

513372

Linux, Scripts y GMT

Matt Miller
2016

*Departamento de Geofísica
Universidad de Concepción*

Contenidos

1	Introducción	3
1.1	El sistema operativo Linux	3
1.2	Hacer una lista de archivos	3
1.3	Creando directorios y cambiando entre ellos	3
1.4	La magia del "TAB" y wildcards(*)	4
1.5	Copiando y moviendo archivos	5
1.6	Ejecutables y permisos	5
1.7	Variables en Bash	7
1.8	Espacio en el disco	7
1.9	Espacio en uso de archivos y directorios	7
1.10	Archivo de datos	7
2	Comandos básicos	8
2.1	More	8
2.2	Head, Tail	8
2.3	Pipes	8
2.4	Awk	8
2.5	Grep	10
2.6	Sed	11
2.7	Sort	12
2.8	Combinación de comandos, pipes	12
2.9	Preguntas	12
3	Printf, expresiones regulares, tar, bashrc	13
3.1	Printf en awk	13
3.2	Substrings en awk	14
3.3	Expresiones regulares	14
3.4	\$PATH y .bashrc	14
3.5	Tarballs	15
3.6	Preguntas	16
4	Scripts, loops y condicionales	17
4.1	Introducción a Scripts en bash	17
4.2	Fijando una variable	17
4.3	Scripts básicos	17
4.4	Aritmética en bash	19
4.5	Expresiones lógicas	19
4.5.1	Operadores de comparación numéricos	19
4.5.2	Operadores de comparación de texto	20
4.6	Loops	20
4.6.1	for	20
4.6.2	While	21
4.7	Condicionales	21
4.7.1	if	21
4.7.2	Case	23

5	Scripts para manipular datos	24
5.1	Introducción	24
5.2	Elegir un rango de variables de un archivo	24
5.3	Calculando el número de réplicas asociadas con cada día	24
5.4	Calcular el número de días después del 27 de febrero	26
5.5	Presentando datos en un terminal	27
5.6	Conclusión	28
6	Instalación GMT	29
7	Mapas, costa y topografía	30
7.1	Colores en GMT	30
7.2	psbasemap	30
7.3	pscoast	31
7.4	Topografía	31
7.5	Iluminación topográfica	34
8	Scripts y GMT	35
8.1	Script para un mapa en GMT	35
8.2	Contornos en GMT	37
8.3	Personalización del mapa con condicionales	38
8.4	Conclusión	40
8.5	Preguntas	40
9	Simbolos y texto en GMT	41
9.1	Simbolos y texto: psxy y pstext	41
9.2	Agregando símbolo al mapa	41
9.3	Agregando texto	41
9.4	Agregando líneas	42
9.5	Agregando vectores	43
9.6	El script	43
10	Simbolos	47
10.1	Simbolos desde un archivo de texto : volcanes	47
10.2	Simbolos con diferentes tamaños y colores	47
10.3	Mecanismos focales	49
10.4	Indent maps	49
11	Grillas	51

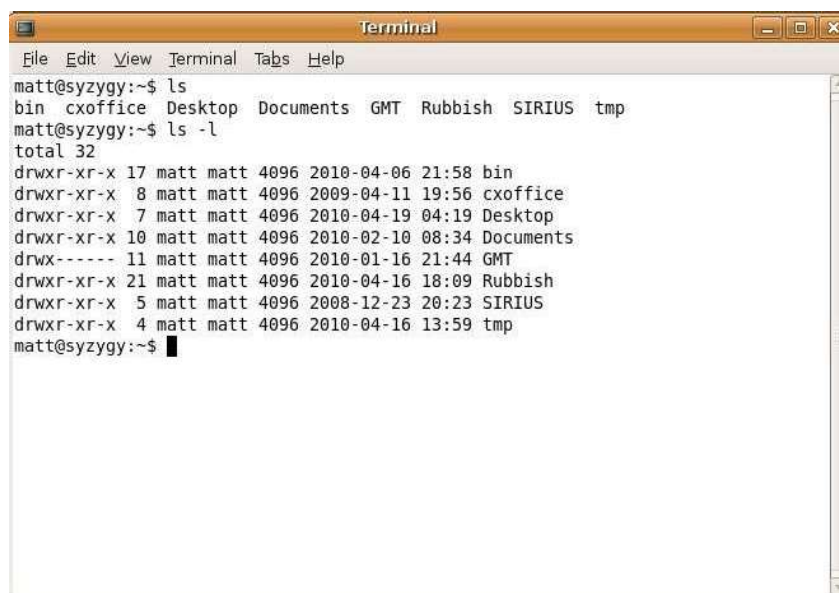
1 Introducción

1.1 El sistema operativo Linux

Todo en Linux es un archivo o un proceso. Un proceso es un ejecutable que se identifica con un PID (process identifier) único. Un archivo es una colección de datos en una variedad de formas (como texto).

Linux tiene un GUI (Graphical User Interface) similar a otros sistemas operativos, no obstante, algunas operaciones no tienen interfaz gráfica, por lo cual se necesita conocer los detalles de bash para acoplar esas operaciones en la manipulación de datos. Para ello, trabajamos en un terminal.

En alguna parte del menú (probablemente aplicaciones) se puede encontrar el terminal. Tome unos minutos para familiarizarse con el menú, y poner el terminal como un ícono de escritorio.



```
matt@syzygy:~$ ls
bin  cxoffice  Desktop  Documents  GMT  Rubbish  SIRIUS  tmp
matt@syzygy:~$ ls -l
total 32
drwxr-xr-x 17 matt matt 4096 2010-04-06 21:58 bin
drwxr-xr-x  8 matt matt 4096 2009-04-11 19:56 cxoffice
drwxr-xr-x  7 matt matt 4096 2010-04-19 04:19 Desktop
drwxr-xr-x 10 matt matt 4096 2010-02-10 08:34 Documents
drwx----- 11 matt matt 4096 2010-01-16 21:44 GMT
drwxr-xr-x 21 matt matt 4096 2010-04-16 18:09 Rubbish
drwxr-xr-x  5 matt matt 4096 2008-12-23 20:23 SIRIUS
drwxr-xr-x  4 matt matt 4096 2010-04-16 13:59 tmp
matt@syzygy:~$
```

Figure 1: Listando archivos

1.2 Hacer una lista de archivos

El comando `ls` presenta los contenidos del directorio en que usted esta trabajando. Simplemente teclee `ls` en el terminal. También prueba el comando `ls -l` (lista formato largo); este tiene más información, por ejemplo, los permisos, el dueño, tamaño y la fecha de la última modificación.

