An Introduction to Advanced UNIX

Part 1

Kevin Keay

April 2008

Last Revised: Apr 20 2008

Introduction

This brief document is a follow-up to *Introduction to the School of Earth Sciences UNIX computer network.* Please refer to that document for information concerning system login and basic use of UNIX and the C shell.

For additional help, please ask Brett Holman or Doug Morrison (support@earthsci.unimelb.edu.au), Kevin Keay (keay@unimelb.edu.au) or any of the Meteorology postgraduate students. A very useful and concise reference book covering UNIX and the C shell is:

UNIX in a Nutshell, O'Reilly and Associates (Engineering Library: 005.43 GILL).

A useful online resource from the University of Surrey, *UNIX Tutorial for Beginners,* is available at:

http://www.ee.surrey.ac.uk/Teaching/Unix/

The above webpage includes a link to download a zip file of the tutorial to your PC.

Another useful web site at the University of Minnesota that concerns C shell scripts is:

Introduction to C-Shell Programming

by Jennifer Dudley. This tutorial is no longer available on the web but there is a PDF version (modified by Kevin Keay) that is retained on the Honours web site.

Note that the scripts are correct for all UNIX systems. However, the example involving `conmap` requires the appropriate 'endian' version for the binary CMP (conmap) file (see *C shell Scripts* - section 8).

All UNIX commands should have a manual (man) page on the machine that you are using e.g. `man awk`. However sometimes the pages are not installed or are in an unexpected location. It may be more convenient to use a search engine like Google to find information on a command.

A remark about the `csh` command

In most cases you will use:

```
#!/bin/csh -f
```

as the first line of your C shell script. You may use `exit` for your last line but this is optional. The –f means fast start (no source of your .login or .cshrc files in /home/user).
Other useful options are:

–n Parse (interpret), but do not execute commands. This option can be used to check C shell scripts for syntax errors.

–x Echo. Set the echo variable; echo commands after all substitutions and just before execution.

See: `man csh` for more details.

After you have created a script use: `chmod 755 myscript` to make it executable, just like a command or program. To run:

```
myscript
```

You can also use the `csh` command at the shell prompt:

```
csh myscript
```

and include additional options without having to change the first line during testing. However, normally you would use the first method i.e. `myscript`.

Comments in a script start with a hash (#) and may be placed anywhere; blank lines are ignored. A command may be extended onto additional lines with a back slash (\) – see *C shell scripts* - section 4 for an example.

Finally, the section: *Lab session – Advanced UNIX* contains a set of basic C shell example scripts, as well as an illustration of the decoding of a NetCDF file and data processing and an example of Generic Mapping Tools (GMT).

C shell scripts

See: *Introduction to C-Shell Programming*.

Supplementary material

1. `command`

A useful way of getting command or program output into a shell variable. The ` character is usually on the key with ~ at the upper left of the keyboard, *not* the one with ". For example:

```
#!/bin/csh -f
set x = `whoami`   # Store the username in $x
echo $x            # This will write the username on the screen
```

Another case:

```
#!/bin/csh -f
set ff = `ls`      # $ff will contain all the filenames in the current folder
@ nf = $#ff        # $nf is the number of filenames
@ i = 1
while ($i <= $nf)
  set f = ($ff[$i])
  echo "File "$i ":" $f      # Write each filename on screen
  @ i ++                     # Increments counter $i by 1
end
```

2. Debugging scripts

Use: `csh -n myscript`

to check the syntax for errors (does not execute the script). The –x option is useful too see: *A remark about the `csh` command.*

3. foreach

This is useful for processing groups of files. For instance:

```
#!/bin/csh -f
foreach f (*.txt)
  echo $f       # Write each filename on the screen
end
```

A number can be assigned to each file.

```
#!/bin/csh -f
@ k = 0      # File counter
foreach f (*.txt)
    @ k = ($k + 1)            # or use:   @ k ++    # Add 1 to $k
    echo "File no: "$k $f     # Write counter and  filename on the screen
end
```

Note that with `echo` the text in "" is written to the screen and that spaces within the double quotes command are written out too i.e. `echo "File no: "$k $f` has more output spaces than in the example.

4. sed

The `sed` command (stream editor) is useful for all kinds of things including renaming files. For instance, if you have a set of filenames like precipitation.yyyy.dat where yyyy is a four-digit year you could rename them to Australian_rainfall.yyyy.txt using the following script:

```
#!/bin/csh -f
foreach f (precipitation.????.dat)
  echo "Input filename:" $f
  # Replace (substitute) 'precipitation' with 'Australian_rainfall' then pipe
  # through sed again to replace 'dat' with 'txt'
  # The '\' at the end of the next line indicates that the command is
  # continued on to the next line – useful for long commands
  set o = `echo $f | sed -e "s/precipitation/Australian_rainfall/" | \
          sed -e "s/dat/txt/"`
```

```
  echo "Renaming $f to $o"
  mv $f $o
end
exit
```

Note: this example is just for illustration but you could modify it to work on your own data.

5. awk

This is a useful command that includes a scripting language based on C. It is particularly useful for dealing with text files that contain columns. For example, you may have a file containing 10 columns but it might be convenient to extract columns 4 and 5 and output these to another file for subsequent processing. The following command does this task:

```
awk '{printf "%7.2f%7.2f\n", $4, $5}' data.txt >! cols_4+5.txt
```

The output file (cols_4+5.txt) contains two columns, each 7 columns wide with 2 decimal places – these are floating point (*real* in Fortran) numbers, hence the 'f' in `%7.2f` (`F7.2` in Fortran):

```
  92.50 -87.50
  90.00 -85.00
 300.00 -85.00
 250.00 -82.50
 302.50 -82.50
  52.50 -80.00
```

If you are not concerned about formatting then a simpler usage is:

```
awk '{print $4, $5}' data.txt >! cols_4+5.txt
```

Note: `print` rather than `printf` (formatted).

`awk` is also useful to output a range of lines from a text file e.g. lines 10-20 as in this example:

```
awk 'NR >= 10 && NR <= 20 {print $0}' test.txt >! out.txt
```

Field $0 is the entire current line of the text file.

