DEPARTMENT OF THE INTERIOR

U.S. GEOLOGICAL SURVEY

A Collection of MAPGEN Command Files Suitable for Hacking

by

James M. Robb[1]

Open-File Report 89-653

November 1989

[1] Branch of Atlantic Marine Geology, U. S. Geological Survey, Woods Hole, MA 02543

30 Nov 1989

U. S. Geological Survey
Branch of Atlantic Marine Geology
Woods Hole, MA 02543

Open-file Report 89-653

A Collection of MAPGEN Command Files Suitable for Hacking

Not quite a follow-the-dots tutorial

or, as the old feller said,
"It ain't gonna be easy, but you can get there from here."

compiled by Jim Robb

I. Introduction

This report is a compilation of MAPGEN command files (Unix executable files, or shell scripts) that can be conveniently modified or used as models to create plottable map-overlay files from digital marine data maintained by the USGS Branch of Atlantic Marine Geology (BAMG). It has three Appendixes: on using a DOS-based PC with Kermit as a remote terminal for the map making process; on digitizing lines from existing maps using the BAMG digitizing tables and application programs; and on Kermit scripts for automating a phone contact with the BAMG central computers.

The MAPGEN map-plotting system, written by Gerald Evenden (Evenden and Botbol, 1985; Evenden, 1986; and aperiodic unpublished updates of program documentation), uses a core of about a dozen programs which create computer files of plotter-useable coordinates known as overlays from files of latitude-longitude data. The most frequently used of those programs are called *mapdef, grid, coast, lines, points,* and *legend.* Each overlay file holds a specific part of the map, such as the geographic grid, a coastline, a title, or a set of bathymetric contours. The overlay files are fed to a plotter via a program called *plotter*, which specifically instructs the plotter how to move its pen.

Making a basemap with MAPGEN programs and a few basic geographic data files is a straightforward operation once the models are established, and the files compiled here will serve to do that quickly. Some of them embody brute-force rather than elegance, but they have worked for me. You can easily change the parameters within the basemap, geographic, and bathymetric command files, and you will quickly have your own map.

The script files for trackline and sample plotting are included here as examples. The data files they were written to process are not part of BAMG's standard support-data files, and rewriting of most of these scripts will be needed for new applications.

Plotting tracklines or sample locations can be complicated because of the wide variety of data formats you may encounter. By default, MAPGEN programs digest a data file with latitude, longitude, time, shotpoint, etc, fields using *^I* (control eye, or tab) as the field delimiter. Point-data files (e. g., core sites) can be reformatted by using the Unix editor or they can be fed through Unix utilities such as *awk* or *sed* and piped into the plotting program. Trackline data, on the other hand, usually require elaborate editing to recast them into forms that will plot as lines with tics and annotations at selectable intervals. There is a program that was written to do this job whose use is illustrated here.

You will need the documentation for the MAPGEN system (Evenden and Botbol, 1985; updated versions of MAPGEN documentation are available within BAMG from Jerry Evenden) and Unix instructions. Hampson and Wright (1988) provide examples of MAPGEN and Unix script files to do a complex map-publishing task. Helpful references for Unix programs and shell script syntax include Kernighan and Pike (1984) or Morgan and McGilton (1987).

II. <u>Organization and contents:</u>

A. The Basemap: map definition, grid, and peripheral information

ddomap	first step: creates a map definition file
dogrid	draws a latitude and longitude graticule
doborder	makes a border around the map
doprojnote	sets up a map projection label
doscalbar	makes a bar scale for the map
doscalfrac	puts a scale-fraction label on the map
dover	makes a creation label (office ID, directory, date, time)
dotitle1	emplaces a main title
dobasemap	a list of the above files except *ddomap* so you can run them as a batch job after you set them up with the editor.

B. Geography: Coastline, Rivers, and Politics

docoast	draws a coastline
dorivers	puts the rivers on your coastline
dostates	draws the state boundaries on the land areas
doeez	draws the EEZ boundary

C. Bathymetric contours

doconxx	plots individual contours from digital data compiled for the GLORIA east-coast atlas
doatconts	plots bathy contours from 36°N to 42°N as a single overlay file: 20 m contour interval on shelf, 100 m on slope and rise
doconshelf	plots 20-m contours on continental shelf as individual overlay files
doconhuns	plots 100-m contours (except the even thousands) as individual overlay files
doconthou	plots the 1000-m contours as individual overlay files (so you can plot them in a different color
docont250	plots 250-m contours (those derived for the GLORIA east-coast atlas) as a single overlay file.
dogeography	another batch list that puts on the coast, the culture, and the 20-m shelf and 100-m slope-and-rise contours

D. Tracklines:

dofarn2	plots Farnella 87-2 cruise track from merge3 navigation format using seltime
doa22	plots AII-89-1 cruise track from WHOI's mbatr navigation format using seltime
lines.par	a parameter file
dois781	plots Iselin 7807-1, illustrating extraction from a TSD file and a data set without even-interval records, where *seltime* is not helpful.
dogy8210	a complex script sequence illustrating *awk, uniq, seltime,* and *sed.*

E. Samples and Sites:

dodsdpwells	plots DSDP well sites
dostratwells	plots East Coast stratigraphic exploration wells

doilwells	plots East Coast oil exploration sites
dongdccores	plots core sites from an NGDC file; illustrates *awk*
docores	plots from NGDC core curator's file
dodives	plots sites from an Alvin-dive file
dolegnl	plots a legend to explain the Alvin-dive symbol
doemerysamps	plots Emery-project (WHOI-USGS) sample sites
dosedlab	plots sample sites of USGS-BAMG sediment analyses
dowhoicores	plots sites of samples from WHOI's sediment lab archive; output of *muddie,* a WHOI program
dodumpsite	plots outline of DWD106 municipal sludge dump site

F. <u>Viewing and Plotting:</u>

preview and *zoom*	plot-viewing programs
plot970	plotter runline for Calcomp 970

III. <u>How to use these examples:</u>

Copy the contents of the directory */eez/demo* from my area on the BAMG Masscomp system (aka geosvy, or Shemp) using *cp,* as *cp /eez/jmr/demo/*/** . or create the files for yourself by typing them into your own machine.

Ensure that the script files are executable *(chmod +x ddomap do*)*. Be sure the files are in a separate directory to isolate the map files from your other files. It will quickly be very messy if you commingle your files. Use a separate directory for each map you make.

The first MAPGEN program to be run (*mapdef,* using the script file I call *ddomap)* calculates an x-y map matrix based on your specifications of projection, scale, boundaries, any rotation of the map, and the width of bordering areas. *Mapdef* creates a file containing the framework used to create plottable overlay files, but which by itself cannot be plotted. The *grid, lines,* and *points* programs, as packaged script files run subsequently, create the overlay files, which are based on the map definition in the .def file and contain the plotter instructions.

The script files for the basemap should run as they appear, and require no alteration to produce a model map. The geographic overlay files that call data in */coast* should also run as they appear (assuming the Branch files of geographic data remain on the system in that directory). Some pathnames of contour-line data refer to files within my home directory and will have to be modified for your situation. On BAMG's Shemp computer change $HOME to */eez/jmr.*

Run *ddomap* then *dobasemap.* Use *preview map.def *.ov* to look at the assemblage or use *preview map.def filename.ov* to see an individual overlay. Try the files individually to see what they create. Use *vi* or *edit* to modify the script files for your own map areas and overlays.

To see an overlay, use *preview* or *zoom* to display it on your terminal; see runlines for those programs below. There is information in Appendix I for those using DOS-machines as terminals, using Kermit and its graphics emulations. If you run a program and then rerun it with changed input parameters (trial and error) to preview it again, the programs will overwrite old overlays, so your directory will be current.

To plot as hard copy, on paper or mylar, etc., see your computer gurus and use *plotter;* I include runlines for the BAMG's Masscomp and the BAMG's Pacific Microcomputers, Inc. system (the unit identified as "Moe") as configured today.

