

Procesamiento de ficheros CSV

Iván López de Munain Quintana

7 de marzo de 2019

I. INTRODUCCIÓN.

En esta primera entrega se va a poner en práctica los conocimientos adquiridos de los comandos *awk*, *sed* y *egrep* para el procesamiento de ficheros CSV. En suma, se van a mostrar cómo escribir programas *bash* y cómo hacer que los ficheros sean ejecutables, entre otras cosas. Antes de comenzar con los ejercicios de la práctica tenemos que descargar los datos que se encuentran en <https://datosabiertos.jcyl.es/web/jcyl/risp/es/ciencia-tecnologia/general/1284166186527.csv>. Para conseguir esto se han ejecutado los siguientes comandos (cabe recordar que se han escrito en el fichero *guion* que se ha convertido en ejecutable mediante la orden *chmod +x guion*):

```
#Comandos incluidos en guion
#crear el directorio para almacenar los datos
mkdir data

#descarga del conjunto de datos y almacenamiento en CataDatosRaw.csv
curl https://datosabiertos.jcyl.es/web/jcyl/risp/es/ciencia-tecnologia/general/1284166186527.csv > data/CataDatosRaw.csv
```

Posteriormente se nos pide que cambiemos la codificación del juego de caracteres del fichero *CataDatosRaw.csv* de la forma ISO-8859-1 a UTF-8 y almacenemos el resultado en *data/CataDatosJCyL.csv* mediante la siguiente orden:

```
#Comando incluido en guion
iconv -f ISO-8859-1 -t UTF-8 data/CataDatosRaw.csv > data/CataDatosJCyL.csv
```

Una vez realizada toda la preparación de los datos ya estamos preparados para normalizar el fichero CSV *CataDatosJCyL.csv* como se mostrará en el siguiente apartado.

II. NORMALIZACIÓN DEL FICHERO CSV.

Para realizar este apartado se ha creado un programa *sed* para facilitar el procesamiento posterior de los datos. El programa en cuestión se llama *ecsv2csv* y se va a utilizar para:

- Eliminación de la primera línea del fichero que solo contiene la fecha de actualización.
- Eliminación de todos los saltos de línea que no supongan cambios de registros.

Para conseguir el primer objetivo solo sería necesario incluir en el fichero *ecsv2csv* la orden *1d* (1->primera línea, d->borrar).

Por contra, la consecución del segundo objetivo es más costosa debido a que solo se quiere borrar los saltos de línea que se encuentren dentro de un registro pero dejando intactos los saltos de línea que separan los registros. Para hacer esto hay que tener en cuenta que dichos registros están separados por *","*. Este apartado es el que mayor dificultad me ha supuesto porque no se me ocurría la expresión

regular adecuada para conseguirlo. Buscando en internet encontré un ejemplo parecido en <https://stackoverflow.com/questions/1251999/how-can-i-replace-a-newline-n-using-sed> que me ayudó para poder resolver este ejercicio. En el fichero *ecsv2csv* incluiríamos lo siguiente:

```
# a sirve para crear labels
# N introduce la siguiente linea al conjunto del patron que estamos buscando
# $! si no es la ultima linea, ba salta al label a
# s/[^\n]/g sustituye los saltos de linea que no estan precedidos por ;
# los sustituye por nada, es decir, los elimina
# g es para que repita esta operacion en todos los casos
:a;N;$!ba;s/[^\n]/g
```

Por último, para ejecutar estos comandos tendríamos que incluir en el fichero *guion* la siguiente orden teniendo en cuenta que queremos almacenar el resultado en *data/CataDatosCSV.csv*:

```
# -f sirve para ejecutar con sed las ordenes del fichero en cuestion
sed -f ecsv2csv data/CataDatosJCyL.csv > data/CataDatosCSV.csv
```

III. OBTENCIÓN DE LA LISTA DE CAMPOS.

Una vez que hemos normalizado el fichero CSV podemos comenzar a realizar distintos ejercicios de procesamiento como por ejemplo obtener los nombres de los campos que componen el conjunto de datos. Para este caso sabemos que está compuesto por 16 campos siendo el último un campo vacío que posteriormente eliminaremos.

