# 4. General features

This section explains a few features common to all the programs in GMT. It summarizes the philosophy behind the system. Some of the features described here may make more sense once you reach the cook-book section where we present actual examples of their use.

## 4.1 GMT defaults

There are more than 50 parameters which can be adjusted individually to modify the appearance of plots or affect the manipulation of data. When a program is run, it initializes all parameters to the GMT defaults, then tries to open the file .gmtdefaults in the current directory. If not found, it will look for that file in your home directory. If successful, the program will read the contents and set the default values to those provided in the file. By editing this file you can affect features such as pen thicknesses used for maps, fonts and font sizes used for annotations and labels, color of the pens, dots-per-inch resolution of the hardcopy device, what type of spline interpolant to use, and many other choices (A complete list of all the parameters and their default values can be found in the *gmtdefaults* manual pages). You may create your own .gmtdefaults files by running *gmtdefaults* and then modify those parameters you want to change. If you want to use the parameter settings in another file you can do so by specifying *+defaultfile* on the command line. This makes it easy to maintain several distinct parameter settings, corresponding perhaps to the unique styles required by different journals or simply reflecting font changes necessary to make readable overheads and slides. Note that any arguments given on the command line (see below) will take precedent over the default values. E.g., if your .gmtdefaults file has *x* offset = 1 as default, the **–X**1.5 option will override the default and set the offset to 1.5 inches (or cm). Default values may also be changed from the command line with the utility *gmtset*.

## 4.2 Command Line Arguments

Each program requires certain arguments specific to its operation. These are explained in the manual pages and in the usage messages. Most programs are "case-sensitive"; almost all options must start with an upper-case letter. We have tried to choose letters of the alphabet which stand for the argument so that they will be easy to remember. Each argument specification begins with a hyphen (except input file names; see below), followed by a letter, and sometimes a number or character string immediately after the letter. DO NOT space between the hyphen, letter, and number or string. DO space between options. Example:

pscoast –**R**0/20/0/20 **–G**200 **–J**M6 **–W**1 **–B**5 **–V** > map.ps

## 4.3 Standardized command line options

Most of the programs take many of the same arguments like those related to setting the data region, the map projection, etc. These 11 options have the same meaning in all the programs (Some programs may not use all of them):

**–B** Defines tickmarks, annotations, and labels for basemaps and axes.
**–J** Selects a map projection or one of several non-map projections.
**–K** Allows more plot code to be appended to this plot later.
**–O** Allows this plot code to be appended to an existing plot.

**–P** Selects Portrait plot orientation [Default is landscape].
**–R** Defines the min. and max. coordinates of the map/plot region.
**–U** Plots a time-stamp, by default in the lower left corner of page.
**–X** Sets the *x*-coordinate for the plot origin on the page.
**–Y** Sets the *y*-coordinate for the plot origin on the page.
**–c** Specifies the number of plot copies.
**–:** Indicates that input file is (*y*,*x*) rather than (*x*,*y*)

These options are described in more detail in the manual pages.

## 4.4 Command Line History

GMT programs "remember" the standardized command line options (See section 4.3) given during their previous invocations and this provides a shorthand notation for complex options. For example, if a basemap was created with an oblique Mercator projection, specified as

**–J**oc190/25.5/327/56/1:500000

then a subsequent *psxy* command to plot symbols only needs to state **–J**o in order to activate the same projection. Previous commands are maintained in the file .gmtcommands, of which there will be one in each directory you run the programs from. This is handy if you create separate directories for separate projects since chances are that data manipulations and plotting for each project will share many of the same options. Note that an option spelled out on the command line will always override the last entry in the .gmtcommands file and, if execution is successful, will replace this entry as the previous option argument in the .gmtcommands file. If you call several GMT modules piped together then GMT cannot guarantee that the .gmtcommands file is processed in the intended order from left to right. The only guarantee is that the file will not be clobbered since GMT now uses advisory file locking. The uncertainty in processing order makes the use of shorthands in pipes unreliable.

## 4.5 Usage messages, syntax- and general error messages

Each program carries a usage message. If you enter the program name without any arguments, the program will write the complete usage message to standard error (your screen, unless you redirect it). This message explains in detail what all the valid arguments are. If you enter the program name followed by a hyphen (–) only you will get a shorter version which only shows the command line syntax and no detailed explanations. If you incorrectly specify an option or omit a required option, the program will produce syntax errors and explain what the correct syntax for these options should be. If an error occurs during the running of a program, the program will in some cases recognize this and give you an error message. Usually this will also terminate the run. The error messages generally begin with the name of the program in which the error occurred; if you have several programs piped together this tells you where the trouble is.

