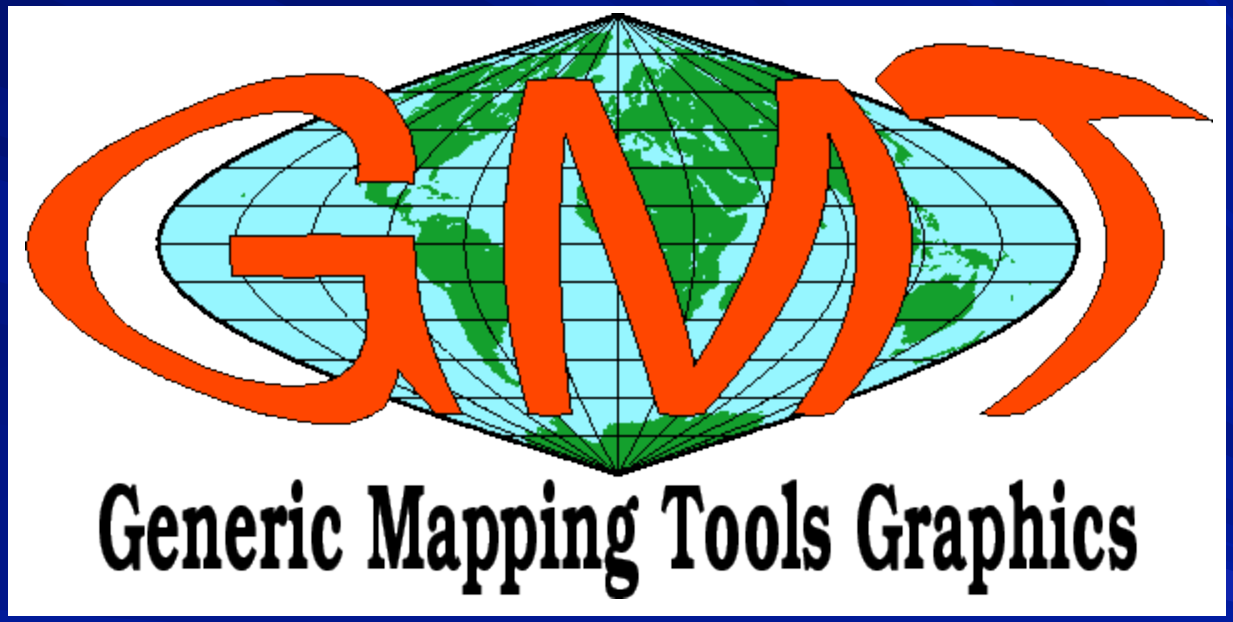
CERI Tools of the Trade Seminar Series

Wed, Oct 17, 2007

Graphics (mapping) with GMT



http://gmt.soest.hawaii.edu/

Has online documentation, mailing list with archives, etc.

Goal – make scientific illustrations ("generic" of GMT is generic to geo sciences) – include

color/bw/shaded topography and bathymetry,

point data (earthquakes, seismic or gps stations, etc.),

line data (faults, eq rupture zones, roads),

vector fields w/ error ellipses,

focal mechanisms,

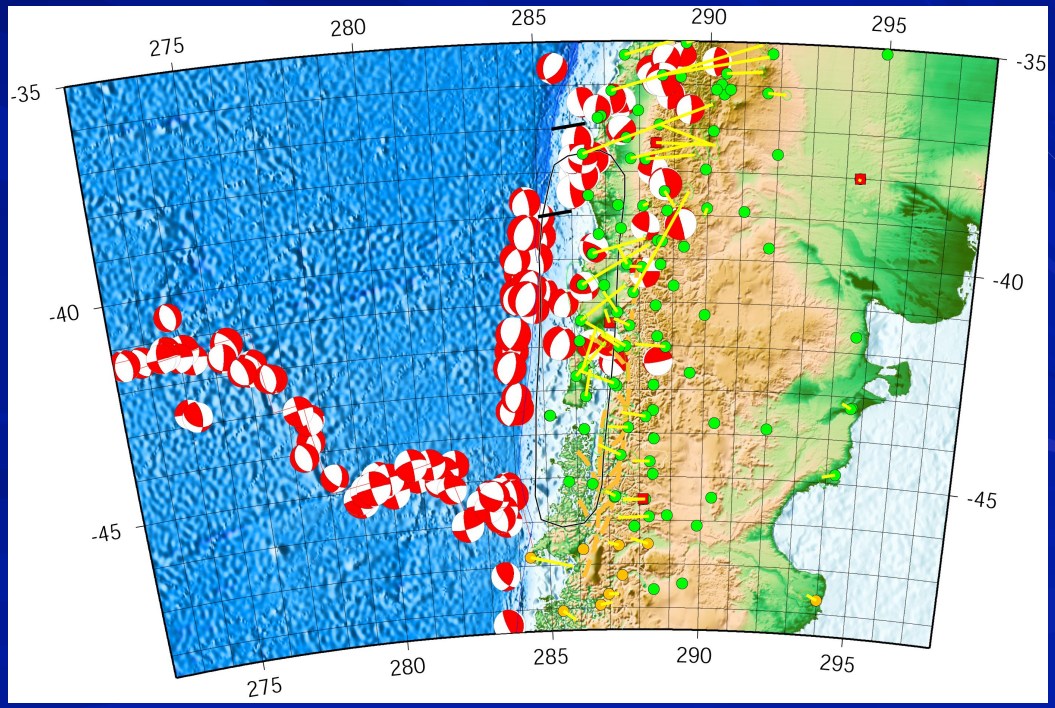
Etc.

Maps

3D surface

Cross sections

Other stuff



From the GMT tutorial

GMT follows the *UNIX* philosophy in which complex tasks are broken down into smaller and more manageable components.

Individual GMT modules are small, easy to maintain, and can be used as any other *UNIX* tool.

GMT was deliberately written for command-line usage, not a windows (or interactive) environment, in order to maximize flexibility.

From the GMT tutorial

We standardized early on to use *PostScript* output instead of other graphics formats.

Apart from the built-in support for coastlines, GMT completely decouples data retrieval/management from the main GMT programs. (puts the onus on user! UNIX philosophy)

GMT uses architecture-independent file formats (flat files – least common denominator).

A software mogul usually Bill Gates, but sometimes another makes a speech. If the automobile industry had developed like the software industry, the mogul proclaims, we would all be driving $25 cars that get 1,000 miles to the gallon. To which an automobile executive retorts, Yeah, and if cars were like software, they would crash twice a day for no reason, and when you called for service, they' d tell you to reinstall the engine. ~ *Anonymous*

What is "Unix Philosophy"?

(can computer operating systems have a "philosophy"?)

Doug McIlroy

(i) Make each program do one thing well.

To do a new job, build afresh rather than complicate old programs by adding new features. (no bells and whistles)

The only way to learn a new programming language is by writing programs in it. *~Dennis Ritchie*

What is "Unix Philosophy"?

Machine shop vs. appliance
(gives you the tools, you to make appliance)

Advantage - POWERFUL

Disadvantages

- Lots of reinventing the wheel

- Requires more educated user (user hostile)

- Requires more work from user rather than developer
(can UNIX/GMT do this? No, but YOU can write a program!)

Programming can be fun, so can cryptography; however they should not be combined. -
Kreitzberg and Shneiderman

"UNIX Philosophy"

(ii) Expect the output of every program to become the input to another, as yet unknown, program.

(GMT breaks this rule a little – final output is usually PostScript – which is input to a specific program called a PostScript interpreter.)

Don't clutter output with extraneous information.

Unfortunately this may make things confusing for the uninitiated user.

Output is for "next program" in a pipe, not the user.

Simple things should be simple and complex things should be possible. -*Alan Kay*

"UNIX Philosophy"

Also idea/use of – redirection (<, << and >, >>), command substitution (`...`)

Idea of "filter" –

program takes input from Standard IN,
does something to it and
sends it to Standard OUT

(notice that – only considers single input source and the "user" is not part of this model).

Put lots of single minded programs in a row to do what you need.

Any fool can write code that a computer can understand. Good programmers write code that humans can understand. *-Martin Fowler*

"UNIX Philosophy"

(ii) Continued

Avoid stringently columnar or binary input formats.

(Avoid, but sometimes necessary. Not closely followed in GMT.)

Don't insist on interactive input.

This does not fit in with use of pipes.

Control implemented by use of "command line switches"

(strictly followed in GMT)

The trouble with programmers is that you can never tell what a programmer is doing until it's too late. *~Seymour Cray*

REVIEW

Write programs that do one thing and do it well.

(lean and mean)

Write programs to work together.

(pipes)

An interactive debugger is an outstanding example of what is not needed - it encourages trial-and-error hacking rather than systematic design, and also hides marginal people barely qualified for precision programming. – *Harlan D. Mills*

Write programs to handle text streams, because that is a universal interface.

(fine if you're a system programmer, not always so useful for scientific data crunching. Scientific data oftentimes binary.)

The UNIX philosophy was followed as strictly as possible in the development of GMT.

(set of 60+ stand alone "filters"/programs

35+ supplementary filters/programs)

Effective use of GMT is really effective application of the UNIX philosophy.

Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it. - *Kernigan*

output is "PostScript" program– generally ascii text, but not too readable. (GMT files can get amazingly BIG)

```
% Map boundaries
%
S 1050 1050 1050 0 360 arc S
S 1050 1050 1074 0 360 arc S
S 24 W
S 1050 1050 1062 -135 -90 arc S
S 1050 1050 1062 135 180 arc S
S 1050 1050 1062 45 90 arc S
S 1050 1050 1062 -45 0 arc S
S 1050 1050 1062 -90 -90 arcn S
S 2 W
S [] 0 B
%
% End of basemap
%
S [] 0 B
%%Trailer
%%BoundingBox: 0 0 647 647
% Reset translations and scale and call showpage
S -295 -295 T 4.16667 4.16667 scale 0 A
showpage
```

So what does/can GMT do?

Filtering 1-D and 2-D data

- simple processing
  - GMT is NOT a general Number Cruncher
    (use FORTRAN == FORmula TRANlator
    or
    C (followed languages A and B)
    or
    something else that's appropriate.)

- output is reprocessed data

So what does/can GMT do?

Filtering 1-D and 2-D data –

Plotting 1-D and 2-D data – includes

- points, lines (symbols, fill, geologic symbols on faults, etc.)
- vector fields
- 2-D images – grayscale and color, illumination
- 3-D perspective of 2-D images
- histograms, rose diagrams
- text
- focal mechanism beachballs

It's OK to figure out murder mysteries, but you shouldn't need to figure out code. You should be able to read it. *-Steve C McConnell*

So what does/can GMT do?

Filtering 1-D and 2-D data

Plotting 1-D and 2-D data

Data preparation

- gridding, resampling, conversion
- contouring
- data base: extraction, merge
- cross section
- projection/map transformation (map sphere to plane)
  ~ output is reprocessed data

Bookeeping and Bunch of other stuff

Program testing can be used to show the presence of bugs, but never to show their absence! -*Edsger Dijkstra, [1972]*

GMT documentation

i) Tutorial

ii) Technical Reference and Cookbook
– both available on web (http://gmt.soest.hawaii.edu/) as

HTML on line
PDF
PostScript

iii) UNIX "man" pages – available on web and also as standard man pages on local installation.

iv) Entering GMT program/filter name all by itself, or certain errors in command specification (switches, not data) – dumps man page to standard error.

Programs must be written for people to read, and only incidentally for machines to execute. *-Abelson and Sussman*

Tutorial –

Installation/Maintenance - done for us (by Mitch – THANKS.
Somewhat complicated, not for average user.)

Setup – basic setup done for us
(don't have to define GMTHOME, path, etc. if
use standard CERI .login and .cshrc files)

- Some common data sets (GTOPO-30, ETOPO-5,
Predicted bathy, etc.) are installed

- ".gmtdefaults" file in your home or working directory
(if you've copied something from the tutorial or
gotten it from someone else and it comes out
"funny", the "default" settings may be the culprit).

Tutorial –

Easiest way to get started

1) Find system with GMT already set up

2) Get working program (shell script) from someone else and modify (hack) it.

Lots examples in

- Tutorial

- available on www

- available from your "friends"

When I am working on a problem, I never think about beauty. I think only of how to solve the problem. But when I have finished, if the solution is not beautiful, I know it is wrong. ~ *R Buckminster Fuller*

Choice of shell/scripting language –

I use tcsh (variant of C Shell) as the basic interactive shell (I think this is the default at CERI. i.e. if you don't know what I'm talking about you're using this one.)

I use the "Bourne Shell" (sh) or "C Shell" (csh) for shell scritps, depending on where the "seed" came from. Other popular shells – ksh, bash, zsh, rc, es. (the "beauty"/"power" of UNIX allows you to roll your own if you don't like what's available.)

Minor differences between tcsh and csh or sh may mean that something that works from the command line does not work in a shell script. Frustrating.

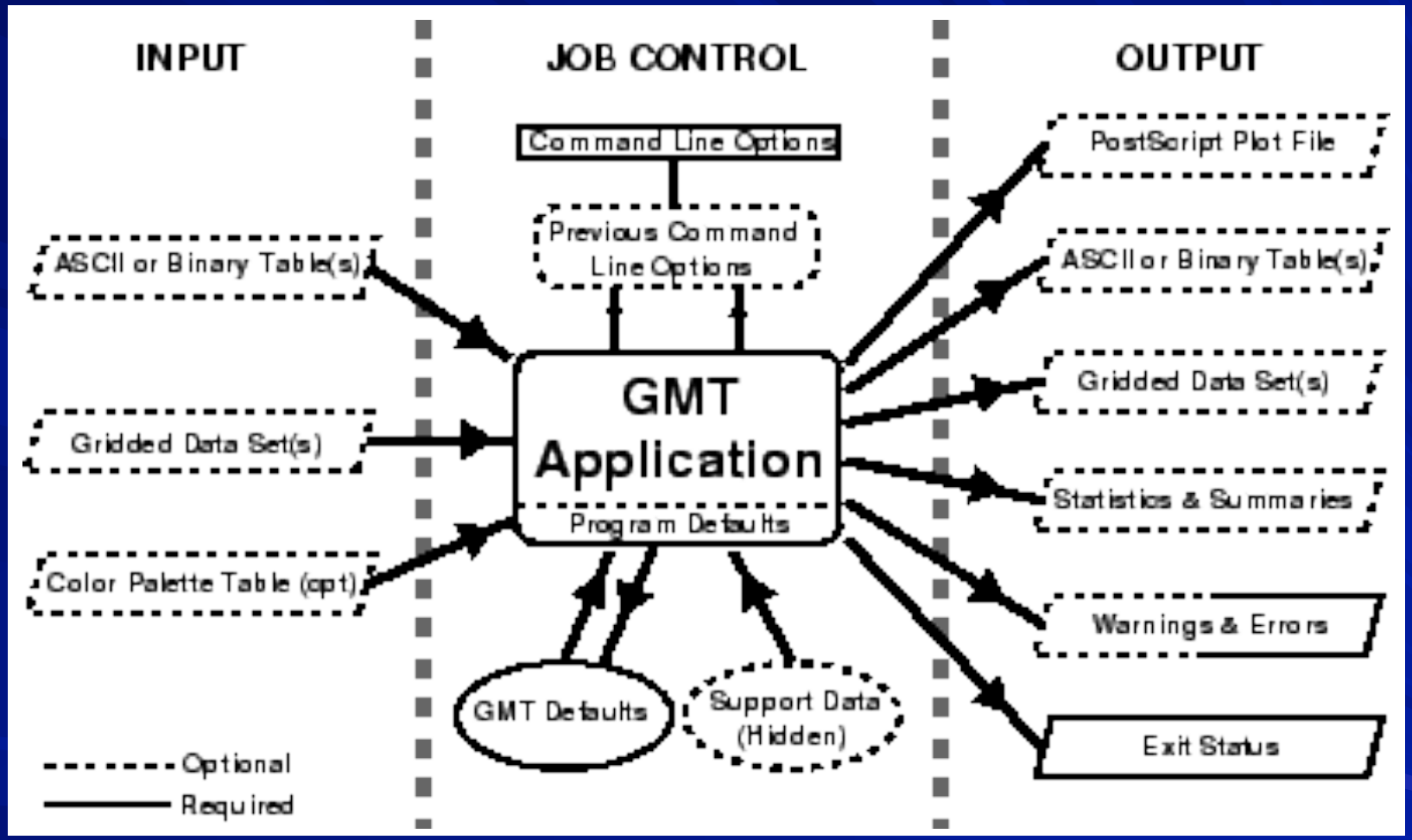
Choice of shell/scripting language –

from the GMT tutorial

"All the examples in this tutorial assumes (sic) you are running the cshell; if you are using something different then you are on your own."

I will take the same attitude – but most of my shells are in sh.

What goes on in GMT



Alzheimer's Law of Programming: Looking at code you wrote more than two weeks ago is like looking at code you are seeing for the first time. *-Via Dan Hurvitz*

Sources of operational parameters/job control

i) command line options/switches or program defaults

ii) carried over from execution of previous commands

III) from your .gmtdefaults file

(first in working directory if ∃

if ~∃, then home directory if ∃

finally, system/program defaults)

Why a defaults file?

- too many parameters to require setting all explicitly (powerful)
- customize – can have different defaults in different directories

Act in haste and repent at leisure; Code too soon and debug forever. -*Raymond Kennington*

Basic GMT use

Most GMT programs read input from terminal (*stdin*) or files, and write output to terminal (*stdout*) (a few write to files).
To write output to files one can use *UNIX* redirection:

```
GMTprogram switches
```

```
GMTprogram switches input-file
```

- Some GMT programs will accept input-file names, pipes and input redirection in lieu of stdin

- Some GMT programs require input-file names (usually if need more than one input file)

Basic GMT use

```
GMTprogram switches input-file > output-file

GMTprogram switches input-file >! output-file
```

-The exclamation point (!) overwrites existing files.
  This may fall over if the file is not pre-existing, behavior may depend on whether you are in an "interactive shell" or "shell script" and which shell you are using.

Basic GMT use

```
GMTprogram switches input-file >> output-file
```

- append output to existing file (cannot be combined with !)

```
GMTprogram switches < input-file >> output-file
```

- Some GMT programs will accept redirected input

```
Someprogram | GMTprogram1 | GMTprogram2 >>
Output-file
```

(or `| lp` if you are brave)

- prepare input using other program and PIPE to GMT
- Some GMT programs will accept piped input

Basic GMT use

```
GMTprogram switches << XXX >> output-file
…Stuff…
XXX
```

- Some GMT programs will accept in-line input – reads whatever follows -- up to character string XXX -- as input.

Usually looks something like

```
GMTprogram switches << END >> output-file
.1  .1
.2  .2
END
```

Can also do with "commnad substitution":

```
GMTprogram switches << FIN >> output-file
`someprogram swithches < input-file…`
FIN
```

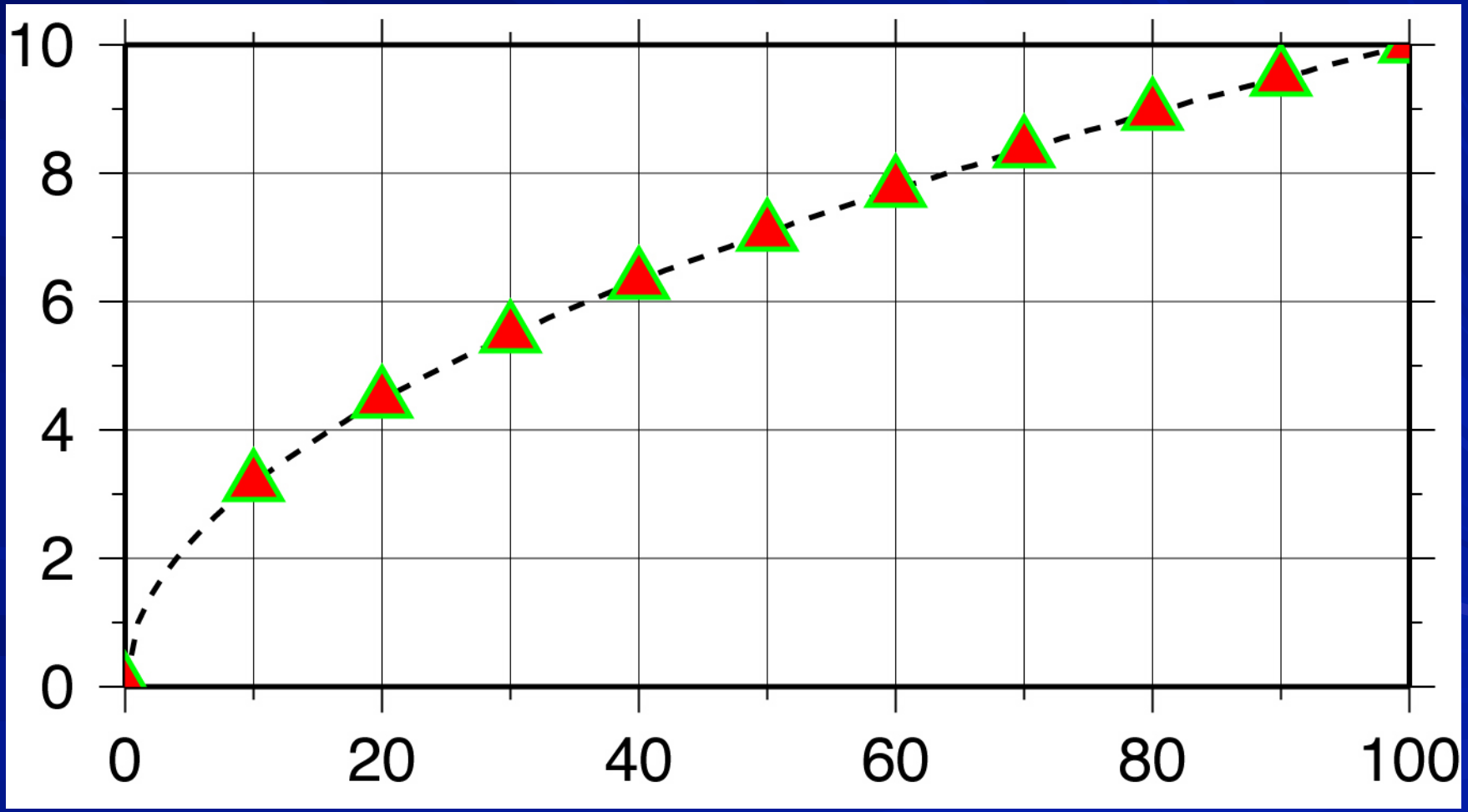
GMT uses all standard UNIX "features" ("tricks")

File name expansion ("wild cards")

`*`	Matches anything
`?`	Matches single character
`[list]`	Matches single characters from list
`[range]`	Matches single characters from range

These are actually specific cases of the more general "regular expression" in UNIX

OK lets look at some examples:



OK lets look at some examples:

1) We start by making the basemap frame for a linear *x*-*y* plot.

2) We want it to go from 10 to 70 in *x*, annotating every 10, and from -3 to 8 in *y*, annotating every 1.

3) The final plot should be 4 by 3 inches in size.

Note GMT does not make any helpful assumptions such as

a) You want to plot the whole x and y range of the data and

b) You want it to fit nicely on the page
You have to specify EVERYTHNG (comes under the excuse of being "powerful")

OK lets look at some examples:

Here's how we do it:

```
psbasemap -R10/70/-3/8 -JX4i/3i -B10/1:."My first
plot": -P >! plot.ps
```

We will first look at how the requirements above are specified to make the map.

This is done using the command line options/switches.

Requirements 1 and 3 are specified to GMT togehter

1) We start by making the basemap frame for a linear *x-y* plot.

3) The final plot should be 4 by 3 inches in size.

psbasemap draws a map frame and sets up the map parameters (so they don't have to be respecified in later GMT program calls)

The **-J** option selects the type of projection

In this case we want a linear x-y plot, or no projection, which is specified by

x or **X**.

There are 25 projections available in GMT, each specified by one letter.
There are no provisions for providing your own projection.

Requirements 1 and 3 are specified to GMT together

The **-J** option also sets the axis scales (distance per unit, **x**) or axis length (**X**)

Where the "unit" is specified in .gmtdefaults or explicitly – inches, **i**, or cm, **c**.

```
psbasemap -R10/70/-3/8 -JX4i/3i -B10/1:."My first
plot": -P >! plot.ps
```

2) We want it to go from 10 to 70 in *x*, annotating every 10, and from -3 to 8 in *y*, annotating every 1.

This is really two conditions

i) We want it to go from 10 to 70 in *x*, and from -3 to 8 in *y*.

Specified by the REGION (-R) option, which (in the usual form) is

`-Rxmin/xmax/ymin/ymax`

2) We want it to go from 10 to 70 in *x*, annotating every 10, and from -3 to 8 in *y*, annotating every 1.

-R*xmin/xmax/ymin/ymax*

Notice that unlike MATLAB, GMT does not make any assumptions about what you want (such as the reasonable one that you just might want the region to show all the input data).

You have to specify every detail. (i.e. powerful)

```
psbasemap -R10/70/-3/8 -JX4i/3i -B10/1:."My first
          plot": -P >! plot.ps
```

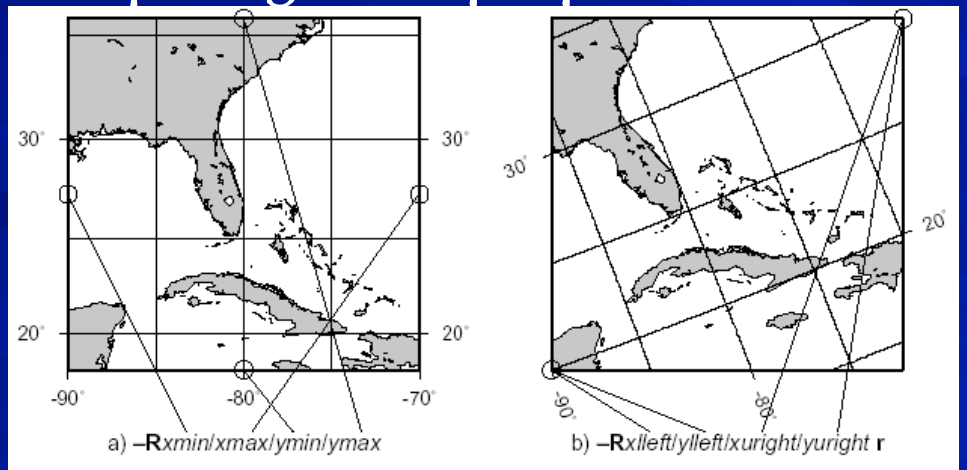
"Feature it" – response of GenRad Development Engineer Dick Benson to bug reports.

