Mapping and modeling Earth Science Data

Segment I: More UNIX and shell scripts

Thorsten Becker

University of Southern California, Los Angeles

Università di Roma TRE, June 2012

UNIX tools: compression and dealing with big files

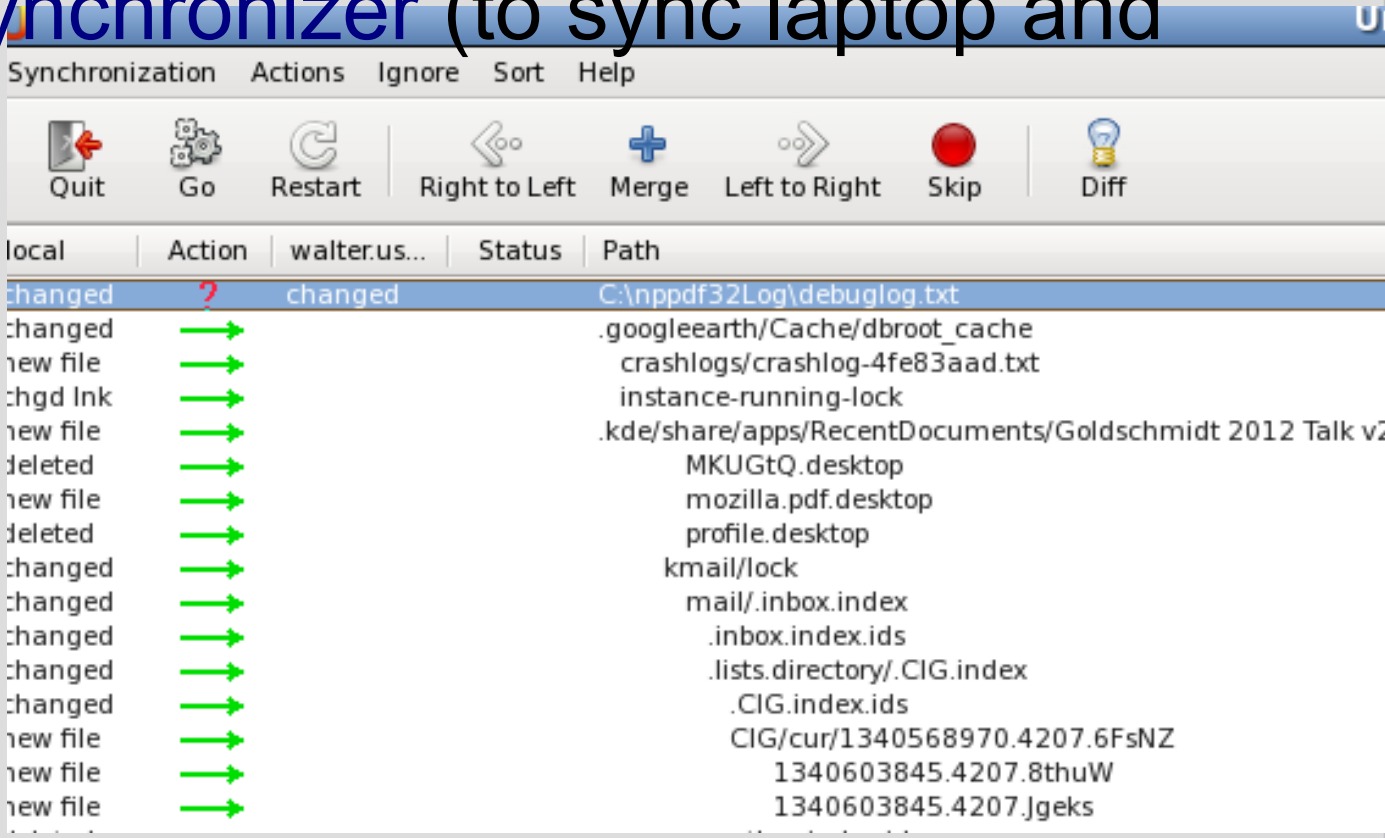
- **gzip**: compress ASCII files, *which can be huge*, to binary
  - compress: gzip file
  - uncompress: gunzip file.gz
- can write gzipped from within C, can use gunzip on the fly (**zcat**): this allows using nice ASCII tools such as awk while storing things compactly
- **bzip2**: smaller files, takes longer

UNIX tools: dealing with big and lots of files

- **tar**: combine lots of files into an archive
- ```
  tar --create --verbose --compress --file my_dir_cfiles.tgz \
  dir/*.c
  ```
- ```
  gunzip my_dir_cfiles.tgz
  tar tf my_dir_cfiles.tar
  tar xf my_dir_cfiles.tar
  ```
- **wget**: download large files
- **rsync**: copy file systems (for backup)