IV. <u>Some notes on Unix quirks:</u>

On Unix operating systems, programs known as shells act as command interpreters. There are several shells which are very similar to each other, but which have small differences among them in the command syntax they recognize and in some of their capabilities. More than one shell is likely to be present and to operate on the same machine at the same time, so a user can invoke the one he or she wants. The most common shells are the Bourne shell and the C shell. The C shell has some conveniences (such as *history* and the tilde (~) as "home directory") that the Bourne shell does not have. The script files contained in this report were created on a system using a C shell as the default for users on terminals, but having a Bourne shell resident and set up to run most scripts. Some of these scripts will run under one of these shells but not the other. A pound sign (#) as the first character in an executable script file will specify the C shell. Other characters in first place invoke the Bourne shell; for convention the use of a : in first place in the scripts of this report is used to guarantee Bourne shell operation.

In order to make the script file work you must make it executable. Use *chmod +x filename.* If you have a group of files that have a common characteristic, you can use wildcards. As these files begin with *do,* you can use *chmod +x do** (except for *ddomap,* which I find convenient to keep separate). Alternatively, you can use the command *sh filename,* which will invoke the Bourne shell and execute the file.

Within a script file a pound sign (#) as the first character on a line <u>not</u> the first line marks a comment. Comments retained in the script files help you modify the script for your next map. One can "comment out" some lines within a file if they're not in use; thus instructions can be kept with the file about where and how the parameters are put in. A minor complication is that the *points* and *lines* programs of MAPGEN accept commands from within a data file where an initial # is the control-line marker. A second # on those lines functions as a comment marker, however, and the part of a line following a second pound sign is ignored by the MAPGEN programs.

The tilde (~), used to mean "home directory" in a pathname, is a C-shell character, and will not be recognized in Bourne-shell scripts. The $HOME notation used in these scripts is recognized by both Bourne and C shells.

Unix script files execute the commands they contain one after another, line by line. However, the command lines themselves (which include any piped-in editing and filtering sequences, the program name, data filename(s), program options and parameters, and output filename, etc.) must be on a single line not interrupted by a carriage return. Long command lines that won't fit across your screen can be made continuous for the computer by terminating segments with a \ at the right end. That is, the shell command interpreter will ignore a carriage return that immediately follows the \ (see the *doconhuns* file below for and example). This feature is useful, but it can be a vexing source of blunders if spaces are inadvertently included or excluded, and the shell consequently misinterprets your command. Some lines in this report were made short just to fit the pages.

In many of the script files that follow there is use of *<<EOF* in the MAPGEN command sequence. This is a Unix capability of directing a standard_in sequence, and allows use of additional lines within the script file (after the first MAPGEN command line) as input until another *EOF* is encountered (see *ddomap* and *dogrid*). This can keep all the input for a single function together in one file.

V. The Script Files:

A. Basemap files

ddomap -- creates a map definition file:

```
mapdef -mcvs map.def <<EOF
77dw 69dw 36dn 43dn
+proj=aea
1000000
0
5 5 76dw 38d 73dw 37d 70dw 38d 73dw 42d
5 5
EOF
mapdef -v map.def >mapnote
## line 1 mapdef command line with EOF directive
## line 2 W,E S,N,[centr mer] geogr bounds
## line 3 projection codes
## line 4 scale denominator
## line 5 rotation CCW degrees normally 0
## line 6 coord pairs: lower left data area x,y in cm default 3
## and if rotated lon-lat coords on map edges, LBRT (minxy maxxy)
## line 7 right and top margins in cm default 3
## the EOF ends the input
## mapdef -v map.def >mapnote makes an information file
```

Explanation:

Line 1 invokes the program *mapdef.* The *m* in *-mcvs* means that the following name (*map.def*) is to be the name of the output file to be created; *c* means create a map definition; *v* means verbose; it causes messages to be sent to the terminal while the program progresses; and *s* means calculate the scales.

The <<EOF is a Unix shell symbol that means read the rest of the script for program input until you get to the EOF line. This allows the whole file definition to be in one file rather than in a separate parameter file. One can also use the command *mapdef -mvcs map.def parameters.par*, where a separate file, *parameters.par*, would contain the projection, scale, bounds, etc. The way it's done here using the <<EOF notation keeps related command items together and allows less cluttered directories.

Line 2 -- Bounds: The order for coordinate pairs is West East South North. Use DMS notation. Follow degrees with d, minutes with ' or m, and use the cardinal-direction letters for clarity. The program accepts 30 or 30N as 30 degrees north and

71W or 71dw or -71d for 71 degrees west. It will also accept decimal degrees and decimal minutes. See the MAPGEN documentation.

For a rotated map or for maps in some non-rectangular projections, you can specify a wider map area than you intend to display for the final map, and then cut the bounds down subsequently (see line 6 below). Such a procedure ensures that you get your data plotted within a full rectangular area. However, the grid annotations (which you will create in a subsequent step using the script *dogrid*) can write over themselves or appear in ways you don't expect (what can't in this business?), so you will find you have to experiment and see what happens.

Line 3 -- Projection: The *mapdef* program relies on another program called *proj* here. *Proj* provides for a large number of projections. Each has a code name and some require other details. In this example aea stands for albers conic equal area projection. You'll have to see the MAPGEN books here.

Line 4 is straightforward. For 1:500000 use 500000.

Line 5 is straightforward after you have used it. CCW means counterclockwise. To recreate Uchupi's bathymetric maps use -47, and change the projection to laea for lambert conic equal area.

Line 6 and 7 specify the data area of the map by specifying the widths of peripheral borders outside the neat lines, to make room for titles, peripheral notes, explanations, etc. MAPGEN recognizes two map areas (windows): the full map sheet and a smaller data area. Projection transformations are calculated within the data area. Titles, legends, and scales, etc. can be plotted inside or outside the data area (see the -w option for *legend*).

Line 6 holds pairs of coordinates; no punctuation necessary. The first pair, x y, in cm, sets the border widths for the lower left hand corner. A blank line or a single dash as place holder invokes 3 cm defaults. Subsequent pairs of longitude-latitude coordinates of points can be used to locate the neat-line borders of the map -- used for rotated maps or to ensure that the projection-calculated data area extends to a rectangular border. These are four more pairs of entries (*lon*gitude-*lat*itude pairs in order of left, bottom, right, top, with no punctuation). They specify the rectangular area you want to display within the larger projection-calculated area you have specified in Line 2.

A single dash in place of any of those pairs goes to the default values of bounds that you specified in line 2.

Line 7 holds x y coordinates of the upper right border area, defaulting to 3 cm if blank. The 5-cm surround used in this *ddomap* example makes more room for odd labels, etc.

The last line (*mapdef -v* etc...) puts a brief description of the map vitals including the plot size into a file called *mapnote,* handy for reference in the directory.

After running *mapdef* (here via *ddomap),* you have a map defined in the map.def file, but you can't display anything with this file alone. The command *mapdef -v map.def* will display a description of the map that is defined (or use *more mapnote*). The best way to see what you have is to create and display a geographic grid. Run *dogrid,* as follows, and you can preview the result on your screen. To preview you must have the terminal emulation worked out first; see Appendix I.

dogrid -- puts a grid overlay on the map; has the graticule (as lines or tics) and lat-lon annotations.

```
grid -m map.def -o grid.ov <<EOF
-f -crp             # font call for annotations
-pi 1   -mi 1       # p parallel, m meridian grid interval
-pu .5    -mu .5    # u major tic size cm
               ## comment the -pu -mu out (using ## as first
               ## characters) to make a line grid
-s .4          # cm size of annotation characters
-d .8          # cm distance of label offset from line
-a rltb        # grid annotations right, left, top, bottom
-p 0           # specifies pen number
EOF
```

Adding a *-i* to the command line of the *grid, lines, points,* or *legend* programs will display the grid on your terminal's screen as it's being created (if you have a graphics-capable terminal). There is an extensive list of fonts that can be used for labels and annotations. See the MAPGEN documentation. The *grid* program can make major and minor tics as well as grid lines.