Prueba el comando `ls -a` (list all). Con esto se obtiene información sobre los archivos/directorios escondidos también. ¿Cómo se esconde un archivo?

1.3 Creando directorios y cambiando entre ellos

El comando `cd` le permite cambiar de directorio. Cuando se pone `cd` sólo en el terminal, se va al directorio `$ HOME`. Es posible verificar dónde estas exactamente con el comando `pwd`, por ejemplo:

Ahora vamos a crear un directorio que se llama *bin* (revisen que estan en su directorio `$HOME` primero con `pwd`):

```
mkdir bin
```



Figure 2: Mostrando ruta actual

donde literalmente **mkdir** significa make directory.

Hace una listado **ls** para corroborar que este directorio ahora existe. Podemos ir al directorio **bin** con el comando **cd bin**.

Una lista aquí va a mostrar que por ahora este directorio esta vacío. Use **pwd** para notar donde esta el directorio **bin** en relación a tu espacio de trabajo **\$HOME**.

El comando **cd ..** te lleva a un directorio más arriba en jerarquía(organización de archivos del sistema Linux), en este caso el espacio **\$HOME**. Notar que **cd** también sirve para ir al **\$HOME** desde cualquier lugar en su computadora.

También podemos crear una jerarquía de directorios añadiendo la opción **-p** al comando **mkdir**. Por ejemplo, creemos el árbol Topografia/Chile/Concepcion

```
mkdir -p Topografia/Chile/Concepcion
```

1.4 La magia del "TAB" y wildcards(*)

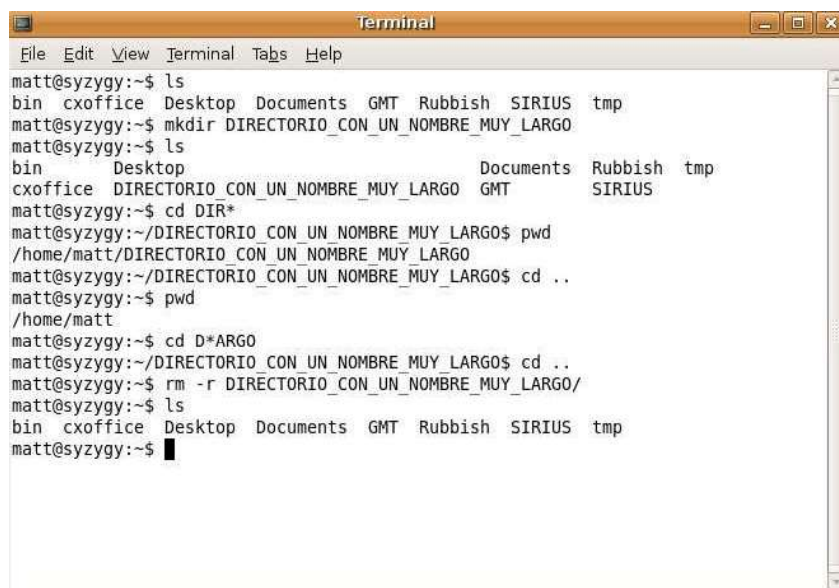
Crean un directorio con un nombre muy largo, por ejemplo:

```
mkdir directorio_con_un_nombre_muy_largo
```

Para cambiar a este directorio no es necesario poner su nombre completo. Si después de poner las primeras letras del directorio, si este es único, tocando la llave **TAB** completará la frase. De otra manera, la estrella (*) puede completar frases únicas también.

Para borrar un archivo hay que usar el comando **rm** , para un directorio **rm -r**, por ejemplo:

```
rm -r directorio_con_un_nombre_muy_largo
```



```
Terminal
File Edit View Terminal Tabs Help
matt@syzygy:~$ ls
bin  cxoffice  Desktop  Documents  GMT  Rubbish  SIRIUS  tmp
matt@syzygy:~$ mkdir DIRECTORIO_CON_UN_NOMBRE_MUY_LARGO
matt@syzygy:~$ ls
bin      Desktop                                Documents  Rubbish  tmp
cxoffice DIRECTORIO_CON_UN_NOMBRE_MUY_LARGO  GMT        SIRIUS
matt@syzygy:~$ cd DIR*
matt@syzygy:~/DIRECTORIO_CON_UN_NOMBRE_MUY_LARGO$ pwd
/home/matt/DIRECTORIO_CON_UN_NOMBRE_MUY_LARGO
matt@syzygy:~/DIRECTORIO_CON_UN_NOMBRE_MUY_LARGO$ cd ..
matt@syzygy:~$ pwd
/home/matt
matt@syzygy:~$ cd D*ARGO
matt@syzygy:~/DIRECTORIO_CON_UN_NOMBRE_MUY_LARGO$ cd ..
matt@syzygy:~$ rm -r DIRECTORIO_CON_UN_NOMBRE_MUY_LARGO/
matt@syzygy:~$ ls
bin  cxoffice  Desktop  Documents  GMT  Rubbish  SIRIUS  tmp
matt@syzygy:~$
```

Figure 3: Usando comandos unix

1.5 Copiando y moviendo archivos

Primeramente vamos a bajar dos archivos que nos serán útiles:

- julday
- calday

Probablemente estos archivos se descargarán directamente al escritorio o al \$HOME, para moverlos al directorio **bin** use :

```
mv Escritorio/Calday bin/
```

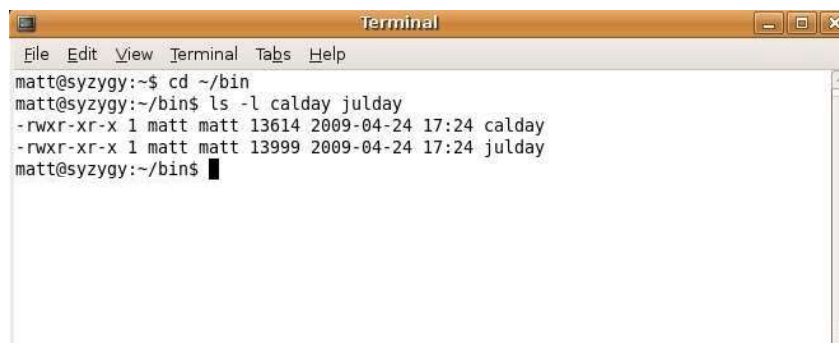
Nota:

Para copiar archivos se usa *cp* en lugar de *mv*

1.6 Ejecutables y permisos

Si tiene Calday, Julday en su directorio **~bin** (~ significa tu espacio \$HOME), revisemos si podemos ejecutarlos. Primeramente vamos al directorio **bin**, luego hacemos una lista en formato largo:

```
cd ~/bin
ls -l
```



```
matt@szygy:~$ cd ~/bin
matt@szygy:~/bin$ ls -l calday julday
-rwxr-xr-x 1 matt matt 13614 2009-04-24 17:24 calday
-rwxr-xr-x 1 matt matt 13999 2009-04-24 17:24 julday
matt@szygy:~/bin$
```

Figure 4: Cambiando de directorios

Hay que fijarse en los primeros 10 caracteres

- 1 d para un directorio - para un archivo
- 2 3 4 Permiso para el usuario
- 5 6 7 Permiso para el grupo (si el espacio esta compartido)
- 8 9 10 Permiso para los demás

En donde,

- r significa permiso para leer
- w indica permiso de guardar/borrar
- x indica permiso de ejecución

Unos ejemplos :

- a) -rwxrwxrwx: Un archivo que todos pueden leer, cambiar/borrar, ejecutar
- b) -rw- - - - - : Un archivo que solamente puede ser leído y cambiado por el dueño de la cuenta.
Por ejemplo, algo de email

Para que los archivos **Julday**, **Calday** puedan ser ejecutables deben tener permiso de ejecución. Hacemos esto con las siguientes instrucciones :

- `chmod a+rxw julday` (para todos/all)
- `chmod u+rxw julday` (para usuario)
- `chmod g+rxw julday` (para grupo)
- `chmod o+rxw julday` (para otros)

Note que con **chmod** se usa + para dar los permisos, y - para quitar permisos.

Ahora hagan una lista a ver si esos programas son ejecutables, si lo son, están listos para usarse.

1.7 Variables en Bash

Existen varias variables en el terminal. Se puede usar el comando **echo** para pedir sus valores. Por ejemplo:

```
echo $USER
echo $HOME
echo $PATH
```

El valor del \$PATH especifica los directorios donde existen ejecutables, comandos y programas. Volveremos a eso en la próxima sección, ya que deseamos que nuestros scripts, y GMT esten dentro de este \$PATH.

1.8 Espacio en el disco

Para revisar el espacio en el disco, use:

```
df -k
```

1.9 Espacio en uso de archivos y directorios

Para comprobar el tamaño de un archivo o directorio usamos el comando **du**, donde para un directorio usamos la opción **-s**. La opción **h** es para ver el tamaño en megabytes.

```
du -sh directorio
```

1.10 Archivo de datos

- global_seismicity_feb27-apr19_2010.txt

Este archivo contiene la sismicidad global desde el 27 de febrero al 19 de abril del año 2010. Creen un directorio de trabajo para este curso y muevan este archivo de sismicidad allá. Explore este archivo, lo pueden hacer con **gedit** o cualquier editor de texto.

2 Comandos básicos

Descargen los siguientes archivos :

- esk_annual_means.txt
- global_seismicity_feb27-apr19_2010.txt
- southern_Chile_and_Argentina.txt

2.1 More

more es un comando para ver archivos de texto en el terminal. Pruebe los siguientes comandos:

```
more global_seismicity_feb27-apr19_2010.txt
```

Aquí tenemos la sismicidad global, de la red global de IRIS, para las primeras semanas después del terremoto magnitud 8.8 en Chile el 2010. Las columnas son (catálogo, día, hora (UTC), latitud, longitud, profundidad, código de la región, código de la zona sísmica, tipo de magnitud, magnitud).

2.2 Head, Tail

Si no quieres ver todas las líneas de un archivo de texto, se puede usar **head** o **tail**. Pruebe lo siguiente:

```
head global_seismicity_feb27-apr19_2010.txt
tail global_seismicity_feb27-apr19_2010.txt
tail -1 global_seismicity_feb27-apr19_2010.txt
```

2.3 Pipes

Se puede ejecutar varios comandos en una sola línea usando un "pipe", | , para mandar la salida de la primera operación a una segunda. Cuando introducimos más comandos abajo, vamos a combinarlos usando esta técnica.

2.4 Awk

awk es una herramienta muy poderosa para manipular datos en columnas. El manual de **awk** es muy grande, así que aquí mostramos solo algunos ejemplos:

- i) Notar el número de la fila (NR = Number of Row):

```
more esk_annual_means.txt | awk '{print NR}'
```

también se puede notar el número de filas en total:

```
more global_seismicity_feb27-apr19_2010.txt | awk 'END {print NR}'
```

- ii) Notar el número de columnas en cada fila (NF = Number of Fields):

```
more esk_annual_means.txt | awk '{print NF}'
```

iii) Elegir solamente líneas con un cierto número de columnas:

```
more esk_annual_means.txt | awk '{ if ( NF == 11 ) print $0 }'
```

aquí \$0 implica toda la línea.

iv) Seleccionar solamente las columnas de datos que quieres:

```
more global_seismicity_feb27-apr19_2010.txt | awk '{print $4, $5, $10}'
```

(en este caso, elegimos el latitud, longitud, y magnitud de los sismos en el catálogo, note que todavía queda un "," entre los valores, podemos sacar eso usando sed (sección 2.6)).

v) Poner texto entre las columnas:

```
more global_seismicity_feb27-apr19_2010.txt | awk '{print "latitud "$4, "longitud "$5, "magnitud "$10}'
```

note que dentro el **awk**, el "," significa un espacio. Podemos obtener el mismo resultado que arriba con:

```
more global_seismicity_feb27-apr19_2010.txt | awk '{print "latitud",$4, "longitud",$5, "magnitud",$10}'
```

vi) Elegir las líneas de un set de datos requeridos:

```
more esk_annual_means.txt | awk '{ if ( $1 != "Year" && NF == 11 ) print $0 }'
```

note que aquí, el condicional dentro del "if" requiere que las líneas tengan 11 columnas, y que la primera columna no se llame Year

vii) Elegir datos solamente entre ciertos valores, por ejemplo terremotos de magnitud 5.6, 5.7, 5.8, 5.9, 6.0

```
more global_seismicity_feb27-apr19_2010.txt | awk '{if ( $10 > 5.5 && $10 <= 6.0 ) print $0}'
```

- viii) Hacer cálculos básicos, por ejemplo para convertir la profundidad de los terremotos (columna 6) en distancia desde el centro de la Tierra, podemos poner:

```
more global_seismicity_feb27-apr19_2010.txt | awk '{print "latitud "$4, "longitud "$5, "dist" 6371 -$6}'
```

- ix) Otro cálculo, para imprimir el año, la declinación y la inclinación del campo magnético, con los ángulos en grados decimales:

```
more esk_annual_means.txt | awk '{if ( $1 != "Year" && NF == 11 ) print $1, $2 - ($3/60), $5 + ($6/60)}'
```

Quizas queremos guardar esta información en otro archivo de texto, que se llama `esk_dec_inc_intensidad.txt`

```
more esk_annual_means.txt | awk '{ if ( $1 != "Year" && NF == 11 ) print $1, $2 - ($3 / 60), $5 + ($6 / 60), $10}' > esk_dec_inc_intensidad.txt
```

- x) Tomar el promedio de todos los valores en una columna (en este caso, tomamos el promedio de la intensidad del campo magnético (nT) en ESK para el último siglo):

```
more esk_dec_inc_intensidad.txt | awk 'BEGIN{sum=0}{sum+=$4}END{print sum/NR}'
```

Aquí, `BEGIN { }` dice que en la primera fila ponga la variable `sum` igual a cero. En la parte siguiente `{ }` dice que sumamos el valor en la columna cuatro línea por línea. `END { }` dice lo que debe hacer `awk` cuando llega a la última fila.