6. cut

This can be used to extract 'segments' of a filename or text string. For instance, you may have a set of filenames like rainfall.yyyymmdd.dat where yyyy is a four-digit year, mm is a two-digit month and dd is a two-digit day. You could input the files to some program e.g. `getmax` (a fictitious program to print out the maximum rainfall on a day to the screen) but want the output to be directed to a file that contains the date. We can extract the date from the filename with the `cut` command. Since the filename comprises segments delimited by dots we can use the –d option with '.' as the segment separator. To select the individual segments (fields) we use the –f option. 'rainfall' is field 1, 'yyyymmdd' is field 2 and 'dat' is field 3 (note that the dot is not included in the field). Here is an example to illustrate this.

```
#!/bin/csh –f
foreach f (rainfall.????????.dat)
  echo "Input filename:" $f
  set d = `echo $f | cut –d. –f 2`     # d is date (field 2)
  set o = (output.$d.txt)
  getmax $f >! $o
end
```

If you wanted to extract the year, month and day from the date segment (which is in shell variable `$d`) you could use the –c option of `cut`:

```
set yr =  `echo $d | cut –c 1-4`     # Use characters 1-4 as year
set mn =  `echo $d | cut –c 5-6`     # Use characters 5-6 as month
set dy =  `echo $d | cut –c 7-8`     # Use characters 7-8 as day
```

7. Character matching

In UNIX it is possible to simplify the command-line representation of files or the search for phrases in text files with character matching. For instance, you may have a set of pressure files ave.pmsl.ncep2.yyyymm.cmp where yyyy is a four-digit year and mm is a two-digit month. With C shell variables `$y` for year and `$m` for month, a particular file is:

```
ave.pmsl.ncep2.$y$m.cmp
```

If you want to specify the winter (JJA) files you could use:

```
ave.pmsl.ncep2.${y}0[6-8].cmp
```

Note: the braces around y prevent any confusion otherwise the shell will be looking for `$y0` rather than `$y`. The expression `[6-8]` means a 6, 7 or 8 in that position e.g. ave.pmsl.ncep2.200207.cmp. An equivalent form is `[6,7,8]`.
If you were interested in the years 1994, 1995, 1996, 2000 and 2002 for JJA you could use:

```
ave.pmsl.ncep2.{1994,1995,1996,2000,2002}0[6-8].cmp
```

or

```
ave.pmsl.ncep2.{199[4-6],200[0,2]}0[6-8].cmp
```

The `[]` are used for single characters to be matched and the `{}` are used for longer text strings.

Note: For a two-digit month the `if` command can be used to include a zero for months less than 10.

```
# Assume we have a month $mn  m=1 … 12
if ($mn < 10) then
   set m =  (0$mn)
else
   set m = ($mn)
endif
# m is a two-digit month e.g. $m = 06 for June
```

8. Answering prompts

Often you will use programs that require answers to questions as well as command-line arguments. A program used by many people in the group for plotting data on a latitude-longitude grid is called `conmap`, which is a Fortran program based on the free NCAR Graphics. Since most people use the Linux machines `conmap7` will be used for illustration. Here is an example.

At the prompt type:

```
conmap7 -SB pmsl.ncep.linux.cmp
```

You will be asked:
Do you want a grey scale rather than colour bands?
Answer n for no.
Enter format for key labels (6 chars) e.g. (f6.1)
Type (f6.1) (appropriate for pressure values)
Do you want to select your own colours? (y/n)
Answer n
Do you want the key?
Answer y
You should then see a message: DO NOT FORGET TO USE GLW_COLOR TO PRINT
A special graphics file called gmeta has been created.
At one stage the script glw_color was used for printing purposes. Today you can use:

```
g2ps -c gmeta
```

where –c means a colour plot.
This creates a Postcsript file called g.ps. You can use convert to change this to a graphics format e.g. PNG, JPEG, to insert into (say) a Microsoft Word document:

```
convert g.ps g.png
```

There are many options for `convert` – see: `man convert`
The Linux version is more recent than the SUN version – see for instance the –trim option.

Now, rather than answering the prompts we can store our responses in a text file and

control the program. Using:

```
nedit icon
```

Type:
n
(f6.1)
n
y

and save it. Then run the script:

```
#!/bin/csh -f
conmap7 -SB pmsl.ncep.linux.cmp < icon
g2ps -c gmeta
convert g.ps g.png
exit
```

The < means read the responses from the file called icon rather than the keyboard.
A variation on this theme is to include the responses in the script using the << operator:

```
#!/bin/csh -f
conmap7 -SB pmsl.ncep.linux.cmp << !    # The responses end at next instance of !
n
(f6.1)
n
y
!
g2ps -c gmeta
convert g.ps g.png
exit
```

A remark on binary CMP (conmap) files

If you are using a Solaris (SUN) machine e.g. atlas, you need to convert the CMP (conmap) file using a program written by Kevin Keay called `binswap`. This is actually written in C!. For example:

```
binswap -c pmsl.ncep.linux.cmp pmsl.ncep.sun.cmp
```

where `binswap` is run on the Solaris machine.
The output CMP file (pmsl.ncep.sun.cmp) can now be read on a Solaris machine.
The reverse procedure is also true: a Solaris CMP file may be converted on a Linux machine with `binswap`.

Lab session – Advanced UNIX

Kevin Keay

Apr 20 2008

1. Log on to one of our Linux machines e.g. cove, tide:
   ```
   ssh -X cove
   ```
2. ```
   cd ~
   mkdir adv_unix
   cd adv_unix
   cp /home/kevin/unix_course/zadv_unix.zip .
   unzip zadv_unix.zip
   ```

   You will see these folders in `adv_unix`:

   ```
   data/  gmt/  lab/  reanal/
   ```

   The Lab session scripts are in `lab`, the decoding of a NetCDF file and data processing examples are in `reanal` and the GMT examples are in `gmt`.

   The scripts have names ending in .csh. When you write your own scripts you may call them whatever you wish e.g. `add_files`.

3. As a quick exposure to writing scripts:
   ```
   nedit s1.csh
   ```

   Type:

   ```
   #!/bin/csh -f
   set x = `whoami`   # Store the username in $x
   echo $x            # This will write the username on the screen
   ```

   Save it.
   At the shell prompt:
   ```

```
csh -n s1.csh
```

This just checks for errors – the script is not executed.
Make the script an executable file: `chmod 755 s1.csh`
Run it:

```
s1.csh
```