The following few files use the program *legend* to put a border and some of the basic information notes on the map. You can combine the border, scale fraction, and scale bar, and other legend products into one file if you want to, but the separate files (and the separate overlays they produce) are convenient because each can be picked and chosen for different maps. See the MAPGEN documentation for *legend.*

doborder -- draws a border around the data area of the map; note that because the border will be rectangular it may not line up with the latitude-longitude lines, depending on the projection you use, and the latitude-longitude annotations may not look right. Albers equal area is narrow at the top, for example. Similarly, unless you have defined a wide area map on line 2 of *mapdef* and then bounded it on line 6, your data may not plot to this border.

```
legend -mo map.def border.ov <<EOF
-w d      # select data window (inside the periphery)
-L 2      # pen for border
> 0       # start draw from righthand edge at bottom
> >
0 >
0 0
> 0
.         # close the line
EOF
```

doprojnote -- labels the projection: you have to edit this so it says the correct projection. Note the use of | to mean refer to centerline for label position. Similarly, > can be used for right or top reference in files like this (see *dotitle1,* below).

```
legend -mo map.def projnote.ov <<EOF
-w d           # data window (origin for subsequent coordinates)
-f -crp        # font call
-s 0.25        # size cm of font
-xy | -4       # xy position cm
               ## | -4 means origin center & 4 cm below data
window
-j c           # center justify text
-t             # title text
Albers Equal Area Conic Projection
.
EOF
```

doscalbar -- puts a scale bar on the map.

```
legend -mo map.def scalbar.ov <<EOF
-w d           # select data window origin
-fs - .3       # font = default, char size cm
```

```
-xy |-4 -3       # point to measure from; | means centerline
                 ## |-4 means 4 cm left of center
-b 10,km,5,5     #scale bar, 10 km/unit, 5 units, 5 intervals left
                 ## of 0
EOF
```

doscalfrac -- puts a scale statement (as Scale 1:500,000) on the map.

```
legend -mo map.def scalfrac.ov <<EOF
-w d             # select data window origin
-f - -s .3       # font = default, char size cm
-j c             # center justify
-xy | -1.5       # measure from x and y; cm; | means x at
                 ## centerline
-d               # specifies scale fraction
EOF
```

dover -- (for do version) puts a USGS-credit, source-directory, and date-of-creation note on the map. Then you know where the map came from when you look at it later. In the form written below it is placed in the lower left corner (-x 4, -y 2), out of the way, but print size is not dependent on map size so the script can be used for most maps without constant readjustment.

```
: ## Bourne shell script
(cat <<EOF;echo USGS BAMG; pwd; date;.)| legend -mo map.def\
 vernote.ov
-w d                  # data window origin
-f -crp               # font call
-s .15                # size cm
-l .40                # leading cm (space between lines)
-p 2                  # pen
-x -4                 # x position cm
-y -2                 # y position cm
-t                    # to print title texts from runline
EOF
```

The *-t in these files* tells the program to expect title text on the next line. Above, *echo USGS BAMG* prints the characters USGS BAMG; *pwd* prints out the name of the working directory, and *date* prints the date/time.

dotitle1 -- puts a title on the map; Note use of | and > to refer to center and top of map's data area. You can change the placement using x-y coordinates (cm) of the map area. Change the coordinates and the fonts for assorted legends or labels (see *dolegn1*, below).

```
legend -mo map.def title1.ov <<EOF
-f -crp             # font call
-s 1.5              # size cm
-w d           # data window origin
-xy | >3       # xy position; | centerline; > top edge
-j c           # text center-justified
-p 0           # pen
-t             # title text
U. S. Mid-Atlantic East-Coast Offshore Region
.              # period ends title text
EOF
```

dobasemap -- a batch-file example to get the basemap done in one command; just make sure the script files it calls hold the right stuff first.

```
dogrid
doborder
doprojnote
doscalbar
doscalfrac
dover
dotitle1
```

B. <u>Geography -- coastlines and other commonly displayed features:</u>

The coast program draws on specifically prepared (packed) data sets that are maintained in the */coast* directory of BAMG. There are DOC files in that directory that explain its contents.

docoast -- puts a coastline on the map; calls a widely-used coastline-data file assembled by the CIA.

```
coast -mo map.def coast.ov /coast/na/cil
```

Further note: Digitized files of coastlines, rivers, political boundaries, and of some bathymetric contour lines are maintained in the /coast directory on the BAMG Unix system. Many of those files are compressed and use the *coast* program for plotting. Other line-data ("vector") files for bathymetry, the EEZ boundary, and navigation files are plotted using the *lines* program. See examples farther below. There is an informative README file in */coast*.

dorivers -- rivers add interest to otherwise boring coastlines

```
coast -mo map.def rivers.ov /coast/na/riv
```

dostates -- adds state boundaries

```
coast -mo map*.def coast.ov /coast/na/pby
```

doeez -- puts that ole EEZ boundary line on the map

```
coast -mo map.def eez.ov /coast/eez
```

C. Bathymetric Contours:

The several scripts below use different ways to get bathymetric contours. A lot depends on the data sets; at BAMG we have at least four: (1) compressed data in the */coast* directory, which is part of the CIA geographic data set, at wide contour intervals; (2) Generally 250-m interval contours digitized for the GLORIA east coast atlas; and (3) 20-m continental shelf contours and 100-m slope and rise contours digitized by National Mapping Division and at the Office of Energy and Marine Geology (OEM) in Reston; (4) several 1000-m contours, ocean-or-worldwide. Please consult the BAMG Database Coordinator.

Lines, such as bathymetric contours, can be plotted with *coast* if the data are in the *coast* format, as in the *doconXX* script below, or they can be plotted using the program *lines,* as in *doconshelf* and other files below.

The *lines* program reads files of latitude longitude points and plots a line. (It can also annotate the data points.) It is governed by commands inserted at the beginning of and within the data stream. It reads data files in sequence, and the first data file can be a *lines.par* (parameter) file which contains the line-drawing commands. In the following files the function of a *lines.par* file is contained in the <<EOF notation, where the program reads to the EOF and is guided by the MAPGEN commands which precede the EOF. It's a way to keep the commands in a single model file, and allows a neater directory list.

docon50, docon100, docon150, docon200 -- these executable files call the digitized-contour files assembled for the GLORIA east-coast atlas by Eric Schmuck, Dave Lubinsky, et al., of BAMG. Examine the */coast* directory. Change the *mgf.** filename to plot the contour you want.

```
lines -mo map.def cont50.ov /coast/bathy/gloria/mgf.0050
lines -mo map.def cont100.ov /coast/bathy/gloria/mgf.0100
lines -mo map.def cont150.ov /coast/bathy/gloria/mgf.0150
lines -mo map.def cont200.ov /coast/bathy/gloria/mgf.0200
```

doconshelf -- this script makes overlays for all the shelf contours using digitized data from the OEM. The *for i* structure, a shell feature, does the listed depths one at a time, and individual *.ov* files are produced.

```
for i in 0020 0040 0060 0080 0100 0120 0140 0160 0180
do
cat - $HOME/data/bathy/atbathy/??.$i.mgf <<EOF | \
lines -mo map.def con.$i.ov
#-p 0          # pen
#-l            # begin line plotting at subsequent data fields
#-d 2,1        # longitude field 2, latitude field 1
EOF
done
```

The $HOME means home directory (for these data on BAMG's Shemp, */eez/jmr*). It is like tilde (~), but is recognized by both Bourne and C shells in script files. The tilde is not a reliable symbol because it is only used by the Bourne shell. Check your own pathnames for the data files. The lone hyphen (-) in the above string shows where the standard_in directed from the <<EOF goes into the string. The \ just breaks the line so it will fit in this presentation here without breaking up the *lines* command line.

doconhuns -- Plots the 100-meter contour lines, omitting the 1000-meter contours: I did this so the 1000-meter lines could be plotted in another color.

```
for i in 0300 0400 0500 0600 0700 0800 0900 1100 1200 1300 \
1400 1500 1600 1700 1800 1900 2100 2200 2300 2400 2500 2600 \
2700 2800 2900 3100 3200 3300 3400 3500 3600 3700 3800 3900 \
4100 4200 4300 4400 4500
do
cat - $HOME/data/bathy/atbathy/??.$i.mgf <<EOF|lines -mo map.def
con.$i.ov
#-p 0          # pen
#-l            # begin line plotting at subsequent data fields
#-d 2,1        # longitude in field 2, latitude in field 1
EOF
done
```

Note that the backslash (\) at the end of lines here allowing a very long line to be input as a command. See the remarks above under "Unix quirks...". The string of numbers is terminated by a carriage return (invisible here) following the 4500, and the command *do* is on the next line.