There are two forms for the –R option

1) For projections where the boundaries follow lines of latitude and longitude ("rectangle" on sphere)– specify sides

2) For regions where the sides do not follow lines of latitude and longitude (will make more sense when we do map projections)– specify corners by appending an "r" to end

Idea of "region" to plot specified this way breaks down for azimuthal projections (outside border of plot is a circle, you really want to specify center and radius) – will see some



2) We want it to go from 10 to 70 in x, annotating every 10, and from -3 to 8 in *y*, annotating every 1.

ii) We want to annotate x every 10, and *y* every 1.

This is specified by the **-B** option (Border?).

This is the most complicated GMT option. Two features are used

Annotation – every 10 for x (first one) and every 1 for y (second one). If you wanted the same annotation for x and y you would not have to do it twice

```
psbasemap -R10/70/-3/8 -JX4i/3i -B10/1:."My first plot": -P >! plot.ps
```

---

it is better to solve the right problem the wrong way than the wrong problem the right way
- McIlroy

Not in specs, but controlled by the **-B** option, the plot title.

This is a little more complicated.

Labels are between colons, with

" . " for plot title,
nothing for x axis label,
" , ' \ " for *y* axis label.

If label/title is more than one word, has to be in double quotes.

```
psbasemap -R10/70/-3/8 -JX4i/3i -B10/1:."My first
plot": -P >! plot.ps
```

---

it is better to solve the right problem the wrong way than the wrong problem the right way
- McIlroy

If this sounds confusing you can look at the man page for psbasemap for the full explanation and more examples.

The man page for the –B option, however, is practically incomprehensible.

The BUGS section of the man page states

"The -B option is somewhat complicated to explain and comprehend. However, it is fairly simple for most applications (see examples). "

Remaining options/switches

-P

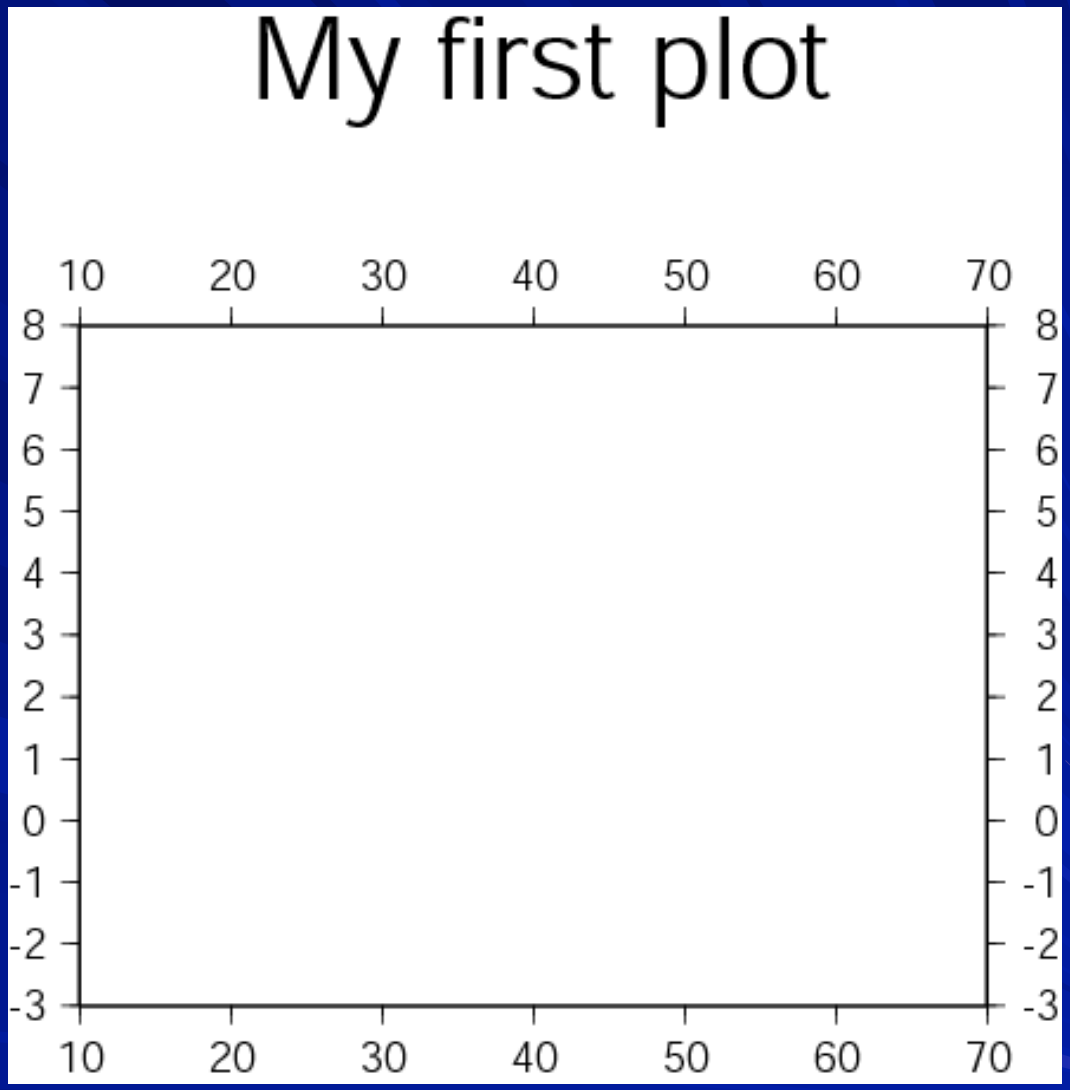
Sets the output to Portrait (long side vertical) mode.
"Default" is Landscape (long side horizontal) mode.

```
psbasemap -R10/70/-3/8 -JX4i/3i -B10/1:."My first
plot": -P >! plot.ps
```

This option actually switches "states".

If .gmtdefaults defines portrait mode as the default, then **-P** will send it to landscape.
(make a figure and see how it comes out, if you don't like the orientation stick in a **-P**).

So, what did we get for all our effort?



Good start – but usually we make plots to show some sort of data – so how do we do that?

Now let's look at a more complicated example:
Lets call it "`full_court_press.sh`"

```
#!/bin/sh
#plot square root x
sample1d -I1 << END | nawk '{print $1, sqrt($1)}' > {$0}_1.dat
0
100
END
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2f1g2WSne -W5t15_15:0 \
-Y2 -P {$0}_1.dat -K > $0.ps
sample1d {$0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \
-G255/0/0 -W5/0/255/0 -O >> $0.ps
```

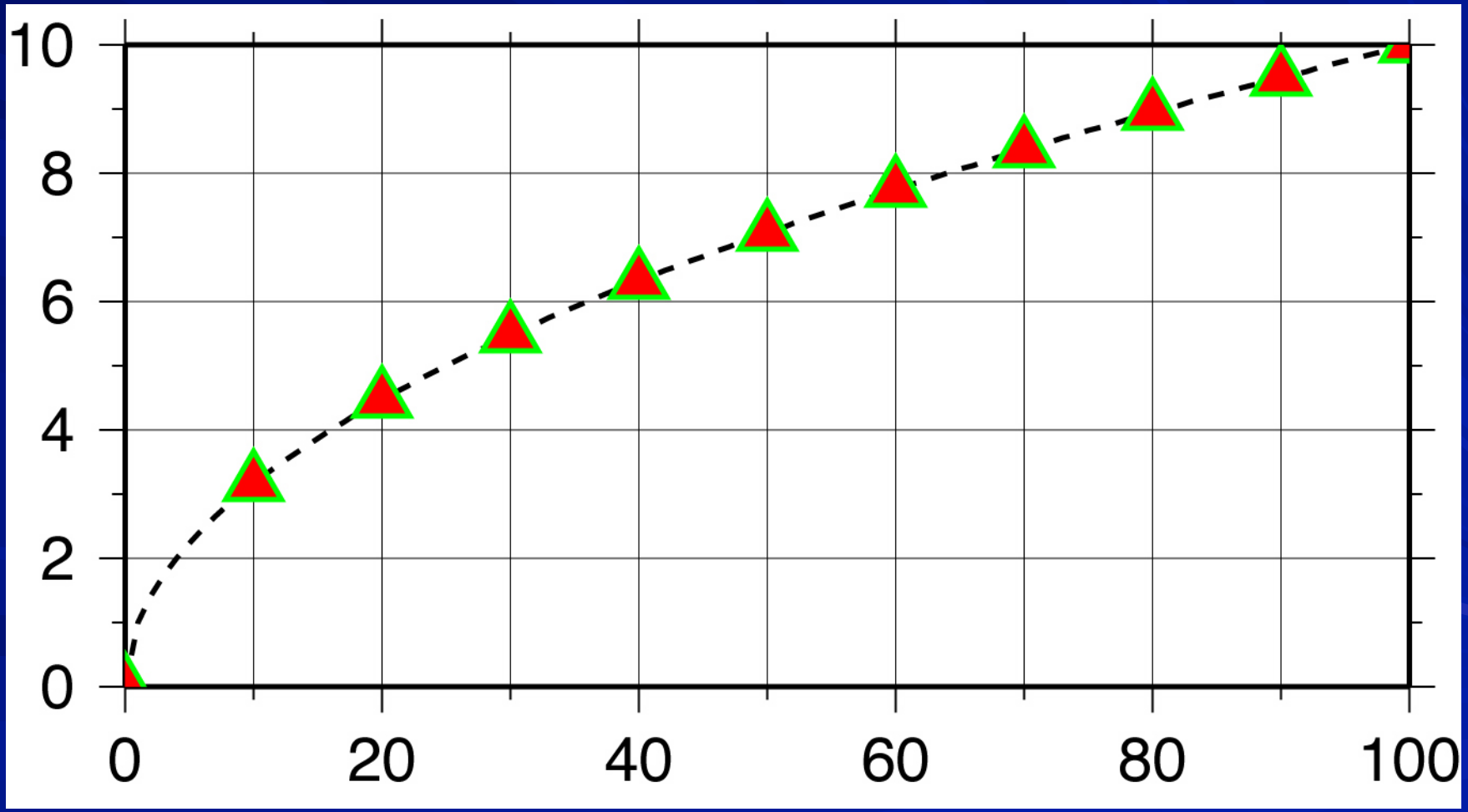
This is a little more than "a little more" complicated.

But it follows the UNIX philosophy – a bunch of simple things stuck together to do something more complex.

Gives you the idea that most useful GMT produced figures are going to be a LOT of GMT calls

Here's what the output looks like

(actually the output is a ascii file containing a PostScript program, this is what it looks like after displaying with GhostScript or printing to a PostScript printer).



Let's look at it piece, buy simple piece.

Set shell

```
#!/bin/sh
#plot square root x
sample1d -I1 << END | nawk '{print $1,
...
```

Set shell to Bourne Shell.

Could also have set it to csh (change first line to **`#!/usr/bin/csh -f`**, this works because this script does not contain anything that is specific to one shell script – such as variable name definition. Use **`-f`**, fast, option which stops it from running your .cshrc).

Next piece

Name the output file.

```
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2f1g2WSne -W5t15_15:0 \
-Y2 -P {$0}_1.dat -K > $0.ps
sample1d {$0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \
-G255/0/0 -W5/0/255/0 -O >> $0.ps
```

Being lazy and disorganized - I don't want to have to type the output file name in lots of times nor keep track of which shell script made which postscript file in my directory.

So I want to find a short and easy way to name the file and I might want to associate the output file name with the name of the shell script that made it.

Enter UNIX argument passing.

When you call a shell script, the system passes a max of 10 predefined, pre-named "arguments" to the shell script.

So if I enter

"`myscript arg1 arg2`"

UNIX automatically gives me (in this case 3 arguments)

`$0`	the name of the shell script
`$1`	the value of arg1 (character string)
`$2`	the value of arg2

All I have to do to use these arguments in my shell script (within some constraints) is stick them in.

The Shell will expand them to their proper values.

So my output file will be named
"full_court_press.sh.ps",

since `$0` will get expanded to "full_court_press.sh"
(the name of the shell script)

Next piece.

Get (actually make) input data – part 1

```
#!/bin/sh
#plot square root x
sample1d -I1 << END | nawk '{print $1, sqrt($1)}' > {$0}_1.dat
0
100
END
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2f1g2WSne -W5t15_15:0 \
-Y2 -P {$0}_1.dat -K > $0.ps
sample1d {$0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \
-G255/0/0 -W5/0/255/0 -O >> $0.ps
```

sample1d, resamples the input - which in this case is redirected (**<<**) to being obtained in-line from this file (from end of the command line – which is somewhat far away – to **END**).

Next piece.
Get (actually make) input data – part 1

```
#!/bin/sh
sample1d -I1 << END | nawk '{print $1, sqrt($1)}' > {$0}_1.dat
0
100
END
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2f1g2WSne -W5t15_15:0 \
-Y2 -P {$0}_1.dat -K > $0.ps
sample1d {$0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \
-G255/0/0 -W5/0/255/0 -O >> $0.ps
```

We have to sepcify the resampling step (**-I1**, which is steps of 1).

We will leave everything else at the default values (which column is the independent variable: zero, type of interpolation, etc.)

sample1d provides to standard out a list of numbers from 0 to 100 in steps of 1.

Next piece.

Get input data – part 2
We want x and sqrt(x)

```
#!/bin/sh
sample1d -I1 << END | nawk '{print $1, sqrt($1)}' > {$0}_1.dat
0
100
END
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2f1g2WSne -W5t15_15:0 \
-Y2 -P {$0}_1.dat -K > $0.ps
sample1d {$0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \
-G255/0/0 -W5/0/255/0 -O >> $0.ps
```

Pipe the resampled data into the program nawk, a handy dandy UNIX tool that processes ascii data files a line at a time.

nawk is a great tool for preprocessing data for GMT.

Next piece.
Generate input data – part 2

Using nawk, one does not have to write programs to make intermediate files in GMT input format, but can go right to the source data file, read it, modify each line into GMT input format and pipe this directly into the GMT program.

```
sample1d -I1 << END | nawk '{print $1, sqrt($1)}' | psxy -
R0/100/0/10 \
-JX4/2 -Ba20f10g10/a2f1g2WSne -W5t15_15:0 -Y2 -P -K > $0.ps
0 0
100 0
END
```

The nawk command says to print the first column (**$1**) and the square root of the first column (**sqrt($1)**)of every line.

We will (break the UNIX philosophy and) make an intermediate file as we will need it more than once.

Next piece.

Plot it

```
#!/bin/sh
#plot square root x
sample1d -I1 << END | nawk '{print $1, sqrt($1)}' > {$0}_1.dat
0
100
END
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2f1g2WSne -W5t15_15:0 \
-Y2 -P {$0}_1.dat -K > $0.ps
sample1d {$0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \
-G255/0/0 -W5/0/255/0 -O >> $0.ps
```

Finally we get to the subject of this ToT – GMT

psxy is the GMT program that plots points and lines.

Next piece.

Plot it

```
#!/bin/sh
#plot square root x
sample1d -I1 << END | nawk '{print $1, sqrt($1)}' > {$0}_1.dat
0
100
END
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2f1g2WSne -W5t15_15:0 \
-Y2 -P {$0}_1.dat -K > $0.ps
sample1d {$0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \
-G255/0/0 -W5/0/255/0 -O >> $0.ps
```

psxy accepts the "standard"/"global" options of the GMT filters that produce PostScript output.

We already know what **-R**, **-J**, **-B** and **-P** do, although the **-B** option here is a bit more complicated looking.

Next piece.

Plot it

```
#!/bin/sh
#plot square root x
sample1d -I1 << END | nawk '{print $1, sqrt($1)}' > {$0}_1.dat
0
100
END
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2f1g2WSne -W5t15_15:0 \
-Y2 -P {$0}_1.dat -K > $0.ps
sample1d {$0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \
-G255/0/0 -W5/0/255/0 -O >> $0.ps
```

Output file `$0.ps` (new - is first instance, append in second – this takes care of UNIX part)

Use \ to continue command on next line

Next piece.

```
...
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2f1g2WSne -W5t15_15:0 \
-Y2 -P {$0}_1.dat -K > $0.ps
...
```

So, what's all that extra stuff on the **-B**? Each of the letters controls a different feature/aspect of the plotting of the axis

a is for annotation spacing

f is for frame (famous map frame of black/white bars – turned off for x/X, ticks)

g is for grid spacing

WSne says to plot the annotation and ticks on the **W**est and **S**outh sides and ticks only on the **n**orth and **e**ast sides. (how would you put annotation without ticks?)

Next piece.

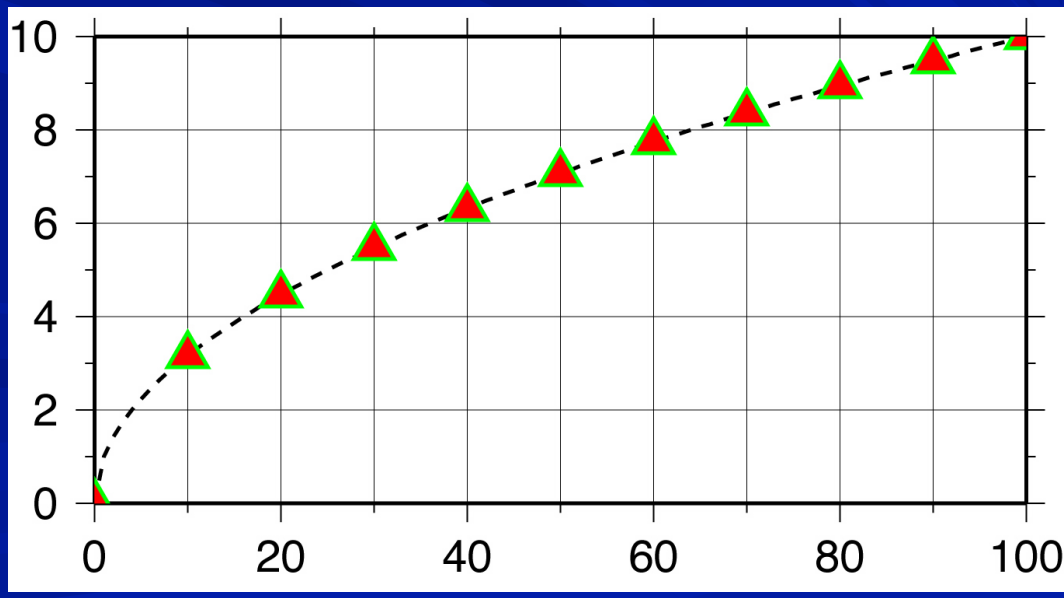
Draw a line `-W5t15_15:0`

```
...
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2f1g2WSne -W5t15_15:0 \
-Y2 -P {$0}_1.dat -K > $0.ps
...
```

Make line 5 units thick (where units depends on the device and default settings) `-W5t15_15:0`

Make it dashed with dashes 15 units long followed by 15 unit long open spaces `-W5t15_15:0`

And a phase offset for the dashes of zero -W5t15_15:0

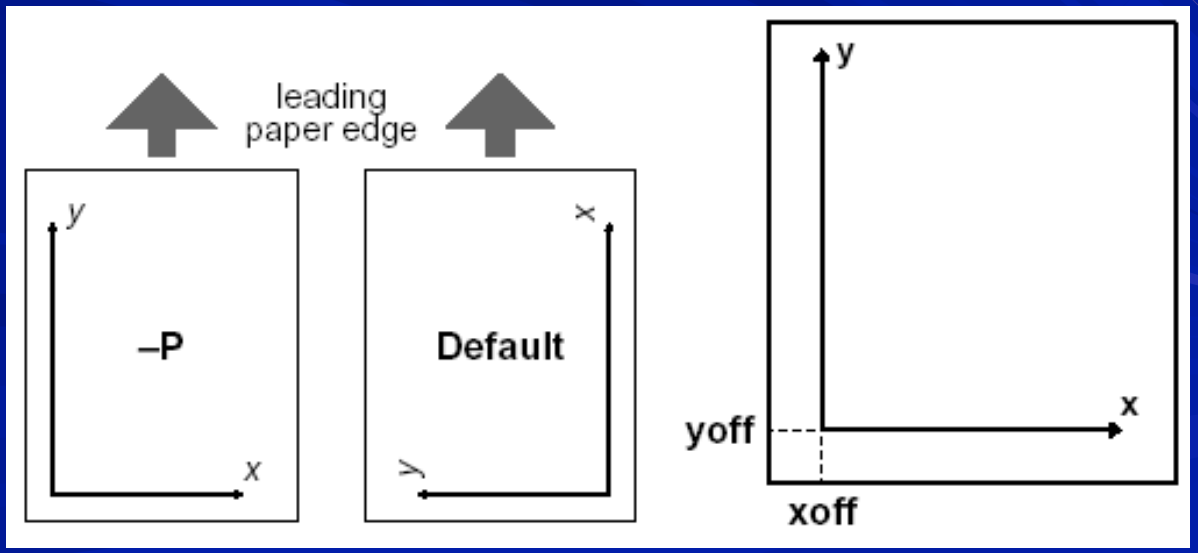


Next piece.

Misc. 1

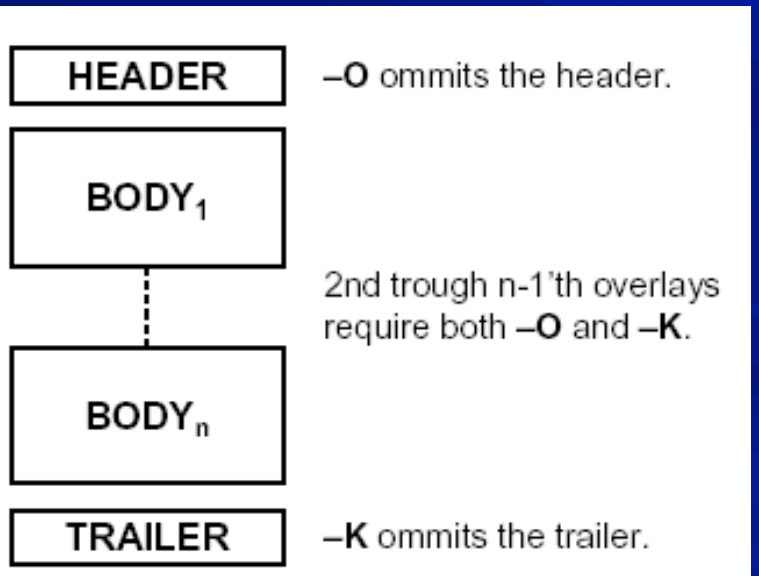
```
#!/bin/sh
#plot square root x
sample1d -I1 << END | nawk '{print $1, sqrt($1)}' > {$0}_1.dat
0
100
END
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2f1g2WSne -W5t15_15:0 \
-Y2 -P {$0}_1.dat -K > $0.ps
sample1d {$0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \
-G255/0/0 -W5/0/255/0 -O >> $0.ps
```

-Y2 offset plot 2 units in the Y direction (else x axis labels get cut off across bottom of plot)



Misc. 2

```
#!/bin/sh
#plot square root x
sample1d -I1 << END | nawk '{print $1, sqrt($1)}' > {$0}_1.dat
0
100
END
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2f1g2WSne -W5t15_15:0 \
-Y2 -P {$0}_1.dat -K > $0.ps
sample1d {$0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \
-G255/0/0 -W5/0/255/0 -O >> $0.ps
```



-K do not close PostScript file (don't output "showpage") so more PostScript can be appended to the file

-O do not initialize PostScript (does not output PostScript header stuff) so this can be appended to existing file (that hopefully does not have a showpage at the end)

Misc. 3

```
...
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2f1g2WSne -W5t15_15:0 \
-Y2 -P {$0}_1.dat -K > $0.ps
sample1d {$0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \
-G255/0/0 -W5/0/255/0 -O >> $0.ps
```

Several common "gotchas"

– no showpage (can see on screen, but does not print – actually prints a blank page) (have a **-K** in last GMT call)

- showpage in middle of file (forgot the **-K** somewhere) – only get part of file on screen or in final print or get ghostscript error message.

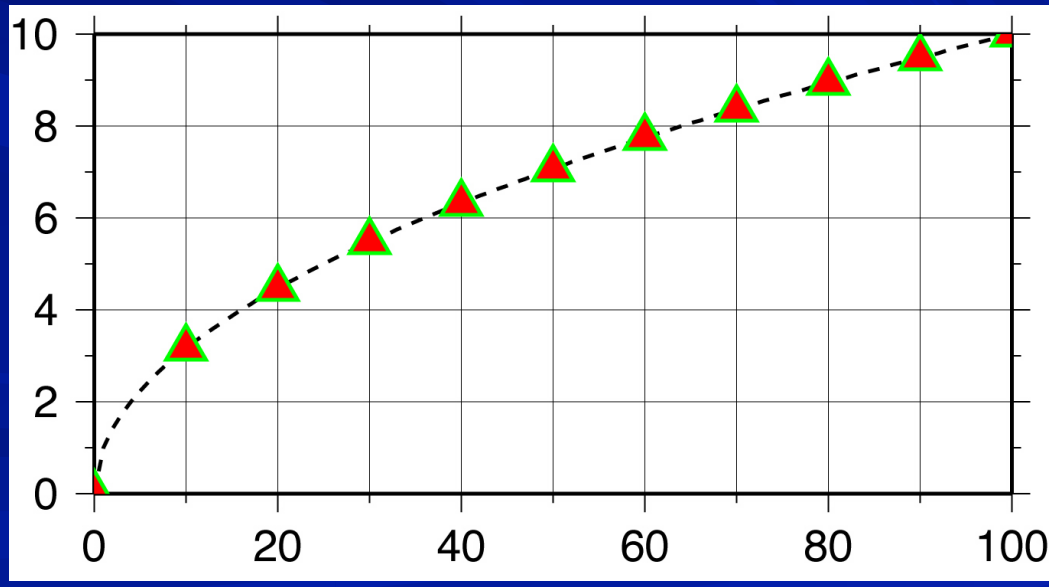
- Have header in middle of file (forgot **-O** somewhere), get ghostscript error message.