You should see your username printed on the screen
e.g. kevin

4. Try out the other scripts: `s2.csh` – `s7.csh` or in UNIX, `s[2-7].csh` (!)

Examples

s1.csh

```
#!/bin/csh -f
set x = `whoami`  # Store the username in $x
echo $x           # This will write the username on the screen
```

s2.csh

```
#!/bin/csh -f
set ff = `ls`     # $ff will contain all the filenames in the
current folder
@ nf = $#ff       # $nf is the number of filenames
@ i = 1
while ($i <= $nf)
  set f = ($ff[$i])
  echo "File "$i ":" $f     # Write each filename on screen
  @ i ++                    # Increments counter $i by 1
end
```

s3.csh

```
#!/bin/csh -f
foreach f (*.pal)
  echo "Input filename:" $f
  set o = `echo $f | sed -e "s/pal/txt/"`
  echo "Renaming $f to $o"
  mv $f $o
end
exit
```

Note: The *.pal files are renamed (moved) to *.txt. The original *.pal files are stored in zpal.zip. Before proceeding, restore these files:

```
unzip zpal.zip
```

s4.csh

```
#!/bin/csh -f
foreach f (pmsl.ncep.????????.cmp)
  echo "Input filename:" $f
  set d = `echo $f | cut -d. -f 3`    # d is date (field 3)
  echo "Date:" $d
  set yr =  `echo $d | cut -c 1-2`    # Use characters 1-2 as year
  set mn =  `echo $d | cut -c 3-4`    # Use characters 3-4 as
month
  set dy =  `echo $d | cut -c 5-6`    # Use characters 5-6 as day
  set hr =  `echo $d | cut -c 7-8`    # Use characters 5-6 as hour
  echo "Year: "$yr "Month: "$mn "Day: "$dy "Hour: "$hr
end
```

s5.csh

```
#!/bin/csh -f
echo "List 1"
ls pmsl.ncep.96060[1-3]*.cmp
echo "List 2"
ls pmsl.ncep.96060[1,3]*.cmp
echo "List 3"
ls pmsl.ncep.9606??{06,18}.cmp
```

s6.csh

```
#!/bin/csh -f
if ($#argv == 0) then
  echo "Usage: s6.csh CMP_file"
  exit
else
  set infile = ($1)   # Assign argument 1 to infile
  echo "CMP file: "$infile  # or ${infile}
endif

# Create plot (gmeta) using instruction file icon (answers to
prompts)
conmap7 -k icon_k.band.s6 -SB $1 < icon
# Convert gmeta to Postscript (g.ps)
g2ps -c gmeta
# Convert g.ps to g.png (PNG file)
convert -trim -density 100 g.ps gs6.png
```

```
echo "Plot file: gs6.png"
# Next line - needs to have \! not just ! - confuses csh
echo "Done\!"

exit
```

Note: The script `s6.csh` uses additional files (icon.k.band.s6 and icon) to work correctly. These files contain parameters and responses for `conmap7`.

s7.csh

```
#!/bin/csh -f

# In this example it is ASSUMED that you will give a conmap
filename
# This becomes argument 1 ($1)

if ($#argv == 1) then
echo "Filename is "$1
conmap -SB $1 << !    # The responses end at next instance of !
n
(f6.1)
n
y
!
g2ps -c gmeta
convert g.ps g.png

else
  echo "ERROR: Give a filename, silly\!"  # Note \! not just !
endif

exit
```

Note: The script `s7.csh` uses the `<< ! … !` structure to give the responses for `conmap7`.

An example of decoding a NetCDF file and data processing

The C-shell scripts and `conmap7/ausmap` instruction files are in: `adv_unix/reanal`.

Assume we have downloaded a NetCDF file called pmsl_198007_part_ncep2.nc from the NCEP Reanalysis (NCEP2) web site. It is located in folder adv_unix/data. A header dump i.e. `ncdump -h pmsl_198007_part_ncep2.nc`, indicates that the pressure variable is named mslp and has units of Pa (we will rescale to hPa). There are 16 maps from July 1 1980 OOUTC - July 4 1980 18UTC.

Hence we may decode all of the maps in the NetCDF file with the command:

```
read_nc2cmp -i ../data/pmsl_198007_part_ncep2.nc -o pmsl_ncep2.cmp \
-u mslp -r NCEP2 -v PMSL -s 0.01
```

Note: \ is the C-shell continuation character.

The multi-map CMP (conmap) file is named pmsl_ncep2.cmp and contains 16 maps.

The individual CMP headers may be listed with:

```
splitcon -L pmsl_ncep2.cmp
```

We may obtain the individual maps as separate files with:

```
splitcon -n -l pmsl_ncep2.cmp
```

The CMP files are: pmsl.ncep2.1980070100.cmp - pmsl.ncep2.1980070418.cmp

We can take the average of these 16 maps with:

```
statconmap pmsl.ncep2.198007*.cmp 1 ave.cmp
```

The '1' indicates that the output CMP file (ave.cmp) is an average file.

We might want to express a given map as an anomaly from this average (usually we would use a longer averaging period). This can be achieved by:

```
conmanip2 -s pmsl.ncep2.1980070218.cmp ave.cmp diff.cmp
```