doconthou -- plots the 200-m and thousand-meter contours. They are put in a separate plot file here so they can be given another color for readability on the map.

```
for i in 0200 1000 2000 3000 4000
do
cat - $HOME/data/bathy/atbathy/??.$i.mgf <<EOF| \
lines -mo map.def con.$i.ov
#-p 3          # pen number
#-l            # begin line plotting at subsequent data fields
#-d 2,1        # longitude field 2, latitude field 1
EOF
done
```

doatconts -- this will plot selected NMD/OEM contours. Usage is: *doatconts depth_number*.

```
for i
do
cat - $HOME/data/bathy/atbathy/??.$i.mgf <<EOF| \
lines -mo map.def con.$i.ov
#-p 3          # pen
#-l       # begin line plotting at subsequent data fields
#-d 2,1        # longitude field 2, latitude field 1
EOF
done
```

docont250 -- plots GLORIA-project 250-m contours as a single overlay file

```
coast -mo map*.def cont250.ov /coast/bathy/gloria_cnt
```

dogeography -- another batch file example; make a list and make its filename executable in the same manner as *dobasemap*.

```
docoast
dorivers
dostates
doeez
doatconts
```

D. <u>Tracklines:</u>

Trackline plots are not simple because they require tics and annotations that can be plotted at different intervals for different scales or purposes. Annotations may represent time, shotpoint, magnetic intensity, etc. MAPGEN's *lines* program accomplishes the task by using two-part data sets; it first creates plots of the lines and associated annotations, then goes on to the second set of data and plots the tics. The input-data file has to be set up in two sections: the list of fixes and annotations that make up the line, and then an appended set of tic locations. Line breaks (where tracks are not to be plotted) are marked by a *#-b* command within the data file, which instructs the plotter not to draw a line between two sequential points. Setting up the data set and the plot parameters (either in a *.par* file or prefixed to the data set) is critical. See the MAPGEN documentation for the *lines* program.

Navigation data come off the ship in various ASCII formats that contain consecutive fixes. There are reformatting routines such as *mbrfmt, cmrfmt, fxrfmt, sgrfmt,* or *mrg3rfmt* (for MBATR, CALCM, FIXSE, SEAG, or merge-merge formats), or one can use *awk* to reformat navigation data for a program called *seltime*, written by Valerie Paskevich, which picks out tics and annotations at selectable intervals, and produces a new data set of two parts, along with default commands for *lines*. See the BAMG documentation for *seltime*.

Trackline data in "*lingen*" format (named from an obsolete MAPGEN program), which include data sets at several tic and annotation intervals, are stored on magnetic tape in the BAMG data library for a number of USGS BAMG cruises until about 1982. Those data must be edited to insert a *#-b* where line breaks occur, and the MAPGEN commands must be updated.

dofarn2 -- Plots Farnella 87-2 tracks in the Mid-Atlantic bight; hourly tics and 12-hour annotations.

```
mrg3rfmt <farn2nav | seltime -a 12 -t 1 | \
lines -mo map*.def farn2.ov
```

Here's what the mrg3 data look like (minus some spaces to get them on the page).

```
FARN2/87 870302 0710 36.7153 -74.6442 395 393 1053317  102
FARN2/87 870302 0712 36.7196 -74.6427 395  393 1053319  100
FARN2/87 870302 0714 36.7239 -74.6411 395  393 1053321   98
```

```
FARN2/87 870302 0716 36.7282 -74.6396 395  393 1053323   95
FARN2/87 870302 0718 36.7324 -74.6381 395  463 1053325   94
```

doa22 -- plots Atlantis II 89-2 tracks from the mbatr format found on the data-library tapes (a WHOI format). This example shows the placement of the lines.par input.

```
mbrfmt <a22nav | seltime -a 12 -t 1 -l lines.par | \
lines -mo map.def a22.ov
```

The *-a* and *-t* call for the annotation and tic intervals to be provided by seltime.

When *-l lines.par* follows the seltime annotation command a *lines.par* file is inserted in the data following the MAPGEN instructions inserted by *seltime* as you see them above. The MAPGEN processes follow the instructions they have last seen, so old instructions are overridden by new instructions that follow them in a data stream. The *lines.par* file is a separate file. In it you can specify font and character sizes, pens (colors) etc. It can be a general file such as the following, and you can use the ## to comment out the parts you don't want:

```
## lines.par
#-l       #start plotting lines
#-c       #start plotting annotations
#-sf F -sc C  -ss S #symbol font,character,size
#-cf F -cc C  -cs S #annotation character font,character,size
##             (See the MAPGEN manual and Replace the caps)
#-p 1          #pen number, pick your color
##-d 2,1       #fields lon, lat
##-f 3         #post field 3
```

dois781 -- plots an older data set where the hourly or 5-minute intervals were not recorded precisely on the hour or the 5-minute times. *Seltime* won't work where there are no properly numbered intervals (i. e., 5, 10, 15 ... minute) records. One approach to get some workable tracklines for your data plotting is to label all the points. The data were stored in TSD format, a formerly used binary data-storage format of BAMG. A translation of the TSD file (using the program *tsd*) produced the following file (Iselin 7807-1 navigation):

```
20 87823122+04  35  16.689 -74 -26.94700 600
20 87823172+04  35  17.759 -74 -26.52700 600
20 87823222+04  35  18.779 -74 -26.18600 600
20 87823272+04  35  19.889 -74 -25.63800 600
```

Then, using *fxrfmt* produced the next data set, to which I have added a 3-line header for identification and a format note in my data directory:

```
## Iselin 7807-1 navigation (aka IS781) from FIXSE format
## lat      lon     yr     mo     dy     hr     min
## ($0,4,8) (15,9) (26,4) (31,2) (34,2) (37,38) (40,41)
   35.27815   -74.44900  1978  8 20 23 12 12
   35.29598   -74.44200  1978  8 20 23 17 12
   35.31298   -74.43633  1978  8 20 23 22 12
   35.33148   -74.42717  1978  8 20 23 27 12
   ...
```

The above data file that I chose to use and to keep in a data directory is then processed with the following script:

```
cat $HOME/data/nav/is781dat \
| sed -e '1,3d' | \
awk '{print substr($0,4,8)"d\t"substr($0,15,9)"d\t"\
substr($0,34,8)}' | lines -mo map.def is781.ov -c "-l -d 2,1 \
-c -f 3 -cs .2 -cf -tri -cx 0.2 -cr o -s -sr o -sf - -sc - \
-ss .3"
```

The *sed* step removes the header; *awk* creates a 3-field data stream of latitude, longitude, day_hour_minute; *lines* plots those data, each point with a tic and an annotation of day_hour_minute. Note the inclusion within the *lines* command of all the plotting specifications. They are enclosed within the double quotes following the *-c*. This script thereby avoids the extra baggage of a special *lines.par* file. Note also the backslashes used as line breaks here so that the Unix shell will treat the script as one line. Be careful with the spaces around the backslashes.

dogy8210 -- A complex sequence of filters, from a TSD tape to plottable file: all in one script. This plots Gyre 82-10 data.

Example data (stage 1) read from the TSD tape, field separator ^I

```
1982/08/27-04:13:08.000  170  39.31365  -71.69895
1982/08/27-04:13:28.000  170  39.31344  -71.69924
1982/08/27-04:13:48.000  170  39.31318  -71.69950
1982/08/27-04:14:08.000  170  39.31292  -71.69972
1982/08/27-04:14:28.000  170  39.31262  -71.69995
```

Here's the script file:

```
cat $HOME/data/nav/gy8210nav | awk '{print ($3)" "($4)\
" "substr($1,1,4)" "substr($1,6,2)" "substr($1,9,2)\
" "substr($1,12,2)" "substr($1,15,2)" "substr($1,18,2)\
" "substr($1,15,2)" "($2)}'| sed -e '/NS/c\
#-b' | sed -e '/  /c\
#-b' | uniq -8 | seltime -a 3 -t 1 | uniq |\
lines -mo map.def gy8210.ov
```

In this example, *cat* feeds the data file through *awk* for reformatting. In *awk* parlance, field 1, here containing date-time in 21 places, is $1 (the year starts with place 1 and has 4 characters); field 2, the at-sea profile line number, is $2; field 3, the latitude $3; and field 4, the longitude, is $4. So, what we get is latitude, longitude, year, month day, hour, minute, second, then a repeated minute field, and the line number. Thus:

```
39.31365 -71.69895 1982 08 27 04 13 08 13 170
39.31344 -71.69924 1982 08 27 04 13 28 13 170
39.31318 -71.69950 1982 08 27 04 13 48 13 170
39.31292 -71.69972 1982 08 27 04 14 08 14 170
39.31262 -71.69995 1982 08 27 04 14 28 14 170
39.31241 -71.70023 1982 08 27 04 14 48 14 170
39.31225 -71.70048 1982 08 27 04 15 08 15 170
39.31203 -71.70076 1982 08 27 04 15 28 15 170
39.31182 -71.70099 1982 08 27 04 15 48 15 170
39.31163 -71.70126 1982 08 27 04 16 08 16 170
```

The *cat* and *awk* stages of the script pass along data with 3 fixes per minute, as was recorded at sea. I repeated the minute field near the end of the line so *uniq* could then delete lines having duplicate minute fields (the *-8* tells *uniq* to skip the first 8 fields before comparing the final part of the line for duplicates). Since the line numbers are duplicates for each data line, they don't interfere.