UNIX tools: storage and backup

- Unison file synchronizer (to sync laptop and workstation)



- *backup all the time* (it's easy to screw up big time), or have your admin backup for you

UNIX tools: awk and sed

- **awk**: (or gawk) powerful language for ASCII data and text manipulations
  - like C, interpreted at run time
  - the best thing since sliced bread
    - **cat** file.dat | gawk '{print($2,cos($5))}' *or*
    - gawk '{print($2,cos($5))}' file.dat
- **sed**: streaming text editor
  - **sed** 's/Bush/Kerry/g' file.dat > new_file.dat
- **perl**: more powerful, more complex

UNIX tools:
Getting smart: a few tricks

- unpacking on the fly:
  - **gunzip** -c newsoftware.tgz | tar xv
- interpreting commands on the fly:
  - **echo** $variable_a `cat file.dat | gawk -f mean.awk`
- tcsh interactive functionality:
  - **foreach** f (*.ps)
    - **convert** $f $f:r.gif
  - **end**

Shells: Job control

- **ps**: list currently running processes
- **jobs**: list current jobs (processes started from shell in background)
- running commands in background
  - **emacs** & (or: **emacs**; CTRL-Z; bg)
  - **echo** mybigjob.exe | nohup (don't quit with shell)
  - **kill** %2 (kill the second job running, % are job IDs)
  - **kill** -KILL 12344 (kill process with PID 12344)
- **top**: show machine load

Shells: Job control example

```
becker@jackie:~ > ps
  PID TTY          TIME CMD
 1758 pts/5    00:00:00 tcsh
 2500 pts/5    00:00:00 ps

becker@jackie:~ > ps aux | tail

becker    1413  0.0  0.0  4348 1004 ?        S    15:10   0:00 /bin/sh /usr/bin/realplay /tmp/youfm_cms.ram
becker    1418  3.2  1.1 79664 12000 ?       Sl   15:10   3:24 /usr/local/RealPlayer/realplay.bin /tmp/youfm_cms.ram
becker    1420  0.0  0.5 25720 5260 ?        S    15:10   0:00 /usr/local/RealPlayer/realplay.bin /tmp/youfm_cms.ram
becker    1421  0.0  0.5 25720 5260 ?        S    15:10   0:00 /usr/local/RealPlayer/realplay.bin /tmp/youfm_cms.ram
becker    1642  0.0  0.1  5200 1776 pts/3    Ss+  16:03   0:00 -csh
becker    1703  0.0  0.1  6624 1764 pts/4    Ss+  16:08   0:00 -csh
becker    1758  0.0  0.1  5328 1984 pts/5    Ss   16:14   0:00 -csh
becker    1888  0.0  1.6 27332 17228 ?       S    16:26   0:01 /usr/lib/acroread/Reader/intellinux/bin/acroread -display :
0.0 -name main -visual default +useFrontEndProgram -xrm *useNullDoc:false -progressPipe 3 -xrm *noPrivateColormap:true
-xrm *exitPipe:4
becker    2501  0.0  0.0  3032  772 pts/5    R+   16:54   0:00 ps aux
becker    2502  0.0  0.0  4212  532 pts/5    R+   16:54   0:00 tail


becker@jackie:~ > kill 1413
...
```

Shells: Parallel processing

```
#!/bin/bash


...

process file1 &
process file2 &
process file3 &

wait

...
```

Shells: Job automation

- **at**

```
echo "echo It\'s 1 a clock and time for lunch" |\
    at 1pm
```

- **cron** crontab file, crontab -l, crontab -r

```
#               field          allowed values
#               -----          --------------
#               minute         0-59
#               hour           0-23
#               day of month   0-31
#               month          1-12 (or names, see below)
#               day of week    0-7 (0 or 7 is Sun, or use names)
# every day at 5am, check my calendar
4 5 *  * * $HOME/progs/batch/check_calendar
# every Sun night at 4am, clean the tmp directory (WARNING!)
17 4 *  * Sun rm -rf $HOME/tmp/* 2> /dev/null
# every week, 6pm Sa backup the home directories
15 18 * * Sat $HOME/progs/batch/backup_home_rsync 0
```

Experimentation time

```
1.echo a b > file1.txt
2.echo c b > file2.txt
3.cat file1.txt
4.cat file?.txt
5.grep b file*.txt
6.ln -s file1.txt file3.txt
7.diff file1.txt file3.txt
8.rm file3.txt
9.diff file1.txt file2.txt
10.paste file1.txt file2.txt
11.ls -la
```

1. `sleep 5`
2. Run the command in the background using &.
3. Run sleep 15 in the foreground, suspend it with Ctrl-z and then put it into the background with bg.
   Type `jobs`. Type `ps`.
   Bring the job back into the foreground with fg.
4. Run `sleep 15` in the background
5. using &, and then use `kill` to terminate the process by its job number.
6. Repeat, except this time kill the process by specifying its PID.