Para realizar esto se va a crear un programa *awk* llamado *campos* que contendrá las siguientes líneas de código:

```
#!/user/bin/awk -f

BEGIN{

    FS=",";

}

NR==1 {

    for(ncampo=1; ncampo<=NF; ncampo++){
        print ncampo " : " $ncampo;
    }

}
```

Este pequeño programa se ejecutará desde el script *guion* tomando como entrada de datos *CataDatosCSV.csv* e incluyendo la siguiente orden:

```
# -f sirve para ejecutar el fichero campos awk
# sed sirve para quedarnos solo con los primeros 15 registros y eliminar
# el ultimo que era vacio, el resultado lo almacenamos en CataDatosCSV.campos
awk -f campos data/CataDatosCSV.csv | sed -n '1,15p' > data/CataDatosCSV.campos
```

IV. OBTENCIÓN DEL NÚMERO DE CAMPO A PARTIR DEL NOMBRE.

Para resolver este ejercicio primero había pensado en realizar un programa *awk* pasando como argumento una expresión regular mediante la opción *-v*. Para realizar esto hay que tener en cuenta que el conjunto de datos *CataDatosCSV.campos* tiene solo dos registros separado por *:* siendo el primero el número correspondiente a cada campo, por lo que obtendríamos el siguiente programa (lo he llamado *campos2num2*):

```
#!/user/bin/awk -f

BEGIN{


    FS=":"; #pongo el delimitador : por el formato que habia guardado antes
}

# x es la expresion regular introducida por el usuario
# se miran los campos de $2 que coincidan con x
$2~x{
    #se imprime el numero correspondiente que esta almacenado en $1
    print "El numero del registro que hay coincidencia es: " $1;
}
```

Posteriormente solo tendríamos que añadir la siguiente sentencia en *guion*:

```
# -v sirve para pasar argumentos al programa
awk -f campos2num -v x='ultima.*' data/CataDatosCSV.campos
```

Posteriormente me di cuenta que en este apartado se pedía un programa *bash* y que el fichero podía ser distinto a *CataDatosCSV.campos*. Este programa tiene el nombre de *campos2num* y al tratarse de un mayor número de líneas de código se adjuntará un recorte del mismo:



```
#!/bin/bash

#expresion regular, se esta diciendo que puede ser cualquier letra
#mayuscula o minuscula repetida varias veces pero no puede contener
#ningun slash (/) porque asi lo diferenciamos de que sea una ruta
expresion='^[a-zA-Z]*'

#sino se especifica una expresion regular se devuelve menos uno

if [[ $1 =~ $expresion ]]; then
    echo -1
    exit 1
fi

#si hay 2 entradas-> expresion, ruta
#datos por defecto-> data/CataDatosCSV.campos

if [ $# -eq 2 ]; then
    dataset=$2
else
    dataset=data/CataDatosCSV.campos
fi

#encontrar igualdades con la expresion regular y quedarnos solo con el primer
#campo, -f1 seleccionamos el primer campo y -d ":" decir cual es el delimitador

numRegistro=$(egrep $1 $dataset | cut -f1 -d ".:")

#sino hay coincidencias se devuelve cero
if [[ $numRegistro = "" ]]; then
    numRegistro=0
fi

echo $numRegistro
```

Figura 1: Código de fichero *campos2num*.

En la Figura 1 se puede ver la consecución del ejercicio, además está explicada la expresión regular y el por qué de las distintas sentencias. En caso de que no se aprecie bien el código está bien comentado en el programa *bash campos2num*.

V. GENERACIÓN DE ALGUNOS LISTADOS.

Por último queremos obtener mediante un programa *awk* llamado *listado* el nombre del conjunto de datos (campo número 1), el cual está seguido de ':' y de la fecha de última modificación del dataset (campos número 14). Posteriormente se quiere imprimir en una línea aparte con dos tabuladores el valor de un campo obtenido por el programa *./campo2num*. Para conseguir esto se utilizará la opción *-v*.

```
#!/user/bin/awk -f

BEGIN{

    FS=":";

}

#saltar la primera linea
NR==1{next;}

# ereg es la expresion regular que condiciona el campo x
# x se obtiene mediante ./campos2num
$x ~ $ereg{

    print $1 ": (" $14 ")\n";

    #siendo x el argumento pasado por -v
    print "\t\t" $x;

}
```

Para ejecutar este programa desde el script *guion* añadimos las siguientes sentencias:

```
#el siguiente codigo es para obtener el mismo resultado que la Figura 3
nURL=$(./campos2num 'Enlace al.*')

#el programa convfecha sirve para modificar el formato de las fechas
awk -f listado -v x=$nURL ereg='.datosabiertos.*' data/CataDatosCSV.csv |
sed -f convfecha > data/Figura3
```

