513372

Linux, Scripts y GMT

Matt Miller
2016

Departamento de Geofísica
Universidad de Concepción

Contenidos

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.

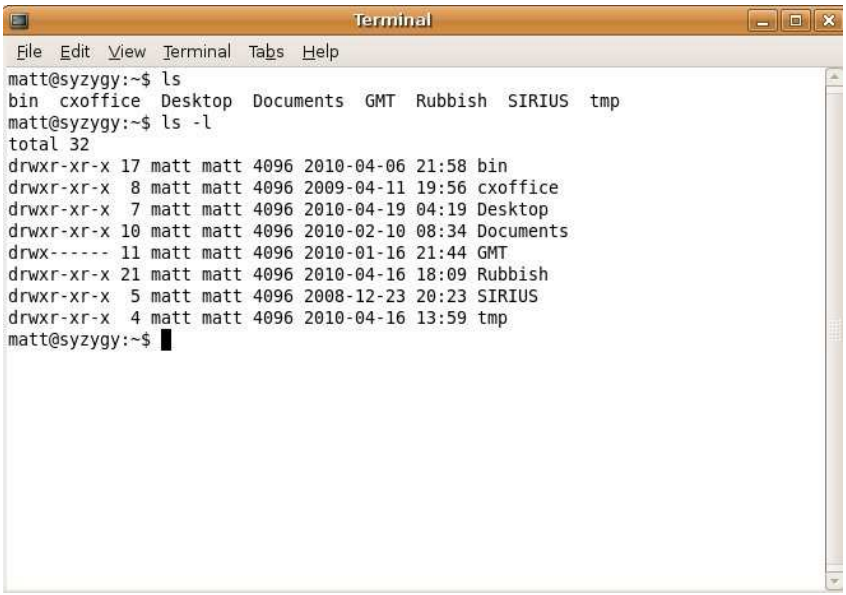


Figure 1: Listando archivos

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 exáctamente 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
```

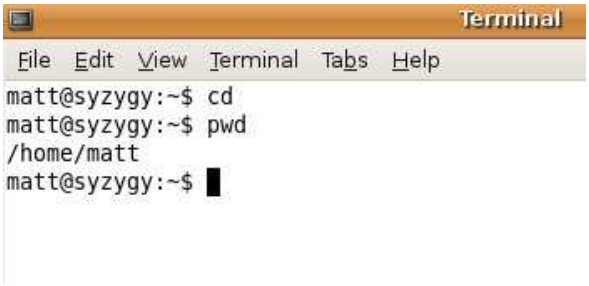


Figure 2: Mostrando ruta actual

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 arriva 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(*)

Creen 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 despúes 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
```

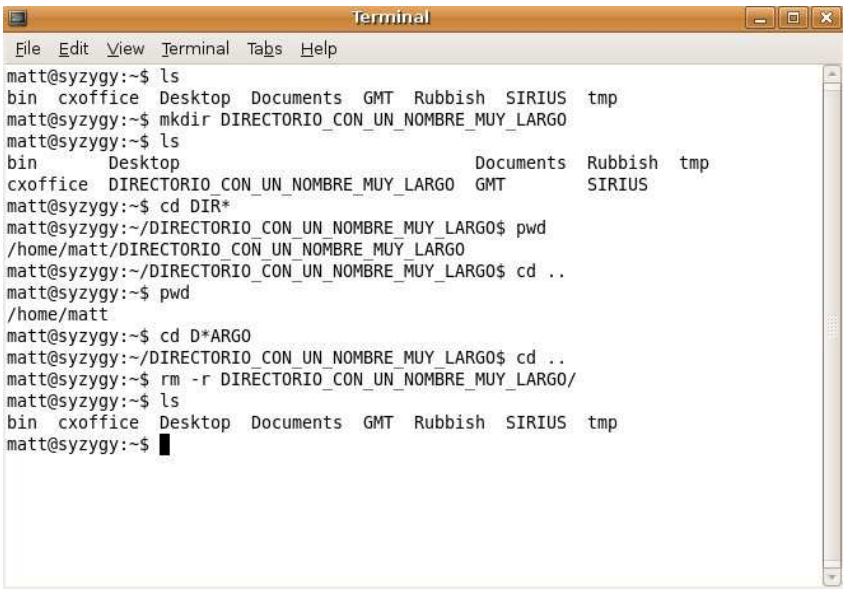


Figure 3: Usando comandos unix

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
```

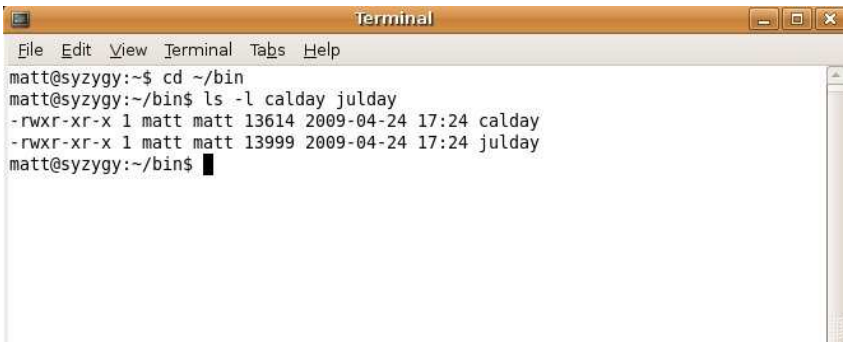


Figure 4: Cambiando de directorios

Hay que fijarse en los primeros 10 carácteres

- **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 leido 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+rwx julday (para grupo)
- chmod o+rwx 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 sismisidad 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 sismisidad allá. Exploren 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 angulos 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 este 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, alfabéticalmente o numéricamente. Por ejemplo para ordenar la columna 1 del archivo de sismisidad 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 $\sim$ /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 desarollado 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 usario. 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
a+b, a-b	adición/sustracción
a*b, a/b	multiplicación/división
a%b	módulo, resto depués de la división
a**b	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 carácteres y números enteros.

4.5.1 Operadores de comparación numéricos

Operador	Singnificado
-eq	igual que
-ne	no igual que
-gt	mayor que
-ge	mayor o igual que
-lt	menor que
-le	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 -1`
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 "Ingrese 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/dia "
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 eligir las líneas que corespondan 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 $día, 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. (prúebalo 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 despues del 27 de febrero. La manera más facil 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 repuesta el día. Podemos aislarlo elegiendo 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.

6 Instalación GMT

Con el conocimiento del uso de un terminal, comandos en bash como awk, sed, grep y sort, y unos scripts básicos para manipular datos con los comandos, estamos listos a installar GMT (Generic Mapping Tools).
La página web de GMT es http://gmt.soest.hawaii.edu/. Aquí, se pueden encontrar ejemplos de scripts en GMT, documentación sobre GMT, y como instalar el programa.

Cada año mas o menos las versiones de GMT se están actualizando. Por eso, para tener la última versión, dependiendo del año en que se empiece a trabajar en el curso, se entregará una hoja aparte de este manual indicando las instrucciones de instalación correspondientes a la versión actual del software GMT.

7 Mapas, costa y topografía

Descargen los siguientes archivos :

- topo_6.2.img.gz
- w100s10.Bathymetry.srtm.grd.gz

Note que estos archivos estan en un formato .gz (gzip). Para extraerlos, use el comando:

```
gunzip topo_6.2.img.gz
```

7.1 Colores en GMT

Colores en GMT están dados en el formato rojo/verde/azul para color, o GRIS para blanco y negro, donde los colores tienen valores entre 0 y 255. Por ejemplo, el color 0 significa 0/0/0 o negro, el color 255 significa 255/255/255 o blanco, el color 128 es gris.

- 255/0/0 es rojo
- 0/255/0 es verde
- 0/0/255 es azul

Cualquier combinación de rojo/verde/azul da un color diferente. Por ejemplo :

255/255/0 es amarillo, 255/140/0 es naranjo. El siguiente link es una guía interactiva para colores en este formato : http://www.switchonthecode.com/tutorials/javascript-interactive-color-picker

7.2 psbasemap

El comando **psbasemap** es usado para generar código postscript, (un gráfico .ps) cuya función es configurar el marco para el mapa. Ahora en este curso, vamos a generar gráficos en formato .ps. Se pueden ver los gráficos con el programa gs, evince, gimp u otros.
psbasemap trabaja con algunos parámetros, **-B** da la anotación de borde del mapa, **-J** la proyección del mapa, y **-R** la región requerida. **-X** y **-Y** son para mover el gráfico horizontal y verticalmente, respectivamente; y **-P** significa "portrait". **-V significa** "verbal". Por ejemplo, para generar un marco Mercator de la región del Bío Bío, podemos usar:

```
psbasemap -Ba1f0.5SEWN -JM16.0 -R-74/-71/-38.5/-36 -X2 -Y6 -P -V > test.ps
```

Para ver el efecto de la -B, siempre la manera más fácil , es probar diferentes situationes, por ejemplo:

```
psbasemap -Ba0.5f0.1g1SeW -JM16.0 -R-74/-71/-38.5/-36 -X2 -Y6 -P -V > test.ps
psbasemap -Ba0.5f0.1g1/a1.0f0.2g0.5NW -JM16.0 -R-74/-71/-38.5/-36 -X2 -Y6 -P -V > test.ps
```

Note que para el formato de la longitud, para cambiar de 286 este a -74 oeste por ejemplo, hay que cambiar los **gmtdefaults** para el formato de los gráficos:

```
gmtset PLOT_DEGREE_FORMAT D
psbasemap -Ba1f0.5SEWN -JM16.0 -R-74/-71/-38.5/-36 -X2 -Y6 -P -V > test.ps
```

También, se puede cambiar la proyección del mapa. Existen varias proyecciones (ver el manual). Por ejemplo, si queremos usar una proyección de Lambert (equal area), usamos **-JA** con el polo de la proyección, en nuestro caso Concepción. Note que el siguiente ejemplo es para el mundo entero, ver la región asociada con el **-R**:

```
psbasemap -Bg30/g15 -JA-73/-36.5/16c -R0/360/-90/90 -X2 -Y6 -P -V > test.ps
```

7.3 pscoast

Ahora vamos a agregar la costa al mapa con **pscoast**, que grafica costas, fronteras y rios. Empezamos con **psbasemap**, para generar el marco, y despues agregamos la costa. Para esto necesitamos dos líneas de código, así que en la primera línea usamos "> test.ps" para generar el archivo .ps, y en la segunda usamos "» test.ps" para agregar más información al .ps que ya existe.
El siguiente ejemplo grafica una projección Lambert de la Tierra, con continentes **-G** en color 215 gris y el mar **-S** en color 255 blanco. Incluye las fronteras nacionales **-N1** con ancho de lápiz de 0.25 puntos, color 128. La resolución de la costa es intermedia **-Di**:

```
psbasemap -Bg30/g15 -JA-73/-36.5/16c -R0/360/-90/90 -X2 -Y6 -P -V -K > test.ps
pscoast -Bg30/g15 -JA-73/-36.5/16c -R0/360/-90/90 -P -V -Di -N1/0.25p/128 -G215 -S255 -O >> test.ps
```

Podemos poner otro circulo al lado del primero para mostrar la Tierra proyectada en su polo opuesto, note la diferencia en el polo de proyección, y el tamaño del mapa entre las -JA's. Note también el -X y -Y en la tercera línea:

```
psbasemap -Bg30/g15 -JA-73/-36.5/16c -R0/360/-90/90 -X2 -Y6 -P -V -K > test.ps
pscoast -Bg30/g15 -JA-73/-36.5/16c -R0/360/-90/90 -P -V -Di -N1/0.25p/128 -G215 -S255 -O -K >> test.ps
pscoast -Bg30/g15 -JA107/53.5/6c -R0/360/-90/90 -X12 -Y12 -P -V -Di -N1/0.25p/128 -G215 -S255 -O  >> test.ps
```

El ejemplo abajo es para la región Bío Bío, con alta resolución de la costa **-Dh**, la costa dibujada como una línea roja **-W1.0p/255/0/0**, fronteras nacionales en gris **-N1/0.25p/128**, todos ríos en el dataset en azul **-Ia/0.25p/0/0/255**, el continente en verde **-G0/255/0** y el mar en azul **-S0/0/255** (*Recuerda que el \ en pscoast es para indicar que el comando completo es en una línea, por lo tanto, al escribirlo debes omitir el* \):

```
psbasemap -Ba1f0.5SEWN -JM16.0 -R-74/-71/-38.5/-36 -X2 -Y6 -P -V -K > test.ps
pscoast -Ba1f0.5SEWN -JM16.0 -R-74/-71/-38.5/-36 -P -V -Dh -W1.0p/255/0/0 \
-N1/0.25p/128 -Ia/0.25p/0/0/255 -G0/255/0 -S0/0/255 -O >> test.ps
```

7.4 Topografía

topo_6.2.img es la topografía global, con una resolución de 2 minutos. En GMT trabajamos usualmente con grillas (grids) de datos. Para convertir este .img a una .grd, usamos el comando **img2grd**

```
img2grd  topo_6.2.img -Gtopo.grd -R-100/-30/-60/20  -m2.0 -T1 -N1 -V
```

Esto va a crear una grilla **topo.grd** de longitud, latitud, altitud en la región definida por **-R**, con cada pixel de 2 minutos de longitud **-m2.0**. Para ver de que consiste la .grd, podemos usar el comando **grd2xyz**:

```
grd2xyz topo.grd
```

Ahora, hagamos un basemap y una imagen de la .grd (grdimage) sobre él. Este comando toma una grilla, una paleta de colores, y llena los puntos en el mapa con los colores definidos en la paleta (*archivo .cpt*). Note que en el comando que sigue, hay que poner el path de su archivo GMT_globe.cpt, y no el mio!

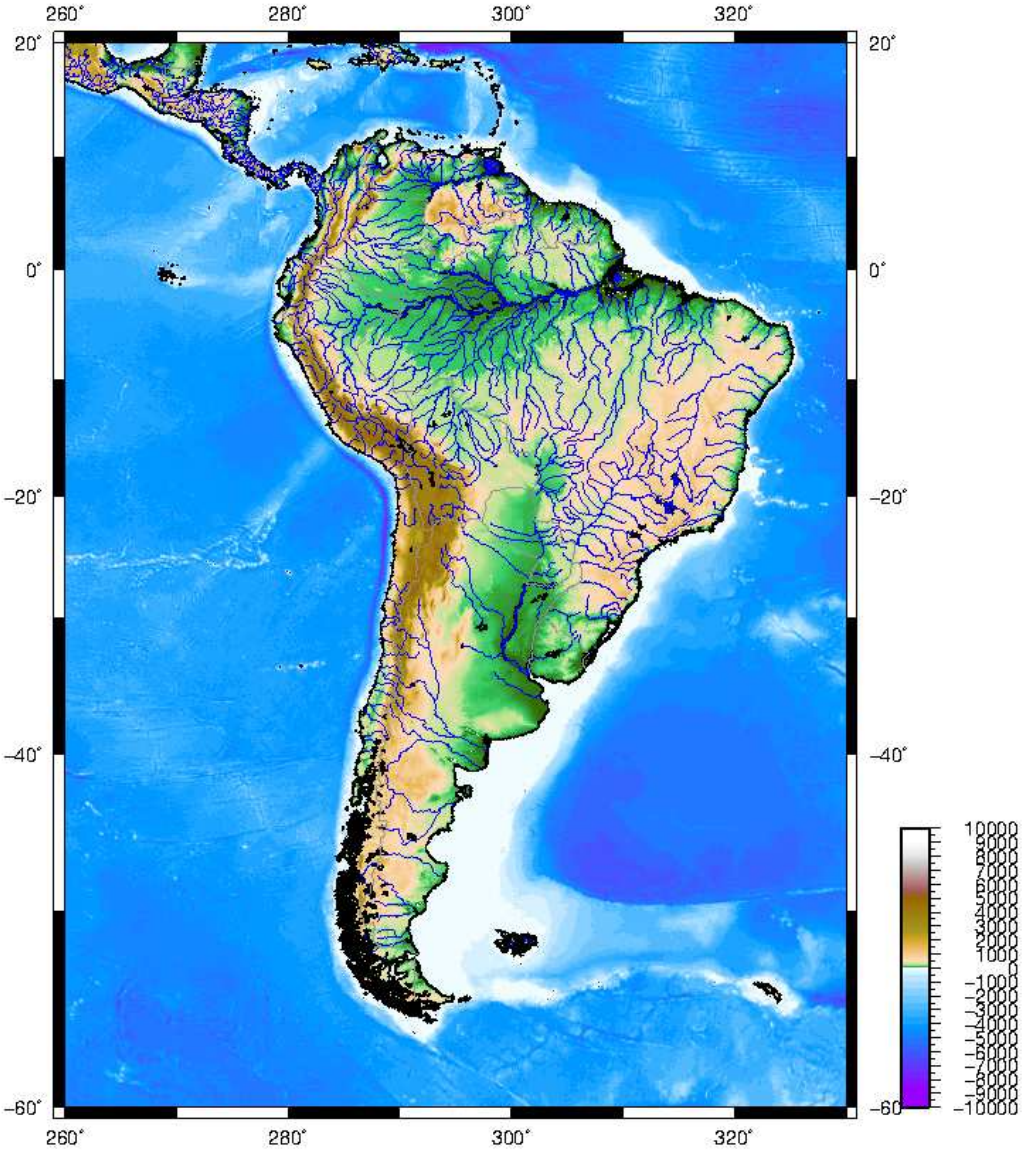
```
psbasemap -Ba20f10SEWN -JM14.0 -R-100/-30/-60/20 -X2 -Y6 -P -V -K > test.ps
grdimage topo.grd -C/home/matt/GMT/GMT4.5.2/share/cpt/GMT_globe.cpt \
-Ba20f10SEWN -JM14.0 -R-100/-30/-60/20 -P -V -O -K >> test.ps
```

Note que estoy usando **-K** aquí, asi que voy a poner más informaición al gráfico antes de cerrarlo, añadamos la costa (*note, no definimos colores aqui con -G y -S, porque ya tenemos los colores de la topografía puestos*):

```
pscoast -Ba20f10SEWN -JM14.0 -R-100/-30/-60/20 -P -V -Dh -W1.0p/0 -N1/0.25p/128 \
-Ia/0.25p/0/0/255  -O -K >> test.ps
```

Por último, ponemos una escala al lado del gráfico con **psscale**. Con este comando, no usamos la opción **-K**, asi que terminamos el gráfico y podemos verlo.

```
psscale  -D15c/2.5c/5c/0.5c -C/home/matt/GMT/GMT4.5.2/share/cpt/GMT_globe.cpt \
-Ba1000f500 -O >> test.ps
```


260°
280°
300°
320°
20°
0°
−20°
−40°
−60°
10000
100
−100
−10000

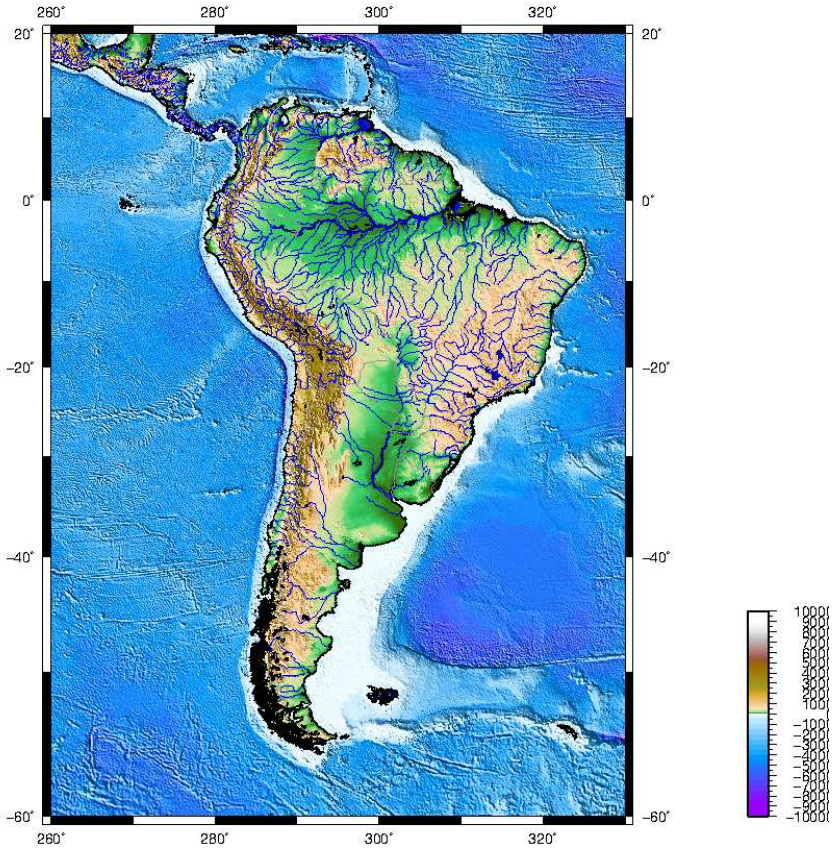
7.5 Iluminación topográfica

Podemos iluminar la topografía de un cierto acimut con el comando **grdgradient**:

```
grdgradient topo.grd -Gtopo.int -A225 -Nt -M
```

Este genera un archivo de intensidad de la pendiente de la grilla, es decir, la intensidad de la sombra. Ahora, podemos repasar los comandos anteriores, con un -Itopo.int en el comando grdimage para iluminar la topografía:

```
psbasemap -Ba20f10SEWN -JM14.0 -R-100/-30/-60/20 -X2 -Y6 -P -V -K > test.ps
grdimage topo.grd -C/home/matt/GMT/GMT4.5.2/share/cpt/GMT_globe.cpt \
-Ba20f10SEWN -JM14.0 -Itopo.int -R-100/-30/-60/20 -P -V -O -K >> test.ps
pscoast -Ba20f10SEWN -JM14.0 -R-100/-30/-60/20 -P -V -Dh -W1.0p/0 \
-N1/0.25p/128 -Ia/0.25p/0/0/255  -O -K >> test.ps
psscale  -D15c/2.5c/5c/0.5c -C/home/matt/GMT/GMT4.5.2/share/cpt/GMT_globe.cpt \
-Ba1000f500 -O >> test.ps
```



8 Scripts y GMT

8.1 Script para un mapa en GMT

Espero que hasta el momento hallan notado algunas cosas :

- GMT es básicamente una serie de comandos bash en un terminal
- Cuando se hace una modificación pequeña a un mapa, es molesto correr todos los comandos de nuevo
- Cuando esta modificación cambia un parámetro que esta en todos los comandos usados, esta modificación tendrá efecto en todas las líneas de comando que lleven este parametro

Por eso existen scripts para generar los mapas. Bajen el script siguiente ejemplo_mapa1.sh
Este script define los parámetros usados para generar el mapa, y después corre la serie de comandos en GMT para ejecutar el dibujo. El script esta abajo, si lo abres con un editor de texto como gedit se separa en parámetros, valores, comentarios y comandos.

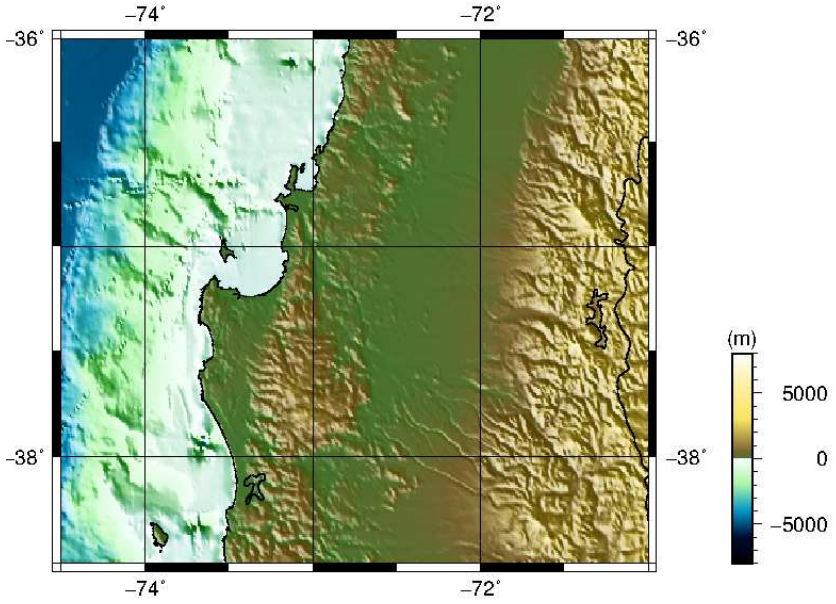
Note los siguientes parámetros :

- **region** - define el nombre del región. Note que el gráfico generado (output file) se llama "${region}.ps"
- **bounds** - define la latitud y longitud de la región
- **xshift**, **yshift** - mueven el mapa horizontal y verticalmente, respectivamente
- **projection** - en este caso uso Mercator de 14 cm
- **palette**, **topography** - hay que cambiar estas variables para dar el camino hacia sus archivos
- **ticks** - El mapa tiene lados con números cada 2 grados, barras cada 0.5 grados, y líneas cada 1 grado
- **illaz** - azimut de la iluminación

También hay parámetros para la costa, su resolución, las fronteras graficadas, y los rios. Note que mi parámetro para los rios (#rivers="-I1/1.00p/0/0/255") tiene un # antes, significando que es un comentario nada más. Con eso, no voy a graficar los rios, pero si quiero graficarlos, simplemente tengo que sacar el "#" para que la línea sea un comando y no un comentario.

```
#!/bin/bash

region="ejemplo_mapa1"
limites="-74.5/-71/-38.5/-36" #Limites de la region a graficar

#Origen (posicion del grafico en la hoja):
    xshift="2.0"
    yshift="6.0"

proyeccion="M14.0"               #Tipo de proyeccion
paleta="/usr/local/GMT4.5.8/share/cpt/GMT_relief.cpt"   #Paleta de colores a usar
topografia="/home/carlos/bin/w100s10.Bathymetry.srtm.grd" #Archivo de topografia
ticks="a2f0.5g1SEWN"  # Describiendo el marco del mapa
illaz="225"     # Direccion del azimut de iluminacion
portrait="-P"    # Si quiero landscape en vez de portrait, dejo esta opcion en blanco
verbose="-V"    # Dejo esta opcion en blanco si no quiero mostrar el "blah blah" de
                # los comandos en la terminal

#Lineas de costa e informacion de los limites fronterizos.
#(Para mas detalles, escribe en la terminal: man pscoast)
linea_costa="1.00p/0/0/0"
resolucion="f"    # Resolucion de las lineas de costa. Puedes elegir:
                # (f)ull, (h)igh, (i)ntermediate, (l)ow, y (c)rude
bordes="-N1/1.00p/0/0/0"  #Limites politicos de los paises

#Tambien se puede aÃśadir:
#rios="-I1/1.00p/0/0/255"

#agua="-S0/0/255" #Esto va a graficar de un solo color todo lo que son masas de agua
#tierra="-G0/255/0" #Esto va a graficar de un solo color todo lo que es tierra

##############################################################################################

# Archivo de salida:
psfile="${region}.ps"

# Construyendo el marco del mapa:
psbasemap -B${ticks} -J${proyeccion} -R${limites} -X${xshift} -Y${yshift} ${portrait} ${verbose} \
-K > ${psfile}

#Cortando la grilla original a los limites establecidos.
#grdcut archivo_entrada.grd -Garchivo_salida.grd -Roeste/este/sur/norte [-V]
grdcut ${topografia} -G${region}.grd -R${limites} -V

#Iluminando la grilla topografica generada en el grdcut desde un azimut especifico para producir
#un archivo de intensidad (.int) de iluminacion:
grdgradient ${region}.grd -G${region}.int -A${illaz} -Nt -M

# Graficando la topografia de la region seleccionada con su iluminacion:
grdimage ${region}.grd -C${paleta} -I${region}.int -B${ticks} -J${proyeccion} -R${limites} \
${portrait} ${verbose} -O -K >> ${psfile}

#Construyendo la escala de colores a partir de la paleta seleccionada:
psscale  -D16c/2.5c/5c/0.5c -C${paleta} -Ba5000f1000/a5000f1000:"(m)": -O -K >> ${psfile}

#Graficando las costas y limites del mapa:
pscoast -B${ticks} -J${proyeccion} -R${limites} ${portrait} ${verbose} -D${resolucion} \
-W${linea_costa} ${bordes} ${rios} ${agua} ${tierra} -O  >> ${psfile}
```

Si ustedes corren este script, con los párametros de la paleta y la topografía cambiados a sus respectivas rutas en su pc, deben generarla siguiente imagen:



Antes de salir de esta sección, revisa que entienden lo que hace el script y que pueden modificar el script para hacer lo siguiente :

1. Cambiar el nombre de la imagen generada a bio_bio.ps
2. Cambiar la región graficada a una que muestre el altiplano de sudamerica llamado altiplano.ps
3. Cambiar la paleta. Prueben globe, ocean, sealand, topo. Noten que algunas de esas son mejores que otros!
4. Cambiar los ejes: números y barras cada 1 grado, y líneas cada 0.2 grado
5. Cambiar el azimut de la iluminación para que sea en la dirección opuesta del ejemplo mostrado aquí
6. Cambiar la resolución de la costa a full
7. Poner rios
8. Sacar del gráfico la escala de la topografía con un gato (#)

8.2 Contornos en GMT

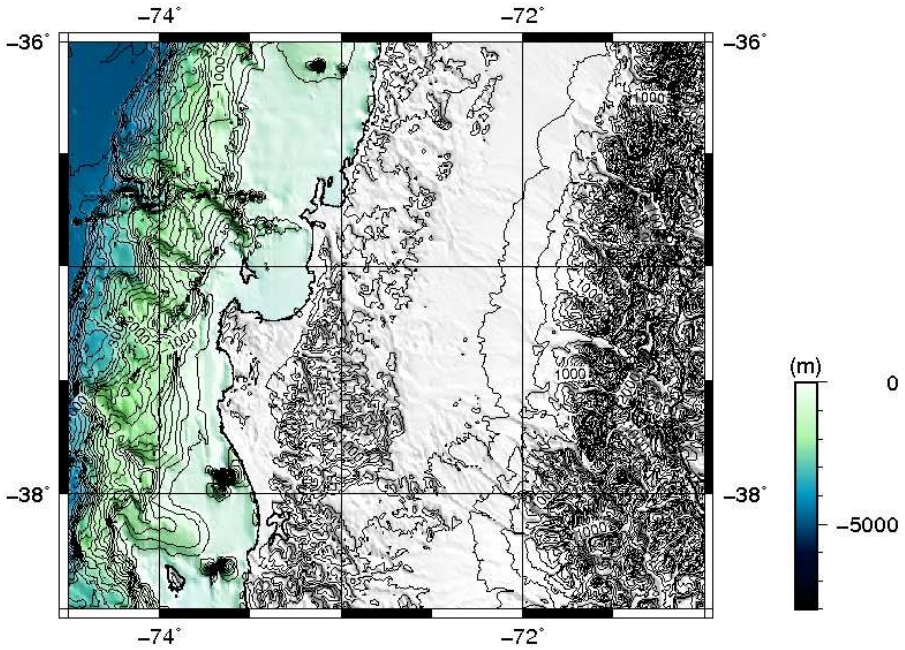
El archivo .grd para la topografía es una grilla, por lo tanto podemos graficar sus contornos usando el comando **grdcontour**. Por ejemplo, puedo poner:

```
grdcontour ${region}.grd -B${ticks} -J${projection} -C200 -A1000+k0/0/0 \
-R${bounds} ${portrait} ${verbose} -W0.1p/0/0/0 -Gd10c -O -K >> ${psfile}
```

dentro del script, entre grdimage y psscale. A continuación se detallan algunos parámetros del comando **grdcontour** y su función.

Parámetro	Función
-C200	contornos cada 200 m
-A1000+k0/0/0	anotación cada 1000 m, color negro
-W0.1p/0/0/0	tamaño y color del lápiz de contornos
-Gd10c	10 cm de anotación entre contornos

El script para generar el siguiente gráfico es **ejemplo_mapa2.sh**, es bastante similar al primero, solamente cambia la paleta e incorpora contornos.



8.3 Personalización del mapa con condicionales

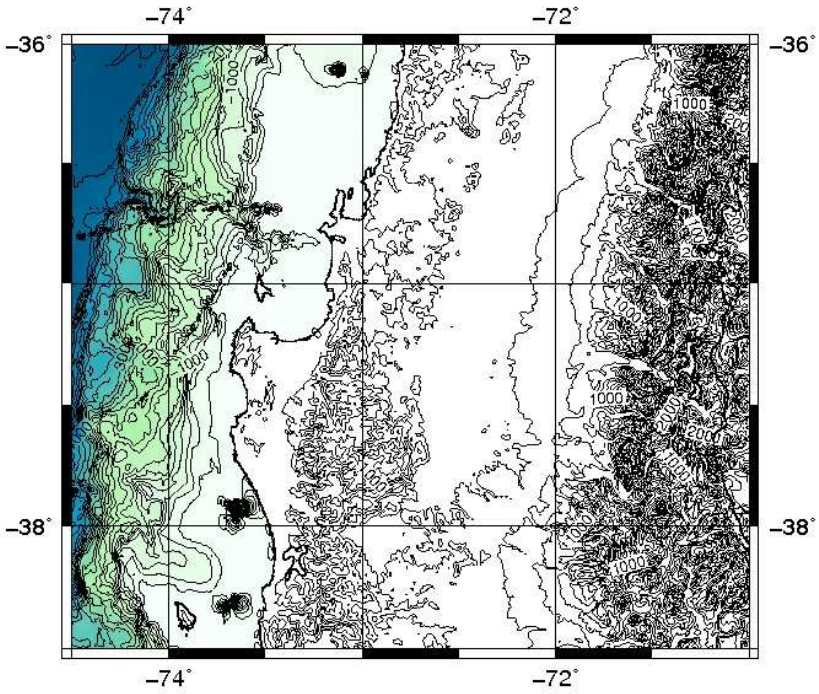
El mapa de arriba es bastante feo. ¿Será mejor si elimino la iluminación de la topografía ?
Se puede hacer eso borrando la opción -I en el comando grdimage, pero con eso hay que recordar como ponerlo de nuevo. Es mejor definir unas variables al inicio del script para decir exactamente que queremos, por ejemplo:
si quiero algo (1)
no quiero algo (0)
Donde "algo" puede ser topografía , iluminación , escala , etc.

```
topo_option=1
illumination_option=0
contour_option=1
topo_scale_option=0
```

Después, en la sección confección del mapa (cuerpo del script), solamente grafica de acuerdo a las opciones declaradas al inicio de script , donde el condicional más apropiado para esta selección es **if**. Note además que las opciones que se declaran al inicio del script (1 o 0) tambien las puede declarar en forma general, usando el condicional *case*. Esto le permitirá declarar las opciones de gráfica como argumentos de entrada , facilitando la tarea de modificar el script poniendo 1 o 0 según lo que desee en cada ocasión.

```
# Aplicando los condicionales:
if [ "${opcion_topografia}" = 1 ]; then
# Cortando la grilla original a los limites establecidos.
# grdcut archivo_entrada.grd -Garchivo_salida.grd -Roeste/este/sur/norte [-V]
grdcut ${topografia} -G${region}.grd -R${limites} -V


if [ "${opcion_iluminacion}" = 1 ]; then
# Iluminando la grilla topografica generada en el grdcut desde un azimut
# especifico para producir un archivo de intensidad (.int) de iluminacion:
grdgradient ${region}.grd -G${region}.int -A${illaz} -Nt -M

# Graficando la topografia de la region seleccionada con su iluminacion:
grdimage ${region}.grd -C${paleta} -I${region}.int -B${ticks} -J${proyeccion} \
-R${limites} ${portrait} ${verbose} -O -K >> ${psfile}

else
# Graficando la topografia de la region seleccionada sin iluminacion:
grdimage ${region}.grd -C${paleta} -B${ticks} -J${proyeccion} -R${limites} \
${portrait} ${verbose} -O -K >> ${psfile}
fi


if [ "${opcion_contorno}" = 1 ]; then
# Graficando los contornos de la grilla:
grdcontour ${region}.grd -B${ticks} -J${proyeccion} -C200 -A1000+k0/0/0 \
-R${limites} ${portrait} ${verbose} -W0.1p/0/0/0 -Gd10c -O -K >> ${psfile}
fi


if [ "${opcion_escala_topo}" = 1 ]; then
# Construyendo la escala de colores a partir de la paleta seleccionada:
psscale  -D16c/2.5c/5c/0.5c -C${paleta} -Ba5000f1000/a5000f1000:"(m)": -O -K \
>> ${psfile}
fi

fi
```

El script entero es **ejemplo_mapa3.sh**. Pruebe este script con varias combinaciones de 1s y 0s en las opciones iniciales para la topografía, y asegurese de entender el gráfico que resulta.
En este caso (topo_option=1, illumination_option=0, contour_option=1, topo_scale_option=0 ,¿ luce mejor ?



8.4 Conclusión

En esta sección vemos que con un script podemos facilmente generar mapas de topografía con opciones de gráfico (región, colores a usar, contornos, ejes, etc.) que son fácil de cambiar.
En la próxima sección trabajamos con el mismo script, y veremos los comandos para poner símbolos en los mapas.

8.5 Preguntas

1. ¿ Como confeccionaría un script que cumpla la misma función anterior usando el condicional **case** ?

9 Simbolos y texto en GMT

9.1 Simbolos y texto: psxy y pstext

Se usa **psxy** para graficar símbolos en un mapa de 2D, y **pstext** para insertar texto.
Los ejemplos abajo son para graficar símbolos sobre un mapa que ya fue creado, tal que las líneas de código dadas acá no son scripts enteros, hay que agregarlas al script ya creado en la sección 8. Note que cuando se pone más código en uno de estos scripts, hay que verificar las opciónes **-K** y **-O** dentro de los comandos.

- **-K** significa que más postscript sera agregado al .ps más tarde, o si no, se termina el .ps. Entonces **-K** debe estar en todos los comandos usados salvo el último
- **-O** significa que lo que grafica el comando va sobre el .ps ya creado. Si no, un nuevo gráfico sera creado. Entonces **-O** debe estar en todos los comandos usados salvo el primero

9.2 Agregando símbolo al mapa

Si quiero poner un símbolo a un mapa, uso **psxy**. Por ejemplo, para poner un símbolo que identifique la ciudad de Concepción en un mapa, puedo poner esta línea de código al final del script **ejemplo_mapa3.sh**.

```
echo -73.03 -36.83 | psxy -J${projection} -R${bounds} ${verbose} \
-Ss1.0c -W0.2c/255/0/200 -G255/175/0 -O >> ${psfile}
```

Esto toma un punto de longitud y latitud (en nuestro caso la UdeC) y lo grafica como un cuadrado de 1.0 centímetro de tamaño **-Ss1.0c**, de un color naranjo **-G255/175/0**, con bordes de 0.2 centímetros de ancho , y de color púrpura **-W0.2c/255/0/0**.
Hay varios símbolos disponibles, por ejemplo círculo **-Sc**, cruz **-Sx**, rombo **-Sd**, hexágono **-Sh**, estrella **-Sa**, triángulo **-St**, triángulo invertido **-Si** entre otros. Juege con ellos, y también pruebe modificar los colores y el borde para que se vea en otra.

9.3 Agregando texto

Para poner texto a un gráfico, por ejemplo, para poner "Concepción" al lado del símbolo de la ciudad, usamos **pstext**.

```
echo "-72.8 -36.83 12 0 4 LM Concepci\363n" | pstext -J${projection} -R${bounds} \
${verbose} -G0 -N -W255 -O >> ${psfile}
```

Esto requiere un poco de explicación :

- La longitud y latitud dónde se requiere empezar el texto estan dadas por -72.8 -36.83
- El 12 significa el tamaño de las letras
- El 4 es el font Times-Roman. Para una lista de los fonts disponibles, escribe pstext -L en un terminal

- El LM significa que el punto donde queremos el text dado por -72.8 -36.83 esta al lado izquierdo (L), al medio (M), del bloque de texto. Las opciones son left/centre/right (LCR) y bottom/middle/top (BMT)

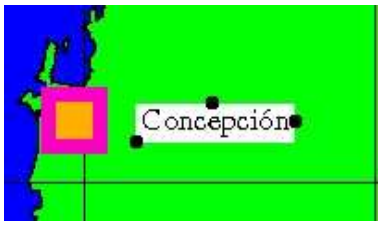


Figure 5: Los tres puntos en el gráfico representan LB, CT, RM

Nota:

- Para poner caracteres especiales, hay que usar código octal
- El default es ISOLatin+, asi que \363 representa la letra "'o", \341 representa la letra "'a", etc
- **-G0** significa que el color del texto es negro
- **-W255** pone un rectángulo de color blanco bajo el texto
- **-N** significa que el texto puede existir fuera de los bordes del gráfico, si no, va a estar cortado por el borde. Revisa eso cambiando la posición del texto

9.4 Agregando líneas

Usamos **psxy** de nuevo para poner líneas entre dos o más puntos. **-W** selecciona el ancho de la línea y su color, aquí 2.0 puntos, **-A** significa que la línea es recta, y no un arco de un gran círculo entre los puntos. El ejemplo abajo va a dibujar una línea negra debajo del texto en la sección 9.2.