- xi) Finalmente, algo que podría servir para reformatear un bloque de datos como

```
0.1 0.2 0.3
0.4 0.5 0.6
0.7 0.8 0.9
```

en solo una columna de datos se puede usar `awk`. Suponiendo que el bloque de datos esta guardado en un archivo llamado `datos`, se puede usar

```
more datos | awk '{i=1; while (i < NF+1) {print $i; ++i}}'
```

para obtener

```
0.1
0.2
0.3
0.4
etc.
```

Revisa que se entiende lo que hace este comando!

2.5 Grep

grep (get regular expression) es un programa que busca patrones en archivos. Solamente las líneas que contienen este patron son mostradas. Por ejemplo, para mostrar los terremotos del catalogo QED/NEIC ponemos:

```
more global_seismicity_feb27-apr19_2010.txt | grep "QED/NEIC"
```

Usamos **grep** con la opción `-v` para buscar las líneas que no tienen el patrón, por ejemplo:

```
more global_seismicity_feb27-apr19_2010.txt | grep -v "QED/NEIC"
```

lo que nos entrega los terremotos que no son de este catálogo.

2.6 Sed

sed (stream editor) es un programa que puede transformar texto, línea por línea. El formato general es:

```
sed 's/algo/algo diferente/g'
```

Por ejemplo, para cambiar las comas en el archivo de los terremotos a estrellas, ponemos:

```
more global_seismicity_feb27-apr19_2010.txt | sed 's/,/*g'
```

Para eliminar las comas, ponemos:

```
more global_seismicity_feb27-apr19_2010.txt | sed 's/,//g'
```

2.7 Sort

El comando **sort** puede ser usado para ordenar las columnas, alfabéticamente o numéricamente. Por ejemplo para ordenar la columna 1 del archivo de sismicidad alfabéticamente tecleamos:

```
more global_seismicity_feb27-apr19_2010.txt | sort -k1
```

Para ordenar columna 6 (profundidad) del archivo de sismicidad numéricamente tecleamos:

```
more global_seismicity_feb27-apr19_2010.txt | sort -k6 -n
```

Para ordenar columna 6 numéricamente, y al revés:

```
more global_seismicity_feb27-apr19_2010.txt | sort -k6 -n -r
```

Noten que **-n** ordena numéricamente, pero si eso no funciona prueba **-g** (general numeric sort) que es mas lento pero acepta mas formatos para los números, incluyendo exponenciales etc.

2.8 Combinación de comandos, pipes

Tomar el archivo de sismicidad, elegir el catalogo FINGER/NEID, sacar las comas, ordenar las magnitudes numéricamente, elegir los datos de la última línea, es decir, los datos del mayor magnitud.

```
more global_seismicity_feb27-apr19_2010.txt | grep "FINGER/NEIC" | sed 's/,//g' | sort -k10 -n | tail -1
```

Elegir las regiones 134 y 135 en la lista de sismicidad:

```
more global_seismicity_feb27-apr19_2010.txt | grep "FINGER/NEIC" | sed 's/,//g' | awk '{if ( $7 == 134 || $7 == 135 ) print$0}'
```

Notar que **||** dentro del **if ()** significa “o”.

2.9 Preguntas

1. ¿ Dónde fue el terremoto más profundo en el catálogo global ?
2. ¿ Cual es el número de réplicas en las regiones 134, 135 el día 28 feb 2010 ?
3. ¿ Cuántos terremotos de magnitud 6.0 o mayor, en regiones 134, 135 ocurrieron entre feb 27 - apr 19 ?
4. ¿ Por cuántos años estuvo la intensidad del campo magnético sobre 48000 nT en la estación ESK ?
5. ¿ Cual fue el promedio de la inclinación y la declinación del campo magnético en ESK entre los años 1910 y 2010 ?

3 Printf, expresiones regulares, tar, bashrc

3.1 Printf en awk

Desde ahora, usamos **awk** para manipular columnas de datos. A veces la salida del comando se ve "sucia", es decir, las columnas no están alineadas, y con los números en el formato original. Más que un problema estético, algunos programas (especialmente los escritos en fortran) necesitan que el orden de los datos en las columnas sea muy específico, porque requieren que los datos esten en sus posiciones exactas en cada línea. Por eso usamos **printf** (print format) en awk para modificar las columnas.

Trabajaremos con el archivo **global_seismicity_feb27-apr19_2010.txt**, modificamos las columnas usando awk, pero ahora en vez de usar print usamos printf.

Pruebe las siguientes sentencias:

```
head global_seismicity_feb27-apr19_2010.txt | awk '{print $5}'
```

Aquí usamos awk para extraer del archivo la columna de las longitudes de los terremotos.

Ahora usamos printf:

```
head global_seismicity_feb27-apr19_2010.txt | awk '{printf "%10s\n", $5}'
```

Aquí especificamos el formato de la columna, en este caso %10s significa que cada entrada en la columna tiene un espacio de 10 caracteres (character string, s) y el \n es un comando para empezar una nueva línea entre las entradas (pruebe el comando sin el \n para ver eso).