The *sed* commands cull unwanted interline data by changing each data line that contains TRANSIT or NSRT (conveniently "NS" is a common string) or double blanks to *#-b.* Note the form of the *sed* commands. The \ line break is critical. It is placed within the single quotes of the *sed* -e statement.

The *seltime* program picks out tics and annotations at the intervals you choose. As input it needs latitude, longitude, year, month, dy, hr, min, sec, using spaces as field separators.

Following *seltime* a second pass through *uniq* culls some lines of duplicate fixes and the excess *#-b's* that were inserted in the interline places.

The first few lines of the prepared data file before it is piped into *lines,* following *seltime,* looks like this:

```
# -p 0       #select pen number
# -c         # select character posting
#  # define location of character string
# -f .27.17
# # define lon/lat (x/y) fields
# -d .13.11,.2.10
# -cr o       # plot characters orthogonal to track
# -cx .3      # char offset from geographic point
# -cs .3      # character size
# -cf -       # select system default for character font
   39.31365   -71.69895   "1982/ 8/27   4:13"
   39.31292   -71.69972
   39.31225   -71.70048
```

Here, within a data header which seltime creates, the *-d .27.17* tells lines the annotation character string format: skip 27 characters (including space characters), then read 17 characters. The fix format *-d .13.11,.2.10* says skip 13 places, read 11 places for the first field, then skip 2 places, read 10 places for the second field. Following some font and format instructions the data comes, and annotated fixes appear within the file at the selected 3-hour intervals, and line breaks are signalled by *#-b.*

The transition within the final data file from fix points for the line plot to a list of tic fixes for a point-symbol plot, that is automatically installed by *seltime,* looks like this:

```
   ...
   38.71787   -72.67151
   38.71771   -72.67197
   38.71776   -72.67232
# -b      # pick up pen
# # tick marks; default values
# # select lon/lat (x/y) fields
# -d .13.11,.2.10
# -p 0     # select pen number
# -lq      # disable line plotting
# -s -sr o # select orthogonal symbol plotting
# -sf -tr -sc 45 -ss .3   # define symbol (-)
   39.31365   -71.69895
   39.28516   -71.73880
   39.25163   -71.79075
   ...
```

E. Sites and Samples

dodsdpwells -- plots Deep Sea Drilling Project well locations and an annotation

```
points -o dsdpwells.ov -m map.def <<EOF - \
$HOME/data/samples/dsdpwells.dat
#-s       # start plotting points
#-c       # start posting
#-d 2,1        # lon, lat fields
#-ss .2        # symbol size cm
#-f 3          # post field 3
#-sf -         # point-symbol font
#-sc 10        # point-symbol character
#-cf -         # char font
#-p 1          # pen number
#-cs .15  # char size cm
#-cx .2   # annotation offset x direction, cm
EOF
```

dostratwells -- Plots stratigraphic wells offshore U. S. east coast from BAMG sediment-analysis laboratory's printer-formatted file

```
: ## requires Bourne shell, start the file, way up top, with : or
blank.
## for stratwells from Larry Poppe's printer-image file
## gives lat, lon, program and site number, year
##
(cat <<EOF; cat $HOME/data/samples/stratwells.dat\
| awk '{print\
substr($0,38,2)"D"substr($0,41,5)"\t"substr($0,48,3)"D"\
substr($0,53,5)"\t"substr($0,20,9)" "substr($0,1,9)}')\
| points -mo map.def stratwells.ov
#-s            # start plotting points
#-c            # start posting
#-d 2,1        # lon, lat fields
#-ss 0.15      # symbol size cm
#-f 3 -cx .2   # post field n, annotation offset cm
#-cs 0.2       # character size cm
#-sf -         # point-symbol font
#-sc 10        # point-symbol character
#-cf -         # char font
#-p 0          # pen number
EOF
```

Awk: see Unix instructions. Briefly, here, the ($0,38,2) means: within each line (the $0) select 2 characters starting at character 38. Those six characters become the first field piped to *points.* The following $0 terms pick out the latitude and longitude in fields 2 and 3.

dosedlab -- plots sample locations from BAMG sediment laboratory files

```
## Plots sedlab.dat, as of 17aug89.
cat <<EOF - $HOME/data/samples/sedlab.dat\
| uniq -4 | points -mo map.def sedlab.ov
#-s             # start plotting points
#-c             # start posting
#-d 6,5         # lon, lat fields
#-ss 0.12       # symbol size cm
#-f 2 -cx .2    # post field n, annotation offset cm
#-cs 0.1        # character size cm
#-sf -          # point-symbol font
#-sc 10         # point-symbol character
#-cf -          # char font
#-p 0           # pen number
EOF
```

dowhoicores -- plots sites and an annotation from WHOI core and sample files on the WHOI's Blue Vax; from output of WHOI's *muddie* program.

```
: ## Run with Bourne shell.
##WHOI's muddie program report format (note the comments
## you can change the annotations by changing the awk statement.
##
## ship   1,3
## cruise 5,3
## leg    12,2
##station 16,4
##samp no.     24,4
##device  31,2
##date    36,6
##lat deg 44,2
##lat min 47,6
##lon deg 54,3
##lon min 58,6
##fix type     66,1
##marsd squar  69,6
##core,dredge no.   76,4
##depth   83,4
```

```
##end depth    89,4
##sample we(?) 96,4
##
## This script plots date and lat and lon; drops duplicate
## lat-lon lines in a sorted file.
##
## requires Bourne shell; put colon or space as first character
in file.
(cat <<EOF ; cat $HOME/data/samples/whoicores.dat| awk '{print\
 substr($0,36,6)"\t"substr($0,44,2)"d"substr($0,47,6)"\t"\
substr($0,54,3)"d"substr($0,58,6)}')\
| uniq -1 | points -mo map.def whoicores.ov
#-s             # start plotting points
#-c             # start posting
#-d 3,2         # lon, lat fields
#-ss 0.15       # symbol size cm
#-f 1 -cx .2    # post field n, annotation offset cm
#-cs 0.15       # character size cm
#-sf -          # point-symbol font
#-sc 10         # point-symbol character
#-cf -          # char font
#-p 1           # pen number
EOF
```

Uniq culls sequentially duplicate lines, here ignoring the first field (-1), thus avoiding plot after plot of the same site for each sample down a core. The data have to be sorted if this is to work, of course. Note also that using *awk,* the *##* symbols are not useful to comment out lines in the data file because awk just looks at whatever comes up in the sequence you give it. The data file has to be clean, or anomalou lines have to be cleaned out with a *sed* filter.

