcionando una amplia gama de posibilidades para el procesamiento de textos. Para aprovechar estas prestaciones se necesita ser capaz de definir y delimitar segmentos de texto específicos dentro de ficheros, desde streams de texto y variables de cadena. Una vez identificado el texto que se busca se puede utilizar una de esas herramientas o lenguajes para hacer lo que se necesite con el texto seleccionado.

En conjunto, a éste lenguaje y a las secuencias en sí mismas, se les llama expresiones regulares (a menudo se abrevia como regexp o regex). Aunque las expresiones regulares son conceptualmente similares a los file globs, existen muchos más carácteres especiales para las expresiones regulares extendiendo la utilidad y las prestaciones de las herramientas que las interpretan.

Hay libros completos sobre las expresiones regulares (como el excelente y muy legible

Mastering Regular Expresions de Jeffrey E. F. Friedl (ISBN: 1-56592-257-3), publicado por O'Reilly & Associates). El examen 101 requiere del uso de expresiones regulares y de las herramientas relacionadas específicamente para realizar búsquedas desde fuentes de texto. Esta sección cubre solo lo básico de las expresiones regulares, sería muy recomendable profundizar en este tema cuando se tenga ocasión.

Sintaxis de las Expresiones Regulares

Sería razonable asumir que hay alguna especificación que define como se construyen las expresiones regulares. Desafortunadamente no hay ninguna. Las expresiones regulares han sido tradicionalmente incorporadas como una prestación en diferentes herramientas con diferentes grados de consistencia e integridad. El resultado ha sido un caso similar al del carro delante del caballo, en el que las utilidades y lenguajes han ido definiendo su propio tipo de sintaxis para las expresiones regulares, cada uno con sus propias extensiones e idiosincrasias. La definición formal de la sintaxis de las expresiones regulares vino después, así como los esfuerzos para hacerla más consistente. Las expresiones regulares están formadas por cadenas estructuradas de texto o secuencias. Estas secuencias están compuestas por dos tipos de caracteres:

Metacarácteres

Al igual que los caracteres de file globbing, los metacarácteres de las expresiones regulares toman un significado especial en el contexto de la herramienta en la que están siendo utilizados. Hay unos pocos metacaracteres que generalmente se consideran como parte del "conjunto extendido" de metacarácteres, específicamente aquellos introducidos en *egrep* después de la creación de *grep*. Ahora la mayoría de esos pueden ser manejados por *grep* utilizando la opción -E.

Ejemplos de metacaracteres serían el símbolo ^, que significa "el principio de una línea" y el signo $, que significa "el final de una línea". En las tablas 7-2, 7-3 y 7-4 se puede ver una lista completa de metacaracteres.

Literales

Todo lo que no es un metacarácter es, simplemente, texto plano o texto literal.
A menudo es de ayuda considerar las expresiones regulares como un lenguaje en sí mismo donde el texto literal actuaría como palabras y frases. La "gramática" del lenguaje estaría definida mediante el uso de los metacarácteres. Los dos se combinarían de acuerdo a unas reglas específicas (las cuales, como se mencionó antes, podrían diferir ligeramente entre las diferentes herramientas) para comunicar ideas y hacer algún trabajo real. Cuando se construyen expresiones regulares se utilizan metacarácteres y literales para especificar tres ideas básicas sobre el texto de entrada:

- Referencia de posición

Se utiliza una referencia de posición para especificar la posición de uno o más conjuntos de caracteres en relación a la línea de texto completa (como, por ejemplo, el principio de una línea).

- Conjuntos de caracteres

Un conjunto de caracteres busca coincidencias en el texto. Podría ser una serie de literales, metacaracteres que coincidan con uno o varios caracteres, o combinaciones de ambos.

- Modificadores cuantitativos

Los modificadores cuantitativos siguen a un conjunto de caracteres e indican el número de veces que el conjunto ha de ser repetido. Estos caracteres "dan elasticidad" a una expresión regular

permitiendo que las coincidencias tengan longitud variable.
La próxima sección lista los metacaracteres más comunes. Los ejemplos dados con los metacaracteres son muy simples y solo pretenden demostrar el uso de ese metacaracter en cuestión. Más adelante se verán expresiones regulares más complejas.

Ejemplos de Expresiones Regulares

Una vez dejados los detalles desagradables, se ponen ejemplos de uso prácticos de expresiones regulares que podrían resultar útiles.

Anclajes

Las referencias de posición se utilizan para describir información sobre posiciones. La tabla 7-2 lista los caracteres de anclaje.

Tabla 7-2. Anclajes de Posición de Expresiones Regulares

Expresión Regular	***Descripción***
^	Coincide con el principio de línea. Esta interpretación sólo tiene sentido cuando el carácter ^ está en el lado izquierdo de la expresión regular.
$	Coincide con el fin de línea. Esta interpretación sólo tiene sentido cuando el carácter $ está en el lado derecho de la expresión regular.