## 4.6 Standard Input or File, header records

Most of the programs which expect table data input can read either standard input or input in one or several files. These programs will try to read *stdin* unless you type the filename(s) on the command line without the above hyphens. (If the program sees a hyphen, it reads the next character as an instruction; if an

argument begins without a hyphen, it tries to open this argument as a filename). This feature allows you to connect programs with pipes if you like. If your input is asciii and has one or more header records, you must use the **–H** option. The number of header records is one of the many parameters in the .gmtdefaults file, but can be overridden by **–H***n_header_recs*. ASCII files may in many cases also contain sub-headers separating data segments; see Appendix B for complete documentation. For binary table data no headers are allowed.

## 4.7 Verbose Operation

Most of the programs take an optional **–V** argument which will run the program in the "verbose" mode. Verbose will write to standard error information about the progress of the operation you are running. Verbose reports things such as counts of points read, names of data files processed, convergence of iterative solutions, and the like. Since these messages are written to *stderr*, the verbose talk remains separate from your data output.

## 4.8 Output

Most programs write their results, including *PostScript* plots, to standard output. The exceptions are those which may create binary netCDF grd-files such as *surface* (due to the design of netCDF a filename must be provided; however, alternative output formats allowing piping are available). With *UNIX*, you can redirect standard output or pipe it into another process. Error messages, usage messages, and verbose comments are written to standard error in all cases. You can use *UNIX* to redirect standard error as well, if you want to create a log file of what you are doing.

## 4.9 *PostScript* Features

*PostScript* is a command language for driving graphics devices such as laser printers. It is ASCII text which you can read and edit as you wish (assuming you have some knowledge of the syntax). We prefer this to binary metafile plot systems since such files cannot easily be modified after they have been created. GMT programs also write many comments to the plot file which make it easier for users to orient themselves should they need to edit the file (e.g., % Start of x-axis). All GMT programs create *PostScript* code by calling the **pslib** plot library (The user may call these functions from his/her own C or FORTRAN plot programs. See the manual pages for **pslib** syntax). Although GMT programs can create very individualized plot code, there will always be cases not covered by these programs. Some knowledge of *PostScript* will enable the user to add such features directly into the plot file. Moreover, GMT *PostScript* can be imported into graphics programs such as IslandDraw and Adobe Illustrator and embellished further.

## 4.10 Landscape and Portrait Orientations

In general, a plot has an *x*-axis increasing from left to right and a *y*-axis increasing from bottom to top. If the paper is turned so that the long dimension of the paper is parallel to the *x*-axis then the plot is said to have Landscape orientation. If the long dimension of the paper parallels the *y*-axis the orientation is called Portrait. (Think of taking pictures with a camera and these words make sense). All the programs in GMT have the same default orientation, which is Landscape. Use **–P** to change to Portrait (Note that PAPER_WIDTH is a user-definable parameter,

by default set to 8.5 inch (or 21 cm). For other paper dimensions you must change this value accordingly).

## 4.11 Overlay and Continue Modes

A typical *PostScript* file has a beginning, a middle, and an end. The beginning defines certain features (e.g., macros, origin, orientation, scale, etc.) which will be needed to create the plot. The middle has the commands which actually do the plotting. The end tells the graphics device to put out the plot ("showpage" in *PostScript*) and reset the graphics state. Many of the illustrations in this cookbook are built up by appending *PostScript* files together. If you do this, the first file needs a "beginning" and no "end", the last an "end" but no "beginning", and the middle files need only a "middle". You accomplish this automatically with the Overlay (**–O**) and Continue (**–K**) options. Overlay indicates that this plot will be laid on top of an earlier one; therefore the "beginning" is not included in the output. The default is always no overlay, i.e. write out the "beginning". Continue indicates that another plot will follow this one later; therefore the "end" is not included in the output. The default is to output the "end". If you run only one plot program, ignore both the **–O** and **–K** options; they are only used when stacking plots.