docores -- plots core sites from a NGDC file

```
: ## requires Bourne shell
## docores (for core curator's format from NGDC)
(cat <<EOF ; cat $HOME/data/ngdccores | awk '{print
substr($0,44,1)" "\
substr($0,20,6)"\t"substr($0,26,2)"D"substr($0,28,2)"."substr($0,
30,2)"\t"\
substr($0,33,2)"D"substr($0,35,2)"."substr($0,37,2)}'\
| uniq -1) | points -mo map.def curacores.ov -
#-s             # start plotting points
#-c             # start posting
#-d 3,2         # lon, lat fields
#-ss 0.15       # symbol size cm
```

```
#-f 1 -cx .2     # post field n, annotation offset cm
#-cs 0.1         # character size cm
#-sf -           # point-symbol font
#-sc 10          # point-symbol character
#-cf -           # char font
#-p 1            # pen number
EOF
```

doilwells -- plots oil well locations from files compiled by Larry Poppe of BAMG; picks out the latitude-longitude and an annotation from Larry's printer format, but you will have to edit the data file manually or use *sed* to eliminate the printout-type headers.

```
:  ## Requires Bourne Shell.
## oil well file format:
##   1,8  date start
##   11,8 date complete
##   21,3 region
##   26,6 company
##   34,7 protraction diagram
##   43,6 block and well number
##   50,8 feet from n/s line
##   59,8 feet from e/w line
##   68,3 degree lat
##   72,6 min lat
##   80,3 degree lon
##   85,6 min lon
##   94,3 water depth m
##   101,3     rkb sea floor feet
##   107,7     rkb TD feet
##   116,4     penetration m
##   124,8     comments
##
##To plot using MAPGEN:
## requires Bourne shell, start file, way up top, with : or
blank.
## gives lat, lon, company_block_and_well_no (in field 3), and TD
in meters.
(cat <<EOF; cat $HOME/data/samples/oilwells.dat\
| awk '{print
substr($0,68,3)"D"substr($0,72,6)"\t"substr($0,80,3)"D"\
substr($0,85,6)"\t"substr($0,26,6)" "substr($0,43,6)}')\
| points -mo map.def oilwells.ov
#-s              # start plotting points
##-c             # start posting
#-d 2,1          # lon, lat fields
```

```
#-ss 0.25       # symbol size cm
##-f 3 -cx .2        # post field n, annotation offset cm
#-cs 0.2        # character size cm
#-sf -sr        # point-symbol font
#-sc 19         # point-symbol character
#-cf -          # char font
#-p 0           # pen number
EOF
```

dodives -- plots Alvin dive locations; DBASE files of Alvin information are maintained by the WHOI data library as part of the Alvin Archives. The example data file below was extracted from DBASE files on a DOS PC and downloaded using Kermit. The fields here are delimited by tabs.

```
## alvindives: dive, date, lat, lon, depth
## extracted from WHOI data library DBASE file, 16 Aug 1989.
1    19640626  41d31N    70d40W    27
2    19640701  41d31N    70d40W    12
3    19640702  41d31N    70d40W    25
4    19640731  41d31N    70d40W    25
```

And, here's the script file:

```
points -o dives.ov -m map.def - $HOME/data/alvindives <<EOF
#-s       # start plotting points
#-c       # start posting
#-d 4,3         # lon, lat fields
#-ss .2         # symbol size cm
#-f 1           # post field 1 (dive number)
#-sf -          # point-symbol font
#-sc 10         # point-symbol character
#-cf -crp       # char font
#-p 2           # pen number
#-cs .25  # char size cm
#-cx .2   # annotation offset x direction, cm
EOF
```

dolegn1 -- puts a note on the map to refer to Alvin dive sites as plotted from *dodives.*

```
legend -mo map.def legn1.ov <<EOF
-w d            # measure for data window
-f -crp         # font call
-s 0.25         # size cm
-p 2            # pen
-xy 27 3        # xy position cm
-t              # text
```

```
+  Alvin dive locations
.
EOF
```

dodumpsite -- Here's a miscellaneous use of the *lines* program. This makes an outline of DWD106 Municipal sludge area; modify it for other rectangles or polygon drawings.

```
## Municipal DWD dumpsite location from NOS chart 13003 dec86
lines -mo map.def dumpsite.ov <<EOF
#-p 4          # pen 4 , pick your color
# -l           # begin line plotting at subsequent data fields
# -d 1,2       # longitude field 1, latitude field 2
72dw 39d       # these are corner points,
72d5w     39d  # five of them close the box
72d5w     38d40     # field delimiter is ^I, or tab.
72dw 38d40
72dw 39d
EOF
```

F. Viewing and Plotting:

Preview and *zoom* are MAPGEN programs for viewing a map on a terminal. Use *preview map.def grid.ov* or **.ov* to see the overlay(s) you have made, or *zoom map.def *.ov*. *Zoom* allows enlarging selected areas of the map on the screen, and can also locate coordinates for labels. See the program documentation. If you are using a remote terminal it must be a graphics terminal and you have to get the emulation right. For DOS-based terminals that use Kermit, see Appendix I.

For ease of trial and error mapmaking insert *alias prevv 'preview map.def'* in your *.cshrc* file, and use *prevv *.ov*. Don't use *prev* for the alias because *prev* exists as another program. Then, with the c shell, you can use *!pr* and *!vi* to let the shell do your repetetive typing as you correct and recorrect your map parameter choices.

plot970 -- A runline for the BAMG Calcomp 970 plotter as run from Shemp. This will plot all your overlays. Check that the T-bar switch on the plotter is set to the Masscomp.

```
plotter -d c970 *.ov >/dev/tty2 &
```

References Cited

Evenden, G. I., 1986, The MAPGEN cartographic system. in Steiger, D. ed., Proceedings 1986 Working Symposium on Oceanographic Data Systems. IEEE Computer Society, p. 239-245.

Evenden, G. I., and Botbol, J. M., 1985, User's Manual for MAPGEN (Unix version): a method of transforming digital cartographic data to a map. U. S. Geological Survey Open-file Report 85-706, 58 pp plus appendixes on font codes and map projections.

Kernighan, B. W. and Pike, R., 1984, The Unix Programming Environment. Prentice Hall, 357 p.

Hampson, J. C., Jr., and Wright, E. L., 1988, GLORIA Atlas Preparation: a basic application of the MAPGEN mapping system as a publishing aid. U. S. Geological Survey Open-file Report 88-287, 27 p.

Morgan, Rachel, and McGilton, Henry, 1987, Introducing Unix System V. McGraw Hill, 612 p.

Appendix I
ver. 2
MAPGEN. Kermit, and your DOS PC
Some laboriously-garnered hints for more comfortable computing

A DOS-based PC with a graphics-capable screen can serve as a terminal for MAPGEN map making by telephone and will also work with the *vi* program on the Unix systems. The commonly used program Kermit provides emulations for VT100 (as vt102) and Tektronix 4014 terminals as well as a couple of others. However, if you identify ypur terminal session to the Unix system as a Graphon 140, vt100-type signals are sent for text editing and tek4014-type signals are sent for graphics, and your screen will switch between the two as it's needed.

Using Kermit, dial and connect the PC to the Unix computer. After the initial logon (username and password) the Unix system will prompt for a terminal type, as *TERMINAL=?(vt100)*. Enter *go* to override the default and set your session up as a Graphon 140 terminal. You will be able to use the terminal as if it were a vt100 with screen editing capability for vi. The screen will change automatically to a tek4014 graphics mode when needed, and the moving finger will draw your map. When the graphics display is complete the machine beeps. After your inspection of the map, push *alt-* (alt minus, both keys simultaneously). The screen will flip back to the vt100 emulation for keyboard work. You can cycle the emulations through Heath-19, and VT-52, and VT102 if you continue to push the *alt-*. You will find the VT102 screen is best for *vi*. You will also find that the map graphics are held on the Tektronix screen, and you can go back and look at them.

With many terminals the automatic 2-part communications, text plus graphics, don't work. Consequently I include a further discussion of convenient ways to flip between Unix TERMCAP ID's at the computer and emulations at the terminal without the nuisance of having to logoff and logon.

There are two ends to the system you are using: the PC serving as a terminal and the Unix computer. The two ends have to be compatible. The Unix system communicates with terminals through individual protocols that it sets up for each session. If your terminal session is identified as using a vt100, the main computer won't send it graphics signals. On the other hand, if the main computer thinks your terminal is a Tektronix 4014 it will send graphics, but it will send the *vi* editing program in "open mode", which is not a screen editor mode, and is a nuisance for you, even though the terminal on your end thinks it's a

VT100. If your PC thinks it's a vt100 it won't draw maps. If your PC thinks it's a tek4014, it will receive graphics signals and draw maps, but there's no cursor, the text characters are not as easily readable, and *vi* won't work as a screen editor. The Unix system will use vt100 and tek 4014 emulations automatically for editing and graphics if you logon as a Graphon 140 terminal.

Terminal emulations at the PC end are created by Kermit or other communications software. The Kermit-provided emulation is identified on the lower right on the banner. There's no banner for graphics. This is written from experience on an IBM Model 50 with a monochrome screen. According to its documentation, Kermit can handle colors, too.