University of Minnesota
Supercomputing Institute
for Digital Simulation and Advanced Computation
1200 Washington Ave S • Minneapolis, MN 55415 • Tel: (612)625-1818

UNIX Reference Card

Warnings!!

When a file has been **DELETED** it can only be restored from a backup. The original is gone!

When a file is **OVERWRITTEN** it has been changed forever! It can only be restored from a backup.

Directory Abbreviations

~	home directory (tilde)
~*username*	another user's home directory
.	current working directory
..	parent of current working directory

Communication

ssh [*options*] hostname
Ssh (Secure Shell) a program for logging into a remote host. Replaces telnet, rlogin, and rsh
options:
-l *login_name* specifies the user to log in on the remote machine

scp [*options*] user@host1:file1 user@host2:file2
Secure copy files between hosts on a network; uses ssh for data transfer.
options:
-p preserve modification times
-r recursively copy entire directories

Comparison

diff [*options*] *file1 file2*
Compare two text files.
options:
-a treat all files as text files
-b ignore repeating blanks and end-of-line blanks; treat successive blanks as one
-i ignore case in text comparison
-q output only whether files differ

File Management

cat [*options*] [*files*]
Read one or more *files* and print them on standard output. Use the > operator to combine several files into a new file; use >> to append files to an existing file.
options:
-n print the number of the output line to the line's left
-s squeeze out extra blank lines

cd [*dir*]
Change working directory to dir; default is the users home directory.

chgrp *newgroup files*
Change the group of one or more *files* to *newgroup*. *newgroup* is either a group ID number of a group name. Only the owner can change the group
options:
-c print information about those files that are affected
-R recursively apply changes to subdirectories

chmod [*options*] *mode files*
Change the access *mode* (permissions) of one or more *files*. Only the owner of a file or a privileged user may change its mode.
options:
-c print information about affected files
-R recursively apply changes to subdirectories
mode:
can be numeric
4 read
2 write
1 execute
or an expression of the form *who opcode permission*. *who* is optional (if missing, default is **a**)
who
u user
g group
o other
a all (default)
opcode
+ add permission
– remove permission
= assign permission
permission
r read
w write
x execute
X set execution permission only if executable by user

cp [*options*] *file1 file2*
cp [*options*] *files directory*
Copy *file1* to *file2*, or copy one or more *files* to the same names under *directory*.
options:
-a preserves attributes of original files
-f remove existing files in the destination
-i prompt before overwriting destination files
-r recursively copy directories
-s make symbolic links instead of copying

file [*options*] *files*
Classify the named *files* according to the type of data they contain.

less [*options*] [*filename*]
A program for browsing or paging through files or other output. Can use arrow keys for scrolling forward or backward.
options:
see **man** pages for options (type: **man less**)

ln [*options*] *sourcename* [*destname*]
ln [*options*] *sourcenames destdirectory*
Create links for files, allowing them to be accessed by different names.
options:
-b backup files before removing originals
-i prompt for permission before removing files
-s create a symbolic link. This lets you see the name of the link when you run **ls -l** (otherwise there is now way to know the name that a file is linked to).

ls [*options*] [*names*]
List the contents of a directory. If no *names* are given, the files in the current directory are listed.
options:
-a list all files, including hidden files
-c list files by status change time
-l long format listing (permissions, owner, size, modification time)

mkdir [*options*] *directories*
Create one or more directories.
options:
-m *mode* set the access *mode* for new directories. See **chmod** for *mode* formats.
-p create intervening parent directories if they don't exist

more [*options*] [*files*]
Display the content of the named *files* one screen at a time. See less for an alternative.
options:
see **man** pages for options (type: **man more**)

pwd
Print the full pathname of the current working directory.

scp [*options*] user@host1:file1 user@host2:file2
Secure copy files between hosts on a network; uses ssh for data transfer.
options:
-p preserve modification times
-r recursively copy entire directories

mv [*options*] *sources target*
Move or rename files and directories. The *source* and *target* determine the result.

source	*target*	*result*
file	*name*	rename file as *name*
file	existing file	overwrite existing file with source file
directory	*name*	rename directory as *name*
directory	existing subdirectory of existing directory	move directory to be a

options:
-b back up files before moving
-f force the move
-i query user before removing files

rm [*options*] *files*
Delete one or more *files*. *Once a file or directory has been removed it can only be retrieved from a backup!*
options:
-d remove directories, even if they are not empty
-f remove files without prompting
-i prompt for file removal
-r recursively remove an entire directory and its contents, including subdirectories. *Be very careful with this option.*