## 4.12 Specifying pen attributes

A pen in GMT has three attributes: *width*, *color*, and *texture*. Most programs will accept pen attributes in the form of an option argument, e.g.,

**–W***width*[/*color*][**t***texture*][**p**]

- → *Width* is normally measured in units of the current device resolution (i.e., the value assigned to dpi in your .gmtdefaults file). Thus, if dpi is set to 300 this unit is 1/300th of an inch. Append **p** to specify pen width in points (1/72 of an inch). Note that a pen thickness of 5 will be of different physical width depending on your dpi setting, whereas a thickness of 5**p** will always be 5/72 of an inch. Minimum-thickness pens can be achieved by giving zero width, but the result is device-dependent.
- → The *color* can be specified as a *gray* shade in the range 0–255 (linearly going from black to white) or using the RGB system where you specify *r*/*g*/*b*, each ranging from 0–255. Here 0/0/0 is black and 255/255/255 is white.
- → The *texture* attribute controls the appearance of the line. To get a dotted line, simply append "**t**o" after the width and color arguments; a dashed pen is requested with "**t**a". For exact specifications you may append **t***string*:*offset*, where *string* is a series of integers separated by underscores. These numbers represent a pattern by indicating the length of line segments and the gap between segments. The *offset* shifts the pattern along the line. For example, if you want a yellow line of width 2 that alternates between long dashes (20 units), a 10 unit gap, then a 5 unit dash, then another 10 unit gap, with pattern offset by 10 units from the origin, specify **–W**2/255/255/0**t**20_10_5_10:10. Here, the texture units can be specified in dpi units or points (see above).

## 4.13 Specifying area fill attributes

Many plotting programs will allow the user to draw filled polygons or symbols. The fill may take two forms:

**–G***fill*
**–Gp***dpi/pattern*[:**B***r/g/b*[**F***r/g/b*]]

In the first case we may specify a *gray* shade (0–255) or a color (*r*/*g*/*b* in the 0–255 range), similar to the pen color settings. The second form allows us to use a predefined bit-image pattern. *pattern* can either be a number in the range 1–90 or the name of a 1-, 8-, or 24-bit Sun raster file. The former will result in one of the 90 predefined 64x64 bit-patterns provided with GMT and reproduced in Appendix E. The latter allows the user to create customized, repeating images using standard Sun rasterfiles. The *dpi* parameter sets the resolution of this image on the page; the area fill is thus made up of a series of these "tiles". Specifying *dpi* as 0 will result in highest resolution obtainable given the present dpi setting in .gmtdefaults. By specifying upper case **–GP** instead of **–Gp** the image will be bit-reversed, i.e., white and black areas will be interchanged (only applies to 1-bit images or predefined bit-image patterns). For these patterns and other 1-bit images one may specify alternative background and foreground colors (by appending :**B***r/g/b*[**F***r/g/b*]) that will replace the default white and black pixels, respectively. Due to *PostScript* implementation limitations the rasterimages used with **–G** must be less than 146 x 146 pixels in size; for larger images see *psimage*. The format of Sun raster files is outlined in Appendix B. Note that under *PostScript* Level 1 the patterns are filled by using the polygon as a clip path. Complex clip paths may require more memory than the *PostScript* interpreter has been assigned. There is therefore the possibility that some *PostScript* interpreters (especially those supplied with older laserwriters) will run out of memory and abort. Should that occur we recommend that you use a regular grayshade fill instead of the patterns. Installing more memory in your printer may or may not solve the problem!

## 4.14 Color palette tables

Several programs, such as those which read 2-D gridded data sets and create colored images or shaded reliefs, need to be told what colors to use and over what *z*-range each color applies. This is the purpose of the color palette table (cpt-file). These files may also be used by *psxy* and *psxyz* to plot color-filled symbols. The colors may be specified either in the RGB system or in the HSV system, and the parameter COLOR_MODEL in the .gmtdefaults file must be set accordingly. Using the RGB system, the format of the cpt-file is:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $z_0$ | $R_{min}$ | $G_{min}$ | $B_{min}$ | $z_1$ | $R_{max}$ | $G_{max}$ | $B_{max}$ | [**A**] |
| … | | | | | | | | |
| $z_{n-2}$ | $R_{min}$ | $G_{min}$ | $B_{min}$ | $z_{n-1}$ | $R_{max}$ | $G_{max}$ | $B_{max}$ | [**A**] |