Por último solo queda mostrar cómo se ha realizado el programa *sed convfecha* para modificar el formato de las fechas. Cabe resaltar que he conseguido obtener la solución esperada pero no de la forma más óptima. Mediante una sola expresión regular solo he conseguido pasar del formato (20180125) al formato [25-01-2018]:

```
# esta expresion regular nos dice que en aquellos sitios donde haya
# cuatro numeros (primera condicion) seguidos de otros dos (segunda condicion)
# seguido de otros dos mas (tercera condicion) los cambiamos a la forma
# [tercera condicion-segunda condicion-primera condicion] y que reptiamos esto
# para todas las coincidencias

s /\([0-9]\{4\}\)\([0-9]\{2\}\)\([0-9]\{2\}\)/[\3-\2-\1]/g
```

El problema lo he encontrado para conseguir obtener [25-ENE-2018] mediante una única expresión regular compacta. Es por esto que he conseguido alcanzar el objetivo pero no de la mejor manera, la solución que he implementado de *convfecha* es la siguiente:

```
#la expresion regular es igual que la explicada anteriormente
s /\([0-9]\{4\}\)01\([0-9]\{2\}\)/[2-ENE-1]/g
s /\([0-9]\{4\}\)02\([0-9]\{2\}\)/[2-FEB-1]/g
...
s /\([0-9]\{4\}\)12\([0-9]\{2\}\)/[2-DIC-1]/g
```

Utilizando estos programas vamos a crear distintos listados (CataListURLs.list, CataListSectores.list y CataListSectorSalud.list) que almacenaremos en el directorio *data*. Cabe resaltar que para el listado que almacena solo los conjuntos del sector salud he incluido también aquellos sectores mixtos como "medio-ambiente, salud". Añadiremos en el script *guion* lo siguiente:

```
#obtencion del listado de url
url=$(./campos2num 'Enlace al.*')
awk -f listado -v x=$url data/CataDatosCSV.csv | sed -f convfecha >
data/CataListURLs.list

#obtencion del listado de sectores
sector=$(./campos2num 'Sector')
awk -f listado -v x=$sector data/CataDatosCSV.csv | sed -f convfecha >
data/CataListSectores.list

#obtencion del listado de sectores de salud
sector=$(./campos2num 'Sector')
awk -f listado -v x=$sector ereg='salud' data/CataDatosCSV.csv |
sed -f convfecha > data/CataListSectorSalud.list
```

Hasta ahora se ha ido indicando qué comandos se han incluido en el script *guion* pero a continuación se va adjuntar un recorte del contenido de forma global para que se vea mejor:

```
mkdir data #crear el directorio para almacenar los datos

curl https://datosabiertos.jcyl.es/web/jcyl/risp/es/ciencia-tecnologia/general/1284166186527.csv > data/CataDatosRaw.csv
#descargar los datos mediante la url

iconv -f ISO-8859-15 -t UTF-8 data/CataDatosRaw.csv>data/CataDatosJCyl.csv
#cambiar la codificación de los datos

sed -f csv2csv data/CataDatosJCyl.csv > data/CataDatosCSV.csv
#ejecutamos csv2csv que elimina la primera línea y elimina los saltos de línea
#excepto en los cambios de campos

awk -f campos data/CataDatosCSV.csv | sed -n '1,15p' > data/CataDatosCSV.campos
#asi obtenemos en el fichero CataDatosCSV.campos los 15 registros

awk -f campos2num2 -v x='ultima.!' data/CataDatosCSV.campos
#programa awk para realizar el apartado 1.5, pero el programa bash se
#encuentra en el fichero Campos2num

./campos2num 'Titu.!'
./campos2num '.*Crea.!'
#estos comandos realizan el ejercicio 1.5

nURL=$(./campos2num 'Enlace al.(!)')
#en nURL almacenamos el numero de campo, en este caso el que contiene las URL
awk -f listado -v x=$nURL ereg='.*datosabiertos.!' data/CataDatosCSV.csv | sed -f convfecha > data/Figura3
#Estos comandos consiguen la misma salida que la mostrada en la figura 3 que se
#almacenara en data/Figura3
#Se hace esto solo con el proposito de mostrar
#que se ha conseguido el objetivo del ejercicio.

#las siguientes ordenes son para conseguir los distintos listados
sector=$(./campos2num 'Sector')
awk -f listado -v x=$sector data/CataDatosCSV.csv | sed -f convfecha > data/CataListSectores.list

url=$(./campos2num 'Enlace al.(!)')
awk -f listado -v x=$url data/CataDatosCSV.csv | sed -f convfecha > data/CataListURLs.list

sector=$(./campos2num 'Sector')
awk -f listado -v x=$sector ereg='salud' data/CataDatosCSV.csv | sed -f convfecha > data/CataListSectorSalud.list
```

Figura 2: Comandos del script *guion*.