Miscellaneous

! Repeat the last command

!*string* Repeat the last command beginning with *string*.

cal [*-jy*] [[*month*] *year*]
Print a 12-month calendar for the given *year* or a one-month calendar of the given *month* and year. No arguments, print a calendar for the current month.
options:
-j display Julian dates
-y display entire current year

clear
Clear the terminal display

history
Display list of most recently executed commands

kill [*option*] *IDs*
Send a signal to terminate one or more process *IDs*.
options:
-l list all signals
-signal the signal number (from **ps -f**) or name (from **kill -l**). You can kill just about any process with a signal number of 9.

man *command*
Display information from the online reference manuals.

jobs [*options*] *job_id*
Display status of jobs in the current session. Simply specifying jobs returns the status of all stopped jobs, running background jobs, and all suspended jobs.
options:
-l provide more information about each job listed
-p display only the process IDs for the process group leaders of the selected jobs

whereis *command*
Locate a *command*; display the full pathname for the *command*.

which [*commands*]
List which files would be executed if the named *commands* had been run.

Searching

egrep [*options*] [*regexp*] [*files*]
grep [*options*] [*regexp*] [*files*]
Search one or more *files* for lines that match a regular expression *regexp*. To include characters such as +, ?, |, ( ), blank spaces, etc. enclose these expressions in quotes. See **man** pages for the differences between **egrep**, **fgrep**, and **grep**.
options:
-c print only a count of matched lines
-i ignore case
-l list filenames but not matched lines
-n print lines and their line numbers
-v print all lines that do not match *regexp*

find [*pathnames*] [*conditions*]
Useful for finding particular files. **find** descends the directory tree beginning at each *pathname* and locates files that meet the specified *conditions*.
options:
-name *pattern* find files whose name matches *pattern*
-print print the matching files and directories using their full pathnames
see **man** pages for options (type: **man find**)

Storage

compress [*options*] [*files*] – compress file
uncompress [*options*] [*files*] – uncompress compressed file
compress reduces the size of the named *files*. When possible the resulting compressed file will have the file extension **.Z**. Compressed files can be restored using **uncompress**.
options:
-d uncompress file, same as **uncompress**
-v prints the percentage reduction
-V prints the version of compress

gzip [*options*] [*files*] – compress file
gunzip [*options*] [*files*] – uncompress gzipped file
GNU compression utility. Renames compressed files *filename.gz*. Uncompress with **gunzip**.
options:
-d uncompress file, same as **gunzip**
-r recursively compress or decompress files within a directory
-v print name and percent size reduction for each file

tar [*options*] [*tarfile*] [*other-files*]
Copy *files* to or restore *files* from an archive. If any files are directories, **tar** acts on the entire subtree.
options:
-c create a new archive
-d compare the files stored in tarfile with other-files
-r append other-files to the end of an existing archive
-t print the names of files in archive
-v verbose, print filenames as they are added or extracted
-x extract *other-files* from archive, or extract all files if *other-files* not specified

System Status

Control-C
Stop (interrupt) job running in the foreground

Control-Z
Suspend job running in the foreground

date [*options*] [*+format*] [*date*]
Print the current date and time. You may specify a display *format*.
options:
see **man** pages for options (type: **man date**)

df [*options*] [*name*]
Report the amount of free disk space available on all mounted file systems or on a given *name*.
options:
-k print sizes in kilobytes

du [*options*] [*directories*]
Print disk used by each named directory and its subdirectories.
options:
-k print sizes in kilobytes
-s print only the grand total for each directory

env [*option*] [*variable=value ...*] [*command*]
Display the current environment or, if an environment *variable* is specified, set it to a new *value* and display the modified environment.
option:
-u unset the specified *variable*

ps [*options*]
Report on active processes.
options:
-a list all processes except processes not associated with the terminal
-e list all processes
-l produce long format listing
-u *list* list for usernames in *list*

quota [*option*]
Display disk usage and limits
option:
-v report quotas even if they haven't been exceeded

Contact Information

Phone: 612-626-0802

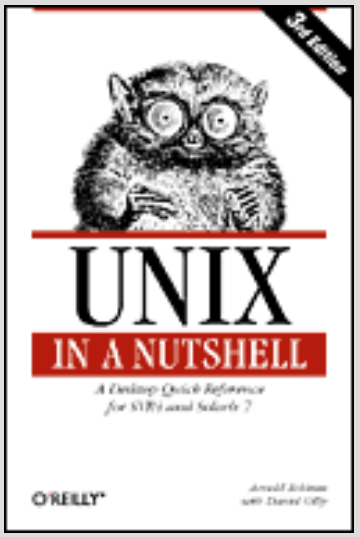
Email: help@msi.umn.edu

WWW: http://www.msi.umn.edu

http://www.msi.umn.edu/user_support

UNIX: UNIX/LINUX references

- UNIX reference card
- Robbins: UNIX in a nutshell
- Siever at al: LINUX in a nutshell
- Online list of UNIX commands
- Local computing center documentation and lecture notes
- google on UNIX: commands, shell scripts, EMACS, awk, ...