Notar que si la entrada es más grande que el espacio disponible, podemos elegir si sale toda la entrada, o solamente la primera parte:

```
head global_seismicity_feb27-apr19_2010.txt | awk '{printf "%7s\n", $5}'
head global_seismicity_feb27-apr19_2010.txt | awk '{printf "%7.7s\n", $5}'
head global_seismicity_feb27-apr19_2010.txt | awk '{printf "%7.6s\n", $5}'
```

También podemos especificar que la entrada en columna sea un entero (integer, i):

```
head global_seismicity_feb27-apr19_2010.txt | awk '{printf "%5i\n", $5}'
head global_seismicity_feb27-apr19_2010.txt | awk '{printf "%7.4i\n", $5}'
```

Podemos especificar el número de decimales de una entrada:

```
head global_seismicity_feb27-apr19_2010.txt | awk '{printf "%10.4f\n", $5}'
head global_seismicity_feb27-apr19_2010.txt | awk '{printf "%10.2f\n", $5}'
```

Expresar un número en forma exponencial:

```
head global_seismicity_feb27-apr19_2010.txt | awk '{printf "%15e\n", $5}'
head global_seismicity_feb27-apr19_2010.txt | awk '{printf "%12.3e\n", $5}'
```

Finalmente podemos hacer todo esto a múltiples columnas, pruebe lo siguiente:

```
head global_seismicity_feb27-apr19_2010.txt | awk '{printf "%15e\ %5.1f\n", $5, $2}'
```

Cada opción dentro del **printf** se aplica a las culumnas 2 y 5 respectivamente.

3.2 Substrings en awk

Algo que es a veces útil con datos, es elegir solamente parte de una columna; por eso usamos `substr($X,Y,Z)` donde `$X` representa la columna X-ésima, `Y` representa el caracter de la columna en donde empezar, y `Z` representa el número de los caracteres requeridos. Eso es más fácil de mostrar con ejemplos:

```
head global_seismicity_feb27-apr19_2010.txt | awk '{print $3, substr($3,1,2)}'  
head global_seismicity_feb27-apr19_2010.txt | awk '{print $3, substr($3,1,4)}'  
head global_seismicity_feb27-apr19_2010.txt | awk '{print $2, substr($2,9,2)}'
```

3.3 Expresiones regulares

Expresiones regulares son una manera poderosa para clasificar partes de texto. Para mostrar algunos ejemplos usamos el archivo **southern_Chile_and_Argentina.txt**

Una expresión de lo más simple es buscar un set de caracteres (aquí uso la opción `-i` para buscar con mayúscula o minúscula)

```
more southern_Chile_and_Argentina.txt | grep -i "volcan"
```

También se puede sustituir un `.` para cualquier caracter, o elegir un set de caracteres con `[...]`:

```
more southern_Chile_and_Argentina.txt | grep -i "a..of"  
more southern_Chile_and_Argentina.txt | grep -i "m[ae]"
```

Si quieres usar uno de los caracteres especiales `? . [] ^ $` tienes que usar un `\` antes del caracter. Por ejemplo, compare:

```
more southern_Chile_and_Argentina.txt | grep -i "3.5"  
more southern_Chile_and_Argentina.txt | grep -i "3\\.5"
```

3.4 \$PATH y .bashrc

En esta sección estamos modificando un archivo fundamental a la operación de un terminal. Si tienen dudas, recomiendo leer más sobre el archivo `.bashrc` (usando google), o pregunta en la clase.

El valor del variable `PATH` contiene los lugares de todos los ejecutables. Es decir, programas y scripts dentro de los directorios en la lista asociada con la variable `PATH` se pueden correr desde cualquier ubicación en el terminal. Para ver el valor de eso, use:

```
echo $PATH
```

Nosotros queremos poner el directorio `~/bin` en esta variable también. Se puede ver exactamente cual es su ubicación en el árbol Linux.

```
cd ~/bin  
pwd
```

Para cambiar la variable, podemos usar el comando:

```
export PATH=ruta del archivo:$PATH
```


Por ejemplo, si mi directorio bin era /home/matt/bin yo uso el comando `export PATH=/home/matt/bin:$PATH` que significa que el valor de PATH esta compuesto de /home/matt/bin: y el valor anterior del PATH.

Después de este comando, el valor del PATH debe cambiar. Se puede revisar con `echo $PATH`; y ahora debe ser posible correr ejecutables en su directorio ~ /bin. Pruebe que reconoce el comando `calday`, teclee:

```
which calday
```

o simplemente corriendo `calday 100 2010` desde cualquier directorio.

Este cambio en el PATH es temporal, pero podemos hacerlo permanentemente modificando el archivo `.bashrc` que se encuentra en \$HOME. Este archivo consiste en una serie de comandos que se ejecutan cada vez que se inicia el terminal.

```
cd
gedit .bashrc
```

Si no existe un `.bashrc`, baja un clón de aca y uselo: `.bashrc`

Vamos a poner una nueva línea al archivo para definir el valor del PATH (`export PATH=ruta del archivo:$PATH`). Ahora, cuando abres un nuevo terminal, se pueden correr tus propios comandos (como `calday` y `julday`) desde cualquier otro directorio.

Nota:

Revisa que cuando abres un terminal, la variable \$PATH tiene el valor requerido, y que se pueden ejecutar los comandos del sistema. Teclee:

```
which gedit
which awk
```

y que también se pueden ejecutar los comandos que están en tu directorio bin:

```
which calday
which julday
```

3.5 Tarballs

tar es un programa que puede comprimir varios archivos y directorios dentro de un solo archivo (es similar al zip, o rar). Por ejemplo, baja el archivo `gmt_files.tar.gz` y ponlo en un lugar temporal.

Podemos extraer el directorio con :

```
tar -xvzf gmt_files.tar.gz
```

en donde,

- x=extract
- v=verbose
- z=zipped
- f=file

(Solo use z para archivos que son zipped (.gz) también).

3.6 Preguntas

1. ¿ Cual sería la sintáxis para comprimir los archivos *file1, file2, file3* en un archivo **files.tar.gz** ?
2. ¿ Cual es el comando para listar los archivos o directorios contenidos en un archivo **tar.gz** sin que este sea descomprimido ?

4 Scripts, loops y condicionales

4.1 Introducción a Scripts en bash

Un script es muy similar a un programa, aunque se puede correr solamente dentro de un contexto específico. Los shell scripts son un tipo de script desarrollado para correr dentro de un terminal, como el de bash.

Todos los shell scripts que encontraremos en este curso (incluyendo los de GMT) empiezan de la misma manera, con un gato-exclamación (#!), donde este informa la ruta del shell que queremos usar para interpretar el script, en nuestro caso /bin/bash.

Ejemplo:

```
#!/bin/bash
# Este es el inicio de un shell script
```

4.2 Fijando una variable

Fijar una variable es simple, por ejemplo, escriban en la terminal:

```
MIVAR="mi variable"
```

Para ver la variable en la terminal, usamos:

```
echo $MIVAR
```

Notar que no hay espacio alrededor del = cuando se fija una variable.