Ejemplo 1
Muestra todas las líneas de fichero1 en las cuales aparezca la cadena "Linux" en el principio de la línea, o sea, que comiencen por dicha cadena:

```
$ grep '^Linux' fichero1
```

Ejemplo 2
Muestra las líneas en fichero1 donde el último carácter es una "x":

```
$ grep 'x$' fichero1
```

Muestra el número de líneas vacías en fichero1 encontrando líneas con nada entre el principio y el final:

```
$ grep -c '^$' fichero1
```

Muestra todas las líneas de fichero1 que contengan sólo la palabra "nulo":

```
$ grep '^nulo$' fichero1
```

Grupos y rangos

Pueden situarse caracteres en grupos y rangos para hacer expresiones regulares más eficientes, como se muestra en la tabla 7-3.

Tabla 7-3. Conjuntos de caracteres de las expresiones regulares

Expresión Regular	***Descripción***
[abc] [a-z]	Grupos y rangos de caracteres independientes. En la primera forma se hace coincidir cualquier carácter independiente de entre los caracteres `a`, `b`, o `c`. En la segunda forma, se hace coincidir cualquier carácter independiente de entre el rango acotado por los caracteres `a` y `z`. Los corchetes son sólo para agrupar y no hacen coincidencia por si mismos.
[^abc] [^a-z]	Coincidencia inversa. Se hace coincidir cualquier carácter independiente que no esté entre `a`, `b`, o `c` o en el rango `a-z`. Se ha de tener cuidado con confundir esta inversión con el carácter de anclaje `^` que se describió anteriormente.
\<palabra\>	Coincidencia de palabras. Las palabras son esencialmente conjuntos de caracteres rodeados de espacios en blanco o situados junto al inicio o fin de la línea o un signo de puntuación. Las barras invertidas son imprescindibles para que se interpreten de esta forma < y >.
. (un punto)	Coincide con cualquier carácter (sólo uno), excepto el de fin de línea.
\	Como se mencionó en la sección de entrecomillado, desconecta (escape) el significado especial de carácter que le sigue convirtiendo metacaracteres en literales.

Ejemplo 1
Muestra todas las líneas de *fichero1* que contengan "Linux", "linux", "TurboLinux" y similares:

```
$ grep '[Ll]inux' fichero1
```

Ejemplo 2
Muestra todas las líneas de *fichero1* que contengan tres dígitos numéricos consecutivos:

```
$ grep '[0-9][0-9][0-9]' fichero1
```

Ejemplo 3
Muestra todas las líneas de *fichero1* que comiencen por cualquier carácter que no sea un dígito numérico:

```
$ grep '^[^0-9]' fichero1
```

Ejemplo 4
Muestra todas las líneas de *fichero1* que contengan la palabra completa "Linux" o "linux" pero no "LinuxOS" o "TurboLinux":

```
$ grep '\<[Ll]inux\>' fichero1
```

Ejemplo 5
Muestra todas las líneas de fichero1 con cinco o más caracteres en una línea (excluyendo el carácter de nueva línea):

```
$ grep '.....' fichero1
```

Ejemplo 6

Muestra todas las líneas no vacías de fichero1 (aquellas que tienen al menos un carácter):

```
$ grep '.' fichero1
```

Ejemplo 7
Muestra todas las líneas de *fichero1* que contienen un punto (como . es un metacarácter, se ha de usar el codigo de escape):

```
$ grep '\.' fichero1
```

Tabla 7-4. Modificadores de Expresiones Regulares

Expresión Regular	***Descripción***
*	Coincide con un número indeterminado (cero o más) del carácter (o expresión regular) que lo precede.
?	Coincide con cero o una instancia de la expresión regular que lo precede. Éste modificador es una función "extendida" y solo se dispone en *grep* cuando se utiliza la opción -E.
+	Coincide con una o más instancias de la expresión regular que lo precede. Este modificador es una función "extendida" y solo se dispone en *grep* cuando se utiliza la opción -E.
\{n,m\}	Coincide con un rango de ocurrencias del carácter o la expresión regular que precede ésta construcción. \{n\} coincide con n ocurrencias, \{n,\} coincide con al menos n ocurrencias, y \{n,m\} coincide con cualquier número de ocurrencias entre n y m inclusive. Las barras invertidas son imprescindibles y permiten la interpretación de { y }.
|	Alternancia. Coincide tanto con la expresión regular especificada antes o después de la barra vertical. Este modificador es una función "extendida" y solo se dispone en *grep* cuando se utiliza la opción -E.