Next piece.
Draw symbols every 10th point

```
#!/bin/sh
#plot square root x
sample1d -I1 << END | nawk '{print $1, sqrt($1)}' > {$0}_1.dat
0
100
END
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2f1g2WSne -W5t15_15:0 \
-Y2 -P {$0}_1.dat -K > $0.ps
sample1d {$0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \
-G255/0/0 -W5/0/255/0 -O >> $0.ps
```

Resample our temporary file – taking every 10th point

Pipe output to psxy



Next piece.
Draw symbols -St0.2

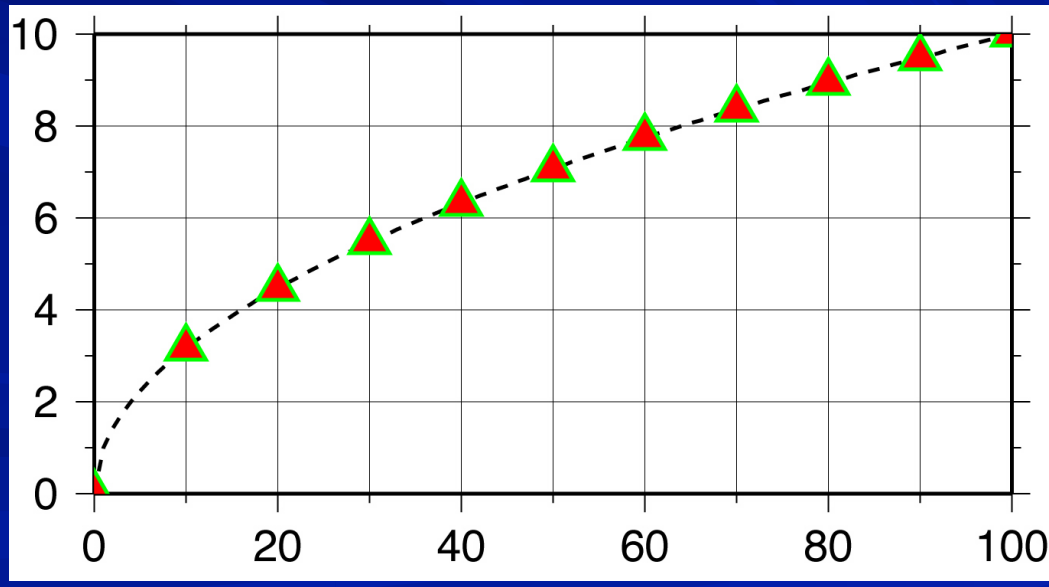
```
...
sample1d {$0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \
-G255/0/0 -W5/0/255/0 -O >> $0.ps
```

Make triangles, t, 0.2 units big -St0.2

Make line outlining/drawing the symbols 5 units wide, and draw them in green (R/G/B) -W5/0/255/0

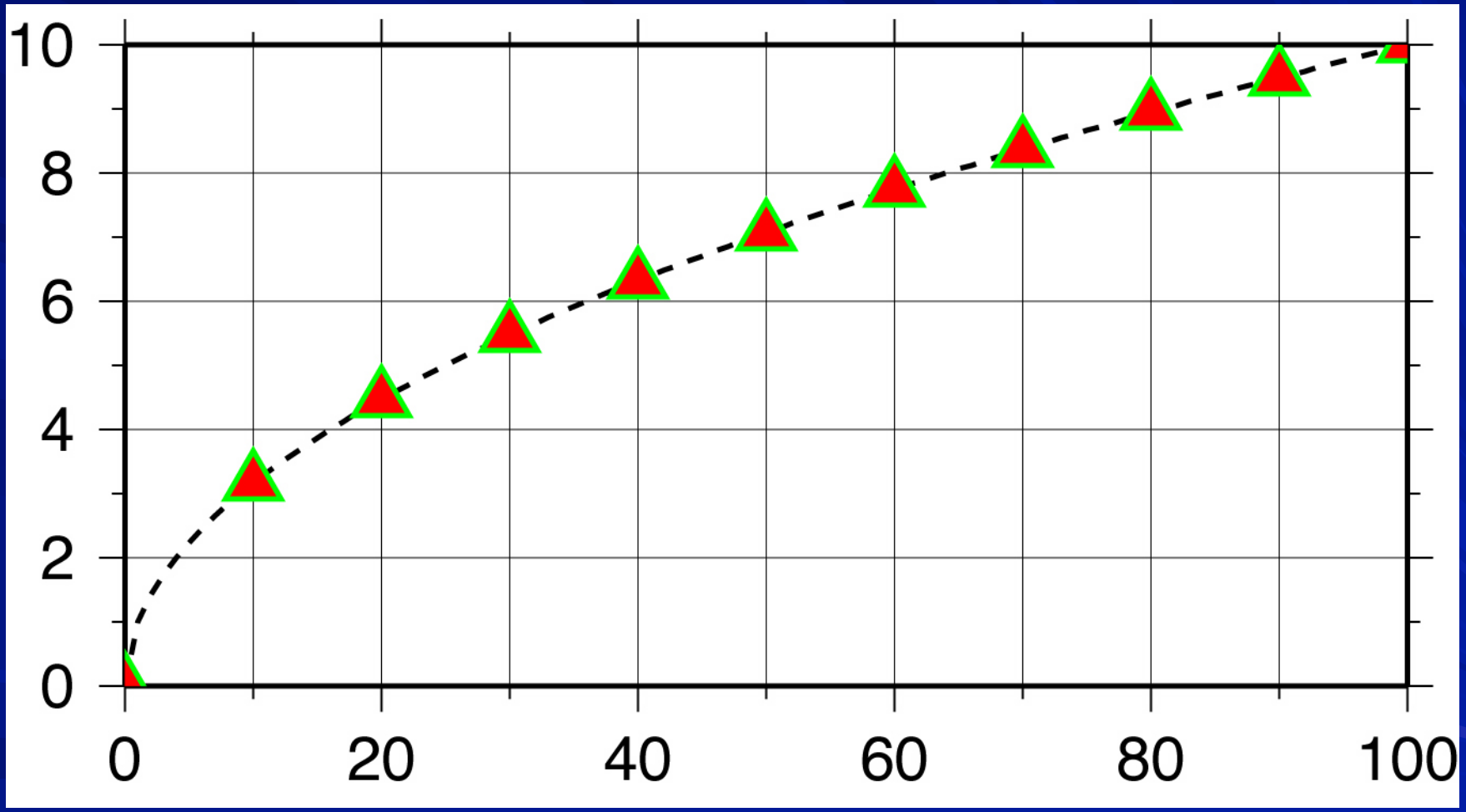
Fill symbols color red (R/G/B) -G255/0/0

Colors specified in R/G/B format (intensity of Red, Green and Blue color guns – primary colors for additive system.



We're done!

That wasn't so bad now, was it?

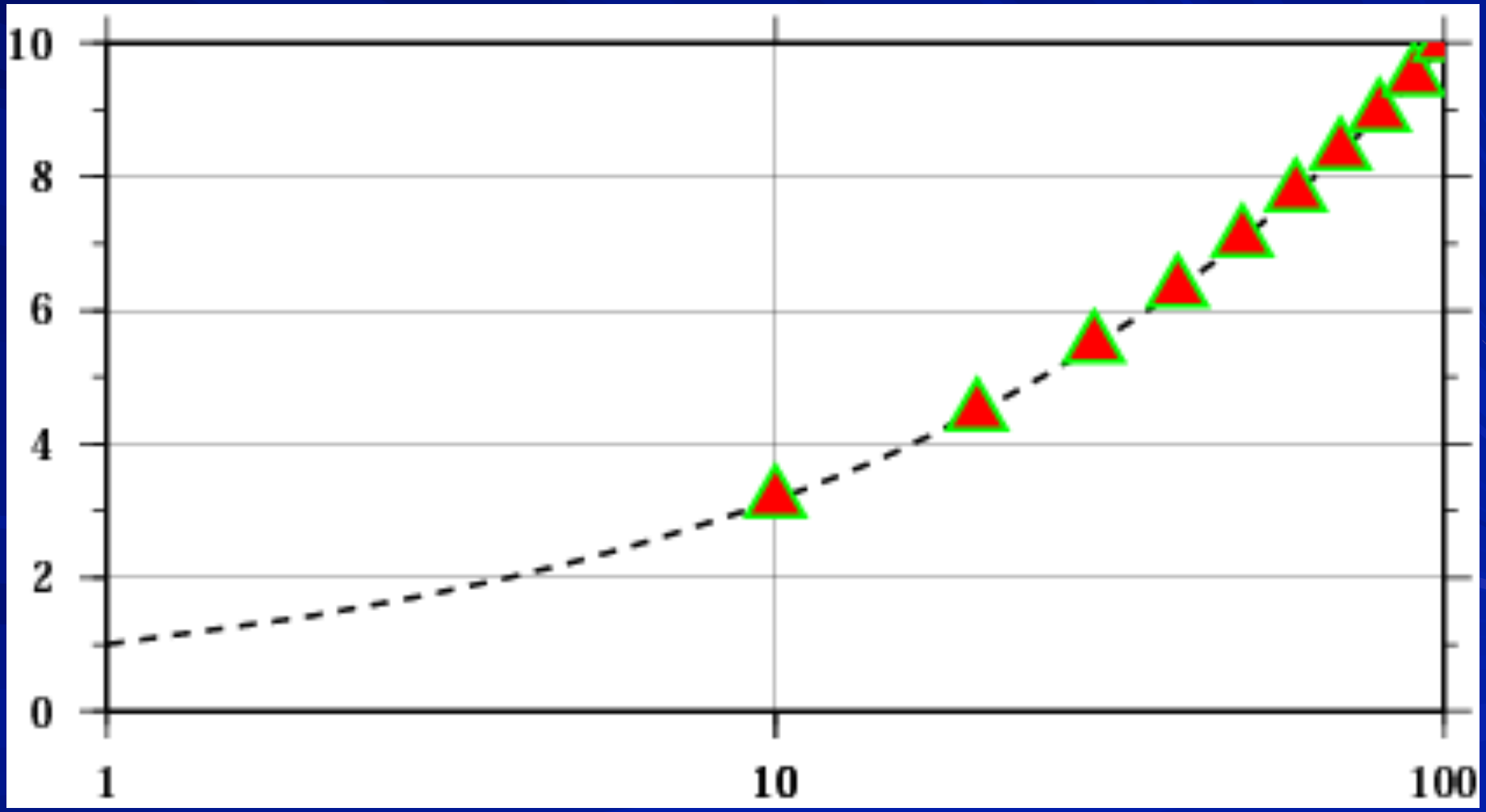


There are two other non-map-projected forms of the x/X projection

Two other non-map-projected forms

First

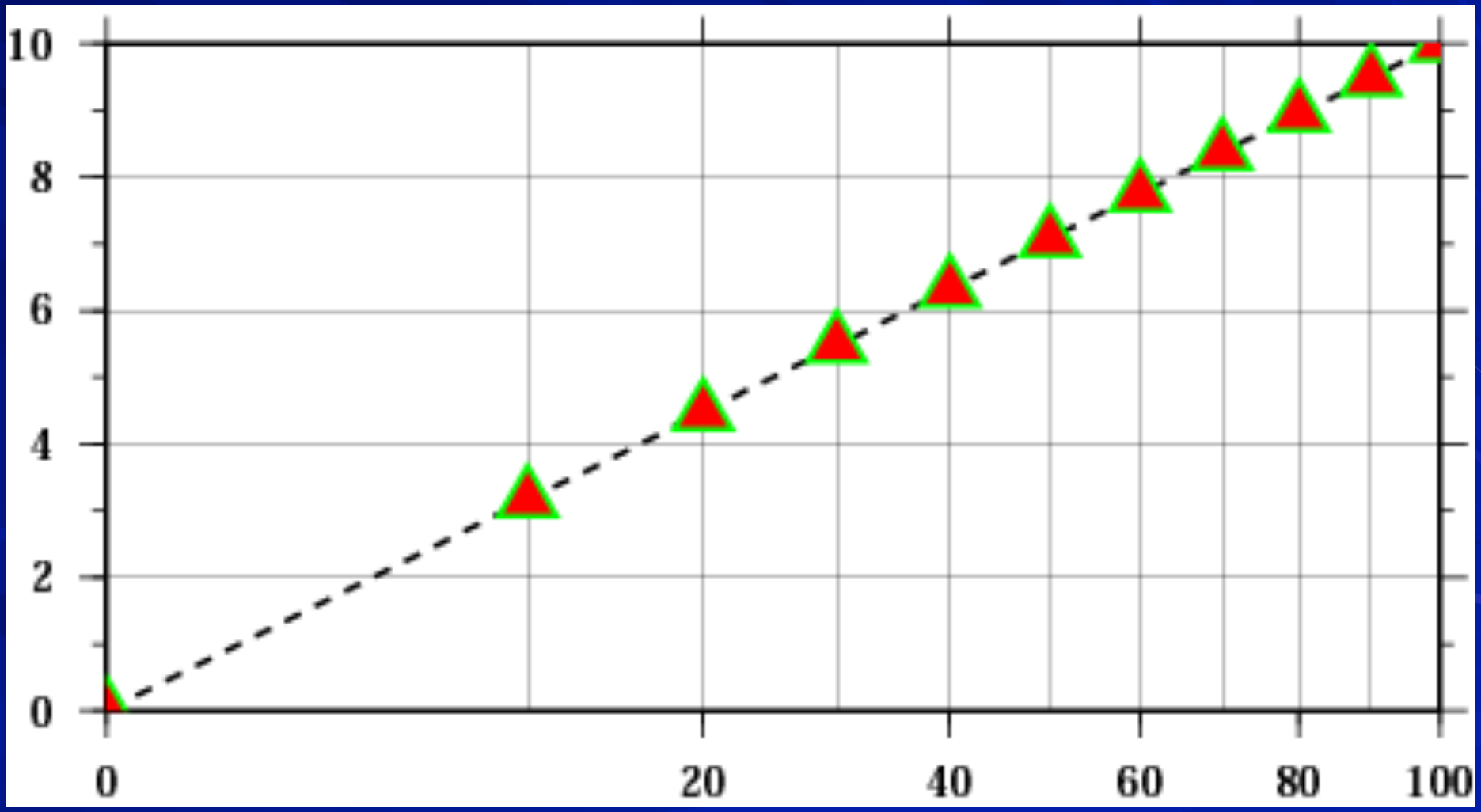
Logarithmic - add `l` (lower case letter L) after scale of axis you want logarithmic `-JX4l/2`



Second

Power/exponential – add `p` and `exponent` after scale of axis you want exponential (can scale axes individually)

`-JX4p0.5/2`



Common command options on first, and possibly subsequent, calls

Need on all calls

-R Define region for plot – will need on first call and at least "–R" on subsequent

-J define projection for plot – will need this on all calls if need to define region

Common command options on first, and possibly subsequent, calls

(Generally) Need on first call only

-B Borders -- annotation, frame, grid. Only need on first (or a single) call.

-P Switch between landscape and portrait modes

-X Shift X axis

-Y Shift Y axis

Common command options on first, and possibly subsequent, calls

Need when needed

-K Don't close PostScript (showpage), use when more will follow

need on all but last GMT call

-O Don't initialize PostScript, use when appending to pre-existing file

- need on all but first GMT call
- use both **-K** and **-O** when putting a large number of GMT call outputs together

Common command options on first, and possibly subsequent, calls

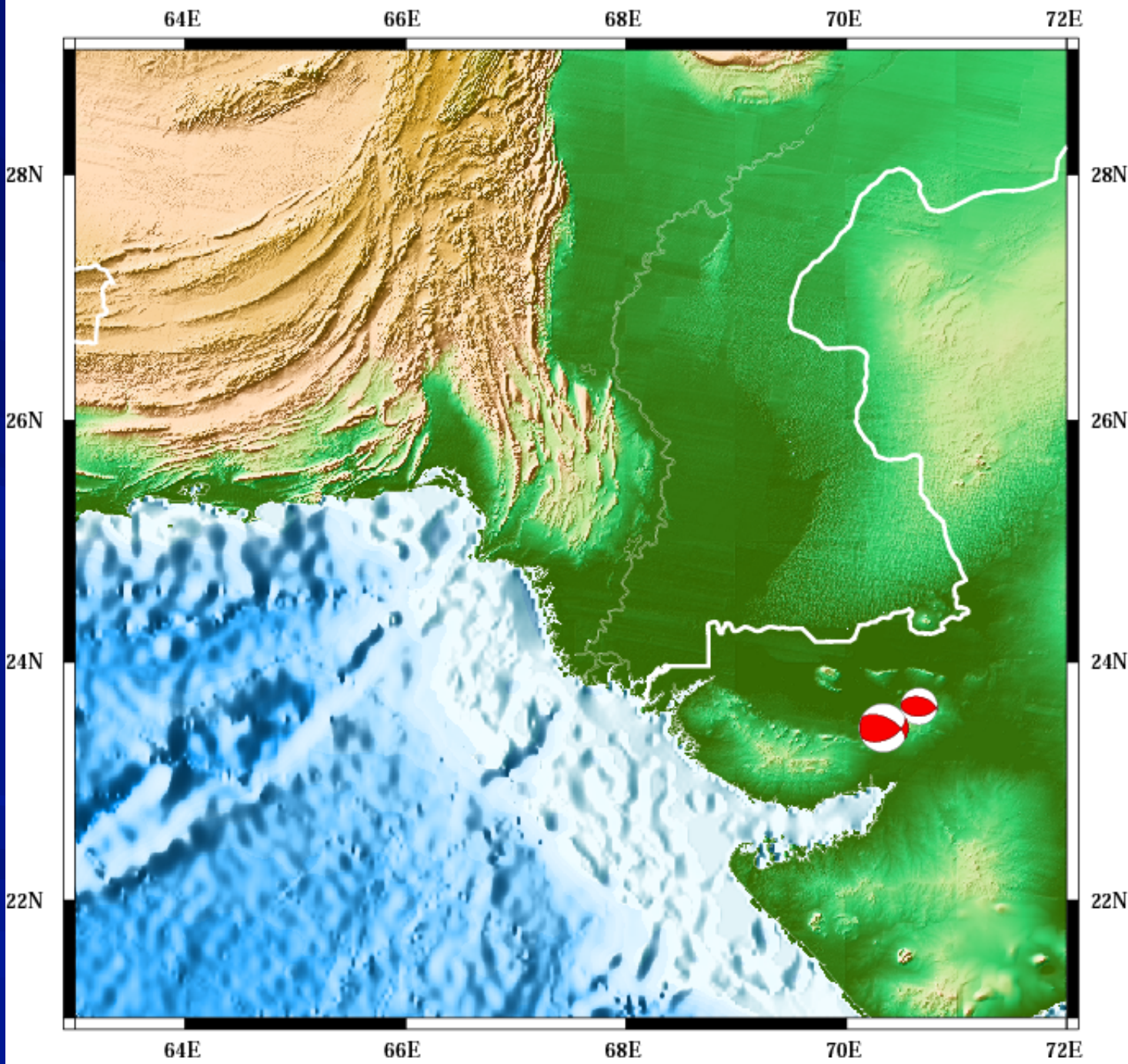
Need when needed

-V Verbose (prints out stuff to standard error for user).

-H Header records (tells GMT to skip first H lines of ascii input file)

How about making pretty MAPS?

(this was made by the shell script I put in Mitch's GMT-ToT web page.)



Map projections available in GMT

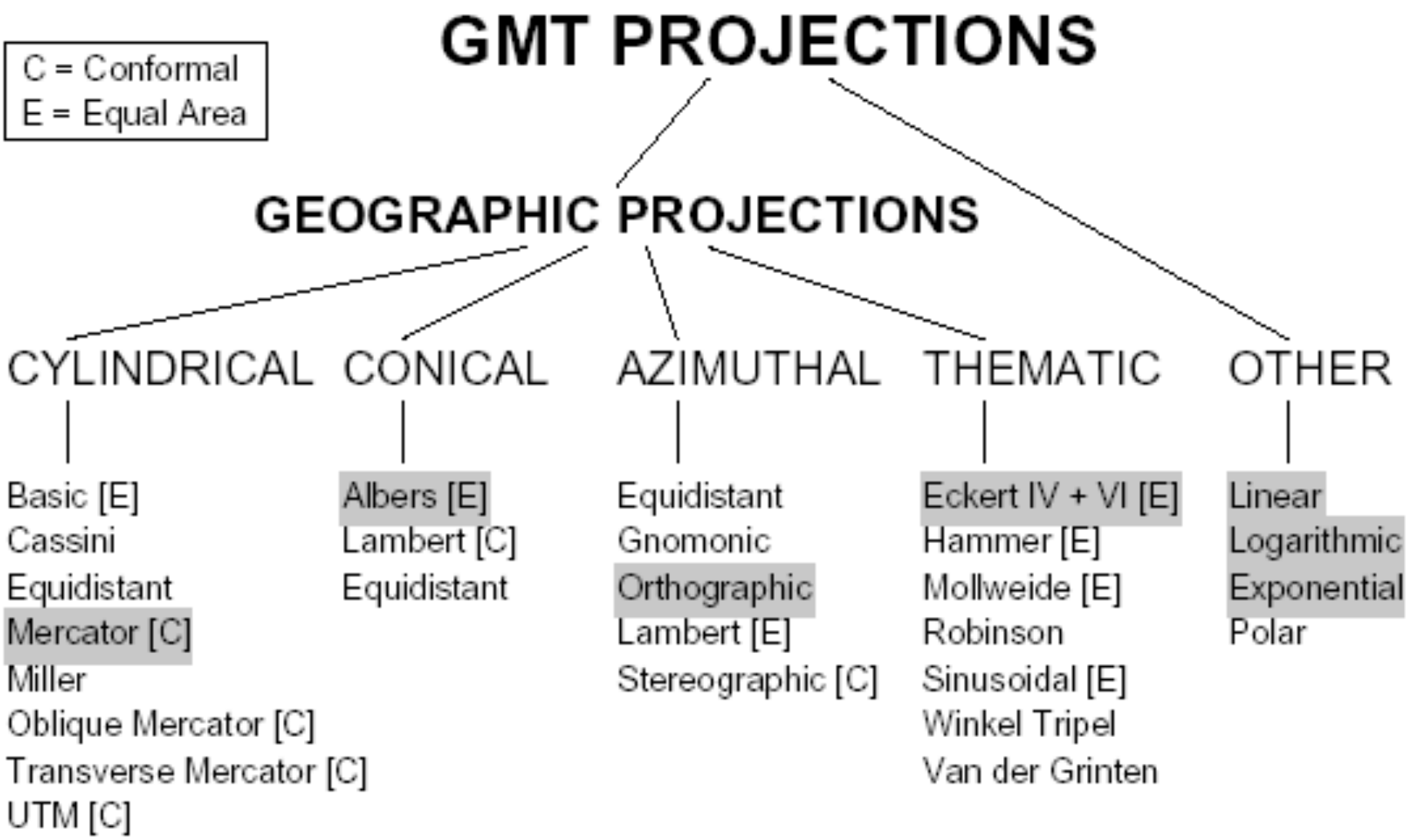


Figure 1.9: The 25 projections available in **GMT**.

List of "standard" command line options.