You can see what kind of terminal the Unix computer thinks you are by looking at TERMCAP and TERM, using the Unix command, *env.* You can change those terminal identification entries. Find the files *.vt100 , .4014sm, and .go140* in */eez/jmr* on Shemp and copy them to your own home directory. The dots just help hide and protect those files within the directory. Then, edit your *.cshrc* file to add an alias line that says **alias vt 'source $HOME/.vt100'**, and **alias tk 'source $HOME/.4014sm'**, or **alias gO 'source $HOME/.go140'**. Then the command vt, tk, or gO will substitute its terminal identification in the TERM and TERMCAP variables of the Unix environment. Then, depending on what terminal you are dealing with, you can match the emulation with the TERMCAP for these three kinds of units.

If you find you must manually switch modes, for fewer keystrokes use the alias capability of Unix. Alias the string **tk; preview map.def** to **prevv**, and the string **vt;vi** to **vv**, so that by entering **prevv** you switch the terminal ID to tektronix and preview the overlays you wish to see; then to use *vi,* after you use the **alt-** to put your own screen in the right mode, edit your trial-and-error MAPGEN files using the **vv** command, which you have made equivalent to *vt;vi.*

Here's what those alias commands, which can go in your *.cshrc* file, look like:

```
alias tk 'source ~/utils/.tek4014sm'
alias vt 'source ~/utils/.vt100'
alias mc 'source ~/utils/.masscomp'
alias prevv 'tk; preview map.def'
alias zm 'tk; zoom map.def'
alias vv 'vt;vi'
```

Pushing DOS

Without logging out of your Unix session you can switch neatly and conveniently back into your PC's DOS and use your word processor or look for a telephone number or run other programs while your Unix job is running in background.

After logon to the Unix through Kermit, you can get back to the Kermit prompt with **^]** (control bracket), or with *control-break* if your system is set up that way. You get a response that gives you no prompt, but puts the cursor at the bottom of the screen. If you push **c**, you get the *Kermit-MS>* prompt. If you wish to work with your DOS files, push **p**, for push, which will bring up your DOS prompt. Do your DOS work, Wordperfect or whatever it may be, and then enter **exit**, which will bring you back to your Unix connection through Kermit. If the *Kermit-MS>* prompt is on the screen while you're on line with the Unix systems, you can use the full command **push**.

What you have done is initiated ("pushed") a second level of DOS commands on top of the first set you read into memory when you booted your machine, and when you exit you drop back down to that former level, which is operating Kermit, which is talking to Unix. This can be confusing and you can pile up several Kermits or other programs on your local machine, all working at once (or trying to), until you run out of memory. So **exit**, if you have pushed.

In order to transfer files between your machine and the Unix machines, while in your Unix subdirectory, enter **Kermit -iwx.** You will get a *Kermit-C>* prompt from a Kermit program operating within Unix. The *-iwx* tells the Unix Kermit that it should be under control of the Kermit on your end, and cautions it not to overwrite any existing files. See the Unix Kermit's help file. Then enter the **^] c** command for the Kermit program on your local machine. To get a file to your local disk from the Unix machine, at the *Kermit-MS>* prompt, enter **get <filename of file on Unix>.** To reverse the transfer, use **send <DOS filename>.** The file will be transferred between the subdirectories you were logged into on each machine.

You can use pathnames, and you can log a different directory or disk (to copy data to a floppy) by using the **cwd** (change working directory) command at the *Kermit-MS>* prompt. Enter

finish at the *Kermit-MS>* prompt, and the Unix-Kermit will be turned off. Enter **c** to get back talking to the Unix, where you started, or **bye**, which will logoff the Unix, disconnect, and come back to DOS. Be aware of the two simultaneously operating programs, which are identified by their prompts, *Kermit-C>* and *Kermit-MS>*.

While you're working in Unix through the Kermit connection, be careful about double pushes of the *Escape* key. Two **escapes** disconnect the phone line. If you blunder and push *escape* twice in sequence while you are using *vi,* the disconnect message from your phone will appear. Reconnect by using the space bar to move the hyphen under the *connect* prompt, and push return, and you will be back in your *vi,* but with a messy screen. Fix the screen display by pushing *escape* again, once, and then pushing **^l (control el)** or if that doesn't work, try **^r (control r).**

There's a 40 or 50 page manual on disk (i.e., floppy) for the January-1988 MS-Kermit version that we have, and help files (*usage kermit*) for the Unix version of Kermit.

Appendix II

How to use the digitizing table for digitizing lines or area outlines

This is a summary of a procedure and of some of the kinks for digitizing features on one map so they can be replotted to another, using equipment and software maintained by the BAMG computer group. Refer also to the *digin* and *proj* manuals on the BAMG Unix systems: use *man digin* or *man proj* on Larry. There is also a Projection Parameters report (see Evenden and Botbol, 1985, and its updates).

The digitizing table in the BAMG computer user's room is wired to Larry. The program *digin* on Larry puts information from the digitizing table into a file in the form of table coordinates. The programs *proj* and *toxy* transform the table-coordinate file into a list of latitude and longitudes which you can use with MAPGEN or ISM. Jerry Evenden wrote the *proj* and *toxy* programs, as well as MAPGEN, and is the expert.

Note before starting: this digitizing procedure requires the map projection of the map you are digitizing *from: not* of the map you intend to make later. There are many map projections and each commonly has choices of parameters depending on global location, country of origin, scale, variety of convention, etc. The accuracy of your digitizing job will depend on your pick of the projection and those parameters. The precision you need depends largely on the scale you wish to plot your results at, compared to the scale of the map carrying the data you are digitizing. To check the job, plot it back as an overlay, and compare it to the original map.

1. Tape map to table. Avoid the table's borders because the digitizing area of the table doesn't extend all the way to the edge.

2. Choose calibration points. These should be points on your map whose longitude and latitude you can determine. They must form a polygon which <u>completely encloses</u> all the area you intend to digitize. Calibration points are simplest for you to work

with when they are at degree intersections (just to avoid typing complicated numbers), but the machine doesn't require that.

There can be more than four points. They require no special placement; they don't have to be on the same parallels or meridians from top to bottom or side to side. The polygon can look like an irregular star. You can do a calibration grid if you want to, so long as the outer polygon encloses the area. Extra points give better statistical closure, but won't repair a wrong choice on the projection.

Mark your calibration points so you will be able to relocate them with the cursor, in sequence. Write down their longitudes and latitudes for later entry into a shell script for processing. You will probably want to be able to recalibrate on the same points again, after you have blundered.

3. Logon to Larry.

4. Enter **digin > datafilename**. Make up a filename.

5. *Digitize the calibration points in sequence.* Set the crosshair on the point and push the **1** key on the cursor. It beeps at each point.

6. Push **F** on cursor to finish the calibration phase.

7. To start the digitizing phase, push **C** for point mode, where you set and choose each point, or push **0**, or **1**, or **2**, or **3**, for stream mode, which will digitize 1, 2, 5, or 10 points per second.

8. *Digitize your data.* For point mode push any numeric key on the cursor at each point. No beeps or no screen change will show. For stream mode, push and hold the **C** key while you slide the crosshair along the line.

9. To suspend digitizing at a line break, push **F**. To restart push a number or **C**, as above. To terminate push **D**.

The **F** inserts a *#-b,* the MAPGEN code for a line break, in the datafile; for ISM usage you will have to edit those later.

On termination (**D**), you should see the Unix prompt on the terminal screen. This is the end of the table phase. There should be a datafile of digitizer counts in your directory under datafilename. You will transform these to longitudes and latitudes using the next procedure.

See the *digin* instructions if you have a lot of lines to digitize. There is an automatic filenaming option for repetitive operations.

10. Check the datafile; eliminate duplicate calibration points. Sometimes the cursors bounce and send multiple points for a single key stroke. Use *vi* or *edit;* look at the file you created; count the calibration points in the initial group (they will be in table coordinates) to be sure there is the proper number. Delete duplicates.

If duplicate points in your data could cause a problem with your project, you might look over the rest of the data file, and you should see the *toxy* instructions for another solution. A few duplicate data points won't hurt most plots, but be sure to fix the calibration points.