The option `-s` is 'subtract' - for usage: `conmanip2`

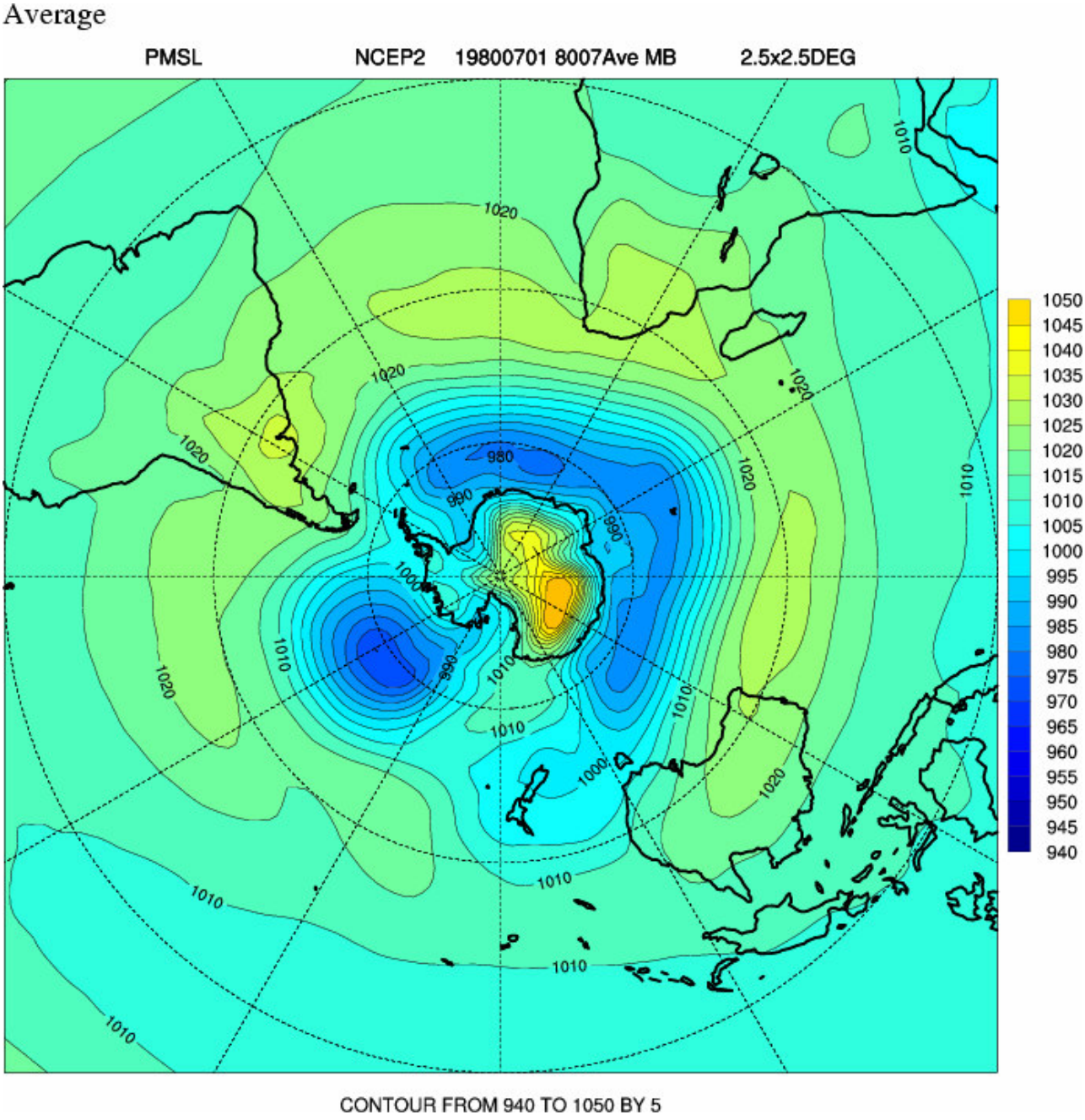
In this example diff.cmp is the anomaly for July 2 1980 18UTC

i.e. (map at this time) - (average of 16 maps).

We can produce a southern hemisphere plot of the average map with:

```
plot_ave.csh ave.cmp
```

Average



Note: For polar stereographic plots, there is a plotting glitch if lon 0 is not duplicated as lon 360. The program `fixcon` (for usage: `fixcon`) will duplicate the lon 0 at lon 360.

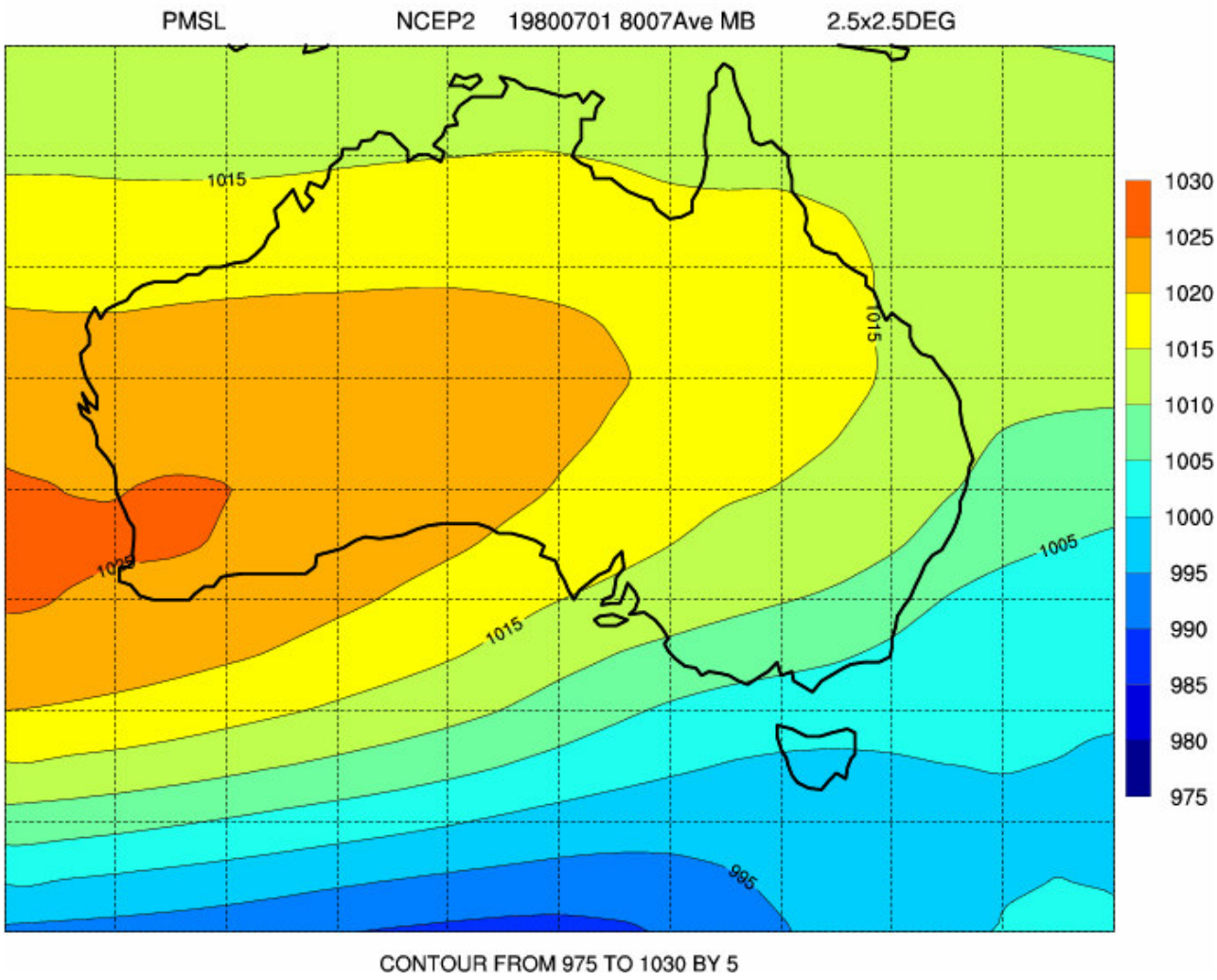
```
fixcon -d diff.cmp diff.cmp

fixcon -d ave.cmp ave.cmp
```