The **-J** option sets the "projection"

One has to look at the man page for each one as "different things vary"

We will now look at the examples from the tutorial

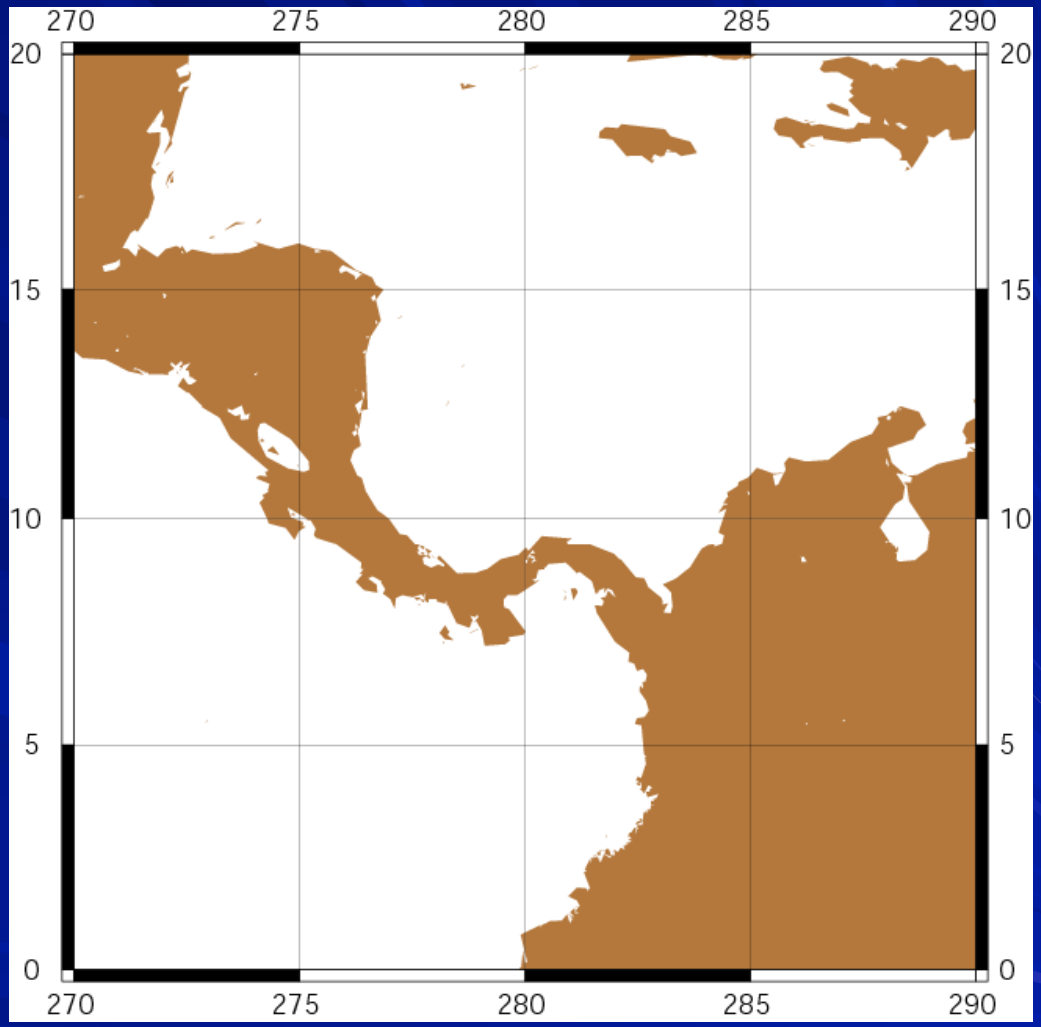
STANDARDIZED COMMAND LINE OPTIONS	
-Bx*info*[*/yinfo*[*/zinfo*]][*WESNZwesnz+*][*:.title:*] Tickmarks. Each *info* is [**a**]*tick*[**m**|**c**][f*tick*[**m**|**c**]][g*tick*[**m**|**c**]][**l**|**p**][:"label":][:,"unit":]	
-H[*nheaders*]	ASCII tables have header record[*s*]
-J (upper case for width, lower case for scale)	Map projection (see below)
-JA*lon*$_0$*/lat*$_0$*/width*	Lambert azimuthal equal area
-JB*lon*$_0$*/lat*$_0$*/lat*$_1$*/lat*$_2$*/width*	Albers conic equal area
-JC*lon*$_0$*/lat*$_0$*/width*	Cassini cylindrical
-JD*lon*$_0$*/lat*$_0$*/lat*$_1$*/lat*$_2$*/width*	Equidistant conic
-JE*lon*$_0$*/lat*$_0$*/width*	Azimuthal equidistant
-JF*lon*$_0$*/lat*$_0$*/horizon/width*	Azimuthal Gnomonic
-JG*lon*$_0$*/lat*$_0$*/width*	Azimuthal orthographic
-JH*lon*$_0$*/width*	Hammer equal area
-JI*lon*$_0$*/width*	Sinusoidal equal area
-JJ*lon*$_0$*/width*	Miller cylindrical
-JKf*lon*$_0$*/width*	Eckert IV equal area
-JKs*lon*$_0$*/width*	Eckert VI equal area
-JL*lon*$_0$*/lat*$_0$*/lat*$_1$*/lat*$_2$*/width*	Lambert conic conformal
-JM*width* or **-JM***lon*$_0$*/lat*$_0$*/width*	Mercator cylindrical
-JN*lon*$_0$*/width*	Robinson
-JOa*lon*$_0$*/lat*$_0$*/az/width*	Oblique Mercator, 1: origin and azimuth
-JOb*lon*$_0$*/lat*$_0$*/lon*$_1$*/lat*$_1$*/width*	Oblique Mercator, 2: two points
-JOc*lon*$_0$*/lat*$_0$*/lon*$_p$*/lat*$_p$*/width*	Oblique Mercator, 3: origin and pole
-JP[**a**]*width*[*/origin*]	Polar [azimuthal] (θ, r) (or cylindrical)
-JQ*lon*$_0$*/width*	Equidistant cylindrical (Plate Carrée)
-JR*lon*$_0$*/width*	Winkel Tripel
-JS*lon*$_0$*/lat*$_0$*/width*	General stereographic
-JT*lon*$_0$*/width*	Transverse Mercator
-JU*zone/width*	Universal Transverse Mercator (UTM)
-JV*lon*$_0$*/width*	Van der Grinten
-JW*lon*$_0$*/width*	Mollweide
-JX*width*[**l**|**p**][*/height*[**l**|**p**]][**d**]	Linear, $\log_{10}$, and x^a–y^b (exponential)
-JY*lon*$_0$*/lat*$_s$*/width*	General cylindrical equal area
-K	Append more PS later
-O	This is an overlay plot
-P	Select Portrait orientation
-R*west/east/south/north*[*/zmin/zmax*][**r**]	Specify Region of interest
-U[*/dx/dy/*][*label*]	Plot time-stamp on plot
-V	Run in verbose mode
-X[**a**]*off*	Shift plot origin in *x*-direction
-Y[**a**]*off*	Shift plot origin in *y*-direction
-c*copies*	Set number of plot copies [1]
-:	Expect *y*/*x* input rather than *x*/*y*

```
pscoast -R-90/-70/0/20 -JM6i -P -B5g5 -G180/120/60 > map1.ps
```

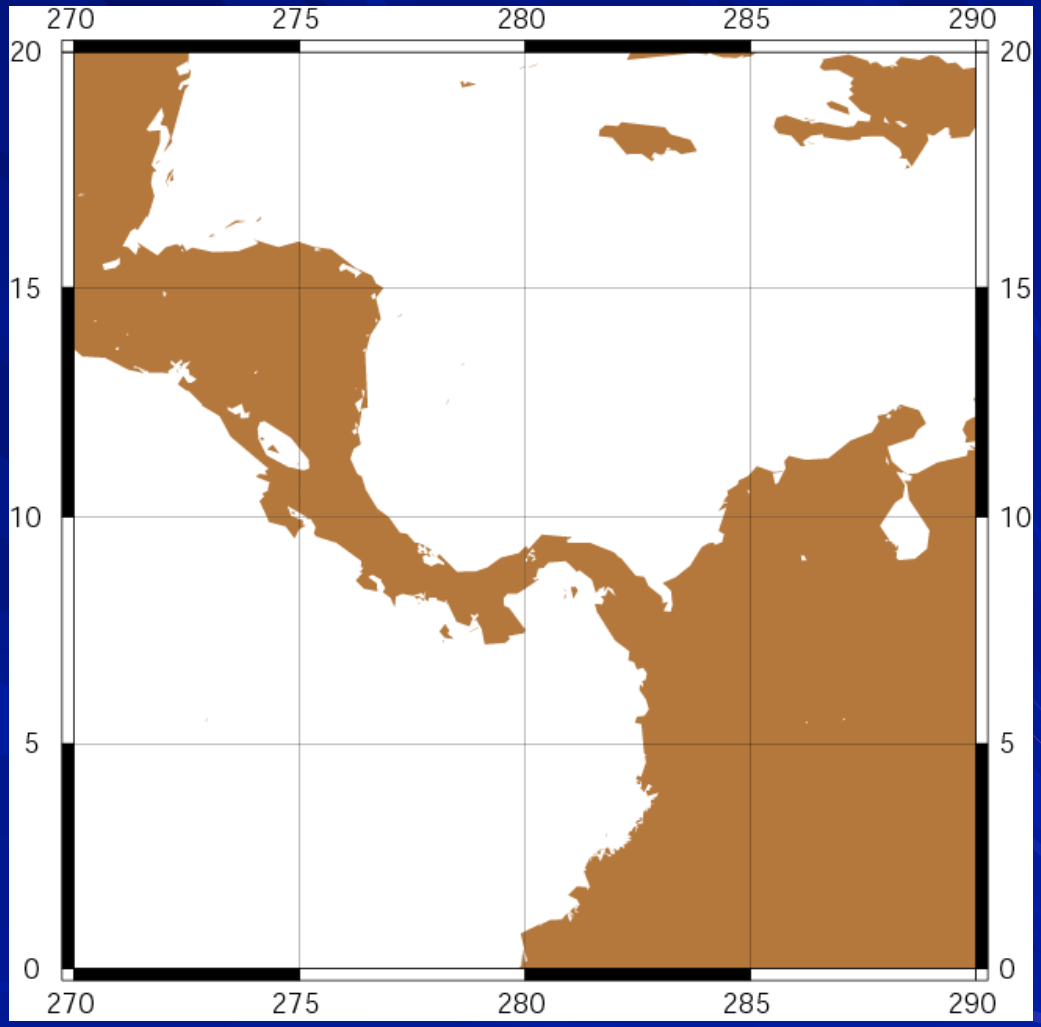
"All" gmt programs plot "maps" through the projection command line option or switch (even the x-y plot).

All projections give you two selections for specifying the scale

(note GMT takes the attitude that a map has to have a predetermined/ known scale – nicely filling the page does not cut it.)



```
pscoast -R-90/-70/0/20 -JM6i -P -B5g5 -G180/120/60 > map1.ps
```



-Jmparameters (Mercator [C]).

Specify one of: **-Jmscale** or **-JMwidth**

Give scale along equator (1:xxxx or UNIT/degree).

-Jmlon0/lat0/scale or **-JMlon0/lat0/width**

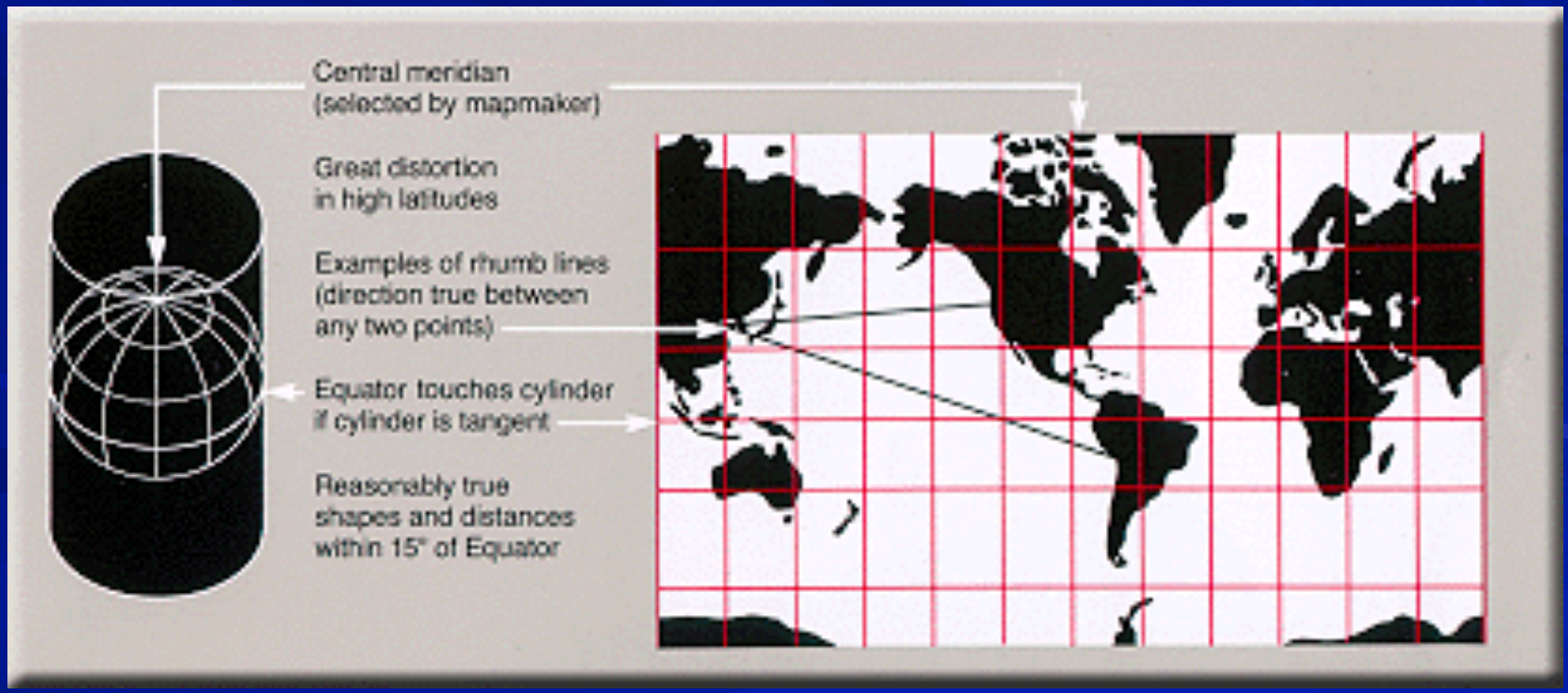
Give central meridian, standard latitude and scale along parallel (1:xxxx or UNIT/degree, UNIT = number inches or cms).

Mercator Projection:

One way to address plotting sphere on a plane (which is whole 'nother subject)

Conformal (maintains shapes)

Cylindrical projection



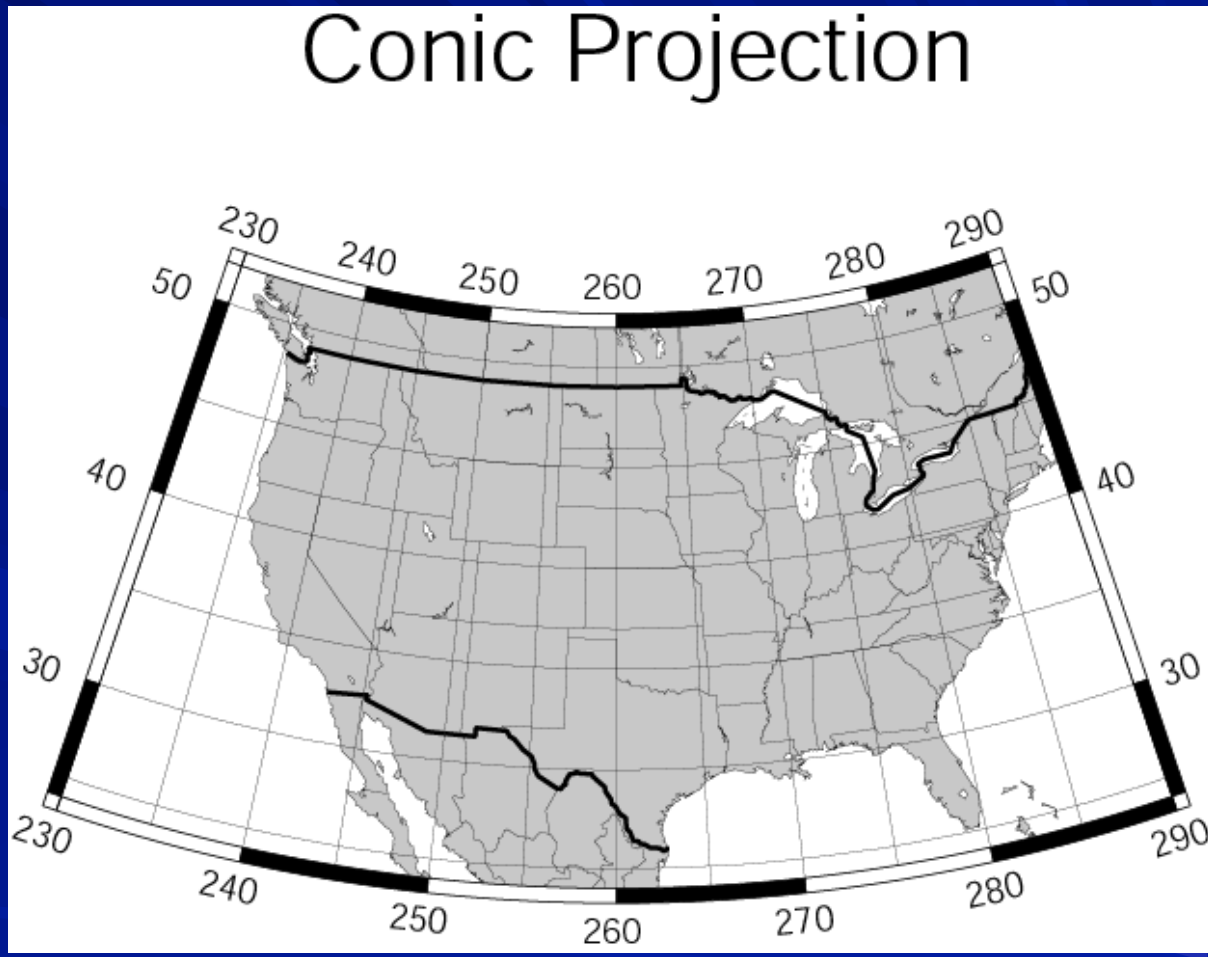
```
pscoast -R-130/-70/24/52 -JB-100/35/33/45/6i -B10g5:."Conic Projection": \
-N1/2p -N2/0.25p -A500 -G200 -W0.25p -P >! map.ps
```

Region is "rectangle" on the spherical earth.

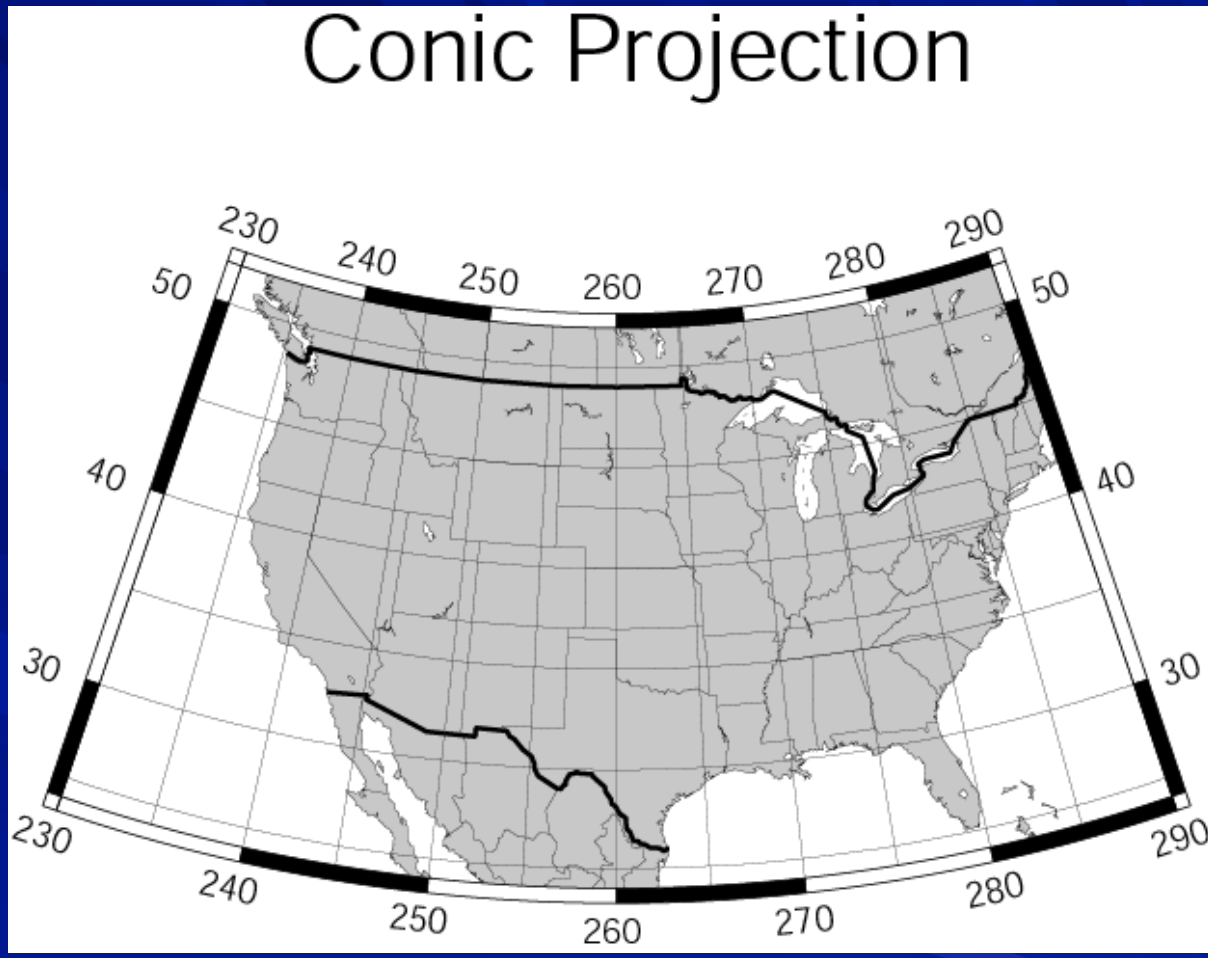
-N for boundaries (international, US/ Canadian/Mexian state boundaries "built in"), rivers.

-A to get rid of small water/island features

Projection (**b**/**B**) – need to know something (center and standard parallels).



```
pscoast -R-130/-70/24/52 -JB-100/35/33/45/6i -B10g5:."Conic Projection": \
-N1/2p -N2/0.25p -A500 -G200 -W0.25p -P >! map.ps
```



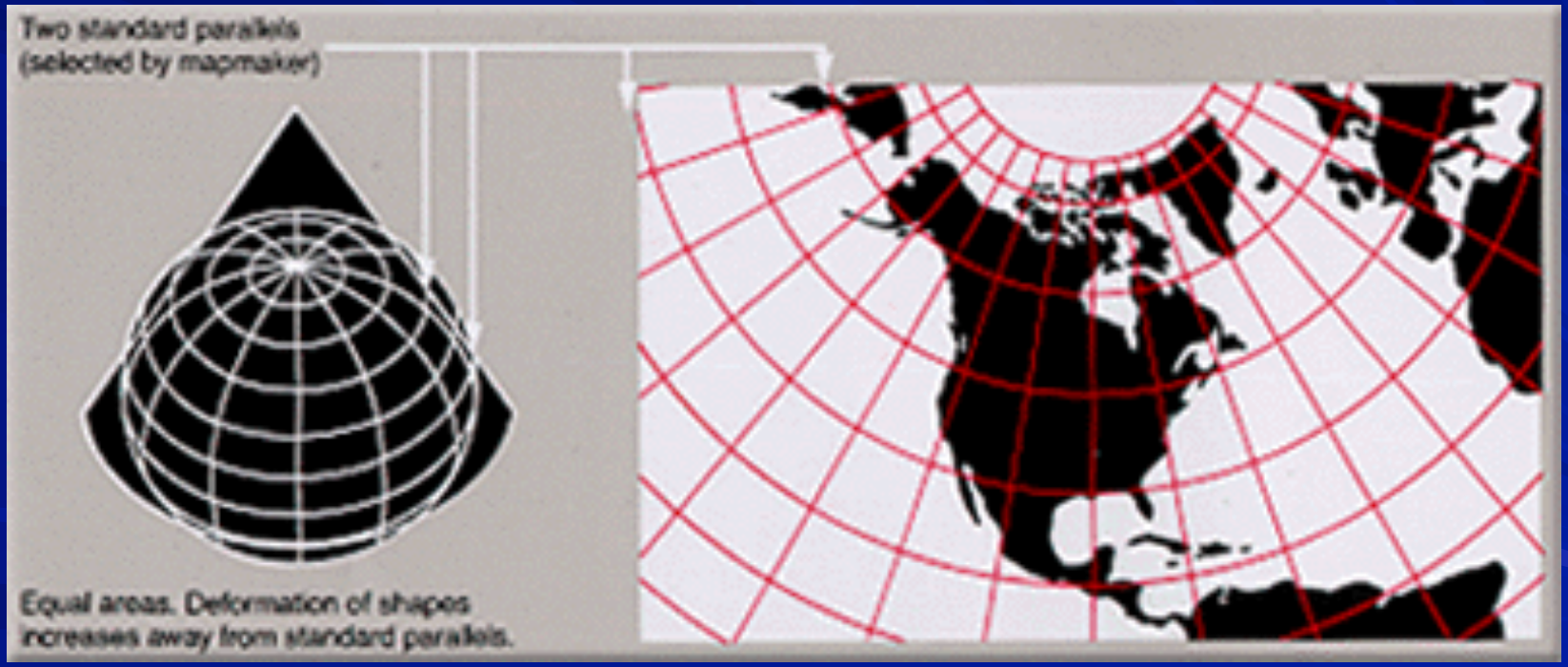
`-Jblon0/lat0/lat1/lat2/scale` or `-JBlon0/lat0/lat1/lat2/width` (Albers [E]).

Give projection center, two standard parallels, and scale (1:xxxx or UNIT/degree).

Albers

Also conformal (maintains/conserves shape)

Conical projection



```
pscoast -R0/360/-90/90 -JG280/30/6i -Bg30/g15 -Dc -A5000 \
-G255/255/255 -S150/50/150 -P >! map.ps
```

azimuthal orthographic projection mimics looking at earth from infinite distance

New option

-Dc

Controls resolution of coastline

f full

h high

l low

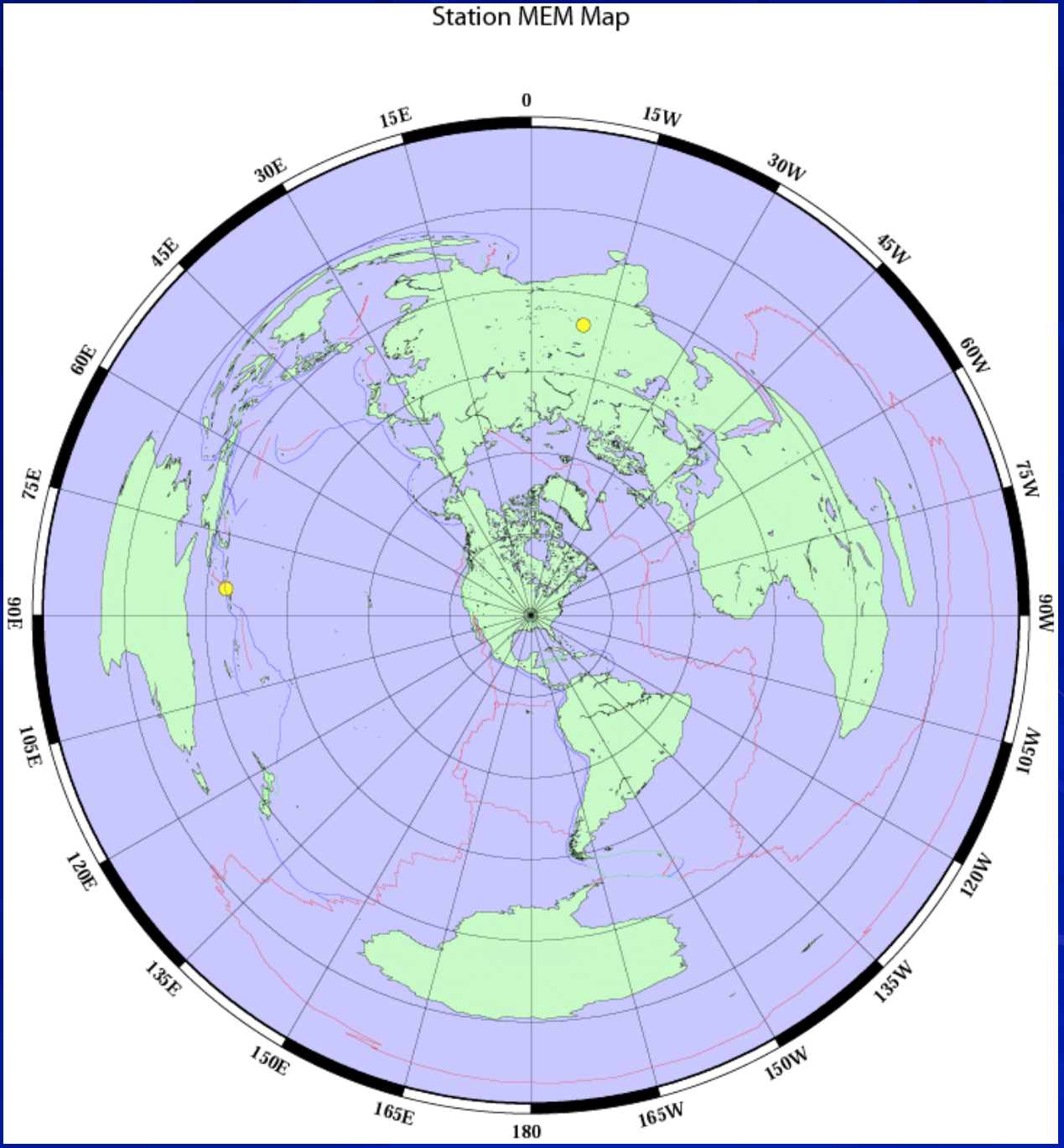
c crude

Helps manage file sizes.