Shells: The environment

- shells: interpret your commands when logged in and using a terminal session
- **csh**, **tcsh**: nice for interactive stuff, syntax close to C, command completion, auto correction
- **bash**, **ksh**: nice for programming
- shells use mostly same commands, but there are differences in the script languages, *e.g.*
  - **export** var=100 (bash)
  - **setenv** var 100 (csh)

Shells: Command history and other feature that save typing

- use up, down, left, right arrows to navigate and edit commands on the command line
- use a bunch of tricks to access and modify last commands, *e.g.*
  - !n: execute last command that starts with “n”
- *auto-completion* (TAB key)
- *auto-correction*
- many more tricks

Shells: Can use variables, and many are predefined (csh example)

```
becker@jackie:~ > setenv region 0/360/-90/90

becker@jackie:~ > echo $region
0/360/-90/90

becker@jackie:~ > echo $HOME
/home/becker

becker@jackie:~ > echo $USER
becker

becker@jackie:~ > env
BIBINPUTS=.:/home/becker/TEX//:
CFLAGS_DEBUG=-g -DDEBUG -DDEBUG -DLINUX_SUBROUTINE_CONVENTION
LDFLAGS=-posixlib -nofor_main  -Vaxlib  -L/usr/lib/gcc/i386-redhat-linux/3.4.3/ -lg2c -lm
MANPATH=/home/becker/progs/man/:/home/becker/progs/man/:
DVIPSHEADERS=/home/becker/TEX//:
SUPPORTED=en_US.UTF-8:en_US:en
SSH_AGENT_PID=31881
HOSTNAME=jackie.usc.edu
DXROOT=
DXMEMORY=128
CONFC=ifort
HOST=jackie.usc.edu
SHELL=/usr/bin/tcsh
FFLAGS_DEBUG=-g -DDEBUG -fpp    -nofor_main  -DDEBUG

....
```

Shells: Startup/configuration

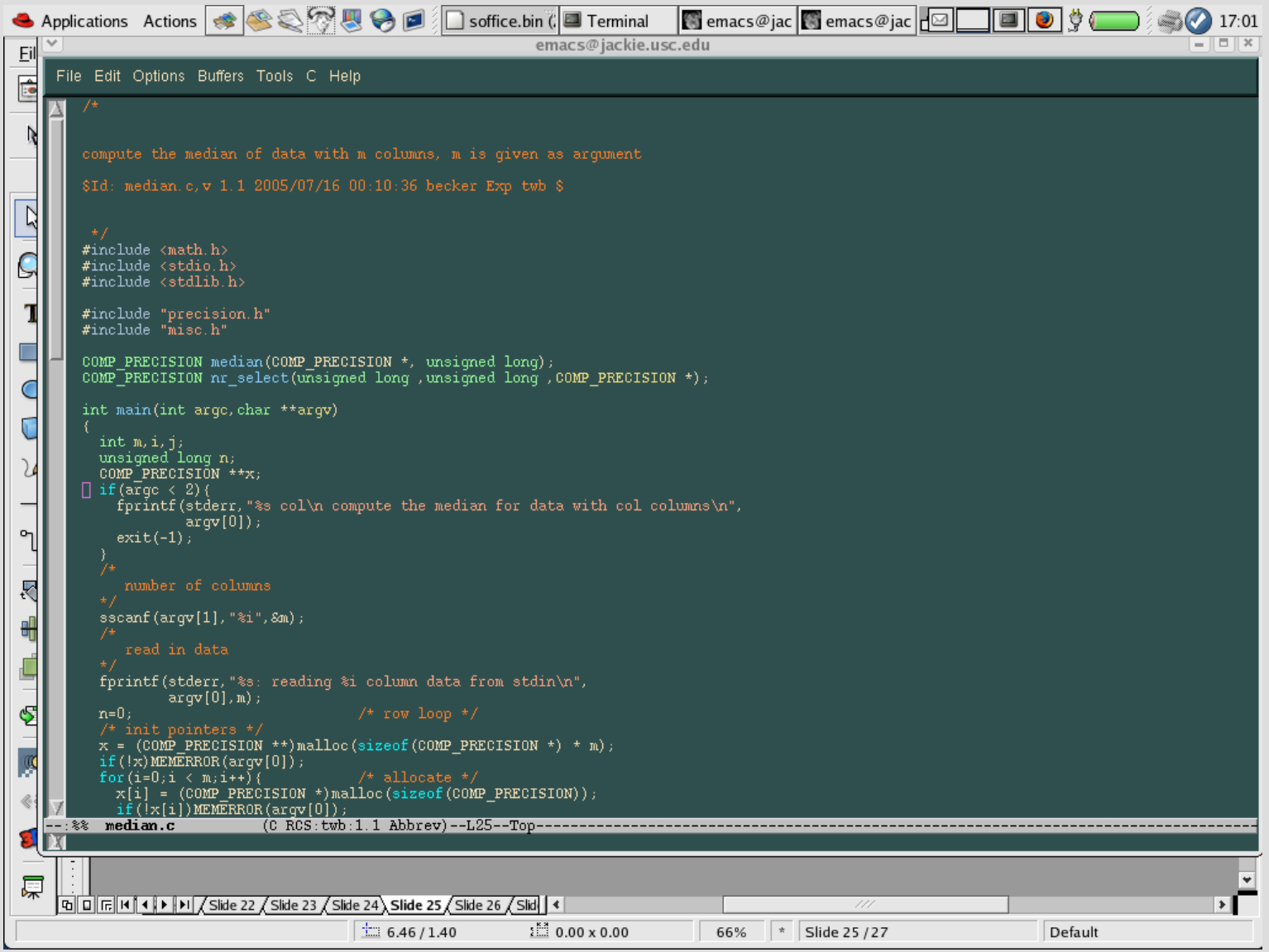
- ~/.login (= $HOME/.login) at startup
- ~/.cshrc every time you start a shell
- those scripts are where you define environment variables and aliases you want to use in all sessions
  - **alias** rm 'rm -i'
  - **setenv** F77 ifort; **setenv** FFLAGS “-O3 -ipo”
- ~/.login is an example of a hidden file
- see references on UNIX and dotfiles.com