Similarly a plot over the Australian region is created with:

```
plot_ave.aus.csh ave.cmp
```

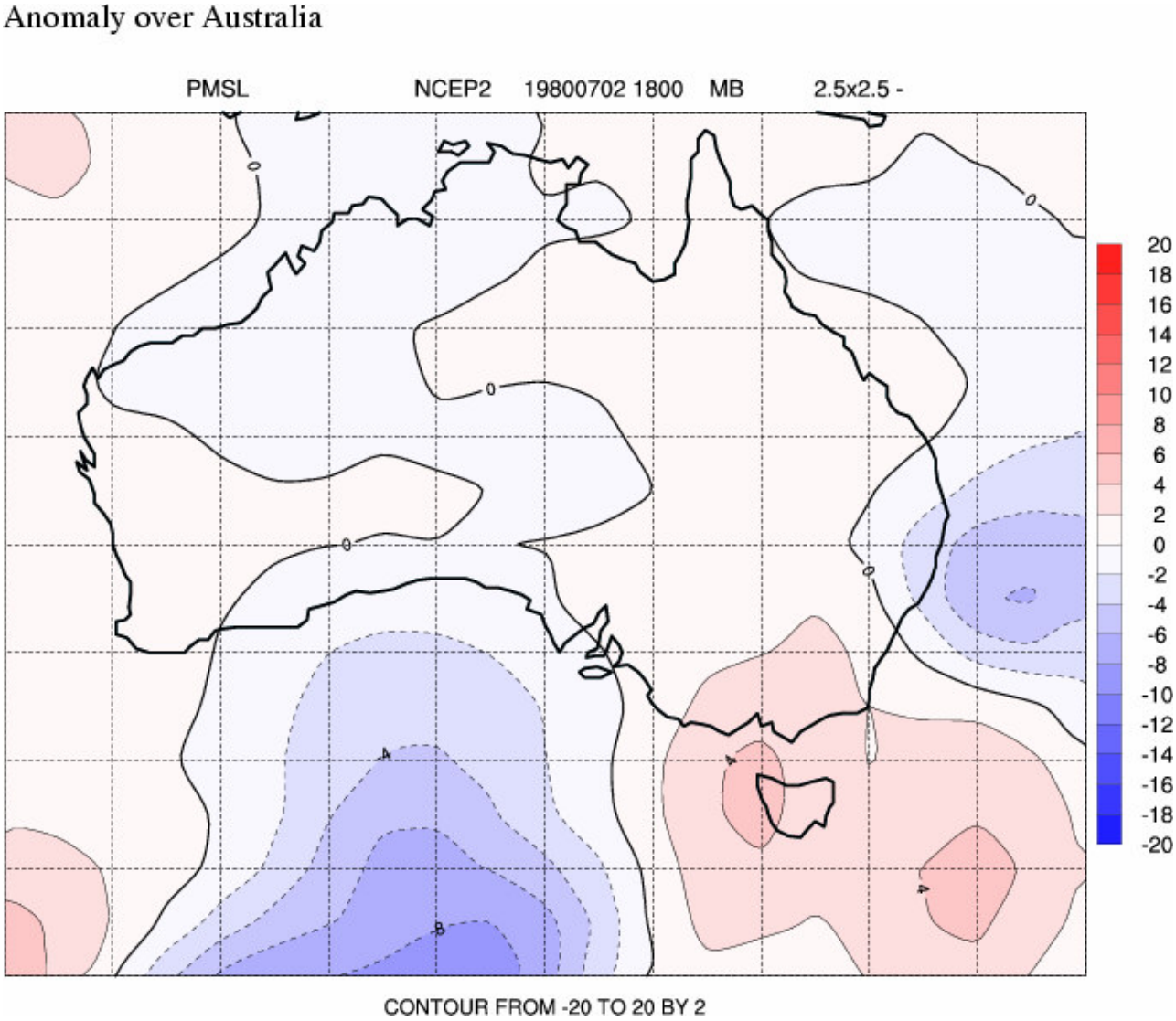
Average over Australian region



A plot of the anomaly (difference) map for July 2 1980 18UTC is produced by:

```
plot_diff.aus.csh diff.cmp
```