Thus, for each "*z*-slice", defined as the interval between two boundaries (e.g., $z_0$ to $z_1$), the color can be constant (by letting $R_{min} = R_{max}$, $G_{min} = G_{max}$, and $B_{min} = B_{max}$) or a continuous, linear function of z. The optional flag **A** is used to indicate anotation of the colorscale when plotted using *psscale*. **A** may be **L**, **U**, or **B** to select anotation of the lower, upper, or both limits of the particular z-slice. However, the standard **–B** option can be used by *psscale* to affect anotation and ticking of colorscales. The background color (for *z*-values < $z_0$), foreground color (for *z*-values > $z_{n-1}$), and not-a-number (NaN) color (for *z*-values = NaN) are all defined in the .gmtdefaults file, but can be overridden by the statements

| | | | |
|---|---|---|---|
| B | $R_{back}$ | $G_{back}$ | $B_{back}$ |
| F | $R_{fore}$ | $G_{fore}$ | $B_{fore}$ |

N $R_{nan}$ $G_{nan}$ $B_{nan}$

which can be inserted into the beginning or end of the cpt-file. If you prefer the HSV system of hue-saturation-value, set the .gmtdefaults parameter accordingly and replace red, green, blue with hue, saturation, value. Color palette tables that contain grayshades only may replace the r-g-b triplets with a single grayshade in the 0–255 range.
Some programs like *grdimage* and *grdview* apply artificial illumination to achieve shaded relief maps. This is typically done by finding the directional gradient in the direction of the artificial light source and scaling the gradients to have approximately a normal distribution on the interval <-1,+1>. These intensities are used to add "white" or "black" to the color as defined by the *z*-values and the cpt-file. An intensity of zero leaves the color unchanged. Higher values will brighten the color, lower values will darken it, all without changing the original hue of the color (see Appendix I for more details). The illumination is decoupled from the data grd-file in that a separate grdfile holding intensities in the <-1,+1> range must be provided. Such intensity files can be derived from the data grd-file using *grdgradient* and modified with *grdhisteq*, but could equally well be a separate data set. E.g., some side-scan sonar systems collect both bathymetry and backscatter intensities, and one may want to use the latter information to specify the illumination of the colors defined by the former. Similarly, one could portray magnetic anomalies superimposed on topography by using the former for colors and the latter for shading.

## 4.15 Character escape sequences

For annotation labels or textstrings plotted with *pstext*, GMT provides several escape sequences that allow the user to temporarily switch to the symbol font, turn on sub- or superscript, etc. within words. These conditions are toggled on/off by the escape sequence @**x**, where **x** can be one of several types. These escape sequences are recognized:

| | |
|---|---|
| @~ | Turns symbol font on or off. |
| @%*fontno*% | Switches to another font; @%% resets to previous font. |
| @+ | Turns superscript on or off |
| @- | Turns subscript on or off |
| @# | Turns small caps on or off |
| @! | Creates one composite character of the next two characters |
| @@ | Prints the @ sign itself |

Shorthand notation for a few special Scandinavian characters has also been added:

| | | | | | |
|---|---|---|---|---|---|
| @E | = | Æ | @e | = | æ |
| @O | = | Ø | @o | = | ø |
| @A | = | Å | @a | = | å |

*PostScript* fonts used in GMT may be re-encoded to include several accented characters used in many European languages. To access these, you must specify the full octal code \xxx (See Appendix F). Also see the definition of (and reason for) WANT_EURO_FONT in the *gmtdefaults* man page. Basically, WANT_EURO_FONT must be set to TRUE for the special characters to be available. Many characters that are not directly available by using single octal codes may be constructed with the composite character mechanism @!.

Some examples of escape sequences and embedded octal codes in GMT strings:

| | | |
|---|---|---|
| 2@~p@~r@+2@+h@-0@- E\363tv\363s | = | $2\pi r^2 h_0$ Eötvös |
| 10@+-3 @Angstr@om | = | $10^{-3}$ Ångstrøm |
| Se@!\304nor Gar@!\317con | = | Señor Garçon |
| A@#cceleration@# (ms@+-2@+) | = | ACCELERATION ($ms^{-2}$) |

The option in *pstext* to draw a rectangle surrounding the text will not work for strings with escape sequences. A chart of characters and their octal codes is given in Appendix F.