11. Using *vi* or *edit,* make a script file (called *digscrpt* or whatever you want) to transform the table's x/y coordinates to lons and lats. Here is a model:

```
#
set a='+proj=lcc'
proj $a <<EOF | toxy - $1 | proj +inv $a > $1.dat
76dw      38d
72dw      34d
78dw      34d       [The data fields must be separated
78dw      38d                 by a tab.]
EOF
```

This file contains a code for the projection of the map you are digitizing from (not the one you intend to make later) and the longitude-latitude coordinates of the calibration points, in the sequence they were entered. It reads your datafile and creates a new file named *datafile.dat.* Use DMS format for the coordinates (see the MAPGEN manual; Evenden and Botbol, 1985). The projection codes and options are in the Projection Parameters manual. End the list with EOF, as illustrated.

What this script file does (for your interest, translated with some help from Jerry Evenden):

# calls the Unix c-shell. (A blank first character would call the Unix Bourne shell, which uses different conventions.)

set a='+proj=lcc' sets a Unix variable (to save some typing). The a, as *set,* is substituted by the system for the $a's where they appear in the next line, and the filename you specify is substituted for the $1.

In this example +proj=lcc specifies the projection: lcc=lambert conic conformal. Other examples, assuming you can accept defaults, follow: (Do check the Projection Parameters report; see Evenden and Botbol, 1985, and its unpublished updates.)

lambert equal area: set a='+proj=laea +lat_0=0n'
utm: set a='+proj=utm +lon_0=75w'
[75w is the central meridian; specify yours; but there are conventions; read the Projection Parameters section of Evenden and Botbol (1985) so you get it right.]
albers equal area conic: set a='+proj=aea'
mercator: set a='+proj=merc'

Note that any options required to describe the projection are to be included within those single quotes.

The program *proj* reads the projection and calibration points as far as the EOF, puts them into a rectilinear x-y coordinate system and sends them to the program *toxy* (the - after *toxy* tells it to read the piped-in data before it reads the datafile (the $1) in its command line. *Toxy* reads the datafile, compares the table counts of calibration coordinates at the top of that file and the list of x-y coordinates that were piped in, and transforms the data coordinates into the x-y system. Then, in turn, another *proj* operation inverts the x-y data into latitudes and longitudes. Clear and simple, huh? The final product goes into a .dat file (>$1.dat); but you can change that suffix if you don't like .dat. See the documentation of the *proj* and *toxy* programs for explanation of options.

12. Make your script an executable file. Enter **chmod +x digscrpt**.

13. Execute the script file by entering **digscrpt datafilename**. *Datafilename* is the file that you created with *digin*. You will get another file, *datafilename.dat*.

14. MAPGEN, using its default format, requires *tabs* (**^I** in ascii) to separate fields in data files. You may want to change the format of the data in the .dat file so the longitude and latitude are separated by a tab, rather than a space. In *vi*, use *:1,$s/ /^I/*, where *^I* is the tab key. If the version of *proj* you are using gives you a tab, you're all set. For ISM you may have

to edit the tabs to blanks and the break points (#-b) to whatever ISM calls for.

15. For a MAPGEN overlay, run the *lines* or *points* program, as:

```
lines -mo map.def overlay.ov lineoptions.par datafilename.bat
```

16. Good luck.

Appendix III

Kermit script files for use with ATT 7404 or 7406 telephone instruments to connect with Shemp, Vax, Mips within the BAMG telephone system

The following are Kermit script files that will connect your PC as a terminal to the Vax or Microvax. When they work they're quicker than typing. Sometimes they don't work because the phones or the machines are slow and *kermit* times out, or because they encounter an unusual response.

Place the following sets of files on your PC in the same directory as your *kermit.exe* file. The batch files (.bat) invoke Kermit and use its "take" function to call the .fil files which hold the kermit scripts.

I. vax.bat

```
@echo off
kermit -f vax.fil
```

II. vax.fil

```
comment start of file
comment kermit script file for network phone logon
comment usage kermit -f vax.fil
clear
set speed 9600
set port 1
set input timeout-action quit
echo Going for Vax via network phone connection...
output \13
pause 0
output \13
pause 0
output \13
pause 0
output \13
input 5 DIAL
output 300\13
comment ;avoiding entire derailing by the occasional
comment ;status prompt without a login prompt
SET INPUT TIMEOUT-ACTION PROCEED
INPUT 1 STATUS
OUTPUT \13
```

```
SET INPUT TIMEOUT-ACTION QUIT
pause 0
output \13
input 5 Local
echo
echo There's the phone box...
echo      Now for the Vax...
echo
pause 2
comment connect on whsys as the box's default
output connect\13
input 5 User
output your_sign_on\13
input 5 password
output your_password\13
comment
comment ..This is an alternative if you want to keep a secret
comment ..word out of your PC file.  Replace the your_password
comment ..lines with the following lines that request your
comment ..password from the keyboard.
comment
comment echo Enter the Password:
comment output @CON
comment
echo
echo OK, Wait for vax inquiry and prompt...
echo
comment wait for vax set term/inquire...
comment respond esc [ <query symbol> 6 ; 2 c (say we are vt102).
comment note syntax for question mark and semicolon
input 18 \27Z
echo
echo Vax's terminal-inquiry signal received...
echo   Responding, wait for the prompt...
output \27[\{63}6\;2c
input 15 $
echo
echo There's the prompt...     Go...
pause 3
connect
comment EOF
```

III. mips.bat

```
     @echo off
     kermit -f mips.fil
```

IV. mips.fil

```
comment start of file
comment kermit script file logon to Mips via Network phone server
comment and whsys vax.
comment usage kermit -f mips.fil or use mips.bat which says that.
clear
set speed 9600
set port 1
set input timeout-action quit
echo Going for Vax via network connection...
output \13
pause 0
output \13
pause 0
output \13
pause 0
output \13
input 5 DIAL
output 300\13
comment ;avoiding derailing by the occasional status prompt
comment ; without a login prompt
SET INPUT TIMEOUT-ACTION PROCEED
INPUT 1 STATUS
OUTPUT \13
SET INPUT TIMEOUT-ACTION QUIT
pause 0
output \13
input 5 Local
pause 0
echo Got the network server; Now for the Vax at Whsys...
comment connect to whsys by default
output connect\13
input 5 User
output your_sign_on\13
input 5 password
output your_password\13
comment
comment echo Enter the Password:
comment output @CON
comment
comment wait for vax set term/inquire...
comment respond esc [ <query symbol> 6 ; 2 c (say we are vt102).
comment note syntax for question mark and semicolon
input 18 \27Z
echo Got the Vax's terminal inquiry signal...
echo  Responding, wait for prompt...
```

```
output \27[\{63}6\;2c
input 15 $
echo Got the Vax prompt, Now going for Mips connection...
output set host mips\13
input 10 User
echo Aha! The Mips speaks...
echo
output your_sign_on\13
input 3 password
output your_password\13
echo OK, Wait for microvax inquiry and prompt...
comment wait for vax set term/inquire...
comment respond esc [ <query symbol> 6 ; 2 c (say we are vt102).
comment note syntax for question mark and semicolon
input 18 \27Z
echo OK, Got the Microvax's terminal inquiry signal...
echo  Responding, wait for the next prompt...
output \27[\{63}6\;2c
input 15 $
echo There it is..     Go...
pause 5
connect
comment EOF
```

V. shemp.bat

```
     @echo off
     kermit -f shemp.fil
```

VI. shemp.fil

```
comment kermit script file for masscomp logon
comment usage kermit -f shemp.fil
clear
set speed 9600
set port 1
set input timeout-action quit
echo
echo Telephoning Shemp...
output \13
pause 0
output \13
pause 0
output \13
pause 0
output \13
input 5 DIAL
```

```
pause 0
output 372\13
comment ;avoiding derailing by the occasional status prompt
comment ; without a login prompt
SET INPUT TIMEOUT-ACTION PROCEED
INPUT 1 STATUS
OUTPUT \13
SET INPUT TIMEOUT-ACTION QUIT
input 5 login:
pause 0
output your_sign_on\13
pause 1
set input echo off
input 5 Password
comment echo Enter the Password:
comment output @CON
output your_password\13
set input echo on
clear
input 5 TERM
comment the go response here invokes a graphon 140 termcap
comment for vt100 respond with carriage return (\13) only.
output go\13
connect
comment EOF
```