Some useful maps

The world centered on Memphis.

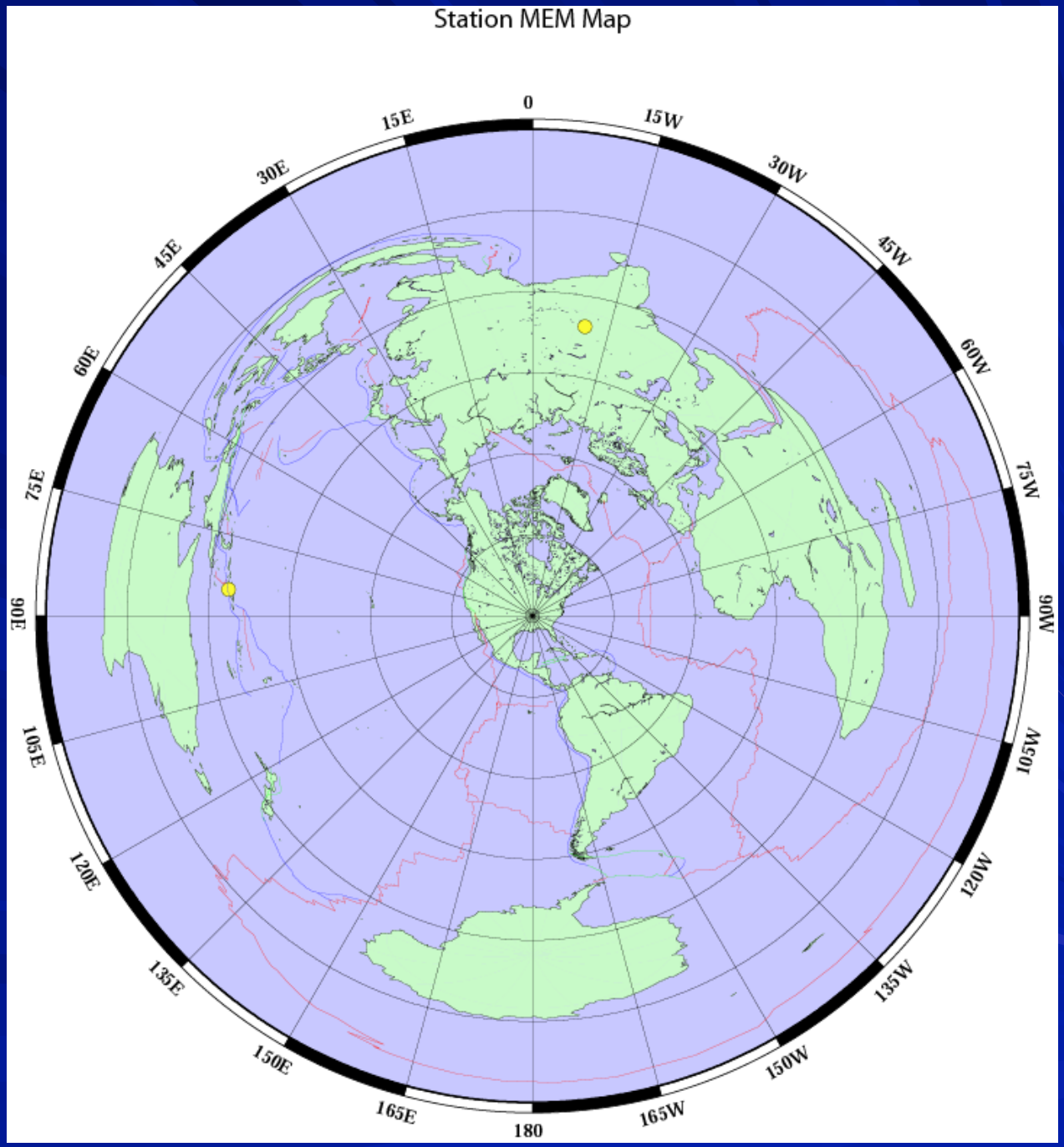
Use to get back azimuth and distance to earthquakes at a glance.



NOTE:

GMT will fall over in this projection if there is land at the anti-pode and you try to fill it.

(fill should be donut between coastline and outside of map but PostScript interpreter – which does fill - will do something else).



Part I of shell script

Set stuff up

```
#!/bin/sh
#call with “stn_az_map lat lon name”
ROOT=/gaia/home/smalley
WORLDCOAST=0/360/0/180
RED=250/50/50
BLUE=50/50/255
GREEN=50/255/50
MOREPS=-K
ADDPS=-O
CONTINUEPS=”-K -O”
FILL=200
SCALE=1.75
XOFFSET=0.75
YOFFSET=1.5
GRIDCNTR=180/90/7/90
OUTPUTFILE=$0_$3.ps
rm $OUTPUTFILE
```

Notice abundant “comments” (use variable names that are self documenting)

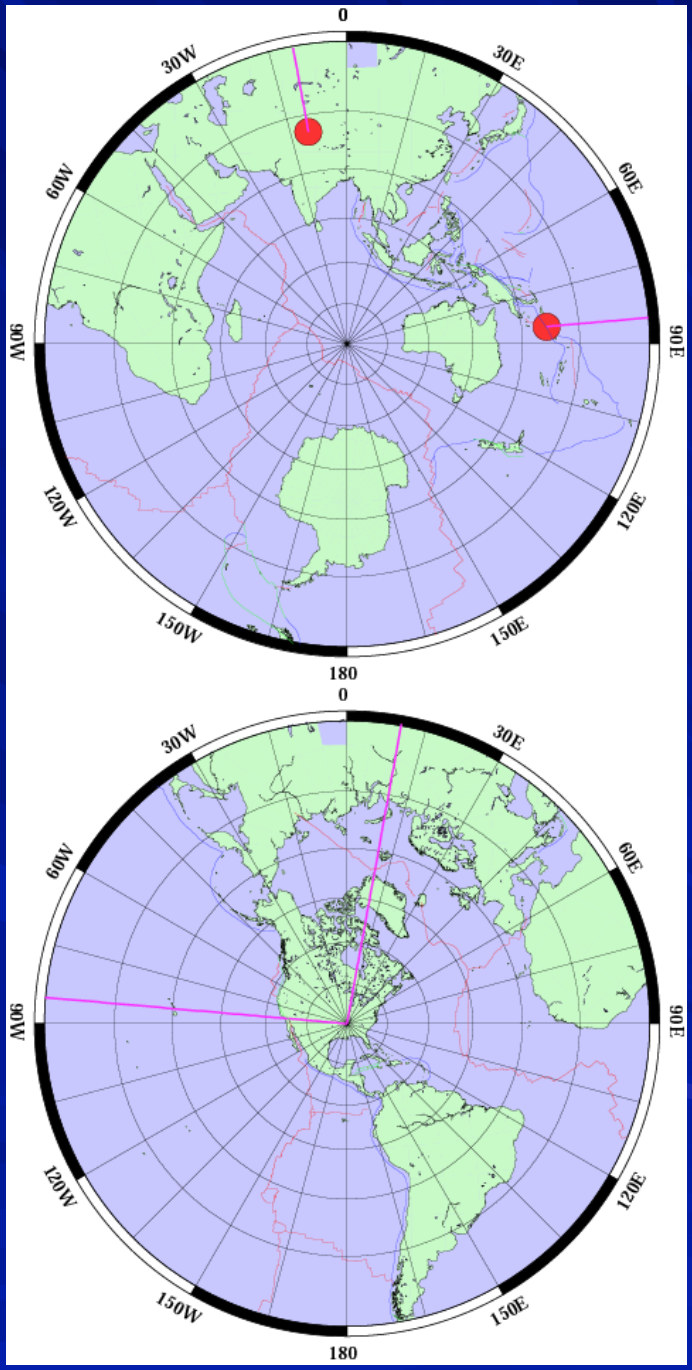
```
#set up map to be centered on lat lon given in command line
#draw crude coastlines, ocean blue, land green
#do not draw lat long grid (no frame specs on -B, could put w/next)
pscoast -R$WORLDCOAST -Je$1/$2/$SCALE/180 -B:."Station $3 Map": -
S200/200/255 -G200/250/200 -W1 -Dc -P $MOREPS -X$XOFFSET -Y$YOFFSET >
$OUTPUTFILE

#set up new map centered on north pole and draw only the lat long grid
psbasemap -R$WORLDCOAST -Je$GRIDCNTR -B15g15 -O -K >> $OUTPUTFILE

#RESET map to be centered on lat lon given in command line
#to put on some earthquake data read from this file
#data specified in lat long order, psxy assumes long lat (x,y) so
#use the "-:" switch to let psxy know (another common gotcha)
psxy -R$WORLDCOAST -Je$1/$2/$SCALE/180 -Sc0.1 -G250/250/50 -W1/0/0/0
$CONTINUEPS -: <<END >> $OUTPUTFILE
-9.09 158.44
35.35 78.13
END

#add plate boundaries, notice don't have to respecify details of region
and projection but do need -R -Je
psxy -R -Je -M$ -W1/$RED $CONTINUEPS $ROOT/ptect/ridges >> $OUTPUTFILE
psxy -R -Je -M$ -W1/$GREEN $CONTINUEPS $ROOT/ptect/xforms >> $OUTPUTFILE
psxy -R -Je -M$ -W1/$BLUE $ADDPS $ROOT/ptect/trenches >> $OUTPUTFILE
```

Another version of an azimuthal, equiangular map centered on Memphis and it' s anti-pode.

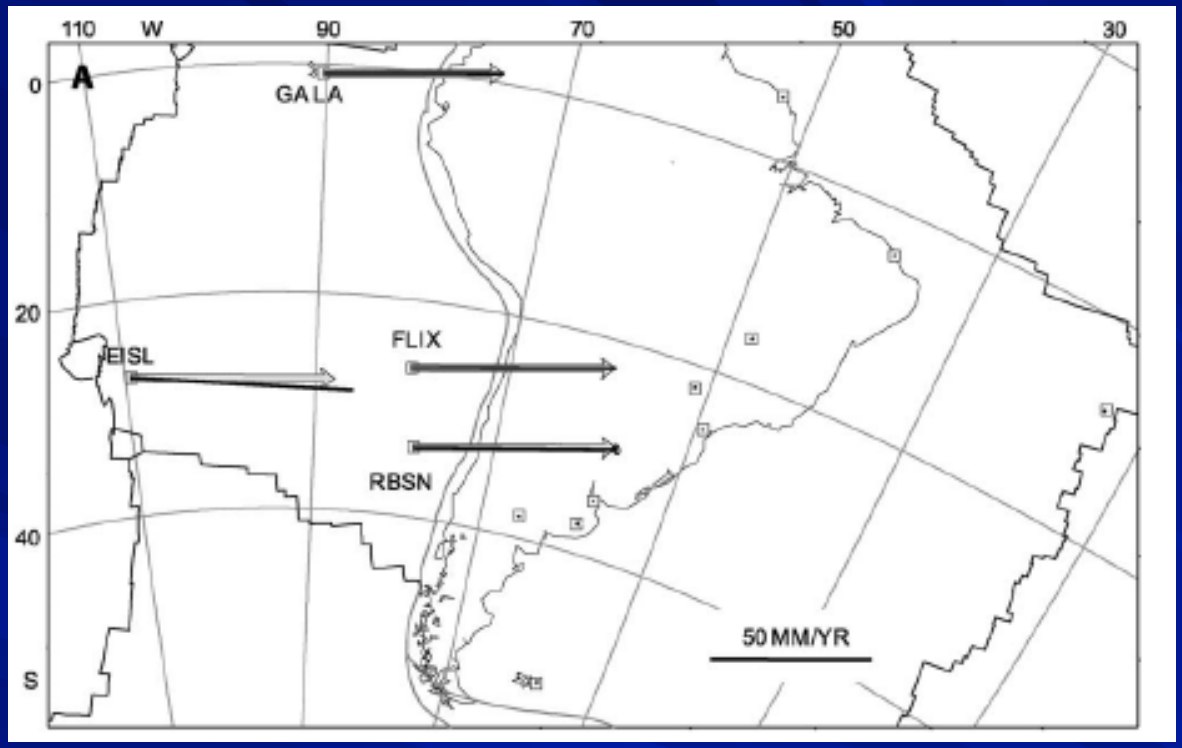
Now it' s a lot easier to identify landmasses on the other side of the globe by their shapes.

Also shows that great circles (the radial lines) converge at the anti-pode.

(also solves antipode fill problem)



Nazca-South America Euler pole



Data plotted in South America reference frame using oblique Mercator projection referenced to Euler pole (points on South America plate have zero – or near zero – velocities.)

Plate motion follows lines of latitude (horizontal lines)

Kendrick et al, 2003

Typical task:

Somebody gives you a file with earthquake data (and if you are lucky a description of the file)

So we have

lat in col 8, lon in col 9, depth in col 10 and magnitude in col 11

```
ZDEQ  64  1  1   5 14 26.76   37.285 143.002  26.9 4.4 0 15 27.0229  5
1.82    7.21    2.95  200.86   9   0   1   5
 DEQ  64  1  1  12 21 58.64   -6.872 129.763 111.1 0.0 0 58 95.0280  7
1.19    6.21    2.69   66.99  17  11   3  27
...
```

Typical task:

We can use the following **nawk** command (can put it into shell script) to produce GMT output for **psxy** – lat long and magnitude for example (**psxy** can scale the symbols from the data – use magnitude for scaling). I usually do it on the fly and pipe or suck it into the GMT program. If it's needed in more than one place – put it in a temporary file.

```
nawk '{print $9, $8, $11}' EBH.HDF
```

Produces the following output for GMT

```
143.002 37.285 4.4
129.763 -6.872 0.0
```

So what do we do with our **nawk** command

```
nawk '{print $9, $8, $11}' EBH.HDF
```

You can put this into GMT several ways

If this is the only file you want to plot – this would work

```
nawk '{print $9, $8, $11}' EBH.HDF | pxsy …
```

If you had a number of files that needed conversion you could do it this way (only need one **psxy** call)

```
psxy … << END …
…
`nawk '{print $9, $8, $11}' EBH.HDF`
…
END
```

Converting each file on the fly.

If you want to do the same thing to a list of files

```
filelist="$SAMDATA/eq-rupt-1995.dat $DEM/eq-rupt-1960.dat"
for FILE in $filelist
do
psxy -R -$PROJ$SCALE -M$ -: $CONTINUE -W$LINETHICK/$PURPLE $FILE \
$VBSE >> $OUTPUTFILE
done
```

Other ways to make list (notice the different kinds of quotes: ", ' and `)

```
filelist=`ls -1 $ROOT/dem/topocontours/andes_3000_*`

contourlist='1 2 3 4'
```

Some other nawk tricks – doing math and passing variables to nawk (quote heaven)

```
SCALE=`echo $STNDTMLON | \
nawk '{print ($1>=0?$1:360+$1)"/"'${jTRESCALE}_1'*'$FACTOR'}' `
```

Another example

Non-simple input data format

We look in the file

```
1975061019 3539818n 682 64w2567  864                                    /              1
 rrd P 2 75 61019 413.86122327430.36 S 3 41       1956160              3199.M  194  0  0  0  0
 cup PD1 75 61019 4 9.46 93528222.26 S 3 48       1566 80       D     -20320    68  0  0  0   0
 csj P 3 75 61019 414.26124428230.46 S 4 41       1983320              4499.   156  0  0  0   0
 pwp P 3 75 61019 412.66103826826.26 S 4 48       171399.M            15599.   213  0  0  0   0
 mtp PC0 75 61019 412.76115926627.06 S 3 41       1875 16       ?       0320     3  0  0  0   0
 abv PC1 75 61019 4 5.56 645 1213.66 S 3 48       1151 80       ?       7320   -58  0  0  0   0
                10
19750617 445237218n4581 65w1307  515                                    /              2
 rrd PC0 75 617 44536.53 755216          48       1309 16       ?      -29          0  0  0  0  0
 abv P 3 75 617 44543.83 908 94          48       1527320              484          0  0  0  0  0
 mtp P 0 75 617 44538.43 856207          48       1454 16               15          0  0  0  0  0
 pwp P 0 75 617 44538.13 768200          48       132                  114          0  0  0  0  0
 csj P 0 75 617 44534.83 73622048.32 S 4 48       1282 16              -599.    369  0  0  0  0
 cup P 0 75 617 44532.48 522196          48        976 16             -101          0  0  0  0  0
                10
```

What's this?

First line – looks like earthquake location information

Pick it apart

```
1975061019 3539818n 682 64w2567  864                                  /          1
```

Year, month, day, hour, minute, second, lat (in degrees, N/S, min, seconds =DMS format), lon (DMS format), other stuff that we can't guess

Is all run together

Next batch of lines looks like phase information then line with “10”

```
rrd P 2 75 61019 413.86122327430.36 S 3 41       1956160              3199.M  194  0  0  0  0
cup PD1 75 61019 4 9.46 93528222.26 S 3 48       1566 80       D     -20320    68  0  0  0   0
csj P 3 75 61019 414.26124428230.46 S 4 41       1983320              4499.   156  0  0  0   0
pwp P 3 75 61019 412.66103826826.26 S 4 48       171399.M             15599.  213  0  0  0   0
mtp PC0 75 61019 412.76115926627.06 S 3 41       1875 16       ?       0320     3  0  0  0   0
abv PC1 75 61019 4 5.56 645 1213.66 S 3 48       1151 80       ?       7320   -58  0  0  0   0
                  10
```

GMT wants lat, long

Fact that fields are run together is a problem

Have to pick input lines apart by column

Have to select lines with earthquake location and ignore those with phase info

```
1975061019 3539818n 682 64w2567  864                                    /          1
 rrd P 2 75 61019 413.86122327430.36 S 3 41       1956160          3199.M  194  0  0  0  0
```

```
#!/bin/sh -f
nawk 'substr($0,19,1) == "n" || substr($0,19,1) == "s" \
{ print (substr($0,19,1) == "s" ? "-" : "") \
substr($0,17,2)+(substr($0,20,2)+substr($0,22,2)/60)/60, \
(substr($0,27,1) == "w" ? "-" : "" ) \
substr($0,24,3)+(substr($0,28,2)+substr($0,30,2)/60)/60}' \
timesortedallc.pha
```

```
#!/bin/sh -f
#make a simple map with point data

LATMIN=10
LATMAX=30
LONMIN=-80
LONMAX=-55
SCALE=0.6
MEDYELLOW=255/255/192
LTBLUE=192/192/255
RED=255/0/0
DONTCLOSE=-K
DONTINIT=-O
CONTINUE="-K -O"
INVLATLON="-:"

pscoast -R$LONMIN/$LONMAX/$LATMIN/$LATMAX -Jm${SCALE} \
-B10 -G$MEDYELLOW -S$LTBLUE $DONTCLOSE -P > $0.ps
psxy -R -Jm${SCALE} -Sc0.2 -G$RED -W1/0 $DONTINIT \
$INVLATLON << END >> $0.ps
`preqs2gmt.sh`
END
```

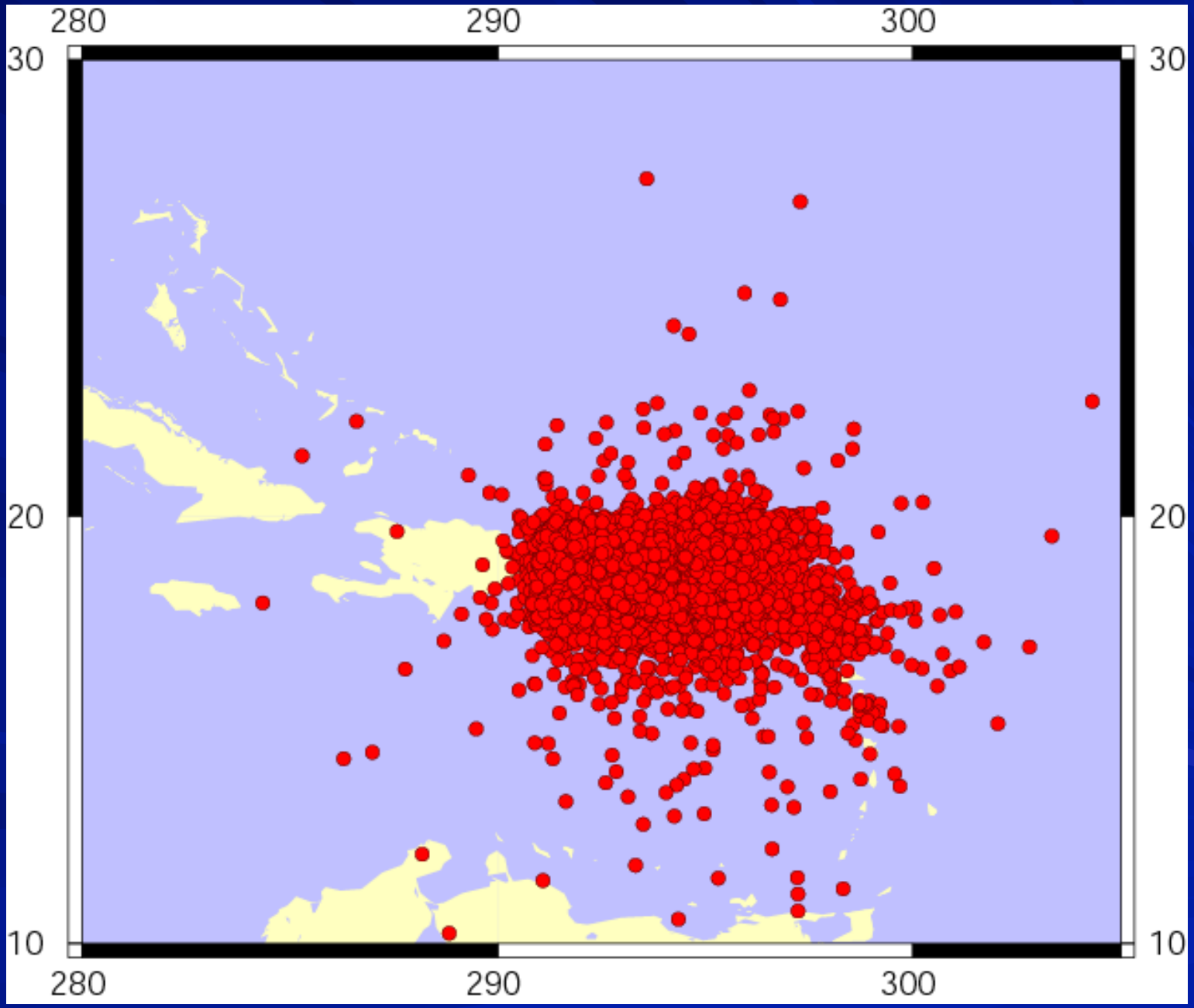
Set it up

pscoast to draw background map

psxy to draw the earthquakes (red circles with black outline)

preqs2gmt.sh to prepare data on the fly

result



Example of GMT man page – expanded for understanding

psxy reads (x,y) pairs from *files* [or standard input] and generates *PostScript* code that will

plot lines, polygons, or symbols

at those locations on a map. If a symbol is selected and no symbol size given, then psxy will

interpret the third column of the input data as symbol size.

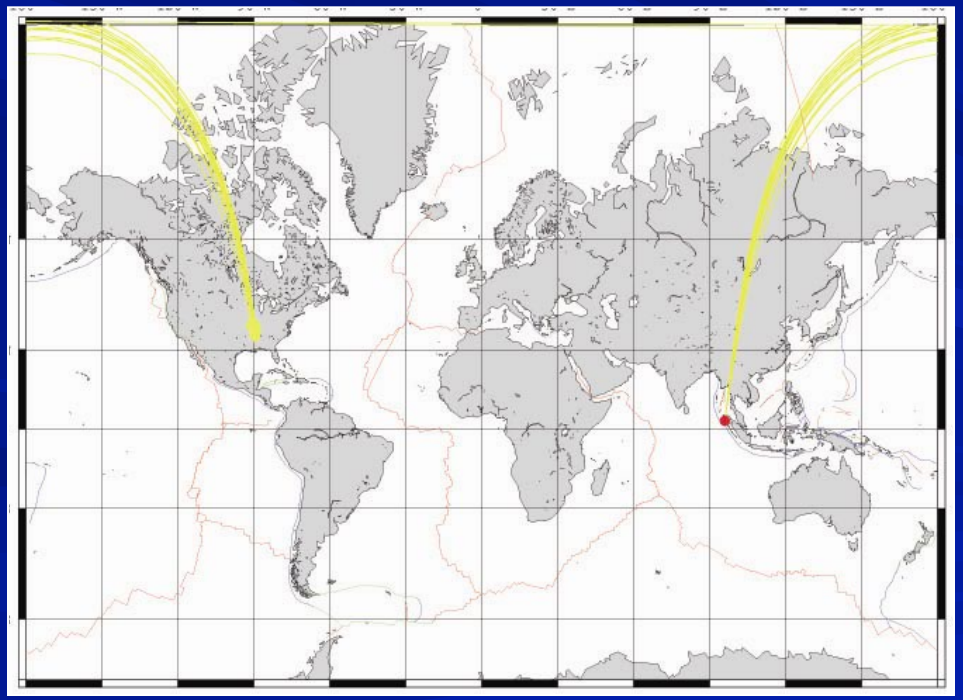
Symbols whose size is <= 0 are skipped.

