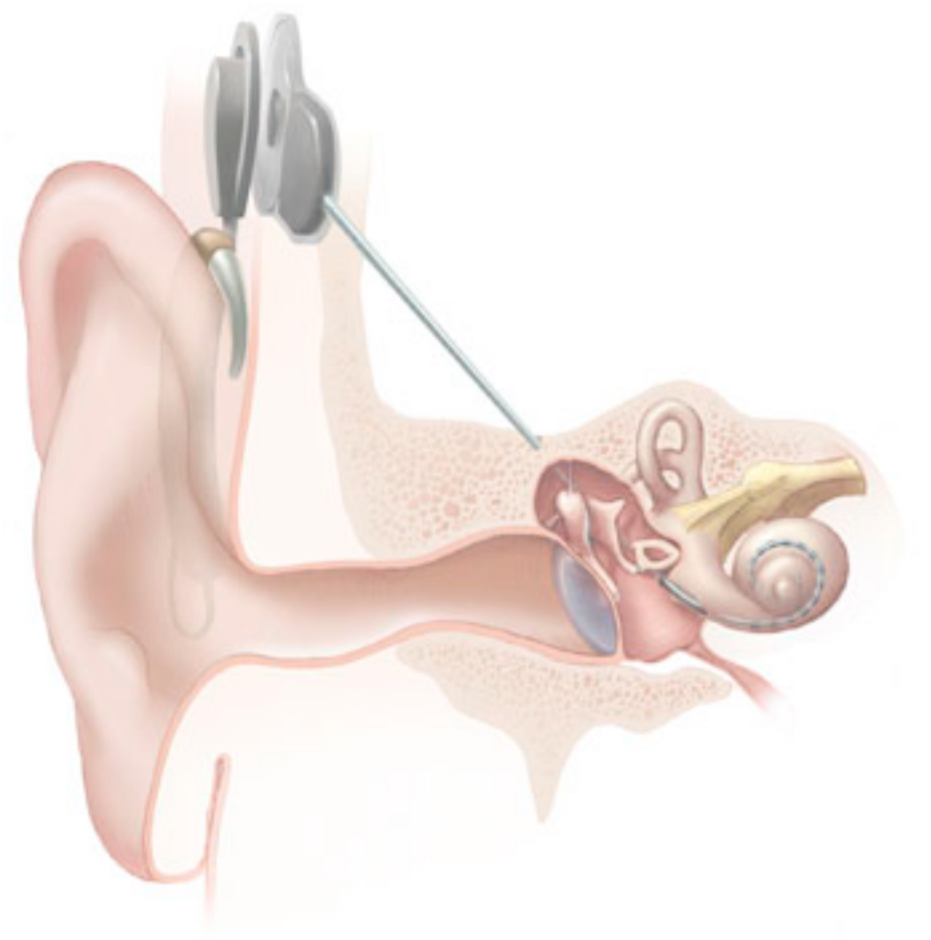
An introduction to the Shell

Software Carpentry Nov 2011 Bootcamp

Cochlear Implants

A cochlear implant is a small electronic device that is surgically implanted in the inner ear to give deaf people a sense of hearing. More than a quarter of a million people have them, but there is still no widely-accepted benchmark to measure their effectiveness. In order to establish a baseline for such a benchmark, our supervisor got teenagers with CIs to listen to audio files on their computer and report:

- the quietest sound they could hear
- the lowest and highest tones they could hear
- the narrowest range of frequencies they could discriminate