Ejemplo 1
Muestra todas las líneas de *fichero1* que contengan "ab", "abc", "abcc", "abccc", y similares:

```
$ grep 'abc*' fichero1
```

Ejemplo 2
Muestra todas las líneas de *fichero1* que contengan "abc", "abcc", "abccc", y similares, pero no "ab":

```
$ grep 'abcc*' fichero1
```

Ejemplo 3
Muestra todas las líneas de fichero1 que contengan dos o más dígitos numéricos consecutivos:

```
$ grep '[0-9][0-9][0-9]*' fichero1
```

Ejemplo 4
Muestra líneas de fichero1 que contengan "fichero" (porque ? puede hacer coincidir cero ocurrencias), “fichero1”, o “fichero2”:

```
$ grep -E 'fichero[12]?' fichero1
```

Ejemplo 5
Muestra todas las líneas de fichero1 que contengan al menos un dígito numérico:

```
$ grep -E '[0-9]+' fichero1
```

Ejemplo 6
Muestra todas las líneas de fichero1 que contengan "111," "1111," o "11111" en una línea y nada más:

```
$ grep '^1\{3,5\}$' fichero1
```

Ejemplo 7
Muestra todas las líneas de fichero1 que contengan cualquier número de tres, cuatro o cinco dígitos:

```
$ grep '\<[0-9]\{3,5\}\>' fichero1
```

Ejemplo 8
Muestra todas las líneas de fichero1 que contengan "Happy", "happy", "Sad", "sad", "Angry", o "angry":

```
$ grep -E '[Hh]appy|[Ss]ad|[Aa]ngry' fichero1
```

Secuencias básicas de las Expresiones Regulares

Ejemplo 1
Coincide con cualquier letra:

```
[A-Za-z]
```

Ejemplo 2
Coincide con cualquier signo (no letra o número):

```
[^0-9A-Za-z]
```

Ejemplo 3
Coincide con una letra mayúscula seguida de cero o más letras minúsculas:

```
[A-Z][a-z]*
```

Ejemplo 4
Coincide con un número de la Seguridad Social de USA (123-45-6789) especificando grupos de

tres, dos y cuatro dígitos separados por guiones:

```
[0-9]\{3\}-[0-9]\{2\}-[0-9]\{4\}
```

Ejemplo 5
Coincide con un importe en dólares utilizando un signo de dólar escapado, cero o más espacios o dígitos numéricos, un punto escapado y dos dígitos más:

```
\$[ 0-9]*\.[0-9]\{2\}
```

Ejemplo 6
Coincide con el mes de mayo y su abreviatura, “may”. La interrogación marca cero o una instancia del carácter *o* :

```
mayo?
```

Utilizando Expresiones Regulares como direcciones en sed

Estos ejemplos son comandos que se proporcionarían a sed. Por ejemplo, los comandos podrían situarse en lugar de comando1 en esta norma de uso:

$ sed [opciones] 'comando1' [ficheros]

Comandos que podrían aparecer en un script independiente de sed.

Ejemplo 1
Elimina líneas en blanco:

```
/^$/d
```

Ejemplo 2
Elimina cualquier línea que no contenga #noborrar:

```
/#noborrar/!d
```

Ejemplo 3
Elimina líneas que contengan sólo espacios en blanco (espacios o tabulaciones). En éste ejemplo tab significa el carácter tab y va precedido por un espacio:

```
/^[ tab]*$/d
```

Ejemplo 4
Elimina líneas que comiencen por puntos o signos #:

```
/^[.#]/d
```

Ejemplo 5
Sustituye cualquier número de espacios por un sólo espacio donde quiera que aparezca en la línea:

```
s/  */ /g
```

Ejemplo 6
Sustituye abc por def de la línea 11 a la 20, donde quiera que aparezca en la línea:

```
11,20s/abc/@@@/g
```

Ejemplo 7
Cambia los caracteres a, b, y c por el carácter @ desde la línea 11 a la 20, donde quiera que aparezca en la línea:

```
11,20y/abc/@@@/
```

En el examen
Asegurarse de que se tienen claras la diferencia entre file globbing y la utilización de expresiones regulares.

Bibliografia y enlaces recomendados

LPIC 1 Certification Bible (Bible) by Angie Nash, Jason Nash
John Wiley & Sons; Bk&CD-Rom edition (July 1, 2001) ISBN: 0764547720

LPI Linux Certification in a Nutshell by Jeffrey Dean
O'Reilly & Associates; 1st ed edition (May 15, 2001) ISBN: 1565927486

CramSession's LPI General Linux Part 1 : Certification Study Guide
CramSession.com; ISBN: B000079Y0V; (August 17, 2000)

Referencias Unix Reviews
http://www.unixreview.com/documents/s=7459/uni1038932969999/

Página LPI: www.lpi.org

Apuntes IBM: http://www-106.ibm.com/developerworks/edu/l-dw-linux-lpir21-i.html

Manuales GPL: http://www.nongnu.org/lpi-manuals/