If no symbols are specified then the symbol code (see -S below) must be present as (specify symbol in) last column in the input.

Example of GMT man page – expanded for understanding

psxy reads (x,y) pairs from *files* [or standard input] and generates *PostScript* code that will

Multiple segment files (lift pen) may be plotted using the -M option. If -S (symbol plotting) is not selected, a (great circle) line connecting the data points will be drawn instead.



Example of GMT man page – expanded for understanding

psxy reads (x,y) pairs from *files* [or standard input] and generates *PostScript* code that will

To explicitly close polygons, use -L.

Shade with -G. If -G is set, -W (line width and color) will control whether the polygon outline is drawn or not.

If a symbol is selected, -G and -W determines the fill color and outline/no out- line, respectively.

The *PostScript* code is written to standard output (screen!).

Things you can plot with PSXY - Point or line data with symbols

star
Bar
Circle
Diamond
Ellipse
front [various symbols such as thrust fault barbs, warm front symbol, etc.]
Hexagon
Invtriangle
Letter
Point
Square
Triangle
Vector
Wedge
cross

Make focal mechanisms – use GMT filter psmeca

make/obtain input file – see psmeca documentation for large number of ways to define focal mechanism data

```
35.59  -90.48 12     220     65      150     4.5975 -0.25 -0.25
35.86  -89.95 16     220     75      150     4.0727 -0.25 0.25
36.37  -89.51 7.5    350     84      145     4.2020 -0.25 0.25
36.54  -89.68 9      85      60      -20     3.7118 0 0.5
36.56  -89.83 8      90      67.5    20      4.1068 -0.25 -0.25
36.64  -90.05 15     304     78      -28     4.6309 0 -0.5
37.16  -89.58 15     140     75      50      4.2547 0.25 0
37.22  -89.31 1.5    280     70      -20     3.5783 -0.25 0.25
37.36  -89.19 16     30      70      170     3.8250 0.25 0.25
37.44  -90.44 15     350     60      135     4.0126 0.25 0.25
37.48  -90.94 5      260     40      -70     4.5728 0.25 -0.25
37.91  -88.37 22     0       46      79      5.2612 -0.35 0.1
38.55  -88.07 15     310     70      0       4.3154 -0.25 -0.25
38.71  -87.95 10     135     70      15      4.9309 -0.25 0.25
```

Make map with focal mechanisms (psmeca)

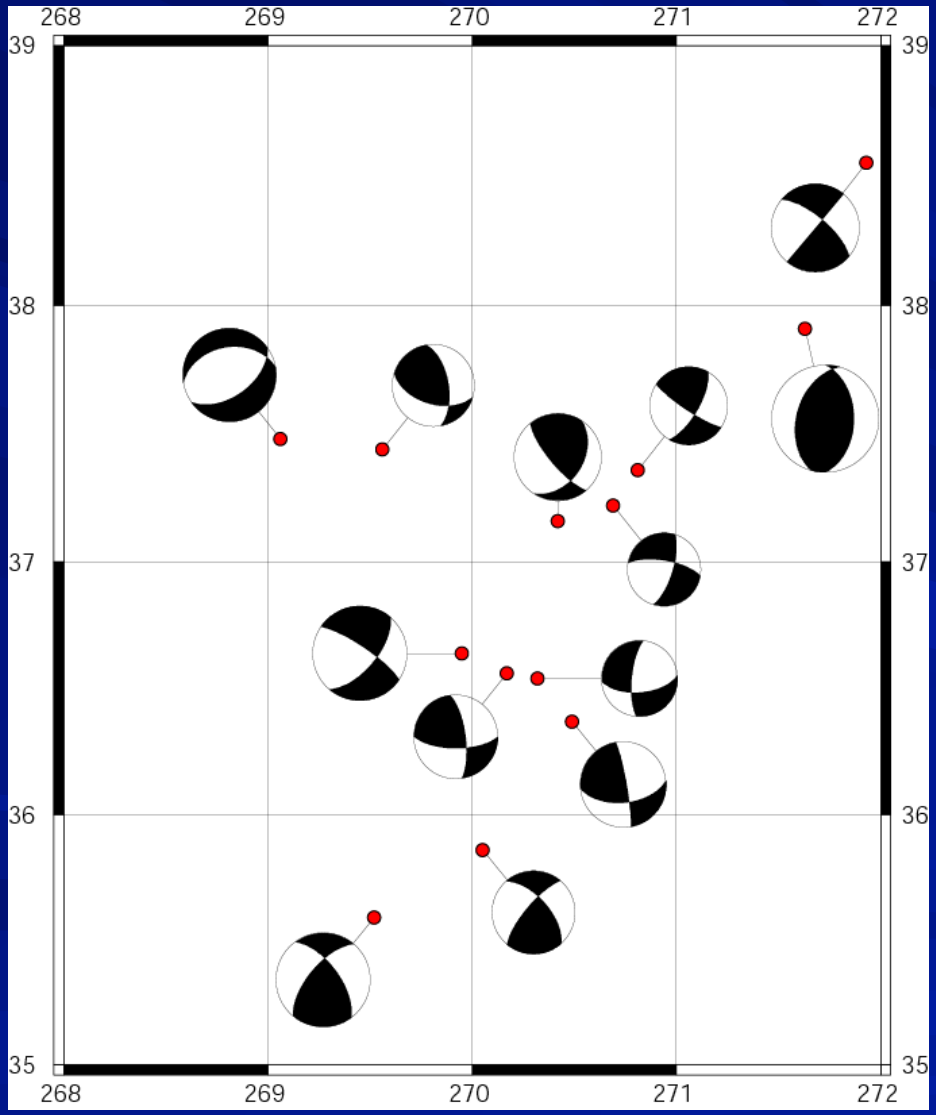
and earthquake locations (psxy)

```
#!/bin/sh -f
REG=-92/-88/35/39
psmeca -R$REG << END -Jm4. -Bg1f1a1 -P -Sa2./0/0 -CP -: -K > $0.ps
`nawk '{print $1, $2, $3, $4, $5, $6, $7, $1+$8, $2+$9}'
practice_data.dat`
END
psxy -R$REG practice_data -Jm4. -Sc0.25 -: -G255/0/0 -W3/0 -O >> $0.ps
```

-S for focal mechanism input format definition

-C for plotting beach ball away from earthquake location and connecting it to point at earthquake location with a line

```
35.59  -90.48 12      220     65      150     4.5975 -0.25 -0.25
`nawk '{print $1, $2, $3, $4, $5, $6, $7, $1+$8, $2+$9}'
```



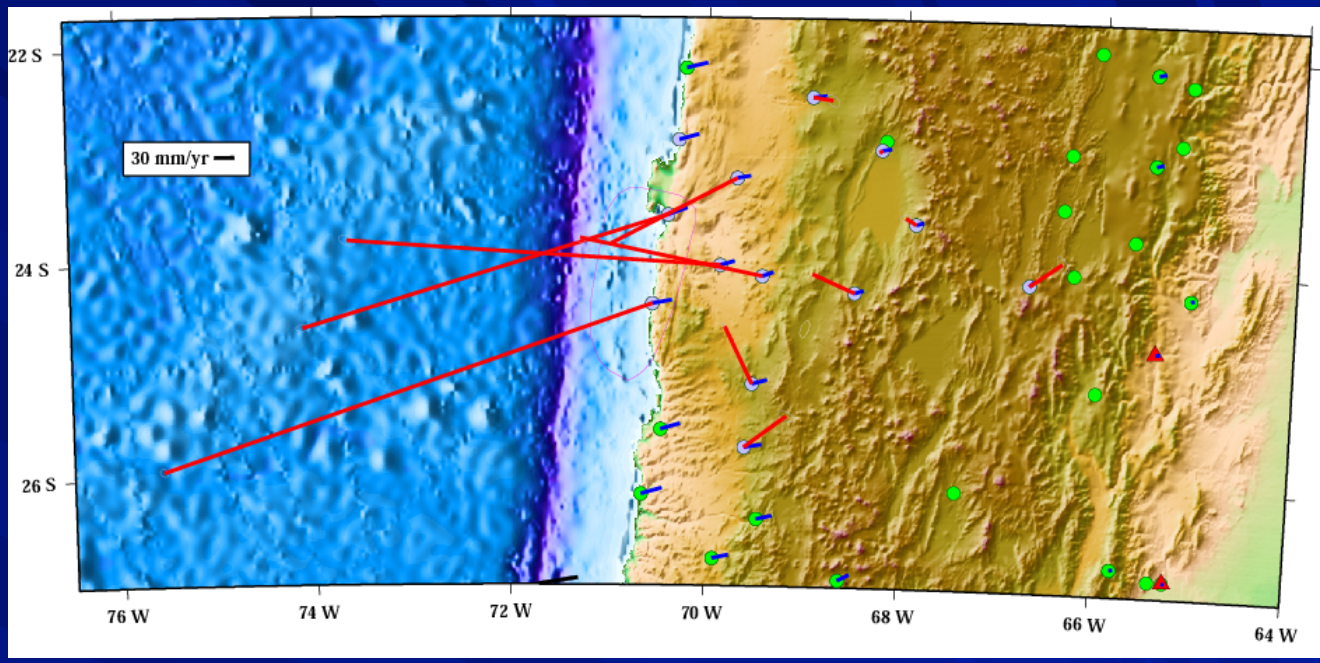
Uses "offsets" specified in columns 8 and 9 to reposition the focal mechanism.

You could put the lat, long you wanted in cols 8 and 9, but why calculate all of them by hand?

You have to specify the offsets or each beachball depending on how things look, no easy way to do automatically.

Plot

- Velocity vectors with error ellipses
- Anisitropy bars
- Rotational wedges
- Strain crosses



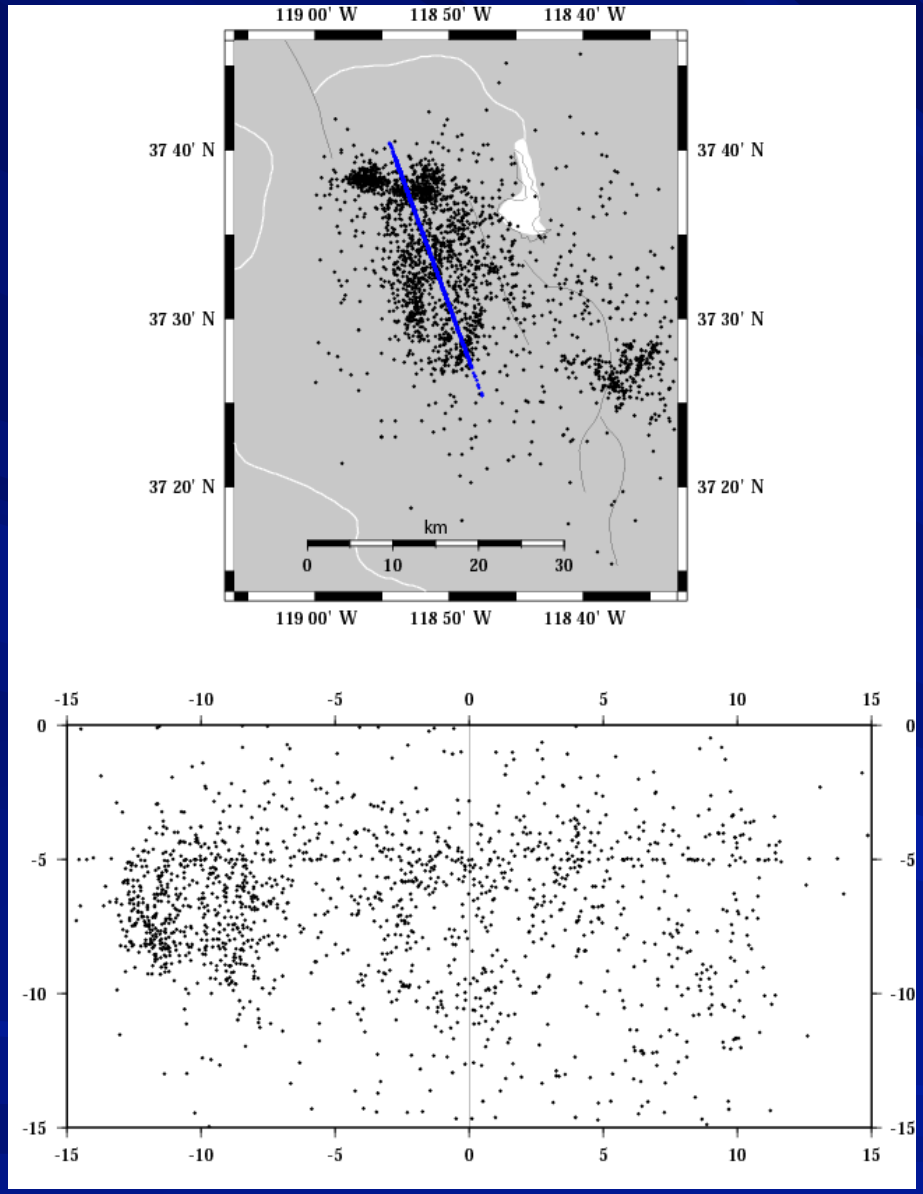

```
psvelo -R -$PROJ$SCALE -Sr$VELLEN/0.95/0 -W1/$PURPLE -G$PURPLE \
$VELARROW $CONTINUE $VBSE andaman_nicobar_coseis.dat \
>> $OUTPUTFILE
```

Various ways to define vector data (ve, vw, or mag, az)

Vector length, error ellipse confidence for plot, label font size

Arrow shaft width, head length and width

Data - Lat lon vlat vlon 1siglat 1siglon corr



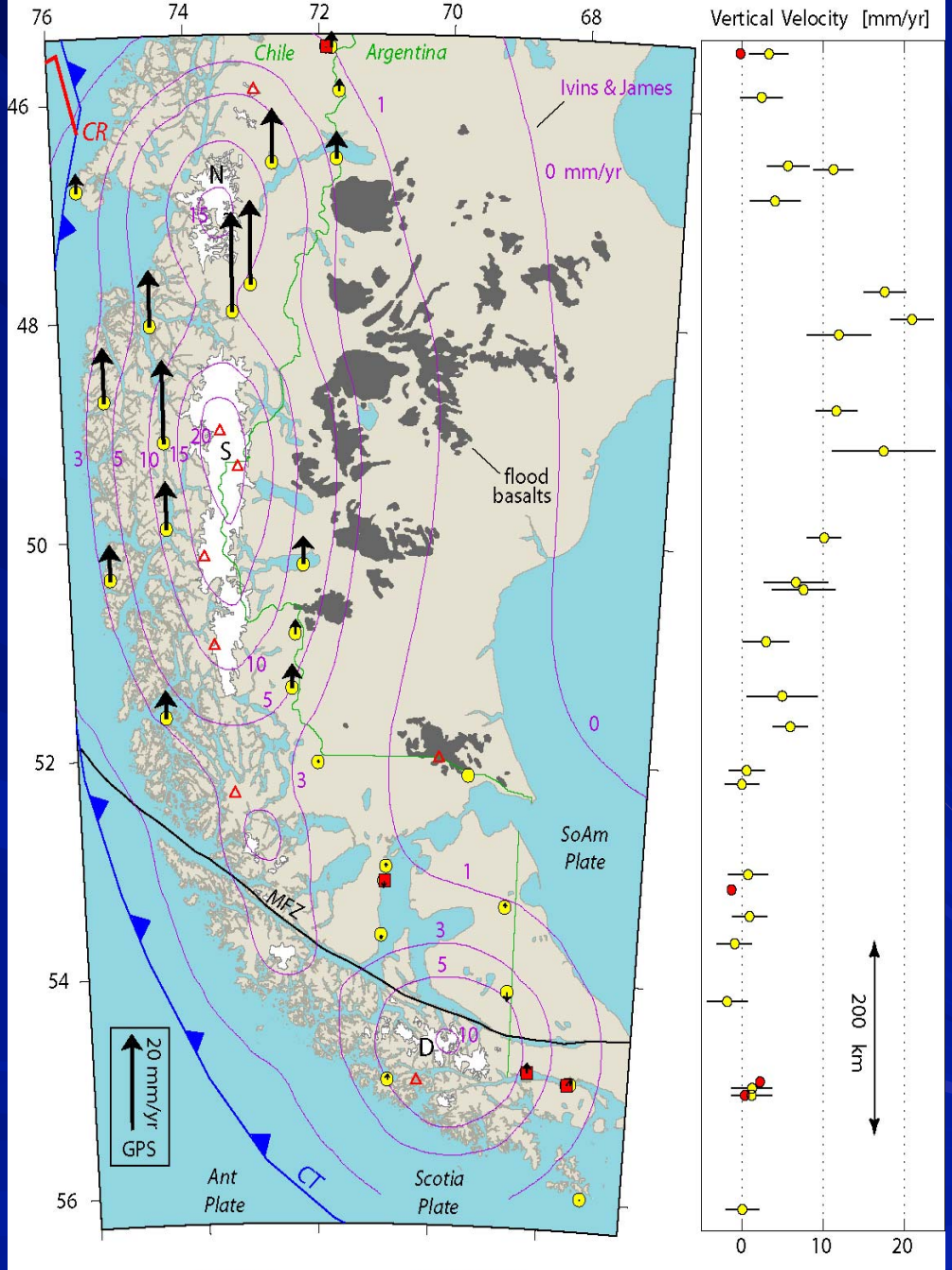
Make a cross section

(2 parts, draw map, draw cross section)

Data and non working version of shell script from
http://www-geology.ucdavis.edu/~gps/GMT/LONG_VALLEY/hypocenter.html

```
# Set PARAMETERS FOR CROSS-SECTION PLOT
center="-118.85/37.55"
azimuth="160.0"
#3.  DEFINE A BOX
width="-5/5"
length="-15/15"
\rm LV_seismicity.tmp
nawk '{print $1,$2, $3}' LV_seismicity.dat | project -C${center}\
-A${azimuth} -Q -W${width} -L${length} -V > LV_seismicity.tmp

# PLOT CROSS-SECTION HYPOCENTERS ON MAP
nawk '{print $6,$7}' LV_seismicity.tmp | psxy  -J${projection} \
-R${range} -P -M -Sc0.03 -G0/0/255 -O -V -K >> ${psfile}
# PLOT CROSS-SECTION BOX
# SET PARAMETERS TO PLOT
brange="-15/15/-15/0"
bprojection="x0.2/0.2"
btick="a5f5g0/a5f5g0"
psxy box_dim -R${brange} -J${bprojection} -B${btick} -W1 -P -O \
-K -X-1.25 -Y-4 -V >> ${psfile}
# PLOT HYPOCENTERS ON CROSS-SECTION
nawk '{print $4, $3*(-1.0)}' LV_seismicity.tmp | psxy -P -M\
-J${bprojection} -R${brange} -Sc0.03 -G0/0/0 -O -V >> ${psfile}
```

Plot contours



ICE AND FIRE

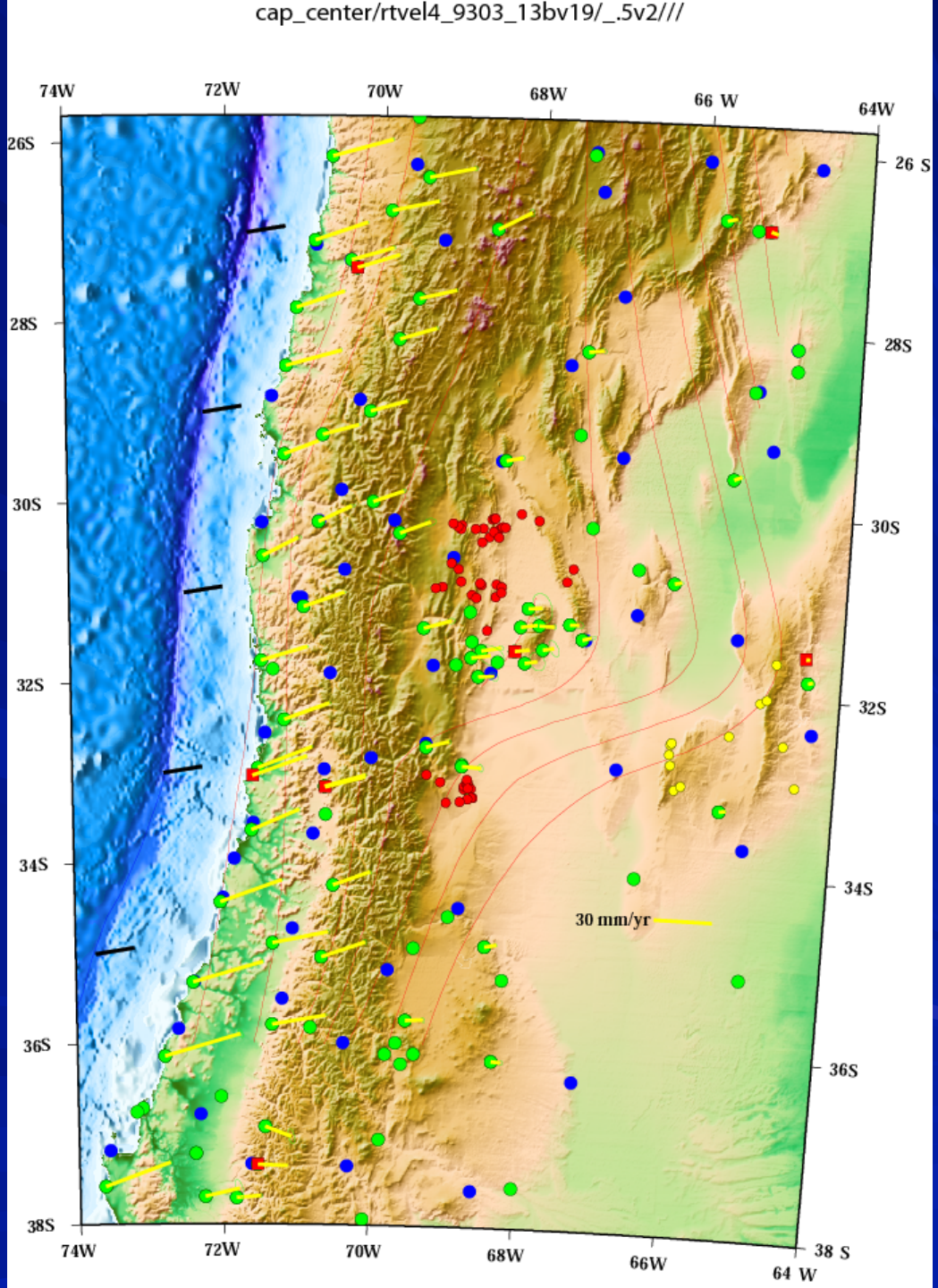
Postglacial rebound in Patagonia

```
echo make pgr contours
PGRFILE=pgr5e18
SPACING=4m
xyz2grd $SAMDATA/$PGRFILE -G$SAMDATA/$PGRFILE.grd -ISPACING /
-: -R$REGION
grdinfo $SAMDATA/$PGRFILE.grd
grdcontour $SAMDATA/$PGRFILE.grd -C1 -Jx1.0 -D$PGRFILE.con -M /
-R$REGION > /dev/null


#have to hand edit the contour file to do 2 things -- as made the
first point in each contour
#is stuck on the end of the new contour seperator line - have to
add <cr>, also does VERY bizzare
#stuff with > for segment seperator, change to $ and works fine.
#exit

psxy -R$REGION -$PROJ/$SCALE -M -W$LINETHICK/$ICECOLOR /
$CONTINUE $PGRFILE.con $VBSE >> $OUTPUTFILE
```

Returning to making pertty MAPS?