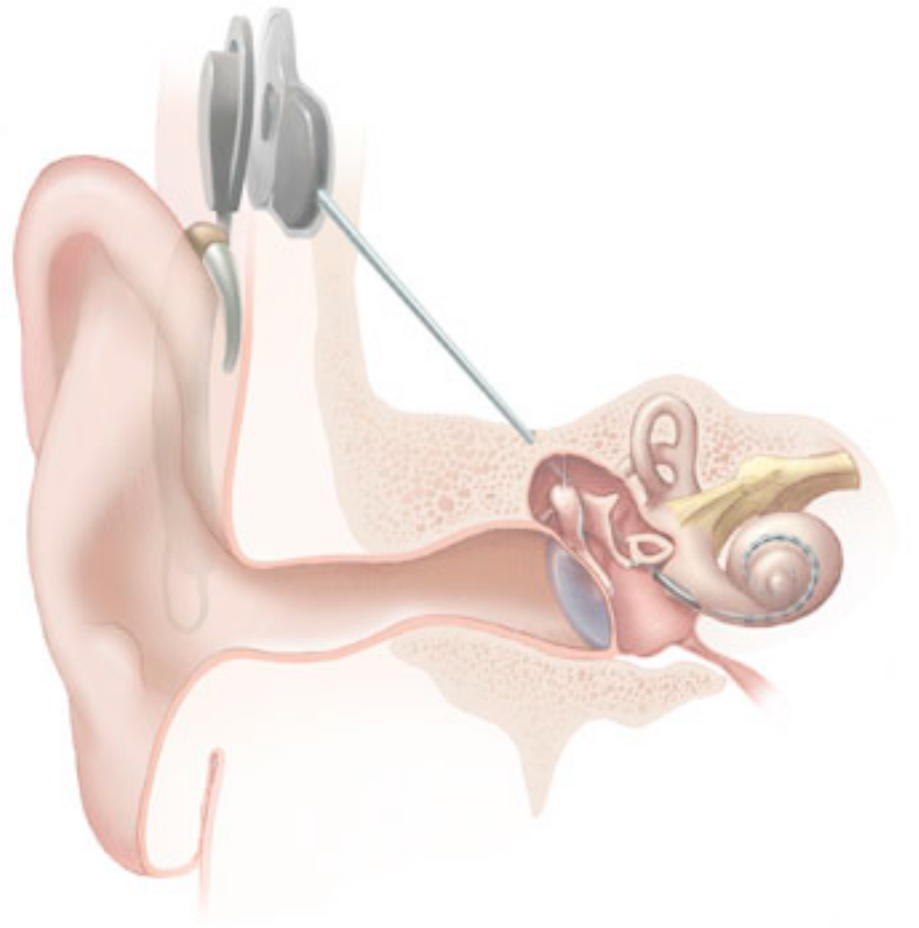

http://en.wikipedia.org/wiki/File:Cochlear_implant.jpg

Cochlear Implants

To participate, subjects attended our laboratory and one of our lab techs played an audio sample, and recorded their data - when they first heard the sound, or first heard a difference in the sound. Each set of test results were written out to a text file, one set per file.

Each participant has a unique subject ID, and a made-up subject name.

Each experiment has a unique experiment ID.



http://en.wikipedia.org/wiki/File:Cochlear_implant.jpg

Cochlear Implants

Our job is to do some preliminary analysis on that data.
We need to:

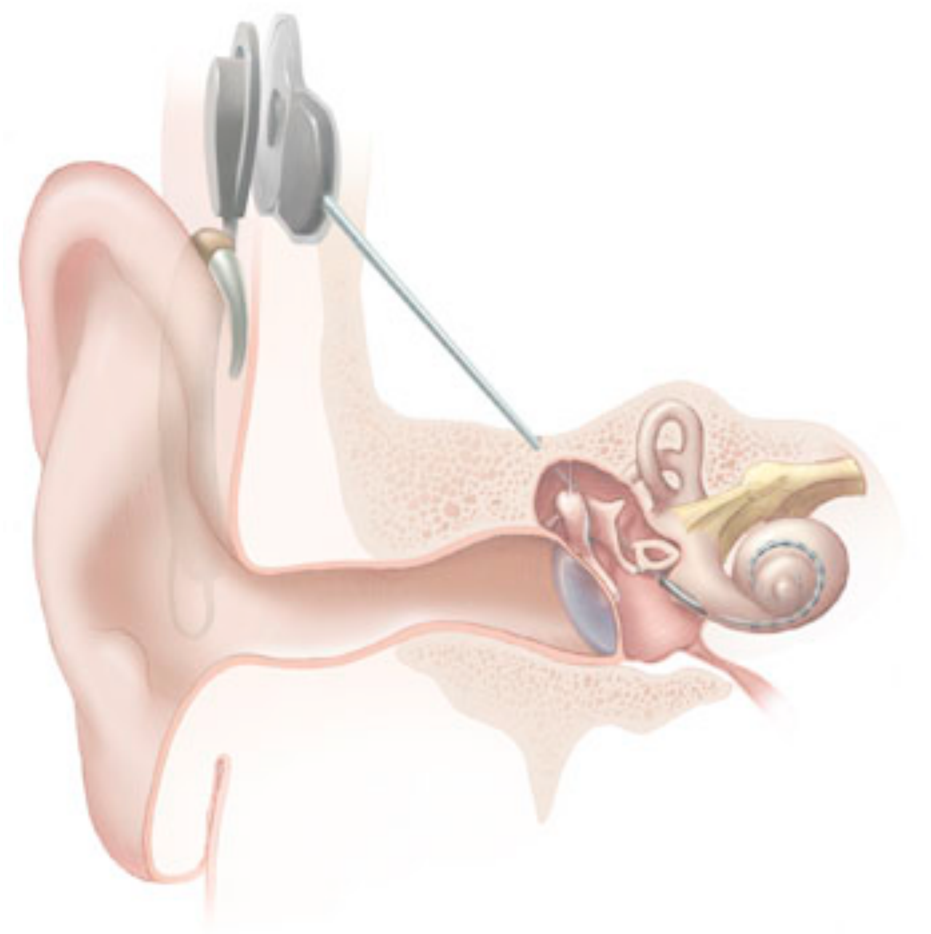
•identify and label files that are missing data (for example, because the participant didn't complete all three tests);

•normalize the data (the first version of the software reported a score for each test in the range 0-9, but it was later "fixed" to report scores in the range 1-10);

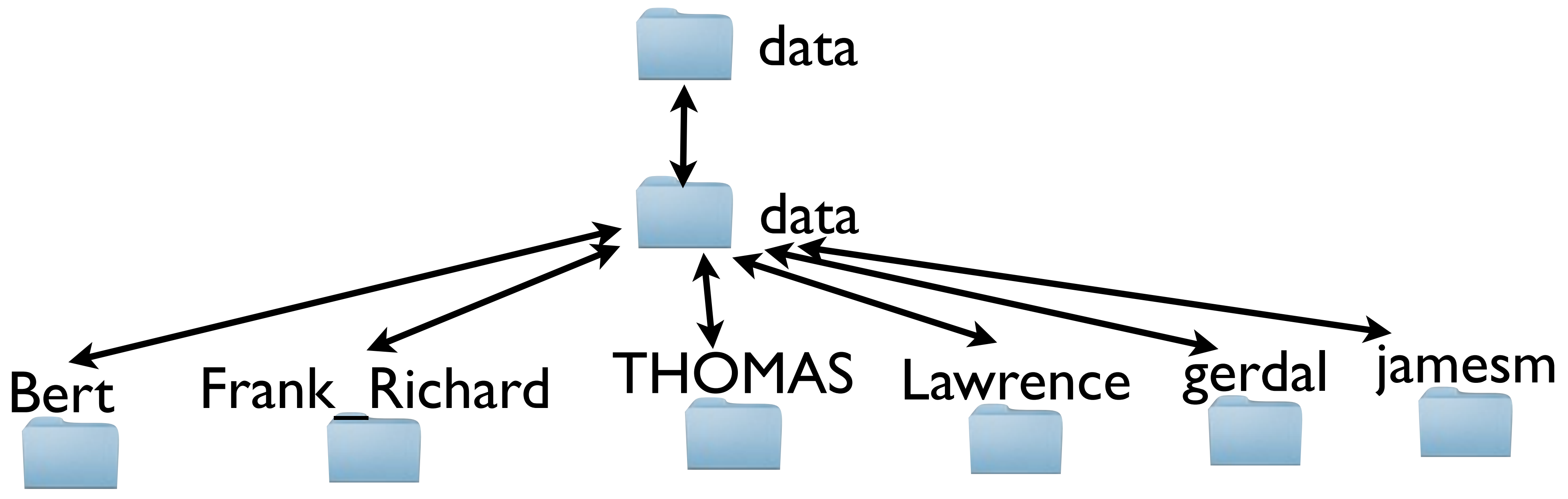
•put the data into a database to make subsequent analysis easier; and

•calculate a few simple statistics, such as average scores for each test by CI model and participant's age and sex.