4.3 Scripts básicos

Tipea lo siguiente dentro de un nuevo archivo de texto, el cual llamaremos **hola.sh**

```
#!/bin/bash
echo "Hola a todos"
```

Ahora en la terminal cambie los permisos al archivo para hacerlo ejecutable:

```
chmod a+x hola.sh
```

Ahora ya puedes correr tu script, teclea en la terminal:

```
./hola.sh
```

Ahora introducimos una variable dentro de un script que estará en un archivo de texto que lo llamaremos (**hola2.sh**)

```
#!/bin/bash
VAR="hola de nuevo"
echo $VAR
```

Cambia permisos y corre el script.

Usaremos en este curso comandos que aceptan parámetros para definir operaciones. Con esto tenemos scripts más flexibles que aquellos que definen los parámetros dentro del script.

Para usar argumentos, a cada variable pasada a nuestro script, se le asigna un número \$1, \$2, \$3 etc. Por ejemplo, (**argumentos.sh**)

```
#!/bin/bash
echo "El primer argumento es" $1
echo "El segundo argumento es" $2
```

Hace el script ejecutable y córrelo! Cuando el script muestra variables como \$1 y \$2 que no están definidas dentro del archivo de texto, entonces las puedes definir como tu quieras, así de la siguiente forma se corre este script:

```
./argumentos.sh 3 4
```

Es decir, en este caso la variable \$1 tomará el valor de 3 y la variable \$2 tomará el valor de 4. Puedes poner expresiones alfanuméricas como letras o palabras y serán asociadas a cada variable. Pero en este caso, como definiste sólo dos, por mas argumentos que pongas en la terminal, aparecerán sólo los dos primeros. Prueba lo siguiente:

```
./argumentos.sh hola mundo
./argumentos.sh hola mundo feliz
```

¿Ves? en este caso “feliz” no aparece, para que aparezca en el script tienes que definir otra variable \$3. Nota que cada variable se diferencia de la otra por el espacio que pongas entre ellas al ejecutar el script.

Siempre los scripts requieren un cierto número de argumentos. Por ejemplo, si no recibe dos argumentos, el script **argumentos2.sh** alerta al usuario, y no corre si no recibe el número de parámetros que requiere:

```
#!/bin/bash

#Condicion que requiere el numero de argumentos sea 2:
if [ $# != 2 ]
then echo "ERROR: El uso correcto es: ./argumentos2.sh arg1 arg2"
exit
fi

#Si pasamos la condicion, corremos lo siguiente:
echo "El primer argumento es" $1
echo "El segundo argumento es" $2
```

Corre este script con y sin dos argumentos para ver la diferencia. Note que hay unos comentarios dentro del script (*líneas que empiezan con #*) y también un condicional [if ... then ... fi]

A veces, se requiere que el script pida algo al usuario. Para eso usamos el comando `read` (**hola3.sh**)

```
#!/bin/bash
echo "Cual es tu nombre?"
read VAR
echo "Hola "$VAR"!"
```

4.4 Aritmética en bash

Los shell scripts son usualmente usados para correr una serie de comandos con ciertos parámetros. A veces un poco de aritmética es útil. Acá hay una lista de algunos operadores que se pueden usar:

Sintáxis	Significado
<code>a+b</code> , <code>a-b</code>	adición/sustracción
<code>a*b</code> , <code>a/b</code>	multiplicación/división
<code>a%b</code>	módulo, resto después de la división
<code>a**b</code>	exponencial

Por ejemplo, prueba los siguientes comandos:

```
echo $[100+5]
echo $[100/3]
echo $[100%3]
echo $[2**4]
```

Note que el comando **awk** también puede hacer aritmética en bash:

```
echo 100 3 | awk '{printf"%0.2f\n", $1/$2}'
```

4.5 Expresiones lógicas

El número de expresiones lógicas que pueden verificarse es muy grande, incluyendo operadores para cadenas de caracteres y números enteros.

4.5.1 Operadores de comparación numéricos

Operador	Singnificado
<code>-eq</code>	igual que
<code>-ne</code>	no igual que
<code>-gt</code>	mayor que
<code>-ge</code>	mayor o igual que
<code>-lt</code>	menor que
<code>-le</code>	menor o igual que

4.5.2 Operadores de comparación de texto

Bash utiliza los comparadores habituales en matemáticas para comparar textos (mientras que para números utiliza los que se han visto anteriormente). Así la lista de comparadores es:

Operador	Singnificado
=	igual
!=	no igual
>	mayor que
<	menor que

4.6 Loops

Si queremos repetir algo varias veces, podemos usar loops dentro de un script :

4.6.1 for

for elige un rango de variables, o una lista de archivos, y aplica comandos a cada uno de ellos. Por ejemplo, en el script **for.sh** imprimimos los números del 1 al 10 en el terminal.

```
#!/bin/bash
for x in {1..10}
do
echo $x
done
```

o si queremos hacer una copia de seguridad *back up* (copiar cada archivo de un directorio a un archivo .bak) podemos usar: (**for1.sh**)

```
#!/bin/bash
for archivo in `ls -l`
do
cp $archivo $archivo.bak
done
```

4.6.2 While

El comando **while** ejecuta el loop de instrucciones situado entre **do** y **done** mientras se cumpla la condición especificada como parámetro en la llamada. El siguiente script muestra lo que hace el condicional **while**, donde se define la variable $x = 0$, y el condicional dice que muestre " x " mientras éste sea menor que 12. El resultado será una sucesión desde 0 (valor inicial) hasta 11 (valor definido por while). Llamaremos a este script (**while.sh**)

```
#!/bin/bash

#Definiendo x igual a 0
x=0

#Definiendo el condicional que dice que muestre el
#numero x mientras este sea menor que 12
while [ $x -lt 12 ]
do
echo $x

#Aumentando x por 1 (para crear una sucesion de x ascendente
#para cada loop del while)
x=$((x+1))

#Fin de condicion while
done
```

4.7 Condicionales

En todo entorno de programación es necesario automatizar la repetición de determinadas acciones un número fijo de veces o en función de que se cumpla o no una condición.