```
psxy -J${projection} -R${bounds} ${verbose} -W2.0p/0/0/0 -A -O << END >> ${psfile}
-72.8 -36.9
-72.3 -36.9
END
```

También podemos tener un archivo con los puntos donde queremos nuestra línea y graficar eso con **psxy**.
Por ejemplo, el archivo **biobio_region.xy** contiene la información sobre las fronteras de la región del Bío Bío. Podemos graficarlo con:

```
psxy biobio_region.xy -J${projection} -R${bounds} ${verbose} \
-m -W1.0p/0/0/0t5_2:2 -A -O >> ${psfile}
```

Note que el archivo **biobio_region.xy** expresa dos líneas, una para la frontera al norte, después una pausa de 1 línea, después la frontera del sur. Por eso uso la opción **-m** para un "multiple segment file", o un archivo de multiples segmentos. Note también que esta línea tiene textura t5_2:2. Eso significa que tenemos una línea con 5 puntos de segmento y 2 puntos de espacio, con

un desplazamiento de 2 puntos de su inicio. Esto no es muy simple de explicar, mejor prueben diferentes combinaciones, como t5_2:0, t2_5:2, e incluso t2_2_8_2:1. Para más información, lean la página *http://gmt.soest.hawaii.edu/gmt/doc/gmt/html/GMT_Docs/node65.html* (SPECIFYING PEN ATTRIBUTES)

9.5 Agregando vectores

Para poner un vector al gráfico, usamos:

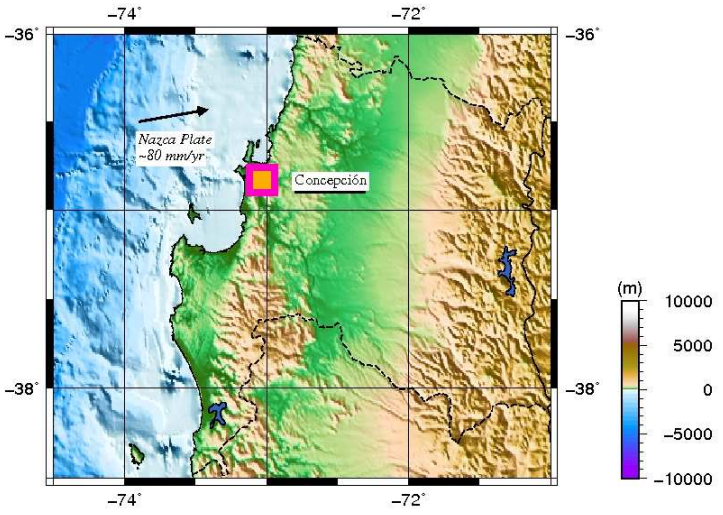
```
echo "-73.9 -36.5 80 2" | psxy -J${projection} -R${bounds} \
${verbose} -SV0.075c/0.3c/0.15c -G0 -O >> ${psfile}
```

Los números son la longitud y latitud del vector, el azimut del vector, y su largo en centímetros. Aquí estoy usando **psxy** con la opción **-SV** para un vector. Los números asociados con el **-SV** son el ancho del vector 0.075 centímetros, el ancho de la flecha 0.3 centímetros, y el largo de la flecha 0.15 centímetros. Prueben eso con diferentes valores. La opción **-G** da el color del vector, en nuestro caso, negro.
Esta flecha representa la velocidad de la placa Nazca, podemos agregar texto a su lado para especificar eso:

```
echo "-73.9 -36.6 12 0 6 LM Nazca Plate" | pstext -J${projection} -R${bounds} \
$verbose} -G0 -N -W255 -O -K >> ${psfile}
echo "-73.9 -36.69 12 0 6 LM ~80 mm/yr" | pstext -J${projection} -R${bounds}  \
${verbose} -G0 -N -W255 -O >> ${psfile}
```

9.6 El script

El script que hace esto es **ejemplo_mapa4.sh**. Note que este script también tiene una opción para el color de los lagos. Es posible limpiar el script con condicionales, como lo que se hizo con la topografía.
El script debe producir un gráfico como este :



El script completo :

```
#!/bin/bash

# Estableciendo las opciones, donde:
# (1) Quiero que aparezca
# (0) No quiero que aparezca

opcion_topografia=1
opcion_iluminacion=1
opcion_contorno=0
opcion_escala_topo=1

#############################################################################################

region="ejemplo_mapa4"
limites="-74.5/-71/-38.5/-36"  # Limites de la region a graficar

# Origen (posicion del grafico en la hoja):
xshift="2.0"
yshift="6.0"

proyeccion="M14.0"                # Tipo de proyeccion
paleta="/usr/local/GMT4.5.8/share/cpt/GMT_globe.cpt"   #Paleta de colores a usar
topografia="/home/carlos/bin/w100s10.Bathymetry.srtm.grd"    #Archivo de topografia
ticks="a2f0.5g1SEWN"
illaz="225"     # Direccion del azimut de iluminacion
portrait="-P" # Si quiero landscape en vez de portrait, dejo esta opcion en blanco
verbose="-V" # Dejo esta opcion en blanco si no quiero mostrar el "blah blah" de
               # los comandos en la terminal

#Lineas de costa e informacion de los limites fronterizos.
#(Para mas detalles, escribe en la terminal: man pscoast)
linea_costa="1.00p/0/0/0"
resolucion="f"  # Resolucion de las lineas de costa. Puedes elegir:
               # (f)ull, (h)igh, (i)ntermediate, (l)ow, y (c)rude
bordes="-N1/1.00p/0/0/0"

#Tambien se puede aÃśadir:
#rios="-I1/1.00p/0/0/255"
agua="-S0/0/255"  #Esto va a graficar de un solo color todo lo que son masas de agua
tierra="-G0/255/0" #Esto va a graficar de un solo color todo lo que es tierra
lagos="-C50/100/200 -A0/2/4"

#############################################################################################

# Archivo de salida:
psfile="${region}.ps"

# Construyendo el marco del mapa:
psbasemap -B${ticks} -J${proyeccion} -R${limites} -X${xshift} -Y${yshift} ${portrait} \
${verbose} -K > ${psfile}

# Aplicando los condicionales:
if [ "${opcion_topografia}" = 1 ]; then
# Cortando la grilla original a los limites establecidos.
# grdcut archivo_entrada.grd -Garchivo_salida.grd -Roeste/este/sur/norte [-V]
grdcut ${topografia} -G${region}.grd -R${limites} -V
```

```
if [ "${opcion_iluminacion}" = 1 ]; then
# Iluminando la grilla topografica generada en el grdcut desde un azimut
# especifico para producir un archivo de intensidad (.int) de iluminacion:
grdgradient ${region}.grd -G${region}.int -A${illaz} -Nt -M
# Graficando la topografia de la region seleccionada con su iluminacion:
grdimage ${region}.grd -C${paleta} -I${region}.int -B${ticks} -J${proyeccion} -R${limites} \
${portrait} ${verbose} -O -K >> ${psfile}

else
# Graficando la topografia de la region seleccionada sin iluminacion:
grdimage ${region}.grd -C${paleta} -B${ticks} -J${proyeccion} -R${limites} ${portrait} \
${verbose} -O -K >> ${psfile}
fi

if [ "${opcion_contorno}" = 1 ]; then
# Graficando los contornos de la grilla:
grdcontour ${region}.grd -B${ticks} -J${proyeccion} -C200 -A1000+k0/0/0 -R${limites} ${portrait} \
${verbose} -W0.1p/0/0/0 -Gd10c -O -K >> ${psfile}
fi

if [ "${opcion_escala_topo}" = 1 ]; then
# Construyendo la escala de colores a partir de la paleta seleccionada:
psscale  -D16c/2.5c/5c/0.5c -C${paleta} -Ba5000f1000/a5000f1000:"(m)": -O -K >> ${psfile}
fi

# Graficando las costas y limites del mapa (para graficar las lineas de costa marinas):
pscoast -B${ticks} -J${proyeccion} -R${limites} ${portrait} ${verbose} -D${resolucion} \
-W${linea_costa} ${bordes} ${rios} -O -K >> ${psfile}
# Graficando las costas y limites del mapa (para incluir los lagos):
pscoast -B${ticks} -J${proyeccion} -R${limites} ${portrait} ${verbose} -D${resolucion} \
-W${linea_costa} ${bordes} ${lagos} -O -K >> ${psfile}

else

# Graficando las costas y limites del mapa (incluyendo un color diferente
# para indicar la tierra y el agua):
pscoast -B${ticks} -J${proyeccion} -R${limites} ${portrait} ${verbose} -D${resolucion} \
-W${linea_costa} ${bordes} ${rios} ${agua} ${tierra} -O -K >> ${psfile}
#Graficando las costas y limites del mapa (para incluir los lagos):
pscoast -B${ticks} -J${proyeccion} -R${limites} ${portrait} ${verbose} ${lagos} -D${resolucion} \
-W${linea_costa} ${bordes} ${rios} -O -K >> ${psfile}
fi

##############################################################################################

# Ahora vamos a poner unos simbolos

# Poner un cuadrado para la ciudad de Concepcion:
echo "-73.03 -36.83" | psxy -J${proyeccion} -R${limites} ${verbose} -Ss1.0c -W0.2c/255/0/200 \
-G255/175/0 -O -K >> ${psfile}
# Poner texto que dice Concepcion:
echo "-72.8 -36.83 12 0 4 LM Concepci\363n" | pstext -J${proyeccion} -R${limites} ${verbose} \
-G0 -N -W255 -O -K >> ${psfile}
# Poner una linea bajo del texto:
psxy -J${proyeccion} -R${limites} ${verbose} -W2.00p/0/0/0 -A -O -K << END >> ${psfile}
-72.8 -36.9
```

```
-72.3 -36.9
END
# Poner la frontera de la region biobio:
psxy biobio_region.xy -J${proyeccion} -R${limites} ${verbose} -m -W1.0p/0/0/0t5_2:1 -A -O -K \
>> ${psfile}