How to do:
color or b&w topo
with shaded topo

How to combine topo and bathymetry

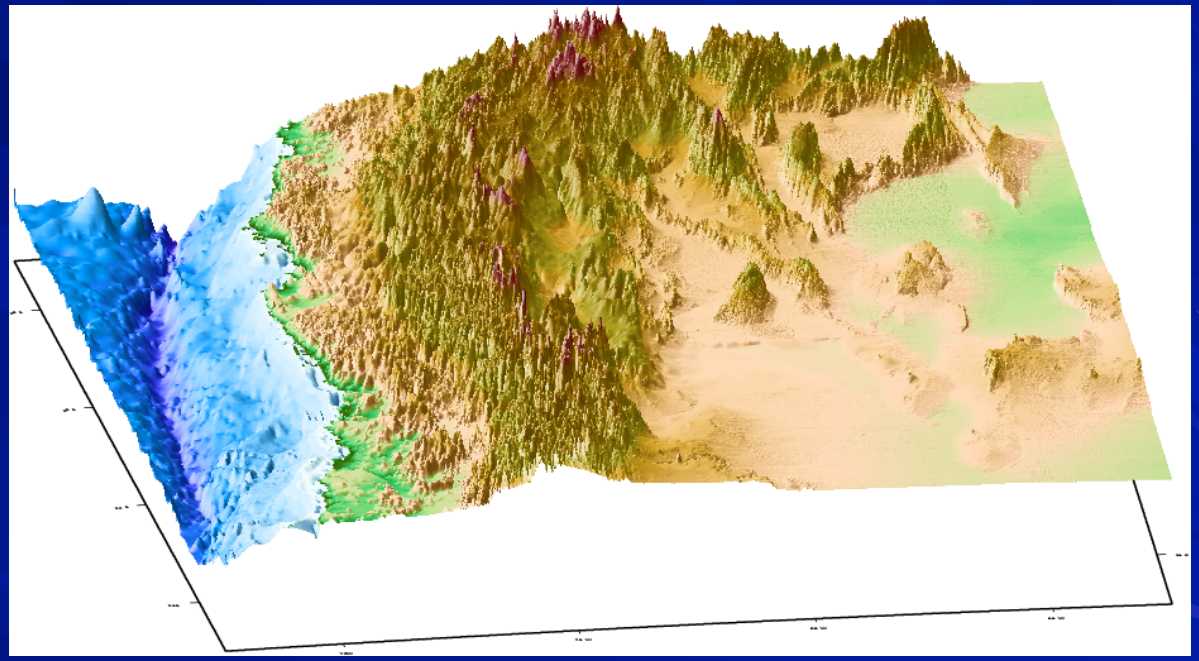


First – have to find data – what' s available

DEM' s (Digital Elevation Models) of world – several resolutions, several kinds of data (GTOPO-30, ETOPO-5 , SRTM, seasat, obs/pred bath, gravity)

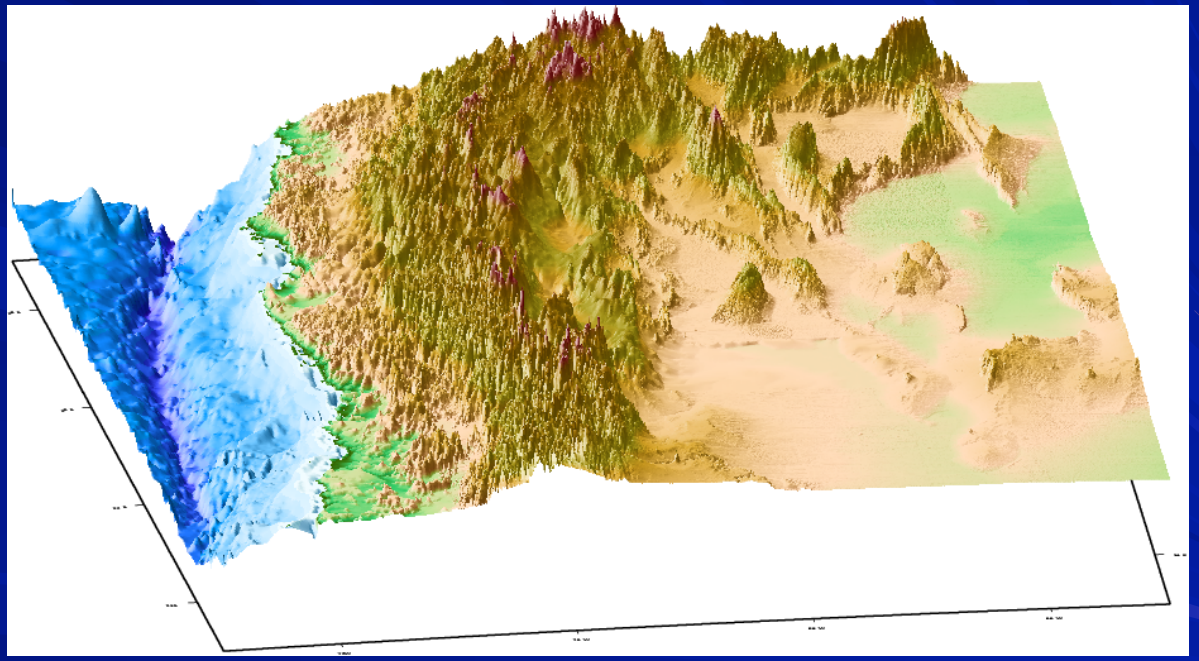
Earthquakes
Moment Tensors
Digitized geologic data
Other Geophys. Data

Roads, Cities, etc.



What tools there are to handle these data sets –
GMT is one of them.

Where to get them?
(we have some online at CERI – makes it easy. Have not fully figured out SRTM yet.)



use **grdraster** to extract a subregion from the global bathymetry data set and make a new grid file for GMT.

grdraster.is not part of "standard" GMT. Is a "supplemental" GMT program.

There are a bunch (order 35-40) of supplemental GMT programs like this around.

Many are written by others and become "attached" to GMT and can be found on the GMT web page, but they are not officially part of GMT.

psmeca and **psvelo** (to draw focal mechanisms and vector fields) are in this class.

use **grdraster** to extract a subregion from the global bathymetry data set and make a new grid file for GMT.

$GRDRASTERREGION has same format at the REGION definition (min lon/max lon/min lat/max lat) and been previously set up to define the region

```
echo do seafloor
DATASET=10
DATAGRID=-I2m/2m
grdraster $DATASET -G${ROOTNAME}_2mtopo.grd $DATAGRID \
-R$GRDRASTERREGION -V
echo done with 2m topo grdraster
```

Let's look at the documentation first

Typing `grdraster` all by itself dumps the man page.
- reports available data sets, unit, data coverage area, spacing and registration (pixel or grid – not important for now, except that when combining data sets they have to be the same).

```
alpaca/smalley 142:> grdraster
grdraster 3.4.3 - Extract a region from a raster and save in a grdfile
usage: grdraster <file number> -R<west/east/south/north>[r] \
[-G<grdfilename>] [-I<dx>[m][/<dy>[m]]][-bo[s][<n>]]
        <file number> (#) corresponds to one of these:

#  Data Description          Unit    Coverage      Spacing Registration
-------------------------------------------------------------------------
1 "ETOPO5 global topography"    "m"     -R0/359:55/-90/90       -I5m     G
2 "US Elevations from USGS"     "m"     -R234/294/24/50         -I0.5m   P
3 "Geo/Seasat grav from Haxby"  "mGal"  -R0/359:55/-90/90        -I5m     G
4 "Geo/Seasat geoid from Haxby" "m"     -R0/359:55/-90/90       -I5m     G
5 "Sea floor age from Cande"    "Ma"    -R0/359:55/-90/90       -I5m     P
6 "Sea floor age from Muller et al., 1998" "Ma" -R0/360/-72/90 -I6m      G
7 "Sea floor age errors Muller et al., 1997" "Ma" -R0/360/-72/72 -I6m    G
8 "1=land, 0=sea bitmask"       "T/F"   -R0/360/-90/90          -I5m     P
9 "USGS/SS ETOPO30s"    "m"             -R0/360/-90/90          -I0.5m   P
10 "2min Observed/Predicted Topo"    "m"   -R0/360/-72/72       -I2m     P
11 "et30wbath"    "m"    -R-78/-63/-25/-12          -I0.5m       P
-------------------------------------------------------------------------
```

First use **grdraster** to extract a subregion from the global data set

```
echo do seafloor
DATASET=10
DATAGRID=-I2m/2m
grdraster $DATASET -G${ROOTNAME}_2mtopo.grd $DATAGRID \
-R$GRDRASTERREGION -V
echo done with 2m topo grdraster
```

We have selected the 2m predicted sea floor topography – data set 10.

We have set the grid to the proper sample spacing (get from previous slide w/ data set properties).

We are going to put the data into a file called **${ROOTNAME}_2mtopo.grd**

Now we do the same for the land topographic data, using GTOPO-30, which only has data for land.

```
echo do topo
DATASET=9
DATAGRID=-I30c/30c
grdraster $DATASET -G${ROOTNAME}_topo.grd $DATAGRID \
-R$GRDRASTERREGION -V
echo done with gtopo grdraster
```

Now we select the ETOTO-30 topography – data set 9.

Notice that the grid has a different sample spacing than the bathymetry, otherwise this code snippet is the same.

The data will go into a file called `${ROOTNAME}_topo.grd`

We now have two complimentary data sets, one for topography and one for bathymetry and we have to combine them.

Unfortunately, they have different sample spacing.

So we have to resample one of the data sets – lets do it to the sea floor (since it has the lower resolution – we will therefore be interpolating)

Use **grdsample** to resample the bathymetry as defined by **DATAGRID** and put in a new file **${ROOTNAME}_30stopo.grd**

```
echo prep and merge bathy
DATAGRID=-I30c/30c
grdsample ${ROOTNAME}_2mtopo.grd -G${ROOTNAME}_30stopo.grd $DATAGRID \
-F -R$GRDRASTERREGION -V
```

Now we use **grdmath** to combine (**AND**) the two data sets (they have distinguishing values in the dataless points).

grdmath uses a stack and RPN – Reverse Polish Notation)

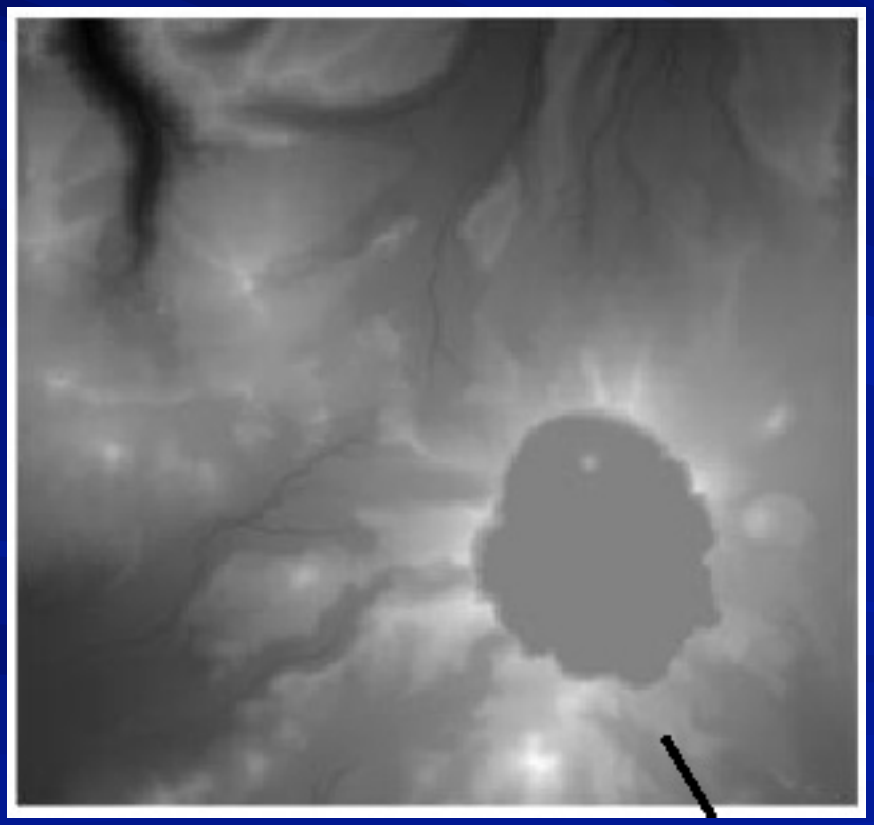
```
grdmath -F -V ${ROOTNAME}_topo.grd ${ROOTNAME}_30stopo.grd AND = \
${ROOTNAME}_topobath.grd
echo done with merge bathy
```

And put the new topo file in **${ROOTNAME}_topobath.grd**

We are now done selecting the topographic and bathymetric data,

which is used to give the coloring or grayscale.

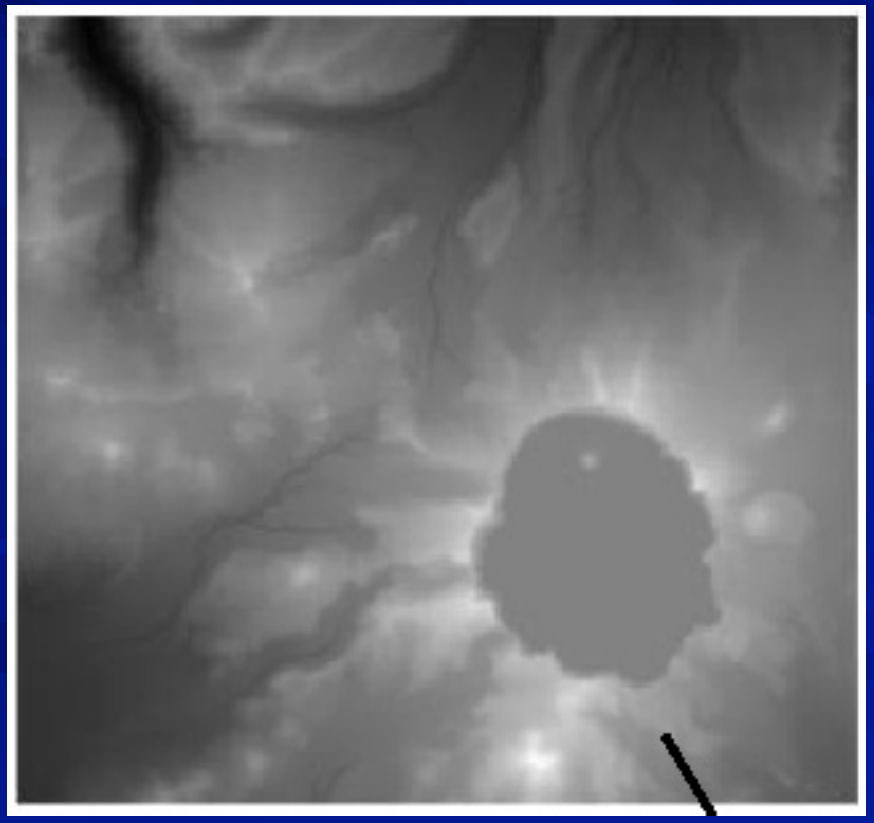
It is very hard, however, for the brain to interpret this view.

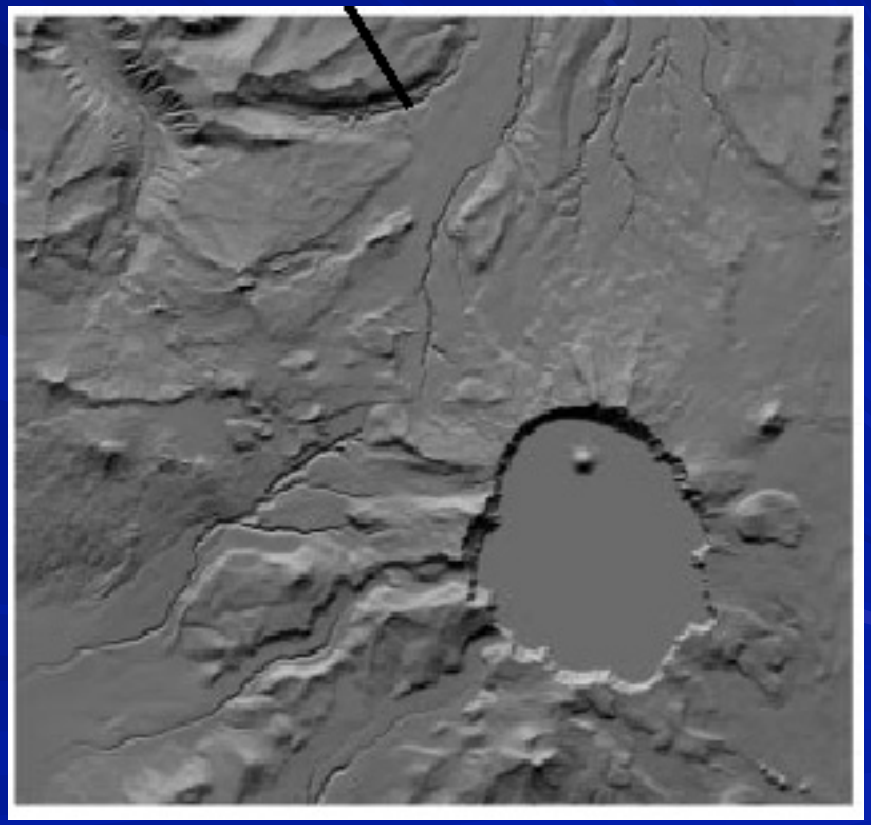


What is this?

One needs to add shadows (shading) for the brain to "get the picture" (and even then there are some problems)

We will therefore "illuminate" the topography and generate an intensity filter to be added to the color or grayscale image.

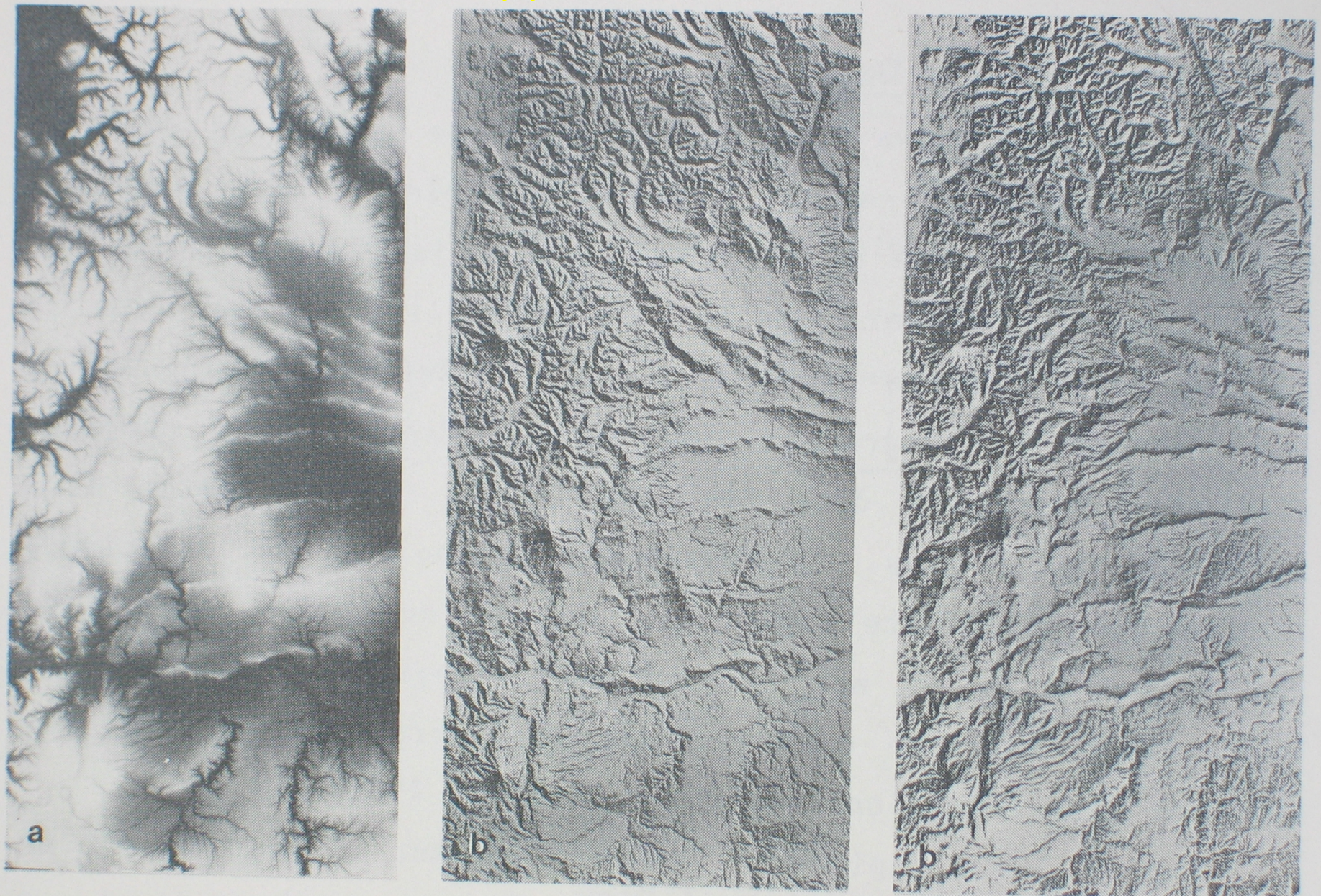




"Raw" data

"illuminated" from upper right

"illuminated" from lower right



GMT has a routine to do this **grdgradient**.

I'll also illuminate the ocean floor and the topography from slightly different angles – to bring out the "best" of both.

After generating the illumination, we have to combine the two files using **grdmath**.

Output files will have **.intns** as extension.

```
NORM=-Nt
BATHILLUM=270
TOPOILLUM=315
grdgradient ${ROOTNAME}_topo.grd -A$TOPOILLUM \
-G${ROOTNAME}_topo.intns $NORM -V

grdgradient ${ROOTNAME}_30stopo.grd -A$BATHILLUM \
-G${ROOTNAME}_30stopo.intns $NORM -V

grdmath -F -V ${ROOTNAME}_topo.intns ${ROOTNAME}_30stopo.intns AND = \
${ROOTNAME}_topobath.intns

INTNSFILE=${ROOTNAME}_topobath
```

So now we have two grid files

One with the topographic data

One with the shading

Now we're ready to plot them together to make the map.

Finally we make our first contribution to the map (PostScript output file) using **`grdimage`**.

`grdimage` can combine the coloring of the data, based on the CPT file, with the shading (which comes from the slopes of the data).

grdimage can combine the coloring of the data, based on the CPT file, with the shading (which comes from the slopes of the data).

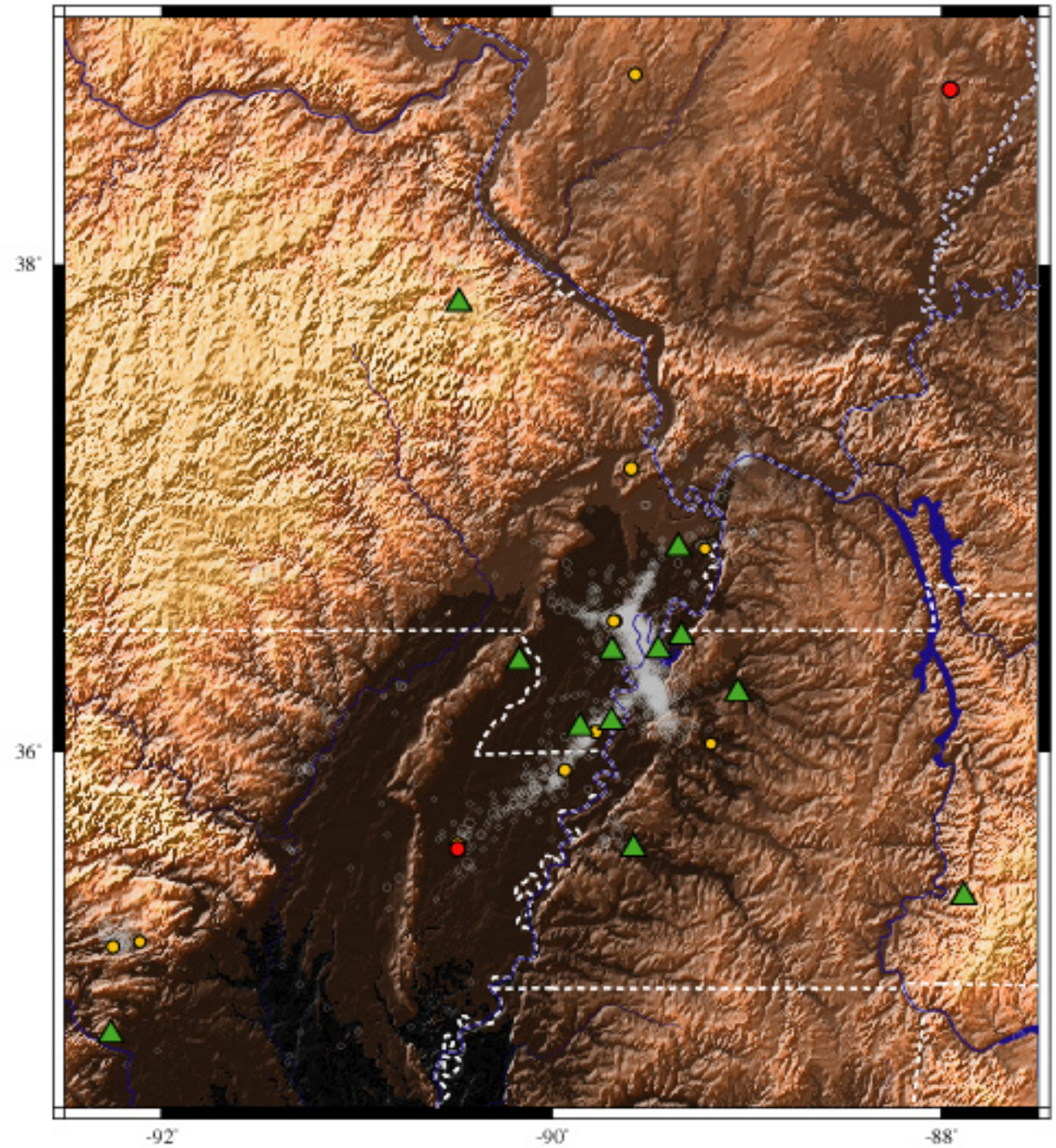
```
echo color topo
CPTFILE=/gaia/opt/gmt/share/GMT_globe.cpt
grdimage $INTNSFILE.grd -I$INTNSFILE.intns -C$CPTFILE -R$REGION -$PROJ
$SCALE $GRID -K -X$XOFFSET -Y$YOFFSET -V $ORIENT > $OUTPUTFILE
echo done with color topo
```

The CPT file is the color table file. GMT has a bunch of them predefined (look in the directory referenced above).

You can also make your own (if you have lots of time)

GMT uses the R/G/B model for color

Now we can add other data (notice **-K**) ~~~ earthquakes, GPS vectors, focal mechanisms, etc.



"copper" cpt file

Now we can add other data – earthquakes, GPS vectors, focal mechanisms, etc.

```
psmeca -R -$PROJ$SCALE -Sd0.2/0/0 -G$RED $CONTINUE -L -W0.5/$BLACK \
india.cmt >> $OUTPUTFILE
```

Again being lazy, I don't like to have to keep track of the last GMT call (to keep track of whether or not I need the -O) so I use $CONTINUE.