Anomaly over Australia

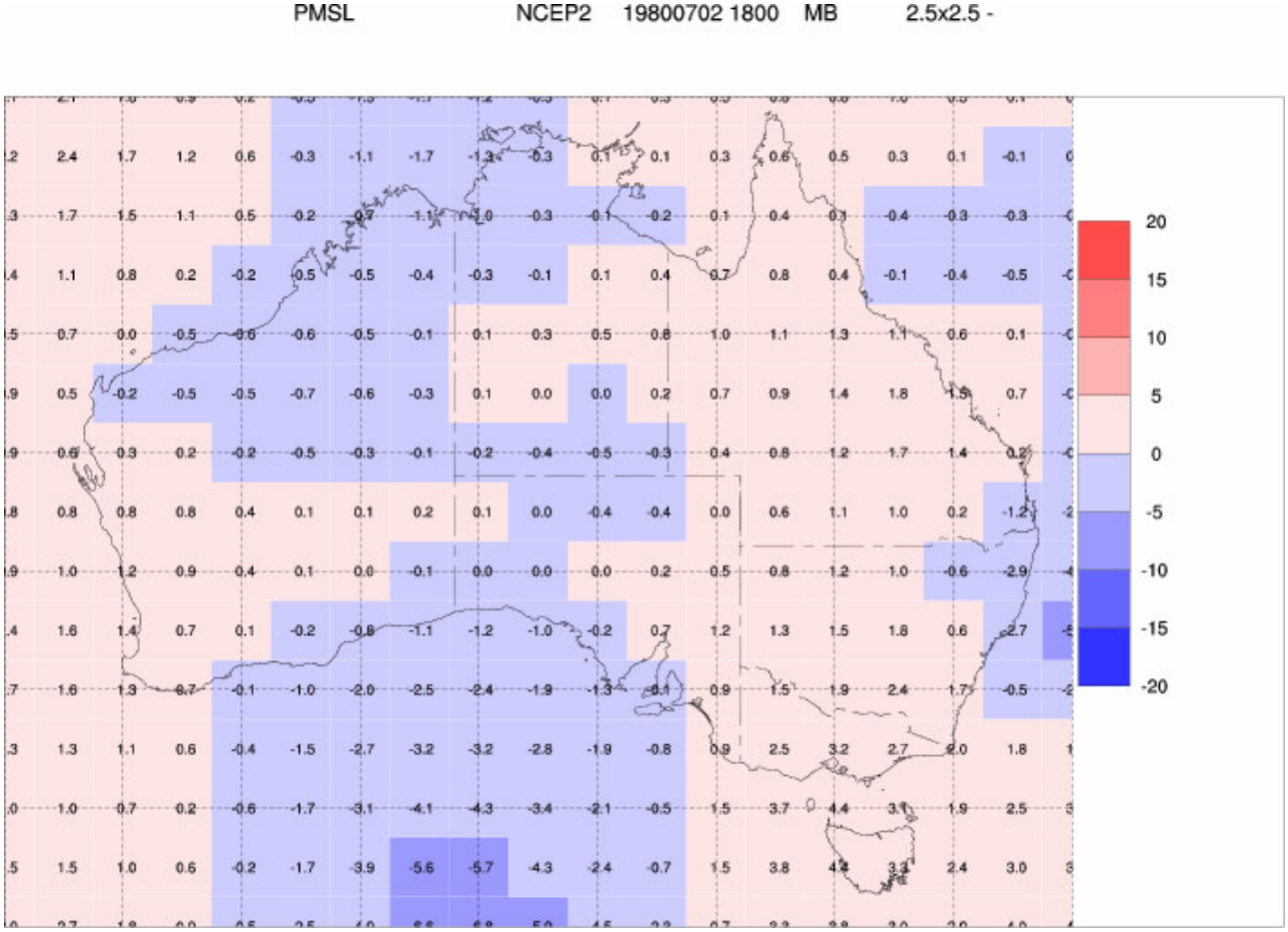


A plot of the actual gridpoint values of the anomalies for this particular time may be created with `ausmap`, using the raster (`-X`) and print (`-P`) options:

```
ausmap -X -P -2 -s diff.cmp < iaus.raster.diff

g2ps -c gmeta

convert -trim -density 100 g.ps gdiff_raster.png
```



Perhaps we want to use the anomaly map with a package like GrADS.

We can convert the CMP file (diff.cmp) to a NetCDF file by:

```
cmp2cdl4 -n nmlist.txt -i diff.cmp -o j.cdl
```

```
ncgen -o diff.nc j.cdl
```

See the namelist file (nmlist.txt) for appropriate NetCDF attribute

values. These need to be changed depending on the variable and data set. `cmp2cdl4` is a recently written program so the documentation is sparse.

The NetCDF file (diff.nc) can read by GrADS:

```
grads
```

(a graphics window will open)

At the grads prompt (ga->) open the NetCDF file (diff.nc):

```
sdfopen diff.nc
```

To check the file information:

```
q file
```

This gives a listing:

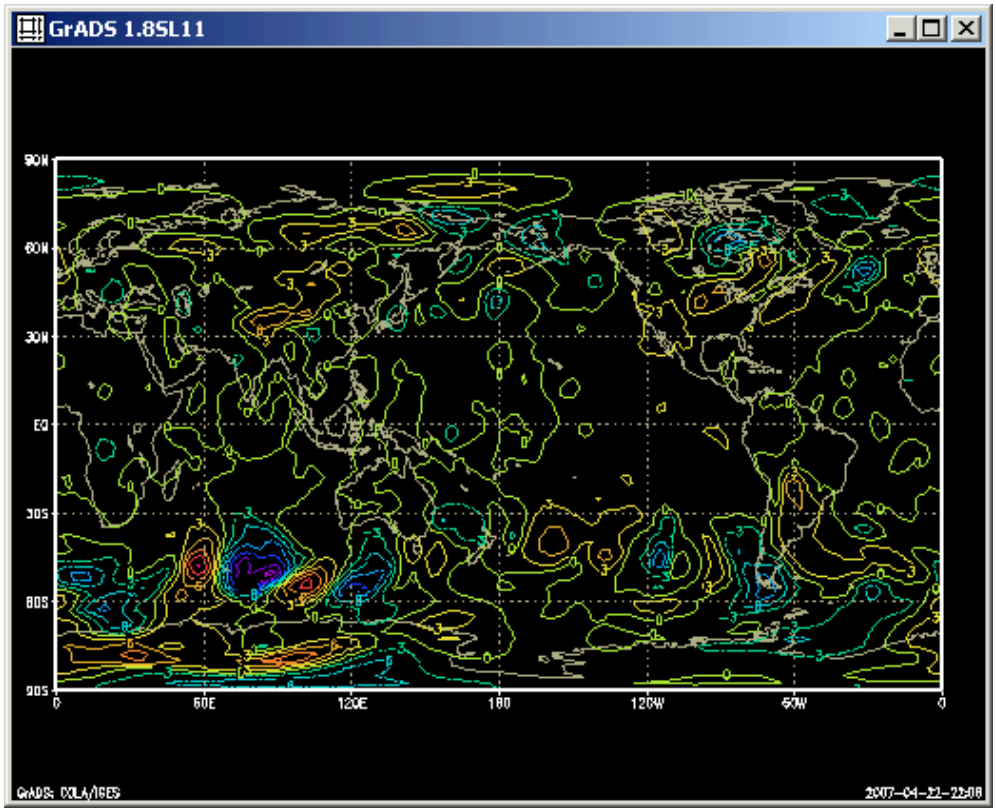
```
File 1 :
  Descriptor: diff.nc
  Binary: diff.nc
  Type = Gridded
  Xsize = 144  Ysize = 73  Zsize = 1  Tsize = 1
  Number of Variables = 1
    diff_mslp 0 -999 diff_MSLP
```

The GrADS variable is named diff_mslp (diff_MSLP is the long NetCDF name).