4.7.1 if

Este comando permite seleccionar entre algunas opciones, por ejemplo, el siguiente script (**if.sh**):

```
#!/bin/bash

#El script pide que ingreses tu edad:
echo "Ingresa su edad"
read edad

#Condicional que selecciona tu condicion laboral
#dependiendo del valor que asignes a la variable $edad
if [ $edad -ge 65 ]; then
echo "Si no te has jubilado, deberias hacerlo"
else
echo "Eres activo laboralmente"
fi
```

En el siguiente script que combina **for**, **while** e **if** vamos a encontrar todos los números primos que se encuentren dentro de una sucesión de números "x" (en este caso entre 2 y 10, porque el 1 no se considera primo). El método para hacerlo consiste en tomar los módulos (o resto) del número "x" dividido por todos los números que sean menores que él mismo; los números primos nunca van a tener un resto cero de esa división, así que de esa forma se pueden identificar. (**primo.sh**)

```
#!/bin/bash
#Corre un loop para x de 2 a 10 (ya que el 1 no se considera entre los
#numeros primos):
for x in {2..10}
do

#Define una variable $primo para ser igual a 1
primo=1

#Define y igual a 2
y=2

#Condicion para hacer el siguiente conjunto de instrucciones solo cuando
#y sea menor que x
while [ $y -lt $x ]
do

#Calcula el resto de la division de x/y
resto=$((x%y))

#Si el resto es cero para cualquier set de $x, $y cambia el variable $primo a cero
if [ $resto -eq 0 ]
then
primo=0
fi

#Aumenta el valor de y por 1
y=$((y+1))
#Termina la condiccion de "while"
done

#Si todavia el valor de primo es 1. Es decir, para cada valor de $x, de todos los
#set de $x, $y que se calcularon en el while (mientras $y<$x), ninguno de ellos tuvo
#resto cero, entonces el numero $x es primo
if [ $primo -eq 1 ]
then
echo $x "es primo"
fi

#Termina el loop de "for"
done
```


4.7.2 Case

La sentencia *case* en bash, es similar a la sentencia *switch* en C, Matlab y Octave. Esta sentencia no es un lazo como *for* y *while*, es decir, no ejecuta un bloque de comandos n veces, en vez de eso, *case* chequea alguna condición y controla el flujo del programa.

Cuando trabajen en la sección de GMT, *case* será una útil herramienta.

Su estructura es la siguiente:

```
case $variable in
  caso 1) bloque de comandos ;;
  caso 2) bloque de comandos ;;
  caso 3) bloque de comandos ;;
  caso n) bloque de comandos ;;
esac
```

A continuación, un sencillo ejemplo que entrega el horario de clases según el día.

```
#!/bin/bash
clear
dia=`date +%u`
case $dia in
  1) echo "09:00 - 12:00 -> Meteorologia Sinoptica , Sala Dgeo"
    echo "16:00 - 17:00 -> Ingles , CFRD" ;;
  2) echo "10:00 - 12:00 -> Estadistica , FM-204"
    echo "17:00 - 20:00 -> Sismologia de volcanes , Sala Dgeo" ;;
  3) echo "10:00 - 12:00 -> Metodos Matematicos de la Geofisica , FM-201"
    echo "09:00 - 11:00 -> Modelacion de Tectonica , A-213"
    echo "18:00 - 20:00 -> Linux, Scripts y GMT , FM-304" ;;
  4) echo "11:00 - 12:00 -> Horario de consulta Linux, Scripts y GMT , Oficina Matt"
    echo "09:00 - 11:00 -> Preparacion Fisica , Casa del Deporte" ;;
  *) echo "Hoy no tengo clases !!" ;;
esac
echo
```

5 Scripts para manipular datos

5.1 Introducción

Supongamos que queremos escribir un script para ver el número de réplicas por día después del terremoto del 27 de febrero 2010 en Chile, en comparación con el número de días después del terremoto. Para eso, necesitamos:

- i) Tomar el día en el archivo **global_seismicity_feb27-apr19_2010.txt** como una variable
- ii) Para cada día, calcular el número de réplicas asociadas a el
- iii) Calcular cuantos días después del 27 de febrero esta ella
- iv) Presentar los datos en el terminal, o en otro archivo

5.2 Elegir un rango de variables de un archivo

En el archivo **global_seismicity_feb27-apr19_2010.txt**, la fecha esta dada en la segunda columna:

```
more global_seismicity_feb27-apr19_2010.txt | awk '{print $2}'
```

Aquí, cada día esta representado varias veces, si queremos una lista donde el día esta representado solo una vez, tenemos que usar el comando **sort**, con la opción **-u** que significa única:

```
more global_seismicity_feb27-apr19_2010.txt | awk '{print $2}' | sort -u
```

En nuestro script, queremos trabajar con cada día individualmente, y podemos usar el loop **for** para generar este efecto. El script toma forma ...

```
#!/bin/bash
for dia in `more global_seismicity_feb27-apr19_2010.txt | awk '{print $2}' | sort -u`
do
#aqui van los comandos para manipular los datos de cada dia
done
```

5.3 Calculando el número de réplicas asociadas con cada día

Desde ahora, tenemos un loop de variables de forma " año/mes/día "

Para hacer la estructura necesaria para aislar el número de réplicas que vamos a incorporar dentro del script, siempre es útil probar en el terminal con una variable de la forma "2010/03/21" (o equivalente). Para probar, podemos definir:

```
dia="2010/03/21"
```

Asi que ahora, dentro del terminal, la variable día tiene esta definición (pruebe echo \$dia). Para contar el número de sismos asociados con cada día, usamos **grep** (para elegir las líneas que correspondan al día que queremos) y **awk** (para contar el número de sismos), por ejemplo:

```
more global_seismicity_feb27-apr19_2010.txt | grep 2010/03/21 | awk 'END {print NR}'
```

y si queremos incorporar la variable \$dia, usamos:

```
more global_seismicity_feb27-apr19_2010.txt | grep $dia | awk 'END {print NR}'
```

Nota que esto es el número de sismos globales de todos catálogos. Nosotros queremos elegir un catálogo y, solamente el rango de latitudes y longitudes que representan el sismo de Chile. Note que si queremos usar límites sobre el latitud y longitud, necesitamos eliminar las comas en el archivo de datos.

Podemos usar un mapa de la área de ruptura, por ejemplo:

http://www.tectonics.caltech.edu/slip_history/2010_chile/preliminary/chile10_map.jpg

para estimar el rango de latitudes y longitudes que queremos. Entonces:

```
more global_seismicity_feb27-apr19_2010.txt | sed 's/,//g' | \\  
grep "FINGER/NEIC" | grep $dia | \\  
awk '{ if ( $4 > -38 && $4 < -33 && $5 > -75 && $5 < -71 ) print $0}' | \\  
awk 'END {print NR}'
```

es la sentencia para contar el número de sismos del catalogo FINGER/NEIC del día \$dia, entre 33° y 38°S y 71° y 75°W. (pruébalo para diferentes días).

Nota:

Cuando tipees esto en un terminal, las instrucciones deben ir en una misma línea. De no ser así, puedes usar \ para indicar que las instrucciones en la línea siguiente son adyacentes a la actual. También se puede usar \\.