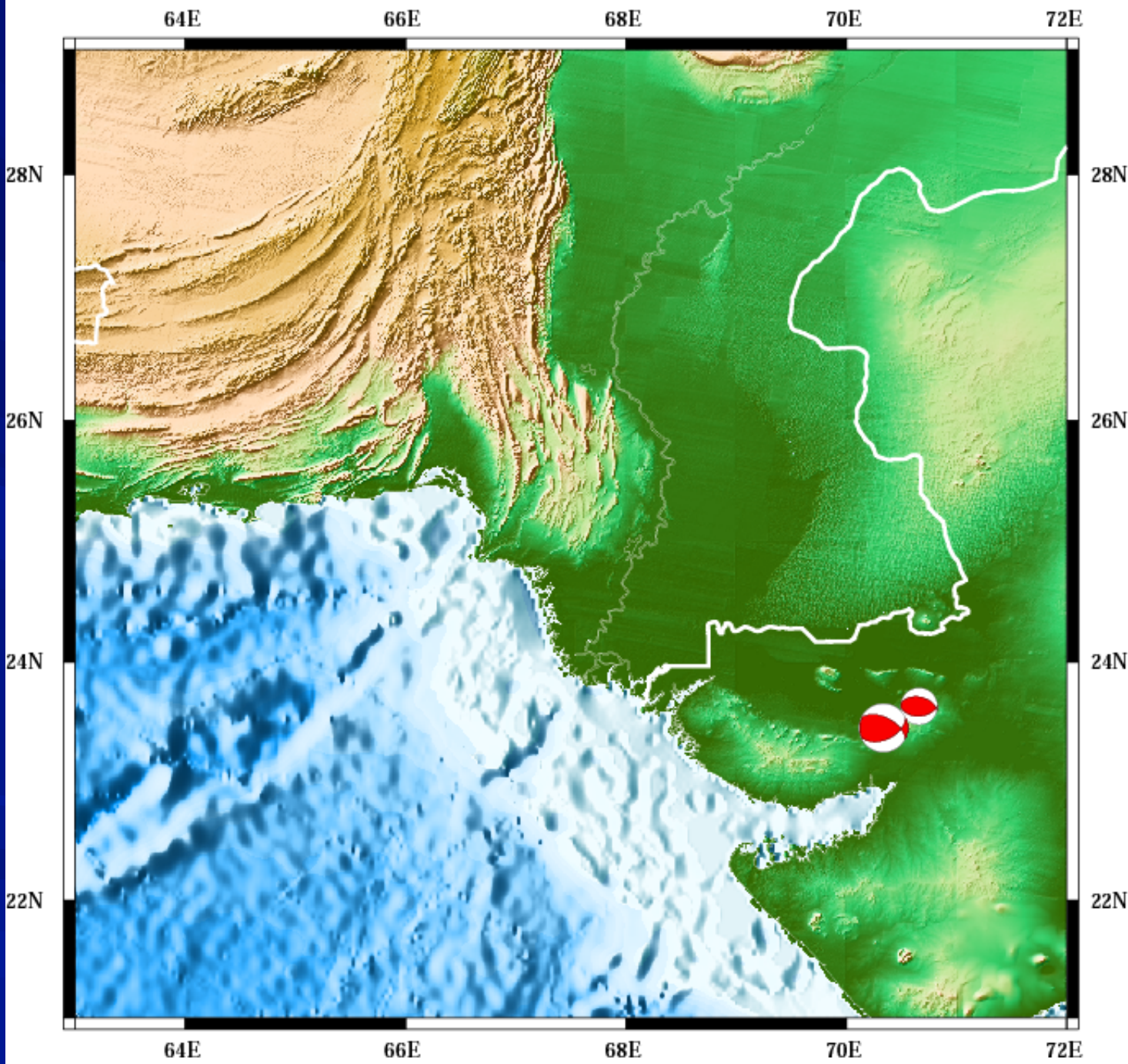
Then I check the output file for a showpage when I'm done – and write the PostScript myself when I need it.

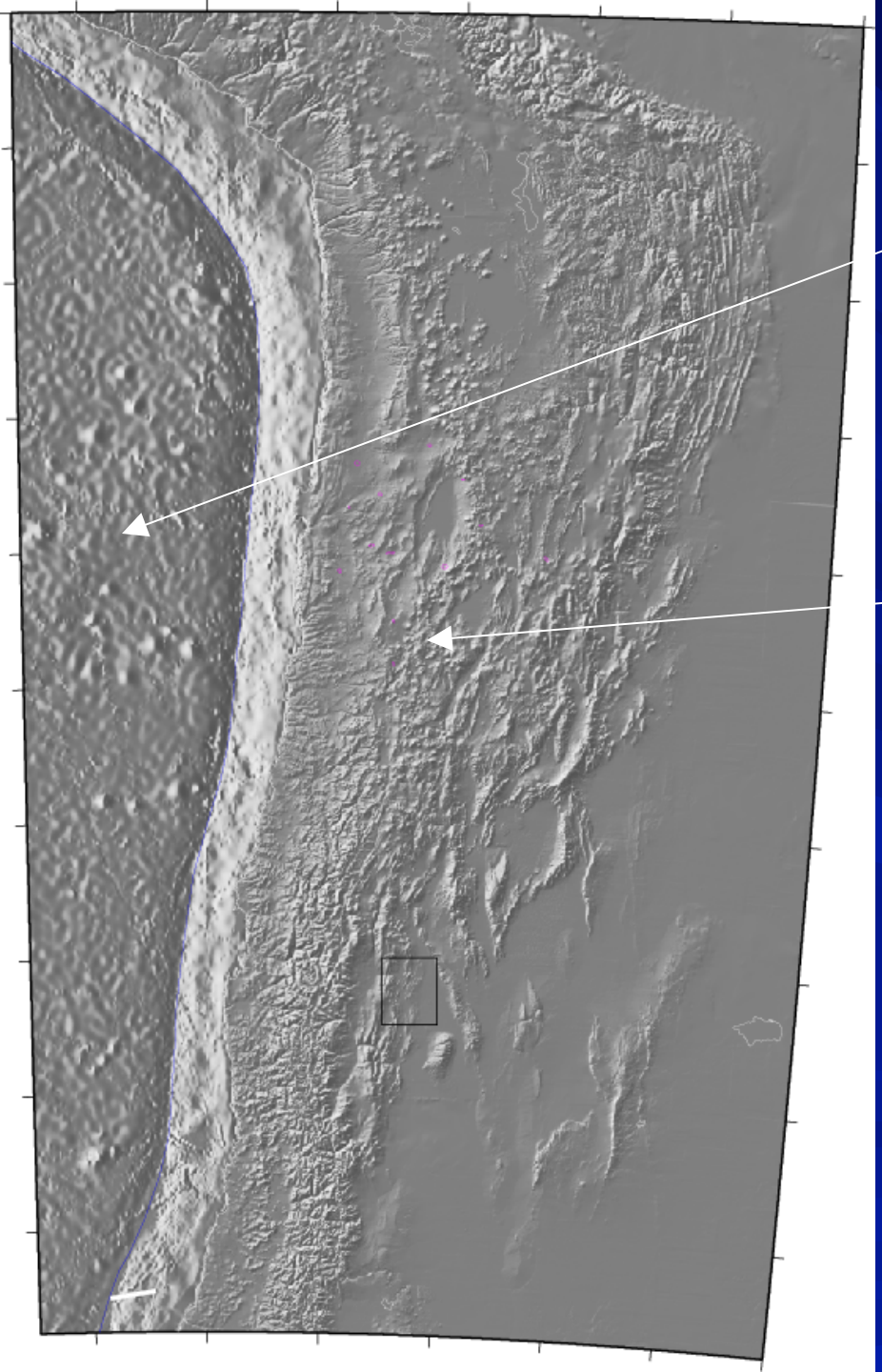
```
echo done with figure - clean up
SHOWPAGE=`tail -1 $OUTPUTFILE | nawk '{print $1}'`
echo check SHOWPAGE -${SHOWPAGE}-
if [ $SHOWPAGE != showpage ]
then
 echo add showpage
 echo showpage >> $OUTPUTFILE
fi
```

We then have to erase all the temporary files we made.

```
if [ $CLEAN = yes ]
then
 echo yes - clean up
 if [ $TOPO != notopo ]
 then

  \rm ${ROOTNAME}.cpt
  \rm ${ROOTNAME}.grd
  \rm ${ROOTNAME}.intns

  \rm ${ROOTNAME}_topo.grd
  \rm ${ROOTNAME}_topo.intns
  \rm ${ROOTNAME}_2mtopo.grd
  \rm ${ROOTNAME}_2mtopo.intns
  \rm ${ROOTNAME}_30stopo.grd
  \rm ${ROOTNAME}_30stopo.intns
  \rm ${ROOTNAME}_topobath.grd
  \rm ${ROOTNAME}_topobath.intns

 fi

 \rm ${ROOTNAME}.nawk
 \rm ${ROOTNAME}.tmp

fi
```

So here's our pretty MAP?





ETOPO -5

global

(5 min)

GTOPO-30

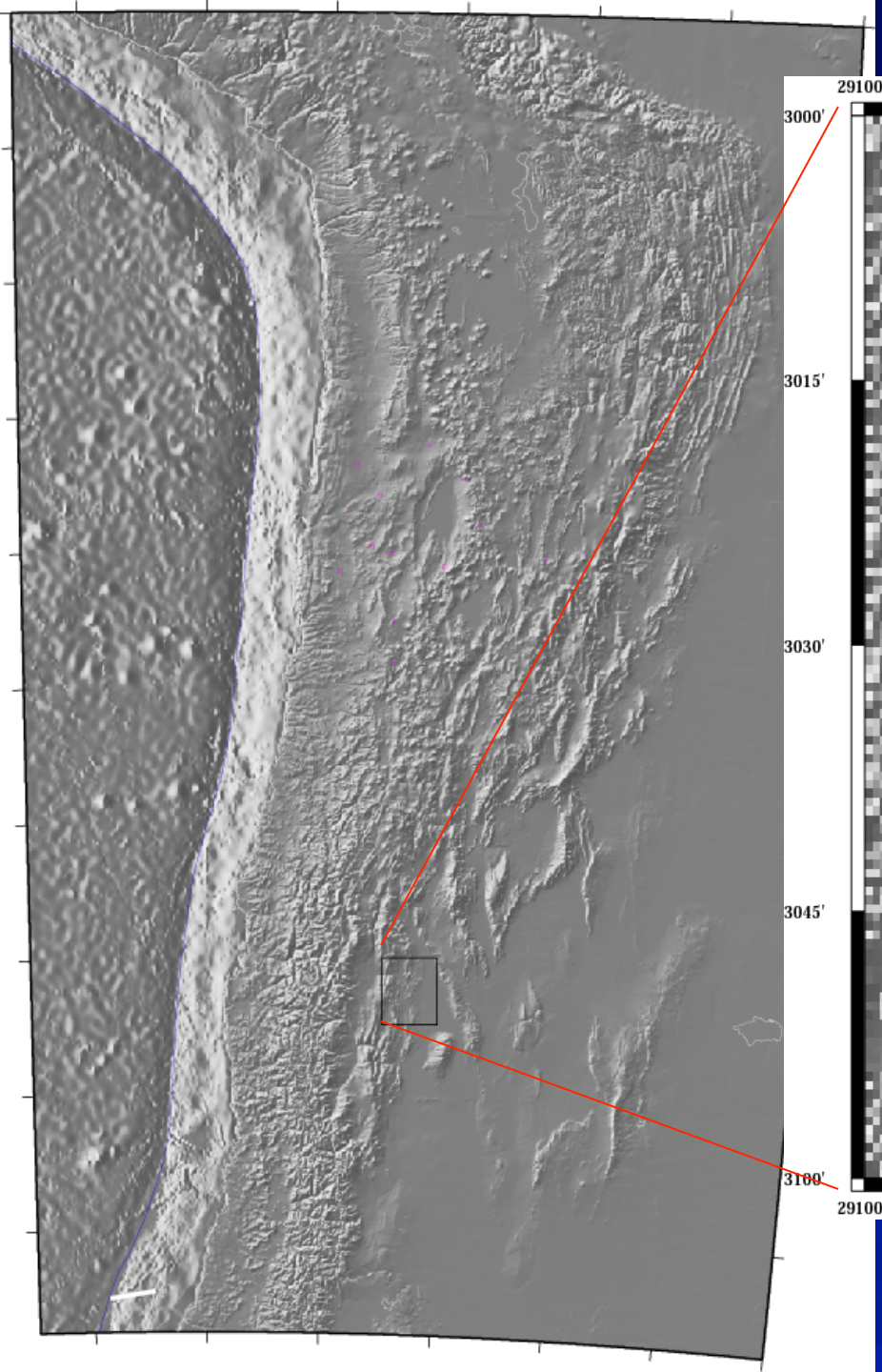
Land only

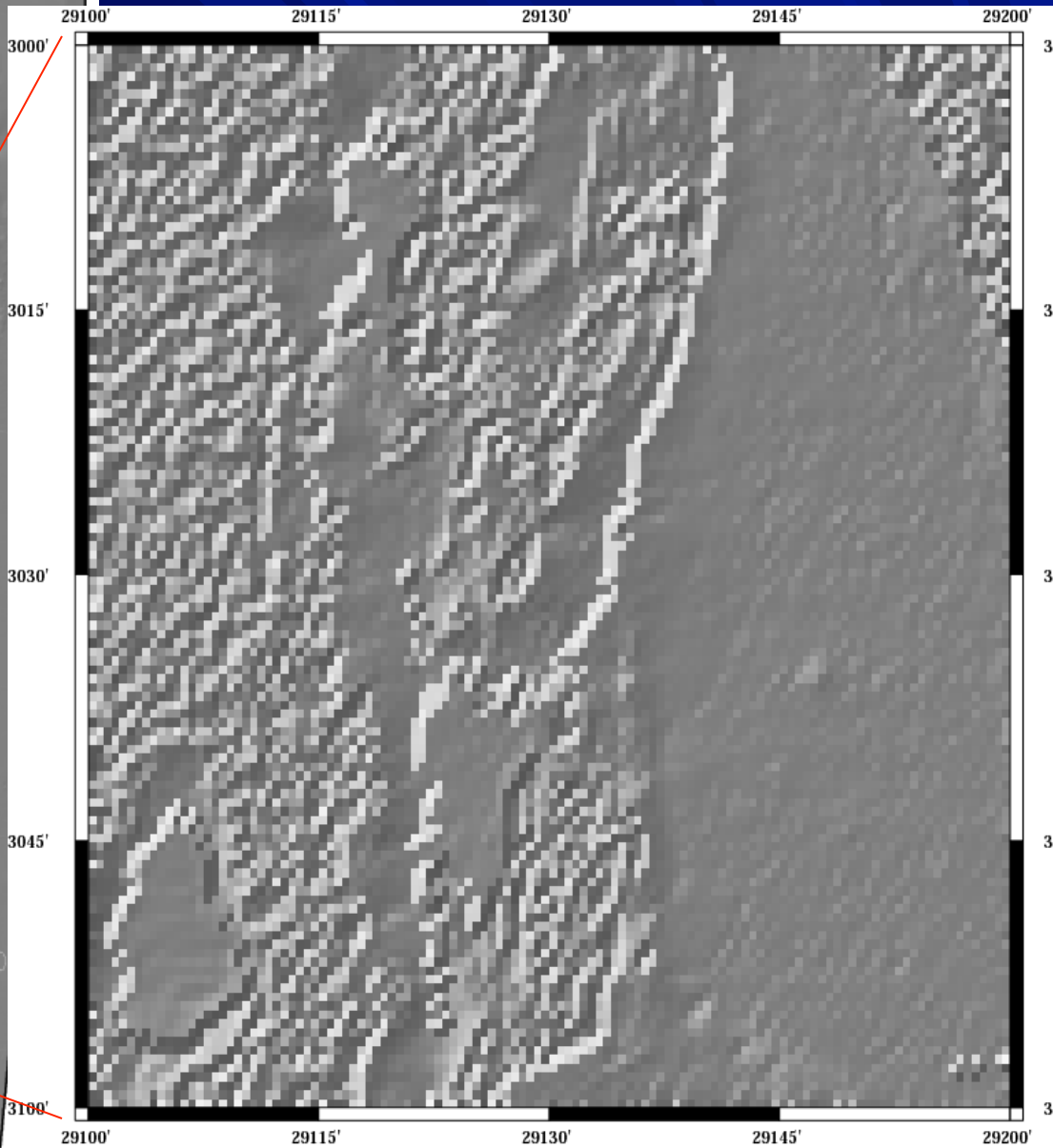
(30 sec)

SRTM

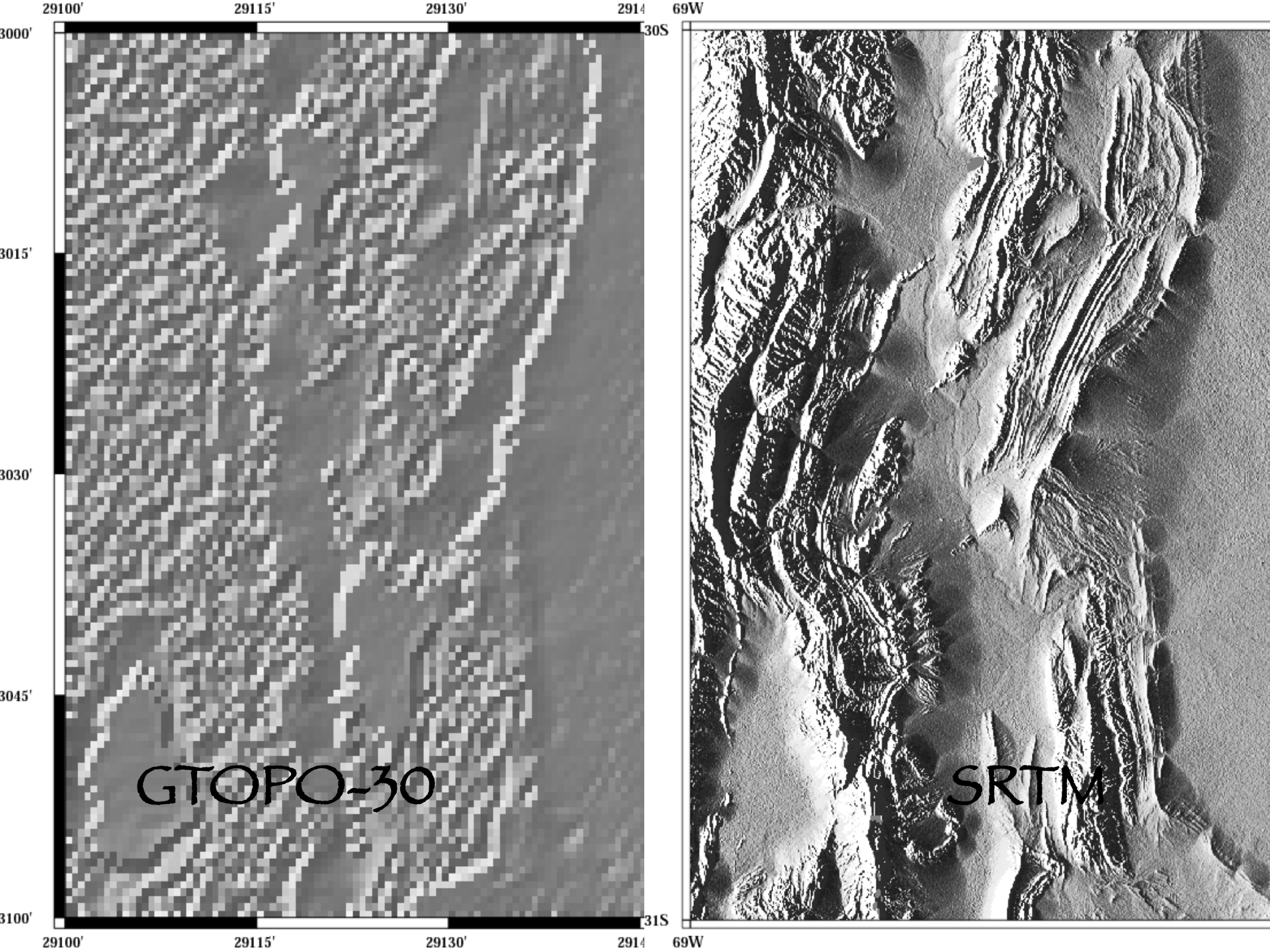
Land only

(3 sec)





GTOPO-30


29100'
29115'
29130'
2914
69W
3000'
3015'
3030'
3045'
3100'
30S
31S
GTOPO-30
SRTM

Plotting a single srtm file

```
#!/bin/sh
\rm tst.grd
grdgradient tile_31_69.grd -A270 -Gtst.intens -Ne0.6 -V \
grd2cpt tst.intens -Cgray > $0.cpt
grdimage tst.intens -Itst.intens -R-69/-68/-31/-30 -Jm7 \
-B1g1a -P -C$0.cpt > $0.ps
```

Plotting multiple 1x1 degree tiles possible, but more slightly complicated (see me).

I can't get SRTM data into grdraster format input file (any volunteers?)

Have covered lots of stuff,

but even more stuff has not been covered

– there are 60 GMT and 35+ Supplemental programs!

Plus power of UNIX to manipulate them.

General GMT shell script will look something like this

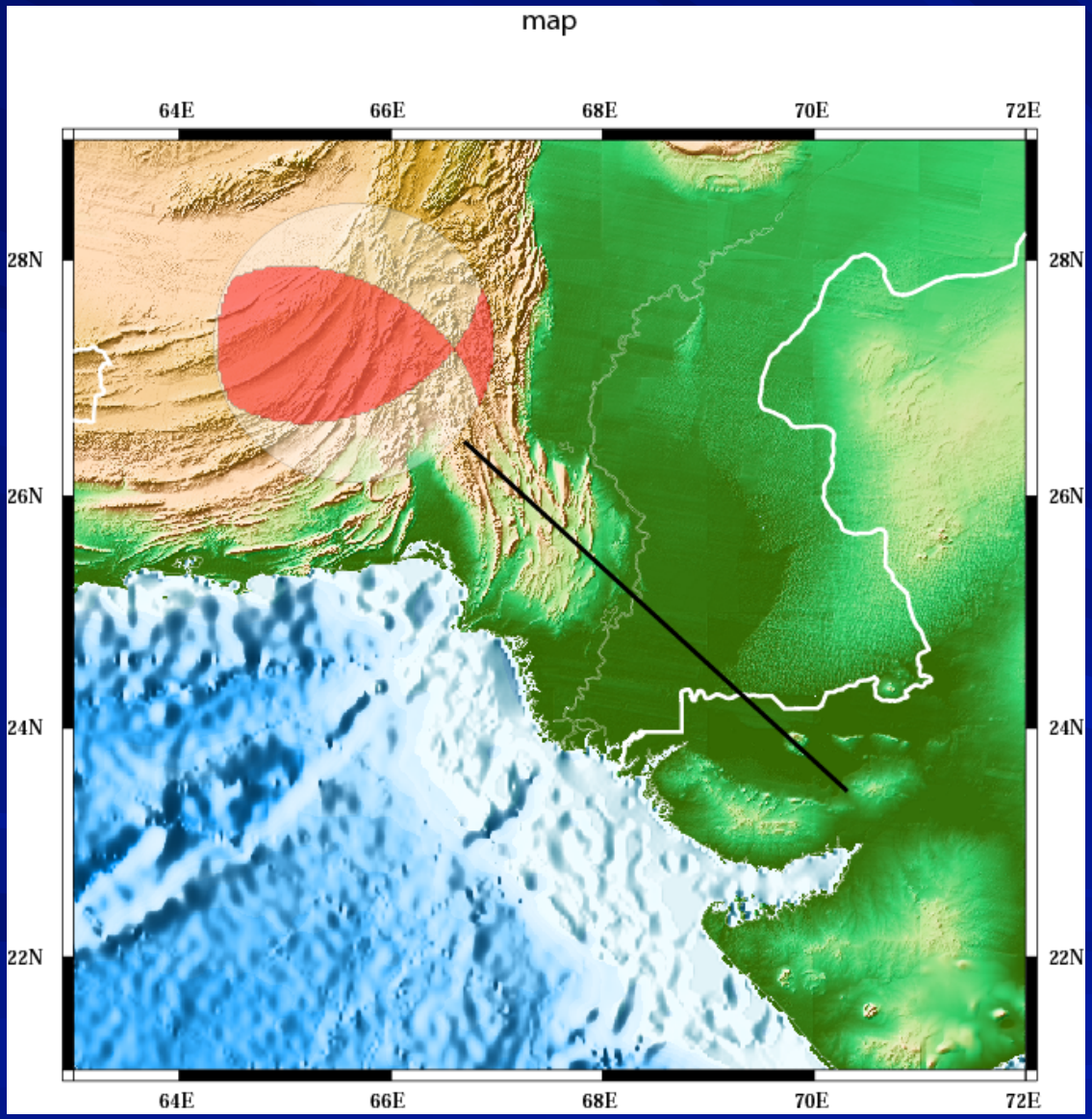
Call to set up base map – this may or may not plot any data

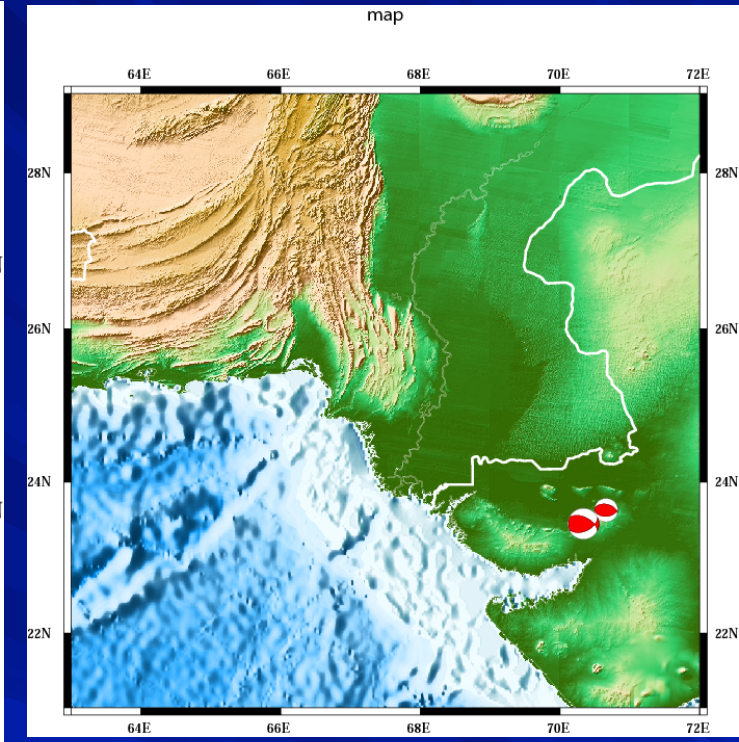
Series of GMT calls to add various kinds of data

Last GMT call "closes" file

Majority of work is in manipulating the data files using all the standard UNIX tools.

Finally, you can put the finishing touches on your figure with Adobe Illustrator (which works with PostScript files)





Lots well documented problems going over to Adobe – principally with annotation/text.

Why is GMT so popular?

The price is right!
(But there's also no such thing as a free lunch!)

Offers unlimited flexibility since it can be called from the command line,
inside scripts, and from user programs.

Has attracted many users because of its high quality *PostScript* output.

Easily installs on almost any (including windows) computer.