To display the single map:

```
d diff_mslp
```

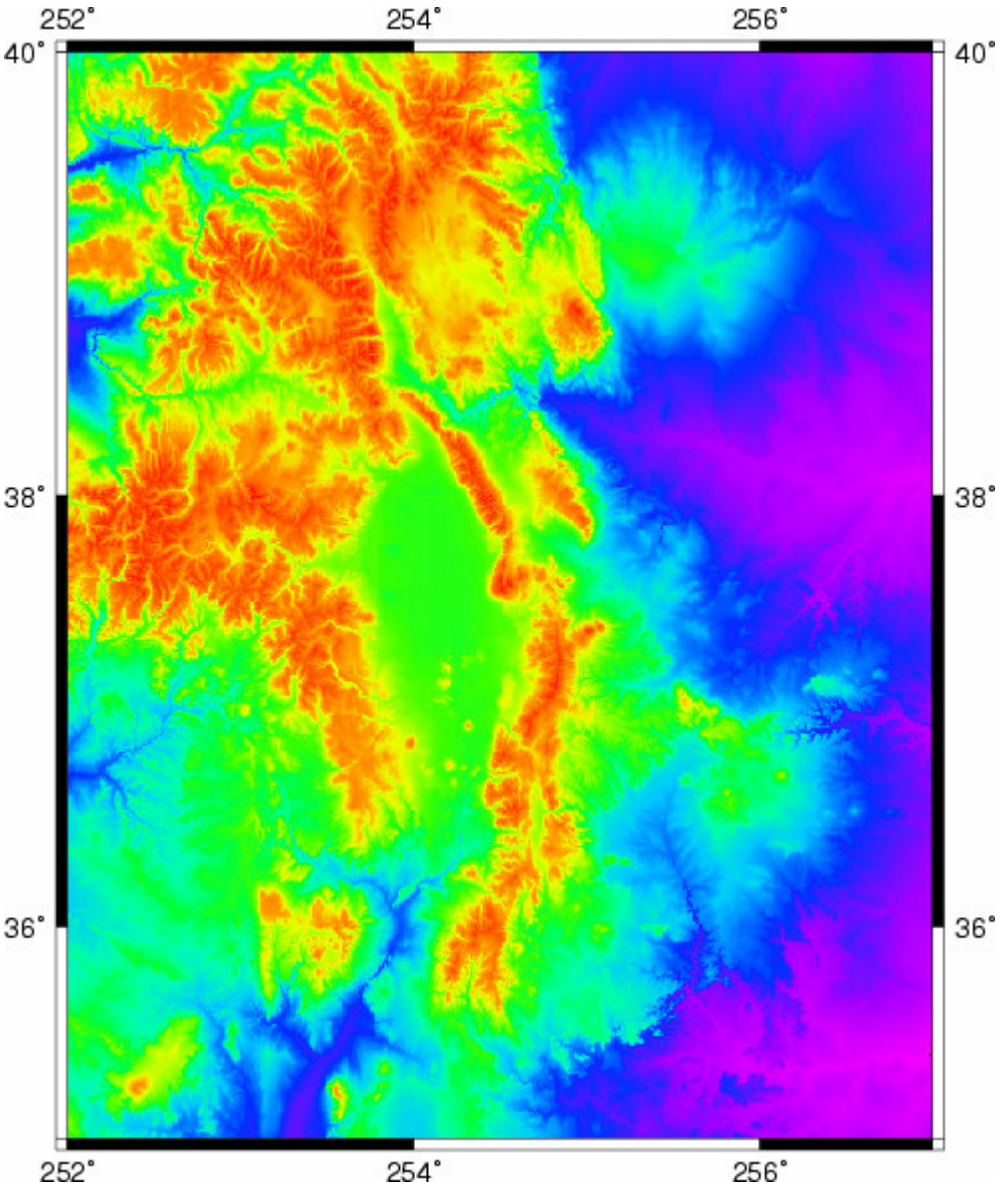
To exit `grads: quit`



An example of Generic Mapping Tools (GMT)

See folder `adv_unix/gmt` and the C-shell script: `run-gmt.eg.txt`
Also see Part 2.

```
# Taken from GMT Tutorial
#
# (1) Topography of Rocky Mountains
grdimage us.nc -JM6i -P -B2 -Ctopo.cpt -V -K > ! topo.ps
convert -trim topo.ps topo.png
```