The experiment has collected 351 files so far, and we expect to get another 30-40 per week for the next couple of months, so we'd really like to automate the four steps above.


http://en.wikipedia.org/wiki/File:Cochlear_implant.jpg

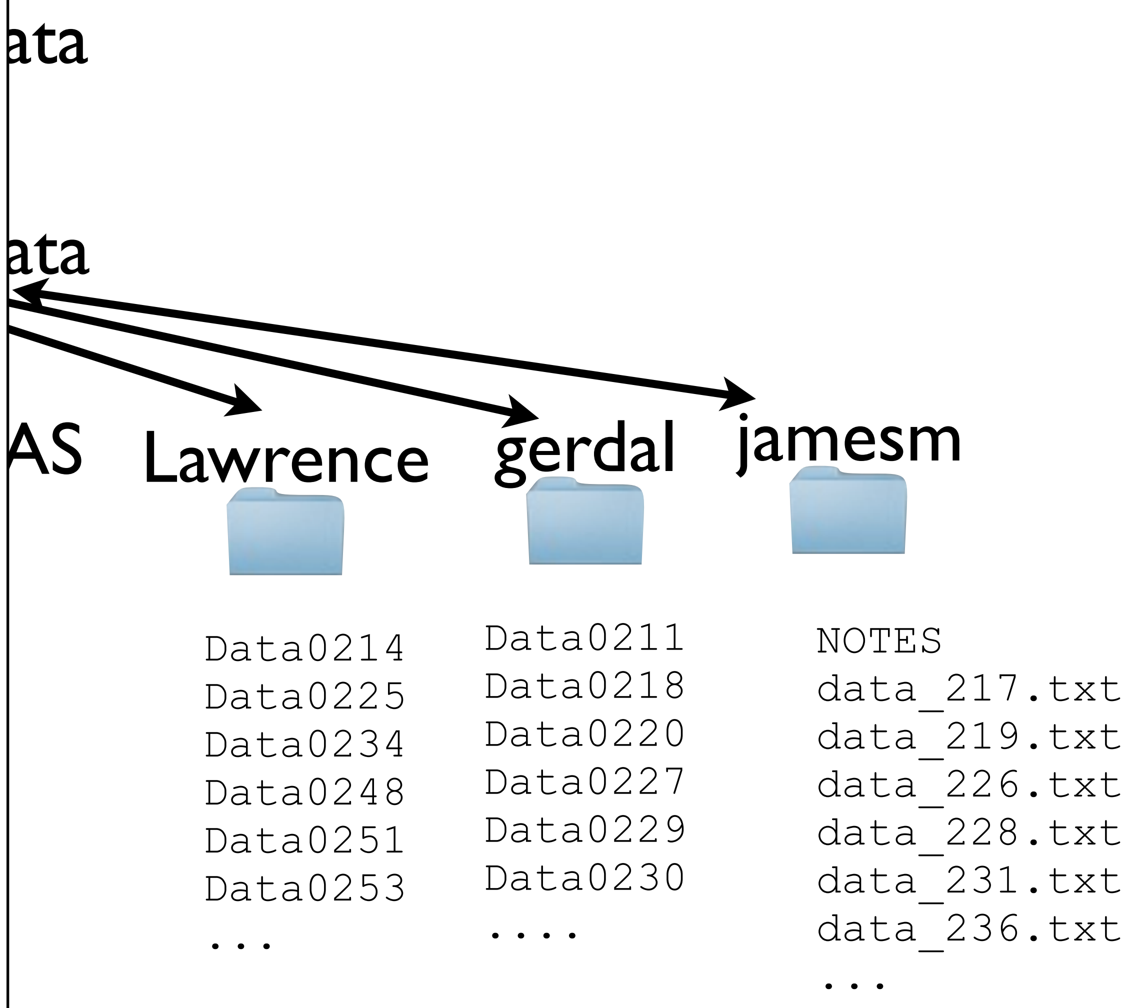
Data is a bit of a mess



audioresult-00215	NOTES	0213	Data0214	Data0211	NOTES
audioresult-00222	data_212	0241	Data0225	Data0218	data_217.txt
audioresult-00223	data_221	0244	Data0234	Data0220	data_219.txt
audioresult-00235	data_224	0249	Data0248	Data0227	data_226.txt
audioresult-00239	data_233	0274	Data0251	Data0229	data_228.txt
audioresult-00246	data_237	0281	Data0253	Data0230	data_231.txt
....	data_247	0287	...	....	data_236.txt
	...	...			...