## 4.16 Embedded grdfile format specifications

A new feature of GMT 3 is the ability to read more than one grdfile format. As distributed, GMT now recognizes six predefined file formats. These are

0. GMT netCDF format [Default]
1. Native binary single precision floats in scanlines with leading grd header
2. Native binary short integers in scanlines with leading grd header
3. 8-bit standard Sun rasterfile (colormap ignored)
4. Native binary unsigned char in scanlines with leading grd header
5. Native binary bits in scanlines with leading grd header

In addition, users with some C-programming experience may add their own read/write kernels and link them with the GMT library to extend the number of predefined formats. Technical information on this topic can be found in the source file gmt_customio.c

Because of these new formats it is sometimes necessary to provide more than simply the name of the file on the command line. For instance, a short integer file may use a unique value to signify an empty node or NaN, and the data may need translation and scaling prior to use. Therefore, all GMT programs that read or write grdfiles will decode the given filename as follows

name[=*id*[/*scale*/*offset*[/*nan*]]]

where everything in brackets is optional. If you only use the default netCDF file format then no options are needed: just continue to pass the name of the grdfile. However, if you use another format you must append the *=id* string, where *id* is the format id number listed above. In addition, should you want to multiply the data by a scale factor, then add a constant offset you may append the /*scale*/*offset* modifier. Finally, if you need to indicate that a certain data value should be interpreted as a NaN (not-a-number) you may append the /*nan* suffix to the scaling string (it cannot go by itself; note the nesting of the brackets!).

Some of the grd formats allow writing to standard output and reading from standard input which means you can connect GMT programs that operate on grdfiles with pipes, thereby speeding up execution and eliminating the need for large, intermediate grdfiles. You specify standard input/output by leaving out the filename entirely. That means the "filename" will begin with "*=id*" since the GMT default netCDF format does not allow piping (due to the design of netCDF).

Everything looks more obvious after a few examples:

1. To write a binary float grd file, specify the name as my_file.grd=1
2. To read a short integer grd file, multiply the data by 10 and then add 32000, but first let all values that equal 32767 be set to the IEEE NaN, specify the filename as my_file.grd=2/10/32000/32767.
3. To read a 8-bit standard Sun rasterfile (with values in the 0-255 range) and convert it to a ±1 range, give the name as rasterfile=3/7.84313725e-3/-1 (i.e., 1/127.5)
4. To write a short integer grd file to standard output after subtracting 32000 and dividing its values by 10, give filename as =2/0.1/-3200

Programs that both read and/or write more than one grdfile may specify different formats and/or scaling for the files involved. The only restriction with the embedded grd specification mechanism is that no grdfiles may actually use the "=" character as part of their name (presumably, a small sacrifice).
One can also define special file suffixes to imply a specific file format; this approach represents a more intuitive and user-friendly way to specify the various file formats. The user may create a file called .gmt_io in the home directory and define any number of custom formats. The following is an example of a .gmt_io file:

```
# GMT i/o shorthand file
# It can have any number of comment lines like this one anywhere
# suffix    format_id   scale       offset      NaN
grd         0           -           -           -
b           1           -           -           -
i2          2           -           -           32767
ras         3           -           -           -
byte        4           -           -           255
bit         5           -           -           -
mask        5           -           -           0
faa         2           0.1         -           32767
```

These suffices can be anything that make sense to the user. To activate this mechanism, set GRIDFILE_SHORTHAND = TRUE in your .gmtdefaults file. Then, using the filename stuff.i2 is equivalent to saying stuff.i2=2/1/0/32767, and the filename land_sea.mask means land_sea.mask=5/1/0/0. For a file intended for masking, i.e., the nodes are either 1 or NaN, the bit or mask format file may be down to 1/32 the size of the corresponding grd format file.

## 4.17 Binary table i/o

All GMT programs that accept table data input may read ASCII or binary data[†]. When using binary data the user must be aware of the fact that GMT has no way of determining the actual number of columns in the file. You should therefore pass that information to GMT via the **–bi**[**s**]*n* option, where *n* is the actual number of data columns (**s** indicates single rather than double precision). Note that *n* may be larger than *m*, the number of columns that the GMT program requires to do its task. If *n* is not given then it defaults to *m*. If $n < m$ an error is generated.

[†] The single exception is *pstext*, since it requires printable text strings of variable length.