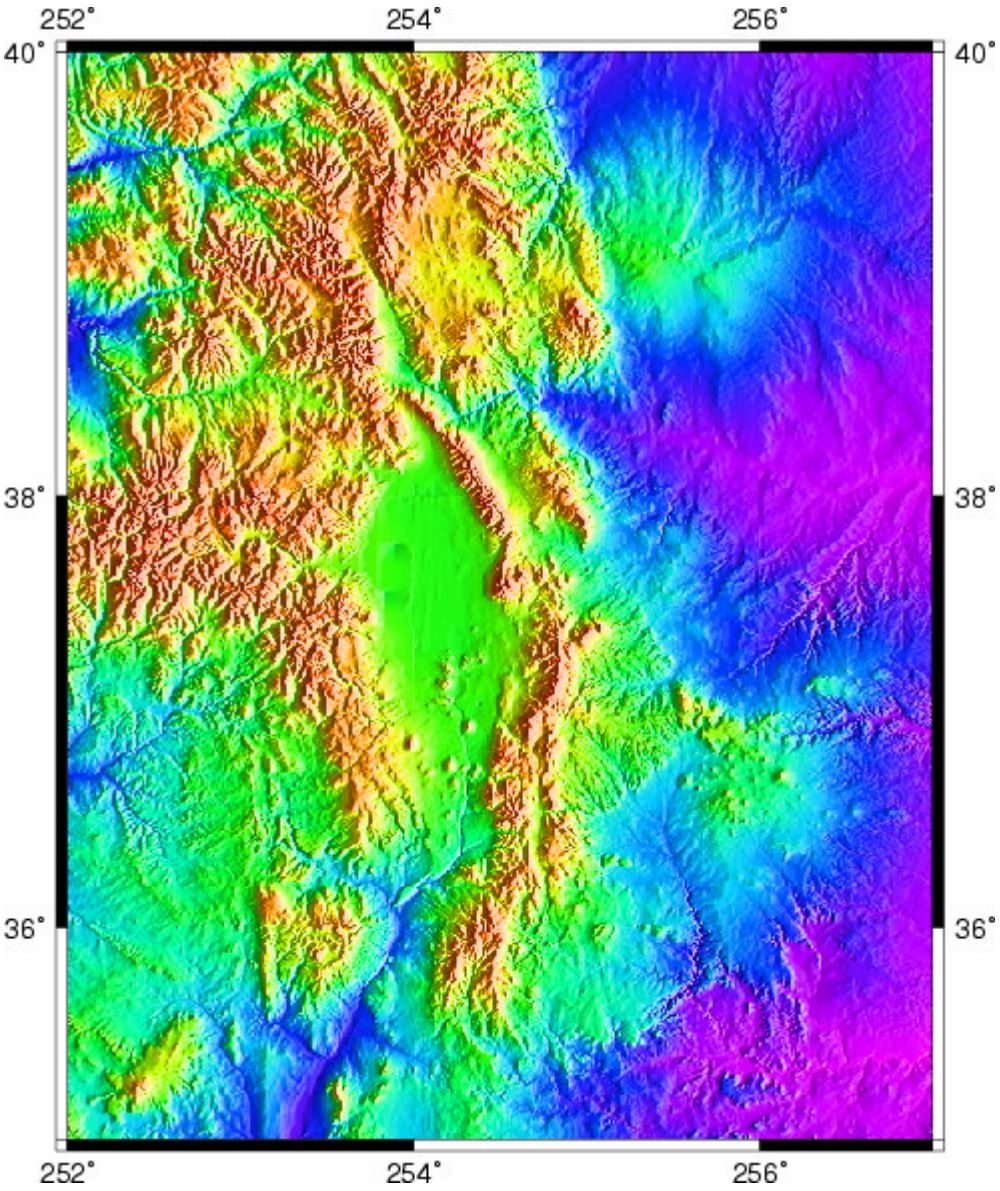


```
# (2) Topography of Rocky Mountains plus artificial illumination based on
      gradient

grdgradient us.nc -Ne0.8 -A100 -M -Gus_i.nc

grdimage us.nc -Ius_i.nc -JM6i -P -B2 -Ctopo.cpt -K > ! topo2.ps

convert -trim topo2.ps topo2.png
```



```
# (3) Tsunami earthquakes in Japan

# Size fof circle reflects magnitude and colour is measure of depth

# We will follow conventional color schemes for seismicity and assign red to shallow
      quakes (depth 0-100 km),

# green to intermediate quakes (100-300 km), and blue to deep earthquakes (depth  300
      km).

# Use UNIX awk command to extract the desired columns from a historical database
      (text format).

# $5 refers to the 5th column etc., and NR is the current record number):

#  Historical Tsunami Earthquakes from the NGDC Database

#  Year  Mo  Da  Lat+N  Long+E  Dep  Mag

#  1987  01  04  49.77  149.29  489  4.1

#  1987  01  09  39.90  141.68  067  6.8

# ...

# The output of awk is:

#

#  149.29  49.77  489  0.082

#  141.68  39.90  067  0.136

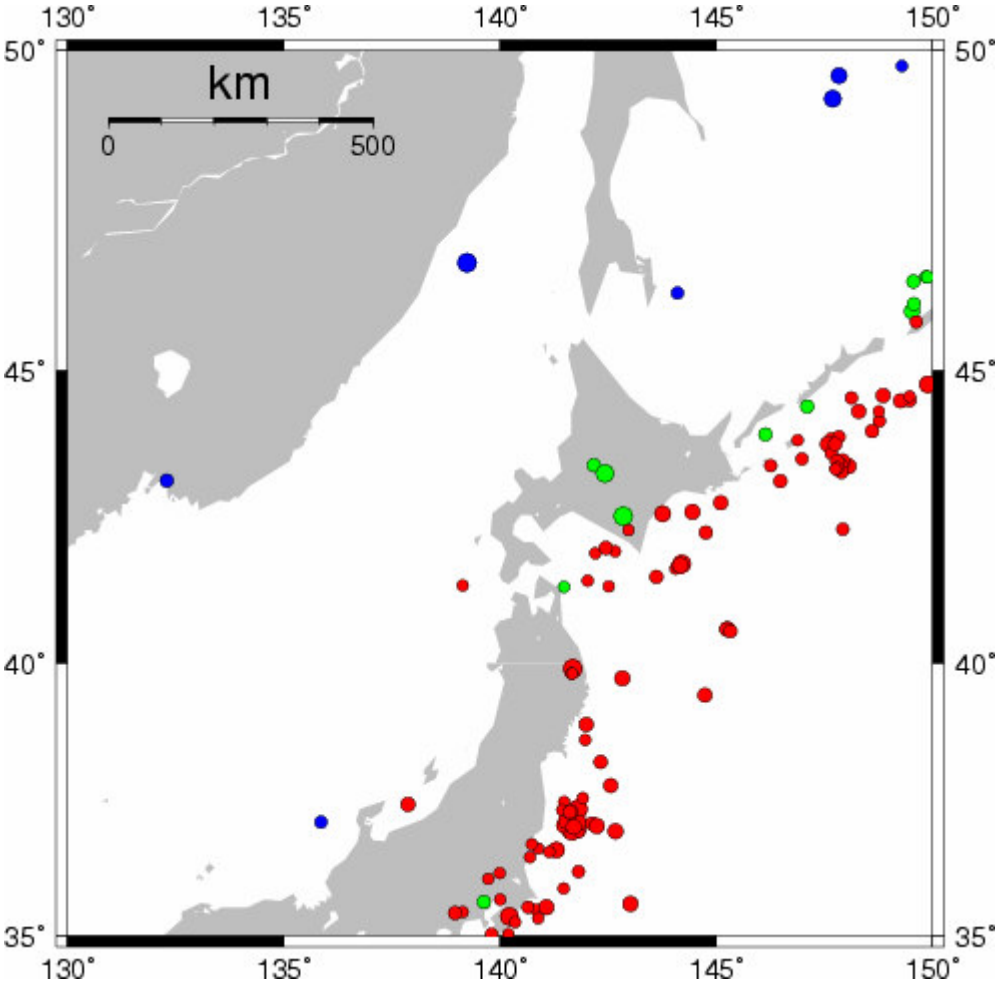
#

awk '{if (NR > 3) print $5, $4, $6, 0.02*$7}' quakes.ngdc >! quakes.d

pscoast -R130/150/35/50 -JM6i -B5 -P -Ggrey -Lf134/49/42.5/500 -K > ! map.ps

psxy -R -J -O -Cquakes.cpt quakes.d -Sci -Wthinnest >> map.ps

convert -trim map.ps map.png
```


130°
135°
140°
145°
150°
50°
45°
40°
35°
km
0
500