Editors:
Editing text or ASCII data files

- **vi**: old school: fast, efficient, bizarre
  - controlled by typing commands like !w, /text
  - good for minor editing tasks, required for admins
- **emacs**: best overall tool
  - GUI, menus
  - flexible, expandable
  - Bizarre
- **gedit**: people like it because it's close to Word
- tons of others, but don't use Word or such, since UNIX expects pure ASCII characters

Editors:

What EMACS looks like



Scripts: Scripts and GUIs

- scripts are the opposite of point-and-click
- need to work hard once to generate template
- benefit forever if you want to produce more products (*e.g.* plots) using different parameters, or if the data has changed
- automate research galore (needed to explore parameter space)
- scripts can serve as documentation of steps taking to analyse data and produce results

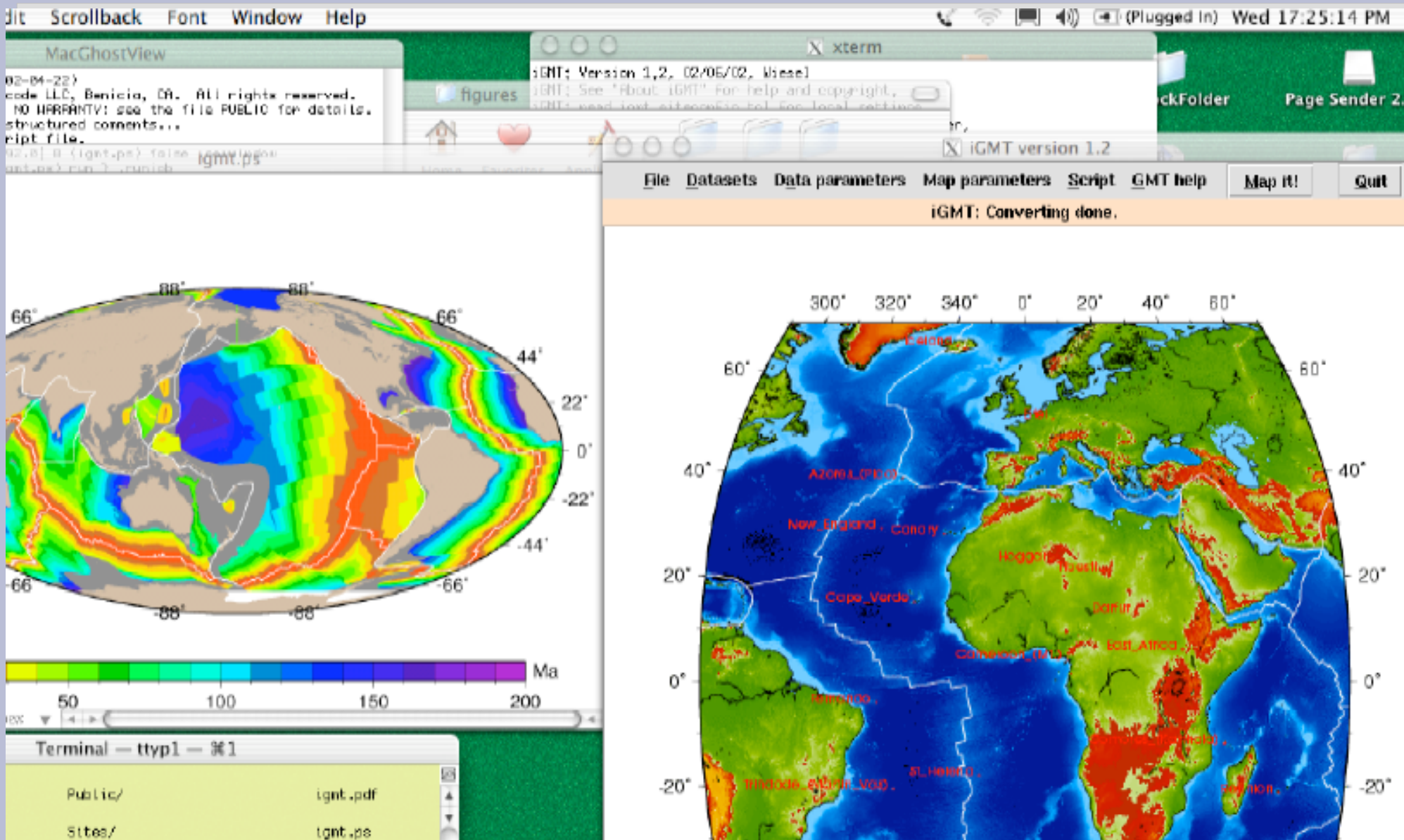
Scripts: An example script

```
#!/bin/bash
#
# run run_fstrack for different models
#
models=${1-"pmDsmean_nt pmDnngrand_nt saf1 saf2 saf3 "}
strains=${2-"2 1 0.5"}

# PBS queues to use
q1="becker64";q2="scec"

c=0
for m in $models;do
    cd $m
    for s in $strains ;do
        if [ $c -eq 1 ];then
            queue=$q1;c=0
        else
            queue=$q2;c=1
        fi
        # regional
        ../run_fstrack 1 0 0 0 $s 1 2 1 -14 0 60 "" 1 $queue
    done
    cd -
done
```

