Sed y Awk.

Teoría de Autómatas y lenguajes formales
Federico Simmross Wattenberg (fedesim@infor.uva.es)
Universidad de Valladolid

Hemos visto que, gracias a la herramienta `grep`, las expresiones regulares nos permiten hacer búsquedas de todo tipo en ficheros de texto. Ahora vamos a dar un paso adelante para ver qué se puede hacer con el texto una vez determinada la coincidencia con una expresión regular. `Sed` y `awk` ofrecen un conjunto de comandos que se pueden aplicar sobre las líneas de texto encontradas para modificarlas a nuestro antojo. Awk es el más potente, pero también el más complejo de los dos.

1. Sed

Sed es un editor de textos en línea de comando. Toma como entrada uno o más ficheros, los procesa según los parámetros especificados, y finalmente escribe el resultado en la salida estándar.

1.1. Funcionamiento de `sed`

Sed procesa su entrada línea a línea, lo cual quiere decir que no es apto para alterar ficheros binarios, ni ficheros con líneas extremadamente largas. Su funcionamiento puede describirse de la siguiente manera:

1- Leer una línea de entrada y copiarla en un buffer interno (el *pattern space*)
2- Si hay algún comando que aplicar en esta línea
    aplicar secuencialmente dichos comandos sobre el *pattern space*
3- Copiar el *pattern space* a la salida estándar
4- Borrar el *pattern space*

Existen comandos y opciones que permiten alterar este bucle de varias maneras. Por ejemplo, se puede impedir que `sed` copie sistemáticamente el *pattern space* a la salida estándar con la opción `-n`. Así, `sed` sólo producirá salida cuando un comando se lo ordene explícitamente.

1.2. Sintaxis de `sed`

```
sed [-n] -e comando [-e comando ...] [fichero_entrada ...]
sed [-n] -f script_sed [-f script_sed ...] [fichero_entrada ...]
```

- Se pueden mezclar las opciones `-e` y `-f` en un solo comando `sed`.
- Si sólo se especifica un comando y ningún script (una sola opción `-e`), se puede suprimir el `-e`.
- Un script de `sed` es un fichero de texto que contiene comandos de `sed`, a razón de uno por línea.

1.3. Formato de los comandos de `sed`

```
[dirección1[,dirección2]]comando[argumentos]
```

Las direcciones permiten seleccionar en qué líneas de la entrada se aplicará el comando. Una dirección puede tomar tres formas:

- **Un número**: especifica un número de línea de la entrada.
- **Un carácter '$'**: especifica la última línea de la entrada
- **Una expresión regular entre caracteres '/'**: especifica aquellas líneas que coinciden con la expresión regular.

Puede haber una, dos o ninguna dirección:

- Ninguna: el comando se aplica a todas las líneas.
- Una: el comando se aplica sólo en la línea especificada.
- Dos: el comando se aplica en todas las líneas que estén entre las dos especificadas (ambas inclusive).

El comando indica qué hacer con cada línea que entra en la(s) dirección(es) especificadas. Generalmente consta de una sola letra y puede aceptar o no algún parámetro. Se encuentran todos detallados en la página del `man` correspondiente.

1.4. Ejemplos

- Quitar las líneas vacías de un fichero:
  ```
  sed '/^$/d' fichero
  ```

- Pasar a mayúscula todas las letras 'a':
  ```
  sed 's/a/A/g' fichero
  ```

- Pasar a mayúscula las letras 'a' de las líneas que empiecen por 'd':
  ```
  sed '/^d/s/a/A/g' fichero
  ```

- Mostrar sólo el *login* de aquellos usuarios que estén a partir de la línea 1100 del fichero /etc/passwd:
  ```
  sed -n '1100,$p' /etc/passwd|sed 's/:.*//'
  ```

2. Awk

Awk funciona de manera similar a sed: busca ciertos patrones en la entrada, y la procesa de la manera especificada. Muchas de las limitaciones de sed desaparecen al utilizar awk, pero esta mayor funcionalidad tiene su coste reflejado en una mayor complejidad del lenguaje.

2.1. Funcionamiento de awk

A diferencia de sed, que sólo proporciona un conjunto de comandos para procesar la entrada, awk reconoce un lenguaje completo, de aspecto similar a C. También tiene mayor potencia a la hora de reconocer patrones en la entrada, ya que permite especificar combinaciones de expresiones regulares en vez de una sola. Tampoco existe el límite de tener que procesar la entrada línea a línea; awk permite escoger el carácter que indica el fin de un registro y procesar la entrada de registro en registro (awk utiliza la palabra 'registro' en vez de 'línea'), y automáticamente separa cada registro en campos separados que pueden utilizarse individualmente.

- Por defecto, un registro es una línea del fichero, es decir, que el separador de registros es '\n'.
- Por defecto, un campo es todo aquello que esté separado por espacios en blanco, es decir, una palabra. Dicho en forma de expresión regular, el separador de campos por defecto es `'[ \t]'` (espacio y tabulador).

2.2. Sintaxis básica de awk

Dependiendo de la implementación concreta, awk reconoce varias opciones. Véase la página del man para más detalles.

```
awk <programa> [fichero_entrada]
awk -f <fichero_programa> [fichero_entrada]
```

Un programa de awk es una secuencia de sentencias patrón-acción, con el formato que se describe a continuación. Las acciones se ejecutarán si en el registro actual se cumple el patrón:

```
patrón {acciones}
```

- Las llaves son necesarias para awk, por lo que suele ser necesario encerrar los programas de awk entre comillas, para evitar que el *shell* las interprete como caracteres especiales.
- Si no hay patrón, se ejecutarán las acciones en todos los registros.
- Si no hay acciones, se ejecuta la acción por defecto: copiar el registro en la salida estándar.

Awk soporta patrones más complejos que sed. En general, pueden ser cualquier combinación booleana de expresiones regulares o relacionales, además de las palabras clave `BEGIN` y `END`, que son ciertas, respectivamente, antes del principio y después del final del fichero.

2.3. Ejemplos

- Quitar las líneas vacías de un fichero:
  ```
  awk '!/^$/ {}' fichero
  ```

- Mostrar el login de los usuarios que están conectados:
  ```
  w|awk 'NR>2 {print $1}'
  ```

- Mostrar el login y el nombre completo de los usuarios cuyo login empieza y acaba en 'a':
  ```
  awk 'BEGIN {FS=":"} /^a[^:]*a:/ {print $1,$5}' /etc/passwd
  ```

3. Ejercicios

Utilícese el fichero de texto 'quijote.txt' para realizar estos ejercicios (excepto el primero de awk). Puede descargarse de la página web de la asignatura: http://duero.lab.fi.uva.es/~valecar/talf

3.1. Sed

1- Quitar las líneas que no acaban en 's'.
2- Mostrar las líneas 3, 4 y 5.
3- Insertar el siguiente texto al principio del fichero:
```
El Quijote
----------
```
4- Poner paréntesis alrededor de todas las letras mayúsculas.
5- Pasar todas las vocales a mayúsculas.

3.2. Awk

1- Mostrar el login de los usuarios conectados (véase el comando `w`).
2- Mostrar la primera palabra de cada línea.
3- Mostrar la última palabra de cada línea.
4- Mostrar las líneas impares.
5- Mostrar en orden inverso las palabras de cada línea