# Poner un vector para la velocidad de la placa Nazca:
echo "-73.9 -36.5 80 2" | psxy -J${proyeccion} -R${limites} ${verbose} -SV0.075c/0.3c/0.15c -G0 \
-O -K >> ${psfile}
# Y texto:
echo "-73.9 -36.6 12 0 6 LM Nazca Plate" | pstext -J${proyeccion} -R${limites} $verbose} -G0 -N \
-W255 -O -K >> ${psfile}
echo "-73.9 -36.69 12 0 6 LM ~80 mm/yr" | pstext -J${proyeccion} -R${limites} ${verbose} -G0 -N \
-W255 -O >> ${psfile}
```

10 Simbolos

10.1 Símbolos desde un archivo de texto : volcanes

El script usado en esta clase es **ejemplo_mapa5.sh** que es una modificación del script anterior.
Podemos usar **psxy** para graficar símbolos desde un archivo de datos, como **southern_Chile_and_Argentina.txt**. De la misma manera que antes, el comando psxy en GMT toma coordenadas de longitud, latitud y grafica un símbolo en esta posición, también podemos mandar varias coordenadas al comando al mismo tiempo con un pipe:

```
more southern_Chile_and_Argentina.txt  | awk '{print $6, $5}' | grep -v lon \
|psxy -J${projection} -R${bounds} $verbose} -St0.5c -G255/0/0 -W2 -O -K >> ${psfile}
```

Aquí tomamos el archivo de datos, aislamos la longitud y latitud de los volcanes usando una combinación de awk y grep, y los graficamos con triángulos de 0.5 cm **-St0.5c**, color rojo **-G255/0/0**, con un borde negro de dos puntos **-W2**. Esta línea de código va dentro de un script, como antes.

10.2 Símbolos con diferentes tamaños y colores

Aquí trabajamos con el archivo de datos **global_seismicity_feb27-apr19_2010.txt**, de la sismicidad global de las semanas siguientes después del terremoto en Chile 2010. Podemos aislar las columnas de longitud, latitud, profundidad y magnitud, si existe una magnitud, usando el siguiente comando :

```
more global_seismicity_feb27-apr19_2010.txt | sed 's/,//g' \
|awk '{if ( $10 > 0 ) print $5, $4, $6, $10}'
```

Si esto es combinado con psxy, la tercera columna da el color del símbolo, y la cuarta su tamaño. Por eso, necesitamos una paleta de colores para los terremotos a diferentes profundidades. Podemos generar una, o podemos modificar una. Por ejemplo, empezamos con **GMT_copper.cpt**, que da diferentes tonos de cobre entre valores de cero y uno (revisa el .cpt), y generamos una nueva paleta con el comando:

```
makecpt -C/home/matt/GMT/GMT4.5.2/share/cpt/GMT_copper.cpt -T0/100/10 -I > depths.cpt
```

La **-T** especifica la escala del .cpt (0 a 100) y el ancho de sus bandas (10). El **-I** especifica que tomamos los colores de GMT_copper.cpt al revés. Pruebe el comando sin el **-I** para ver la diferencia. Esta paleta de colores no es perfecta, pero se nota que la paleta original (GMT_copper.cpt) es muy simple, y es fácil generar otras paletas entre 0 y 1 con diferentes colores, y después modificar el rango con la opción **-T** en makecpt.
Ahora estamos listos para graficar terremotos con:

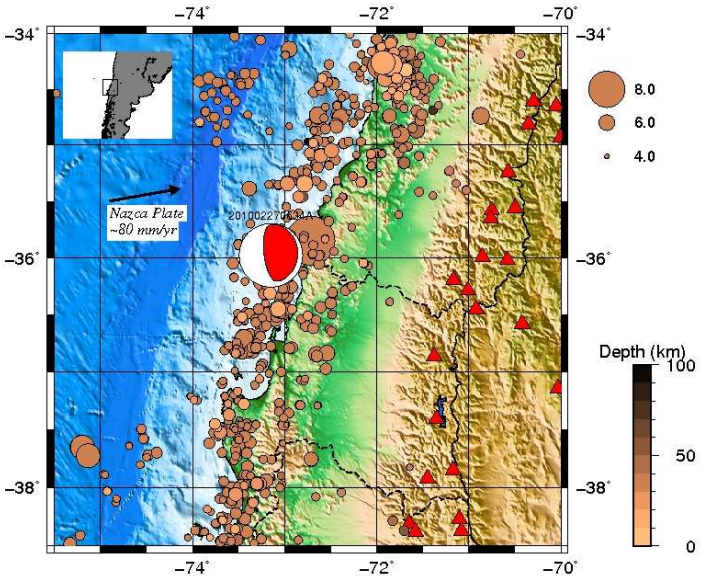
```
more global_seismicity_feb27-apr19_2010.txt | sed 's/,//g' | awk '{if ( $10 > 0 ) \
print $5, $4, $6, ($10*$10*$10/500)}' | sort -k3 -r | psxy -J${projection} \
-R${bounds} ${verbose} -Sc -Cdepths.cpt -W2 -O -K >> ${psfile}
```

Nota:

- *Se usa sort -k3 -r para ordenar los terremotos por su profundidad. Esto grafica los mas profundos primeros*
- *Note que este catalogo era preliminar, entonces las profundidades de los terremotos no son muy precisas*
- *El color del símbolo depende de su profundidad -Cdepths.cpt. Se puede graficar la escala con : psscale -D16c/2.5c/5c/0.5c -Cdepths.cpt -Ba50f10/a50f10:"Depth (km)": $verbose -O -K » $psfile*
- *Los símbolos van a ser círculos -**Sc**; si no especifico un tamaño aquí, va a tomar el tamaño de la cuarta columna de los datos. En nuestro caso usamos $10*$10*$10/500 para que un terremoto de magnitud 8 tenga un tamaño de ∼1 cm, y de magnitud 4 tenga tamaño ∼0.12 cm. Por supuesto, puedo poner una escala al lado del mapa con :*

```
echo "-69.5 -34.5 35 1.024" | psxy -J${projection} -R${bounds} ${verbose} /
-Sc -Cdepths.cpt -N -W2 -O -K >> ${psfile}
echo "-69.2 -34.5 12 0 1 LM 8.0" | pstext -J${projection} -R${bounds} ${verbose} /
-G0 -N -W255 -O -K >> ${psfile}
echo "-69.5 -34.8 35 .432" | psxy -J${projection} -R${bounds} ${verbose} -Sc /
-Cdepths.cpt -N -W2 -O -K >> ${psfile}
echo "-69.2 -34.8 12 0 1 LM 6.0" | pstext -J${projection} -R${bounds} ${verbose} /
-G0 -N -W255 -O -K >> ${psfile}
echo "-69.5 -35.1 35 .128" | psxy -J${projection} -R${bounds} ${verbose} -Sc /
-Cdepths.cpt -N -W2 -O -K >> ${psfile}
echo "-69.2 -35.1 12 0 1 LM 4.0" | pstext -J${projection} -R${bounds} ${verbose} /
-G0 -N -W255 -O -K >> ${psfile}
```

Donde se grafican: Símbolos de un cierto tamaño al lado del mapa, y la magnitud que coresponde.



El **-N** en el comando permite graficar cosas afuera del borde del mapa.
En conclusión, podemos tomar un set de datos y graficarlo con símbolos cuyos tamaños y colores dependen de los datos. En este caso los datos son tomados de:

http://www.iris.edu/SeismiQuery/sq-events.htm
Ahora, tomando los resultados de la sismicidad global durante esta época, usando el catálogo "PREFERRED", va a tener las profundidades y ubicaciones de los sismos con mayor precisión.
Este ejemplo usa sismicidad, pero se puede extender a cualquier set de datos que tengan latitud, longitud y ciertos valores que se necesitan graficar como símbolos.

10.3 Mecanismos focales

El comando **psmeca** es para graficar mecanismos focales de terremotos. Por ejemplo, podemos buscar el mecanismo focal del terremoto del 2010-Chile en el Harvard CMT catalog, y elegir que su formato este en *GMT psmeca input*. Para graficarlo, usamos el comando:

```
echo "-73.15 -35.98 23 1.04 -0.04 -1.00 0.30 -1.52 -0.12 29 X Y 201002270634A" \
|psmeca ${portrait} -J${projection} -R${bounds} ${verbose} -L -Sm1.0c -G255/0/0 \
-O -K >> ${psfile}
```

Note que si queremos graficar varios mecanismos focales, podemos guardar ellos en un archivo .txt y correr el comando:

```
more mecanismos.txt | psmeca ${portrait} -J${projection} -R${bounds} \
${verbose} -L -Sm1.0c -G255/0/0 -O -K >> ${psfile}
```

Esta sección es especifica a sismología, no a geofísica en general, así que voy a dejarlo hasta aquí. Los sismólogos entre ustedes pueden leer mas sobre los mecanismos focales y el manual para **psmeca** para más conocimiento.

10.4 Indent maps

Siempre, cuando se grafica una región, es necesario poner un mapa al lado con esta región marcada en un mapa general del país o continente. Para esto, podemos graficar la costa de una región diferente en la esquina de nuestro mapa:

```
bounds2=-90/-50/-50/-25
pscoast -JM3c -W1p -R${bounds2} -Y11.25 -X0.25 -Dh -G128 -S255 \
-O -K >> ${psfile}
```

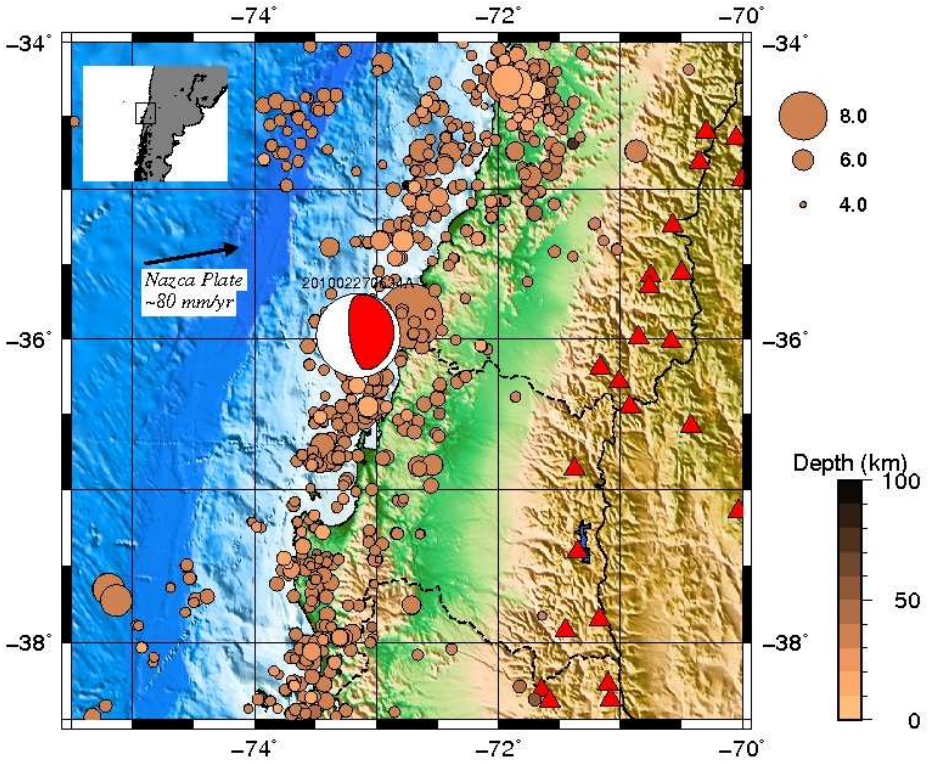
Note que el **-Y** y **-X** son elegidos para ubicar este mapa pequeño en la esquina.
Para marcar el borde de la región original, podemos usar psxy de nuevo :