Podemos definir este número a ser la variable **num**:

```
num=`more global_seismicity_feb27-apr19_2010.txt | \\  
sed 's/,//g' | grep "FINGER/NEIC" | grep $dia | \\  
awk '{ if ( $4 > -38 && $4 < -33 && $5 > -75 && $5 < -71 ) print $0}' | \\  
awk 'END {print NR}'`
```

y revisarlo con:

```
echo $num
```

Ahora estamos listos para agregar esta línea al script que queremos crear :

```
#!/bin/bash

# Haciendo un loop sobre cada dia:
for dia in `more global_seismicity_feb27-apr19_2010.txt | awk '{print $2}' | sort -u`
do

# Contando el numero de sismos del catalogo FINGER/NEIC, del dia $dia,
# entre 33 y 38 grados S y 71 y 75 grados W. Esta cantidad de sismos se
# llama num

num=`more global_seismicity_feb27-apr19_2010.txt | sed 's/,//g' \\  
| grep "FINGER/NEIC" | grep $dia \\  
| awk '{ if ( $4 > -38 && $4 < -33 && $5 > -75 && $5 < -71 ) print $0}' \\  
| awk 'END {print NR}'`

done
```

Notar que \\ es para indicar que la sentencia continúa, función similar a los .. en Matlab.

5.4 Calcular el número de días después del 27 de febrero

Seguimos en el terminal con la variable `dia="2010/03/21"`. Queremos cambiar esta fecha para el número de días después del 27 de febrero. La manera más fácil de hacer eso es usando el comando "julday", que ya tenemos en el sistema. Recuerde:

```
julday 03 21 2010
```

lo que nos da el día del año (080) de esa fecha, es decir,

Calendar Date 03 21 2010

Julian Date 080 2010

Ahora, con ayuda del comando `julday`, vamos a cambiar la variable `$dia` a un nuevo formato. Para eso reemplazamos los "/" con " ", y reordenamos las columnas. Para sacar los "/", usamos `sed`.

```
echo $dia | sed 's/\/// /g'
```

Ahora cambiamos el orden de las columnas con `awk`:

```
echo $dia | sed 's/\/// /g' | awk '{print $2, $3, $1}'
```

así tenemos la fecha en el formato correcto para poder usarlo con `julday`. Definimos esto como una variable temporal, en este caso, `temp`:

```
temp=`echo $dia | sed 's/\/// /g' | awk '{print $2, $3, $1}'`
```

Revisa que es lo que se obtiene al tipear `echo $temp`.

Estamos listo para correr el comando `julday` con la variable `$temp`.

```
julday $temp
```

Pero solamente queremos como respuesta el día. Podemos aislarlo eligiendo la línea que solamente contiene la palabra "Julian":

```
julday $temp | grep Julian
```

el día es la tercera columna de esta línea:

```
julday $temp | grep Julian | awk '{print $3}'
```

OK, pero nosotros queremos el número de ese día después del 27 febrero (día 58 - revisalo con `julday 02 27 2010`). Entonces necesitamos sustraer 58 en el comando `awk`:

```
julday $temp | grep Julian | awk '{print $3-58}'
```

También queremos este número como una variable, vamos a llamarlo `tiempo`, para el tiempo en días después del terremoto. En el script:

```
tiempo=`julday $temp | grep Julian | awk '{print $3-58}'`
```

Ahora estamos listos para agregar estas líneas al script que esta en preparación:

```
#!/bin/bash

for dia in `more global_seismicity_feb27-apr19_2010.txt \\  
| awk '{print $2}' | sort -u`  
do

# contar el numero de sismos del catalogo FINGER/NEIC, del dia $dia,  
# entre 33 y 38 grados sur y 71 y 75 oeste. Llama este numero num

num=`more global_seismicity_feb27-apr19_2010.txt | sed 's/,//g' \\  
| grep "FINGER/NEIC" | grep $dia \\  
| awk '{ if ( $4 > -38 && $4 < -33 && $5 > -75 && $5 < -71 ) print $0}' \\  
| awk 'END {print NR}'`

# extraer el numero de dias despues del terremoto esta la fecha $dia  
# llama este tiempo

temp=`echo $dia | sed 's/\\/ /g' | awk '{print $2, $3, $1}'`  
tiempo=`julday $temp | grep Julian | awk '{print $3-58}'`

done
```

5.5 Presentando datos en un terminal

Podemos usar echo para poner los datos en el terminal, es decir:

```
echo $tiempo $num
```

O en el script anterior,

```
#!/bin/bash

for dia in `more global_seismicity_feb27-apr19_2010.txt \\  
| awk '{print $2}' | sort -u`  
do

# contar el numero de sismos del catalogo FINGER/NEIC, del dia $dia,  
# entre 33 y 38 grados sur y 71 y 75 oeste. Llama este numero num

num=`more global_seismicity_feb27-apr19_2010.txt | sed 's/,//g' \\  
| grep "FINGER/NEIC" | grep $dia \\  
| awk '{ if ( $4 > -38 && $4 < -33 && $5 > -75 && $5 < -71 ) print $0}' \\  
| awk 'END {print NR}'`

# extraer el numero de dias despues del terremoto esta la fecha $dia  
# llama este tiempo

temp=`echo $dia | sed 's/\\/ /g' | awk '{print $2, $3, $1}'`  
tiempo=`julday $temp | grep Julian | awk '{print $3-58}'`  
# presentar los datos en el terminal

echo $tiempo $num

done
```

Ahora el script esta listo, 7 líneas de código para obtener lo que queremos.

En los archivos en la página web <http://www.mttmllr.com/GMT>, se llama **sismos_por_dia.sh** (buscalo!). Hazlo ejecutable y córrelo !

```
./sismos_por_dia.sh
```

Si queremos poner la salida del script dentro de un nuevo archivo, por ejemplo dia_sismos.txt, usamos:

```
./sismos_por_dia.sh > dia_sismos.txt
```

5.6 Conclusión

El script que hemos creado podría ser hecho a mano en alrededor de una hora (cierto, es solamente un máximo de 2600 líneas para revisar). La ventaja de hacerlo en forma de un script es que, si es necesario, se puede cambiar facilmente el catalogo usado, los límites de latitud y longitud, poner un límite sobre magnitud, etc. También, se puede aplicar eso a diferentes grupos de datos, diferentes catalogos, etc., con poca revisión. Finalmente, con el conocimiento de manipular datos así, usando siempre los mismos comandos como sed, grep, awk y sort, se puede trabajar con cualquier set de datos seguro en el conocimiento de que no importa el formato, se puede extraer la información requerida.

Ahora estamos listo a poner los datos en forma gráfica.