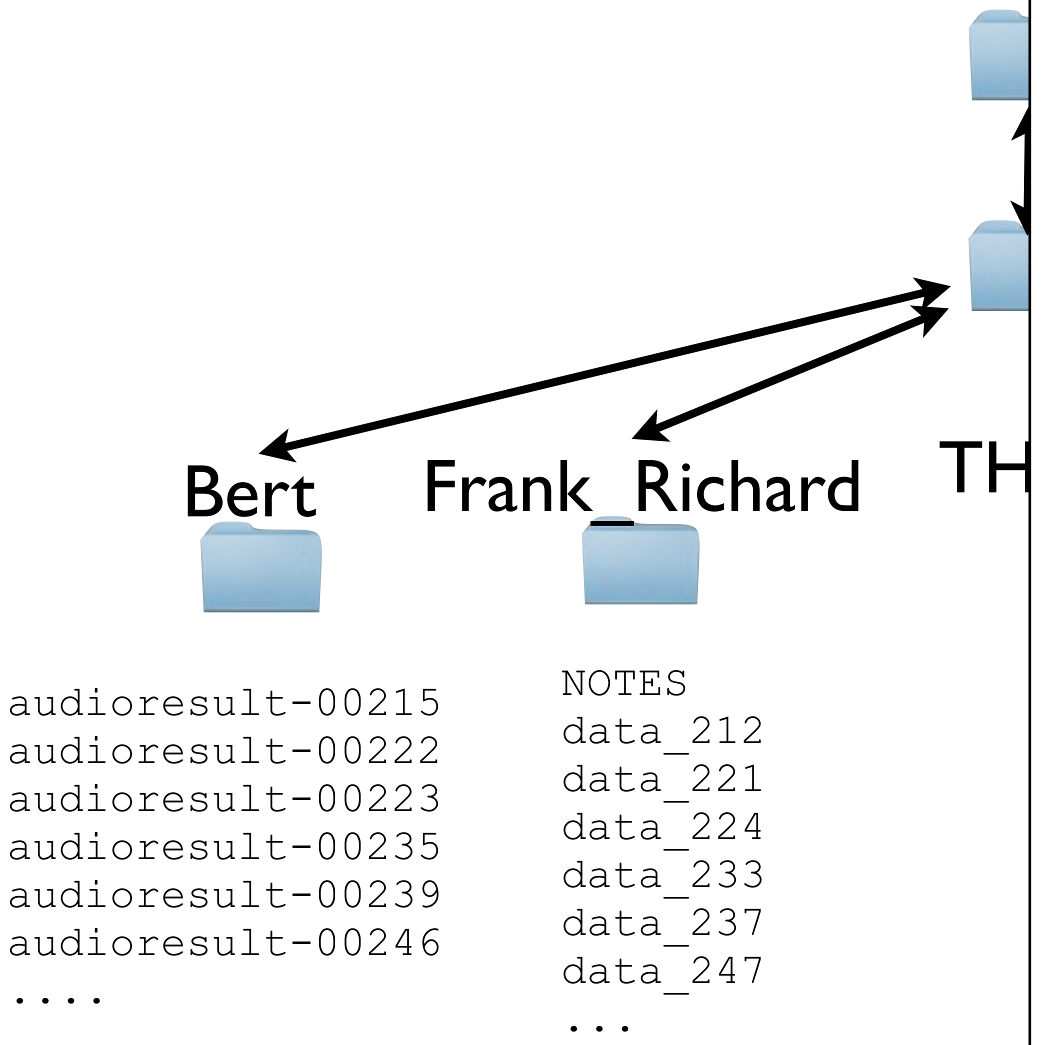
Data is a bit of a mess

- Inconsistent file names
- Some directories have extraneous NOTES file
- multiple directories.



```
Data0214      Data0211      NOTES
Data0225      Data0218      data_217.txt
Data0234      Data0220      data_219.txt
Data0248      Data0227      data_226.txt
Data0251      Data0229      data_228.txt
Data0253      Data0230      data_231.txt
...           ....          data_236.txt
                            ...
```