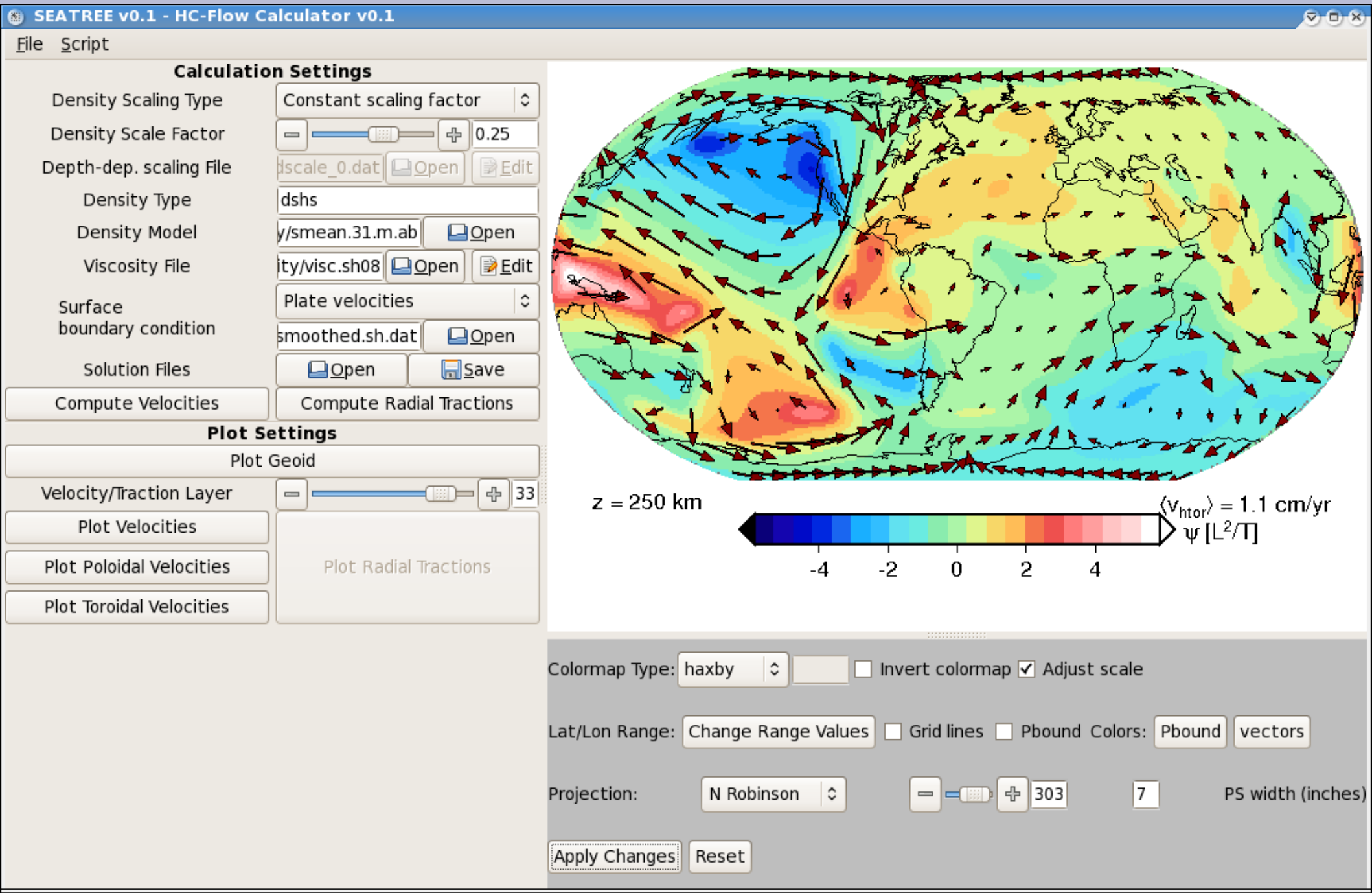
Scripts: scripting languages

- csh, tcsh
- bash
- perl
- *python*
- Scripted programs
  - gnuplot
  - Matlab
  - GMT
- Script whenever you can (because you'll want to reproduce things exactly)

Scripts: Visual scripting languages: Tcl/TK (*e.g. iGMT*), *GTK, Tkinter, Qt*



GUIs: Python and Gtk: SEATREE



Scripting

- Can be the way to go if individual processing steps are not time-sensitive
- If speed is an issue, need to compile from higher level language such as script
- For basic LINUX automatization tasks, bash and awk are very useful
- Python seems to be a nice middle ground for more advanced projects

Bash script programming

```
#!/bin/bash
#
# run run_fstrack for different models
#
models=${1-"pmDsmean_nt pmDnngrand_nt saf1 saf2 saf3 "}
strains=${2-"2 1 0.5"}

# PBS queues to use
q1="becker64";q2="scec"

c=0
for m in $models;do
    cd $m
    for s in $strains ;do
        if [ $c -eq 1 ];then
            queue=$q1;c=0
        else
            queue=$q2;c=1
        fi
        # regional
        ../run_fstrack 1 0 0 0 $s 1 2 1 -14 0 60 "" 1 $queue
    done
    cd -
done
```

Bash scripting

- Command line arguments

```
file=${1-”tmp.grd”}
```

- If, case statements

```