```
echo ${bounds} | sed 's/\// /g' | awk '{printf"%s %s\n %s %s\n %s \
%s\n %s %s\n %s %s\n", $1, $3, $2, $3, $2, $4, $1, $4, $1, $3}'   \
| psxy -R${bounds2} -JM3c -A -W0.5p -O >> ${psfile}
```

Si no es obvio por que se usa un **printf**, pruebe los siguientes tres comandos en un terminal:

```
bounds="-75.5/-70/-38.5/-34"
echo ${bounds} | sed 's/\// /g'
echo ${bounds} | sed 's/\// /g' \
| awk '{printf"%s %s\n %s %s\n %s %s\n %s %s\n %s %s\n", \
$1, $3, $2, $3, $2, $4, $1, $4, $1, $3}'
```

Acá esta la imagen final usando el script generado en esta sección, llamado **ejemplo_mapa5.sh**. Espero que ahora ustedes puedan tomar un set de datos que consiste en latitud, longitud y unos valores, y representarlo gráficamente:



11 Grillas

En esta sección trabajamos con el set de datos **bouguer.xyz** que representa mediciones de terreno, en este caso, datos tomados de la anomalía de Bouguer de gravedad.
El archivo contiene tres columnas: longitud (x), latitud (y), y anomalía de Bouguer en mgal (z); y representa una serie de mediciones del Altiplano. Se puede usar sort para ver los límites de latitud y longitud que contienen los datos. Estos datos vienen de la Universidad Libre de Berlin.
Este set de datos no es continuo, es decir, las mediciones están tomadas donde existen caminos o aceso al Altiplano. Podemos usar el comando **surface** para generar una superficie, es decir, una grilla continua que pase por los puntos donde existen mediciones. Este comando requiere un factor de tensión entre 0 y 1 que impone restricciones sobre la curvatura de la superficie para evitar oscilaciones o falsos máximos/mínimos en la superficie, y un límite de convergencia para definir cuando la superficie esta lista, esto es, que tiene buen parecer a los datos. Probemos el siguiente comando en el terminal:

```
surface bouguer.xyz -Gbouguer.grd -R-71.5/-64/-29.6/-24.9 -I1m -T0.25 -C0.1 -V
```

Aquí tomamos el set de datos de gravedad bouguer.xyz y generamos una grilla (bouguer.grd) que pasa por sus puntos dentro de una cierta región que contiene los datos (-R), con una resolución de un minuto, es decir, $(1/60)^o$ para la grilla **-I1m**, un factor de tensión de 0.25 **-T0.25**, y un límite de convergencia de 0.1 mgal con **-C0.1**, esto significa que cuando la iteración para generar la grilla esta cambiando los puntos un valor menor que 0.1, se detiene la iteración y tenemos nuestro resultado final.
Con este comando hemos generado una grilla *bouguer.grd* que representa los datos. Se puede revisar los contenidos de cualquier grilla con el comando **grd2xyz** para ver en el terminal los puntos de la grilla. Revisen *bouguer.grd* usando este comando.
También queremos una paleta de colores con un rango suficiente para cubrir los valores de la grilla. Podemos crear una pero en este caso es más fácil usar una paleta ya creada para el gráfico. Usamos la paleta de colores **GMT_red2green.cpt** para mostrar eso. Si miran la paleta, se puede ver que la paleta varía de -1 (rojo) a +1 (verde), mientras que los datos tienen valores en los cientos de mgal, por lo cual usaramos el comando **grd2cpt** para generar una paleta apropriada para los datos.

```
grd2cpt bouguer.grd -C/usr/local/GMT4.5.8/share/cpt/GMT_red2green.cpt -T= > bouguer.cpt
```

Con el comando de arriba se toma una paleta de colores y se cambia su rango para que se aplique a los valores de una cierta grilla. Recuerden cambiar el camino .cpt a la ubicación de sus paletas. El **-T=** es una opción para que la paleta de colores generada *bouguer.cpt* sea simétrica, así la anomalía de Bouguer de valor cero será puesta en blanco, anomalías negativas en rojo y anomalías positivas en verde. Revisen la paleta y la grilla para los datos y noten que con esta paleta podemos graficar los datos con un rango de colores aceptables.
Ahora estamos listo para generar/modificar un script para graficar los datos. Descargen el script **ejemplo_bouguer.sh** que maneja el set de datos.
Este script es una modificación de los scripts anteriores. Empieza definiendo unas variables (hay que cambiar "palette" y "topography" para dar los caminos a sus archivos), después grafica el basemap y la topografía, con iluminación, usando psbasemap, grdgradient y grdimage. Los próximos comandos son los de **surface** y **grd2cpt** para definir la superficie que representa los datos y la paleta con que vamos a graficarla.
Después vienen las próximas dos líneas que usan el comando **grdsample**:

```
grdsample ${region}.int -Gtopo.int -R-71.5/-64/-29.6/-24.9 -I1m -V -F
grdsample bouguer.grd -Gbouguer2.grd -R-71.5/-64/-29.6/-24.9 -I1m -V -F
```

¿ Que estoy haciendo aqui? - Bueno, siempre la anomalía de gravedad está asociada con la topografía, recuerden los cursos de Geofísica de la Tierra Sólida o Geodesia. Aquellos que no han hecho estos cursos noten que la interpretación de la imagen no es necesaria en el curso de GMT, por los cual mostraré la anomalía de gravedad y la topografía en la misma imagen. Para esto, puedo usar el comando **grdsample** para forzar **-F** los puntos de la grilla de Bouguer *bouguer.grd* y la grilla de la iluminación topográfica **region.int** a exáctamente los mismos puntos, cada grilla en la misma región **-R** con la misma resolución de 1 minuto **-I**. Estos comandos crean las grillas *bouguer2.grd* y *topo.int* respectivamente. Ahora puedo graficar la grilla de Bouguer en colores de la paleta bouguer.cpt pero con iluminación asociada con la topografía:

```
grdimage bouguer2.grd -Cbouguer.cpt -B${ticks} -J${proyeccion} -Itopo.int -R${limites} \
${portrait} ${verbose} -O -K >> ${psfile}
```

Las cinco líneas siguientes tienen la función de:

1. Graficar los puntos de datos **bouguer.xyz** como círculos en el mapa para ver donde la grilla es confiable.
2. Poner contornos a la superficie.
3. Graficar la costa.
4. Graficar los lagos en un cierto azul definido por ${lagos}.
5. Poner una escala al lado del gráfico para la paleta de colores usada.

```
more bouguer.xyz | psxy -J${proyeccion} -R${limites} ${verbose} -Sc0.1c \
-Cbouguer.cpt -W3 -O -K >> ${psfile}
grdcontour bouguer2.grd -B${ticks} -J${proyeccion} -C50 -A100+k0/0/0+g255/255/255 \
-R${limites} ${portrait} ${verbose} -W0.5p/0/0/0 -Gd5c -O -K >> ${psfile}
pscoast -B${ticks} -J${proyeccion} -R${limites} ${portrait} ${verbose} \
-D${resolucion} -W${linea_costa} ${bordes} ${rios}  -O -K >> ${psfile}
pscoast -B${ticks} -J${proyeccion} -R${limites} ${portrait} ${verbose} ${lagos} \
-D${resolucion} -W${linea_costa} ${bordes} ${rios} -O -K >> ${psfile}
psscale  -D15.7c/2.5c/5c/0.5c -Cbouguer.cpt ${verbose} -Ba100f20:"(mGal)": \
-O -K >> ${psfile}
```

Hemos llegado a la última parte del script. En esta estamos graficando un perfíl de la grilla. Primeramente usamos el comando **project** para generar un archivo *track.xy* que consiste de puntos intermedios entre dos ubicaciones de longitud/latitud con cierto espaciado entre los puntos.

```
project -C-73/-28 -E-62/-28 -G0.01 > track.xy
```

En este caso estamos definiendo un perfil a latitud 28° sur que cruce los Andes y pase por los puntos de datos. El espaciado entre los puntos es 0.01 grados (revisen el archivo track.xy). Ahora queremos tomar los datos de las grillas de Bouguer y de topografía a los puntos a lo largo del perfil; pora eso usamos **grdtrack** :

```
grdtrack track.xy -Gbouguer2.grd > bouguertrack.xydz
grdtrack track.xy -G${region}.grd > topotrack.xydz
```

Estamos generando archivos con 4 columnas: longitud, latitud, distancia a lo largo del perfil y el valor interpolado de la grilla en esos puntos. Noten que el perfil tiene un largo de 9.709 grados o 1078 kilómetros. Para el gráfico del perfil, queremos definir ejes entre 0 a 1078 kilómetros y -500 a 500 mgal. Los ejes deben estar abajo del mapa, entonces usamos **-X0 -Y-4**, con esta ubicación relativa a la ubicación del mapa original, es decir 4 centímetros abajo de él. También queremos que el ancho del perfil sea el mismo al mapa, para eso usamos **-JX14/2** para que el perfil sea 14 cm en largo y 2 cm en alto.