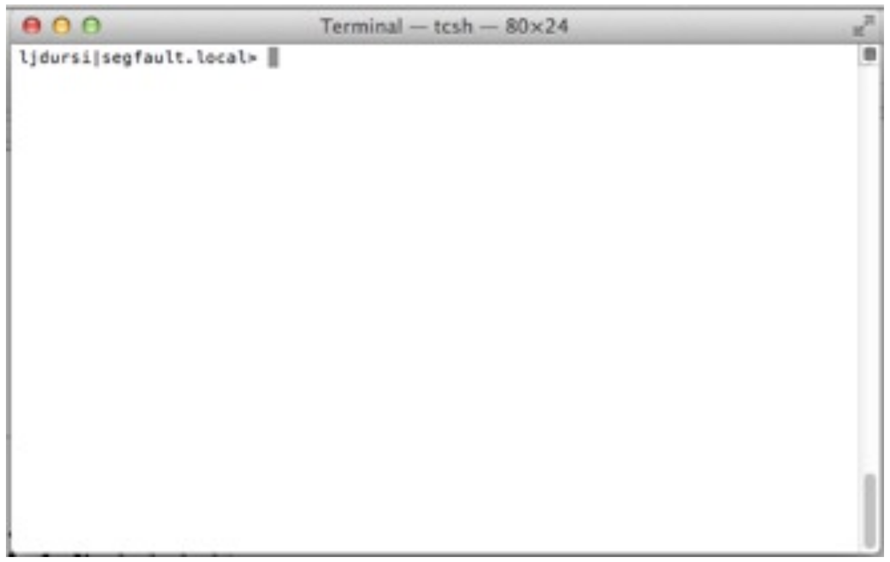
Data is a bit of a mess



- Our job, by end of this session:
- Make **one** directory (alldata)
- have all *data* files in there, all with .txt extension
- Get rid of NOTES files.

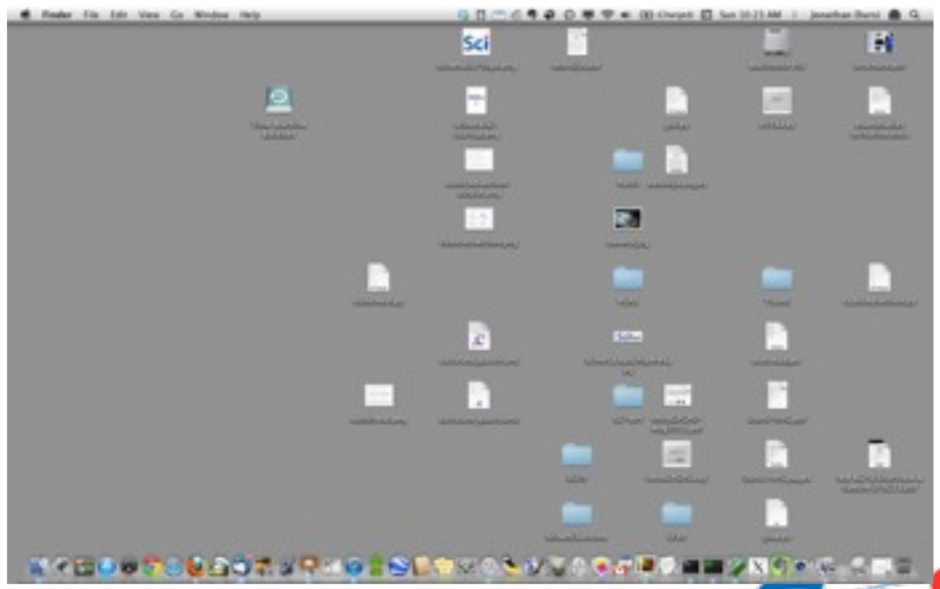
Shell vs GUI

- Presents a Command Line Interface (CLI, or CUI) vs GUI interface to your computer.
- Why on earth would you use a command line interface?