if [[ $file = ”tmp.ps” && $i -eq 1 ]];then
 ...
fi
```

- Loops

```
while [ $i -le $n ];do
 ...
 ((i=i+1))
done
```

Bash scripting

- Command evaluation
  `variable = `grd2max $file.grd``
  where `grd2max` is another script:

```
#!/bin/bash
#
# program to find the maximum value of an Netcdf grd file using
# the GMT tool grdinfo
#
pname=`basename $0`
if [ $# -ne 1 ]; then           # need at least one argument
    echo $pname: usage: > /dev/stderr
    echo $pname file    > /dev/stderr
exit

if [ ! -s $1 ];then             # file does not exist
    echo $pname: error: file $1 is not there > /dev/stderr
    exit
fi
grdinfo -C $name | gawk '{print($7)}'
```

Bash scripting

- Functions
- Traps

```
tmpn=/tmp/$USER.$HOST.$$.`basename $0`
trap "rm -f $tmpn.* ; exit" 0 1 2 15
```

- Problems:
  - Sensitive to white space (`if [ $i -eq 5]` won't work)
  - Usually only deals with integers (`=` vs. `-eq` comparison)

Experimentation time

- Write a script that outputs:

```
1
2
3
4
5
OK
```

- Write a script that takes input *n* `myscript n` and writes

```
1
2
...
n
OK
```

- Write a script that works such that

```
>hello_user.sh
Hello lebowski!
Your home is: /nihilist/lebowski
Today's date is: Sun Jun 24 18:19:42
CDT 2012
```