```
psbasemap -R0/1078/-500/500 -JX14/2 -X0 -Y-4 -Ba200f100 -P -O -K -V >>  ${psfile}
```

Ahora graficamos la anomalía de Bouguer a lo largo del perfil. Multiplicamos la tercera columna por 111 para convertir entre grados y kilómetros, graficamos una línea con **psxy**:

```
more bouguertrack.xydz | awk '{print $3*111, $4}' | psxy -R0/1078/-500/500 \
-JX14/2 -B -P -W2/0/0/0t20_10:10 -V -O -K >> ${psfile}
```

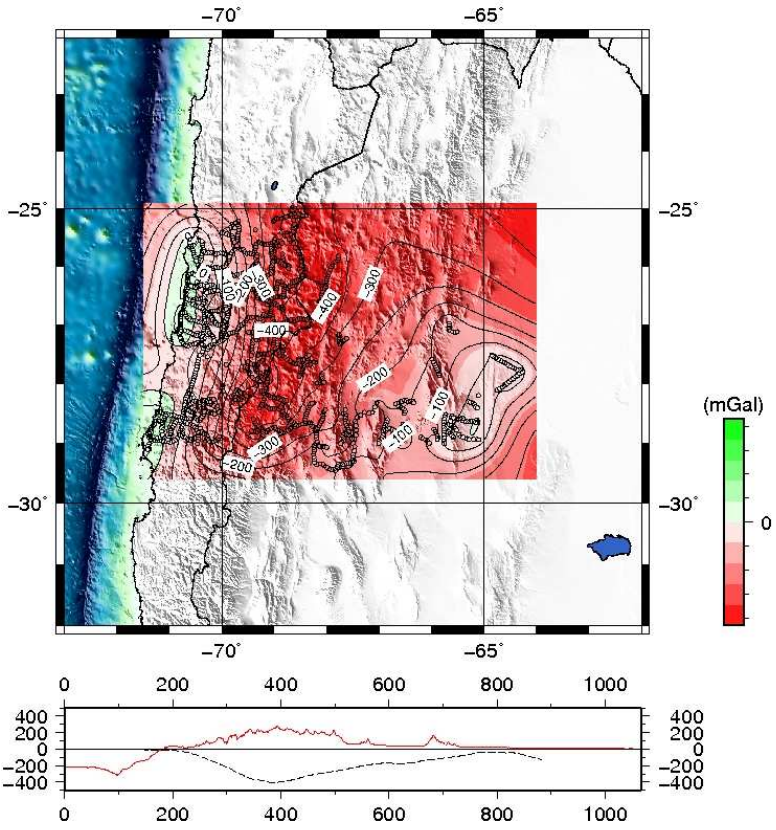
Noten la línea -**W2/0/0/0t20_10:10**, estoy tomando una línea con un ancho de 2 puntos -**W2**, de color negro 0/0/0, con rayas de 20 puntos de largo t20 separadas con espacios de 10 puntos (_10), y la primera raya tiene un largo de 10 puntos solamente (:10). Prueben diferentes combinaciones de valores acá para generar diferentes líneas punteadas, prueben t20_10_5_10 también - ¿ que hace?.
El perfil de topografía es de una línea sólida roja. Note el escalamiento de los valores de topografía ($4/20) para que podamos usar los mismos ejes. Con eso la topografía en kilómetros tiene una exageración de un factor de 50.

```
more topotrack.xydz | awk '{print $3*111, $4/20}' | psxy -R0/1078/-500/500 \
-JX14/2 -B -P -W3/255/0/0 -V -O -K >> ${psfile}
```

Finalmente ponemos una línea entre los puntos (0,0) y (1078,0) para que el perfíl se vea un poco mejor:

```
psxy << END -R0/1078/-500/500 -JX14/2 -B -P -W1/0/0/0 -O >> ${psfile}
0 0
1078 0
END
```

En este caso, los valores entre el comando psxy « END (literalmente, correr psxy usando las siguientes líneas de texto hasta que encuentras una línea que dice "END") y el END representan el equivalente a un archivo .xy.
Este script debe introducir una manera de poner datos tomados en terreno a un mapa que representa la región. Debe generar la imagen que sigue. El código completo del script viene después de la imagen.



El script completo :

```
#!/bin/bash

region="ejemplo_bouguer"
limites="-73/-62/-32/-22"

#Origen (posicion del grafico en la hoja):
xshift="2.0"
yshift="10.0"

proyeccion="M14.0"               #Tipo de proyeccion
paleta="/usr/local/GMT4.5.8/share/cpt/GMT_ocean.cpt"    #Paleta de colores a usar
topografia="/home/carlos/bin/w100s10.Bathymetry.srtm.grd" #Archivo de topografia
ticks="a5f1g5SEWN"    # Describiendo el marco del mapa
illaz="225"     # Direccion del azimut de iluminacion
portrait="-P" # Si quiero landscape en vez de portrait, dejo esta opcion en blanco
verbose="-V" # Dejo esta opcion en blanco si no quiero mostrar el "blah blah" de
               # los comandos en la terminal

#Lineas de costa e informacion de los limites fronterizos.
#(Para mas detalles, escribe en la terminal: man pscoast)
linea_costa="1.00p/0/0/0"
resolucion="f"   # Resolucion de las lineas de costa. Puedes elegir:
               # (f)ull, (h)igh, (i)ntermediate, (l)ow, y (c)rude
bordes="-N1/1.00p/0/0/0"  #Limites politicos de los paises

#Tambien se puede aÃśadir:
#rios="-I1/1.00p/0/0/255"

agua="-S0/0/255" #Esto va a graficar de un solo color todo lo que son masas de agua
tierra="-G0/255/0" #Esto va a graficar de un solo color todo lo que es tierra
lagos="-C50/100/200 -A0/2/4"

################################################################################

# Archivo de salida:
psfile="${region}.ps"

# Construyendo el marco del mapa:
psbasemap -B${ticks} -J${proyeccion} -R${limites} -X${xshift} -Y${yshift} ${portrait} \
${verbose} -K > ${psfile}

# Cortando la grilla original a los limites establecidos.
# grdcut archivo_entrada.grd -Garchivo_salida.grd -Roeste/este/sur/norte [-V]
grdcut ${topografia} -G${region}.grd -R${limites} -V

# Iluminando la grilla topografica generada en el grdcut desde un azimut
# especifico para producir un archivo de intensidad (.int) de iluminacion:
grdgradient ${region}.grd -G${region}.int -A${illaz} -Nt -M

# Graficando la topografia de la region seleccionada con su iluminacion:
grdimage ${region}.grd -C${paleta} -I${region}.int -B${ticks} -J${proyeccion} -R${limites} \
${portrait} ${verbose} -O -K >> ${psfile}

################################################################################
```

```
# Generando la grilla (.grd) de anomalias de Bouguer a partir de los datos de
# gravedad proporcionados por bouguer.xyz
surface bouguer.xyz -Gbouguer.grd -R-71.5/-64/-29.6/-24.9 -I1m -T0.25 -C0.1 -V

# Creando la paleta de colores asociada a la nueva grilla de gravedad:
grd2cpt bouguer.grd -C/usr/local/GMT4.5.8/share/cpt/GMT_red2green.cpt -T= > bouguer.cpt

# Generando nuevas grillas de iluminacion topografica (topo.int) y de bouguer
# (bouguer2.grd), de tal manera que se inserten en los mismos puntos de una
# misma region (-R):
grdsample ${region}.int -Gtopo.int -R-71.5/-64/-29.6/-24.9 -I1m -V -F
grdsample bouguer.grd -Gbouguer2.grd -R-71.5/-64/-29.6/-24.9 -I1m -V -F

# Graficando la nueva grilla de bouguer, pero con iluminacion asociada a la
# topografia:
grdimage bouguer2.grd -Cbouguer.cpt -B${ticks} -J${proyeccion} -Itopo.int -R${limites} \
${portrait} ${verbose} -O -K >> ${psfile}

# Graficando los datos de bouguer.xyz como circulos en el mapa para ver donde la
# grilla es confiable:
more bouguer.xyz | psxy -J${proyeccion} -R${limites} ${verbose} -Sc0.1c -Cbouguer.cpt \
-W3 -O -K >> ${psfile}

# Insertando contornos a la nueva superficie bouguer2.grd que fue creada:
grdcontour bouguer2.grd -B${ticks} -J${proyeccion} -C50 -A100+k0/0/0+g255/255/255 \
-R${limites} ${portrait} ${verbose} -W0.5p/0/0/0 -Gd5c -O -K >> ${psfile}

# Graficando las costas y limites del mapa:
pscoast -B${ticks} -J${proyeccion} -R${limites} ${portrait} ${verbose} -D${resolucion} \
-W${linea_costa} ${bordes} ${rios}  -O -K >> ${psfile}

# Agregando al grafico los lagos con un color determinado en la variable $lagos:
pscoast -B${ticks} -J${proyeccion} -R${limites} ${portrait} ${verbose} ${lagos} \
-D${resolucion} -W${linea_costa} ${bordes} ${rios} -O -K >> ${psfile}

# Agregando una escala de magnitudes la grilla de bouguer en base a su paleta de
# colores:
psscale  -D15.7c/2.5c/5c/0.5c -Cbouguer.cpt ${verbose} -Ba100f20:"(mGal)": -O \
-K >> ${psfile}

# Generando un perfil a los a lo largo de la latitud 28S entre las longitudes
# 73W y 62W, el cual se guarda en un archivo track.xy
project -C-73/-28 -E-62/-28 -G0.01 > track.xy

# Tomando los datos de las grillas de bouguer y de topografia que se encuentran
# a lo largo del perfil creado y guardandolos en archivo (.xydz):
grdtrack track.xy -Gbouguer2.grd > bouguertrack.xydz
grdtrack track.xy -G${region}.grd > topotrack.xydz

# Creando los bordes de un nuevo mapa, con el objetivo de graficar en el los
# perfiles creados:
psbasemap -R0/1078/-500/500 -JX14/2 -X0 -Y-4 -Ba200f100 -P -O -K -V >>  ${psfile}

# Graficando la anomalia de bouguer a lo largo del perfil como una curva discontinua:
more bouguertrack.xydz | awk '{print $3*111, $4}' | psxy -R0/1078/-500/500 -JX14/2 \
-B -P -W2/0/0/0t20_10:10 -V -O -K >> ${psfile}
```

```
# Graficando la topografia a lo largo del perfil como una curva roja:
more topotrack.xydz | awk '{print $3*111, $4/20}' | psxy -R0/1078/-500/500 -JX14/2 \
-B -P -W3/255/0/0 -V -O -K >> ${psfile}

# Agregando una linea recta para ver mejor las variaciones de la anomalia y
# topografia a lo largo del perfil seleccionado:
psxy << END -R0/1078/-500/500 -JX14/2 -B -P -W1/0/0/0 -O >> ${psfile}
0 0
1078 0
END
```