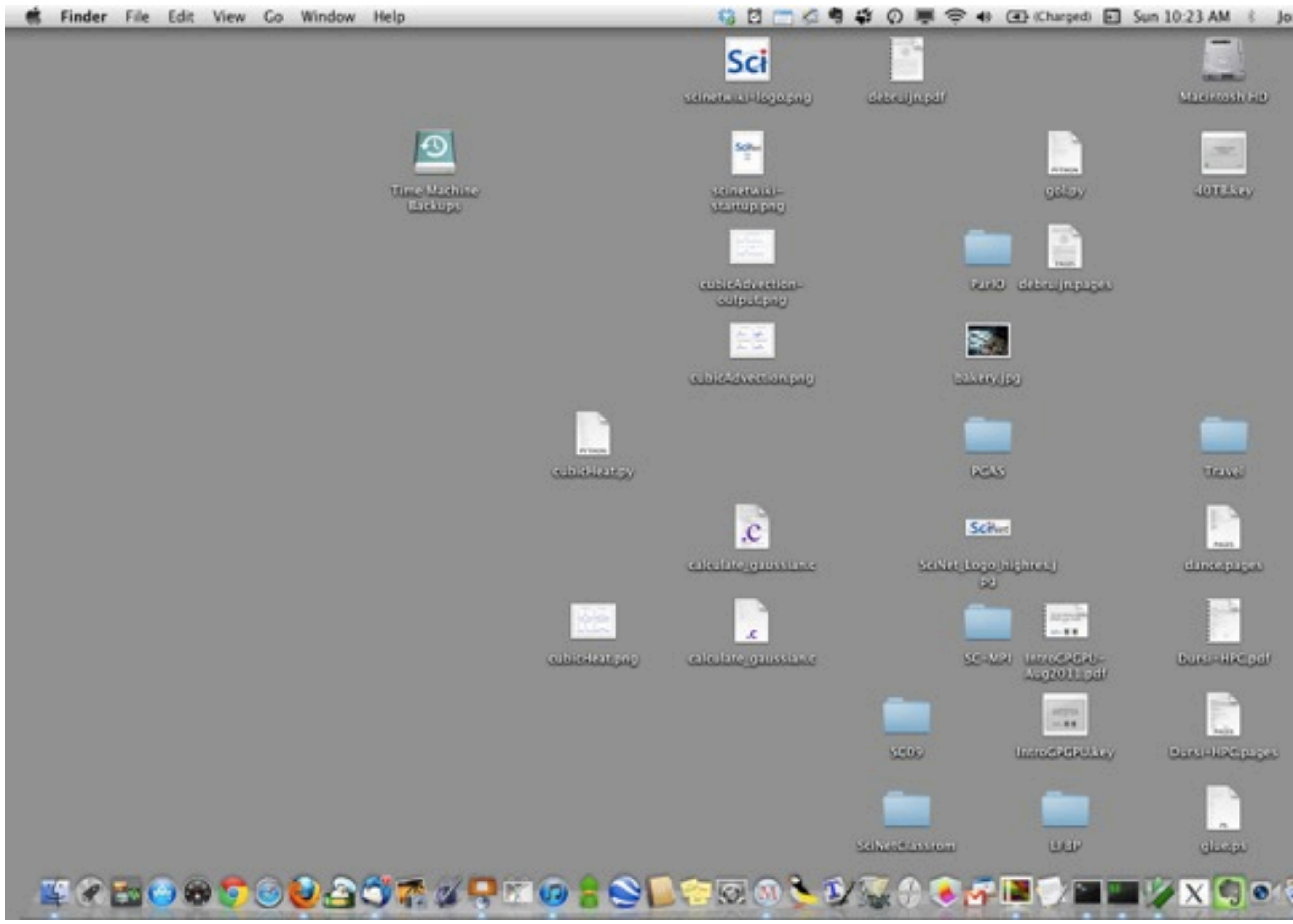
```
ljdursi|segfault.local>
```

vs.



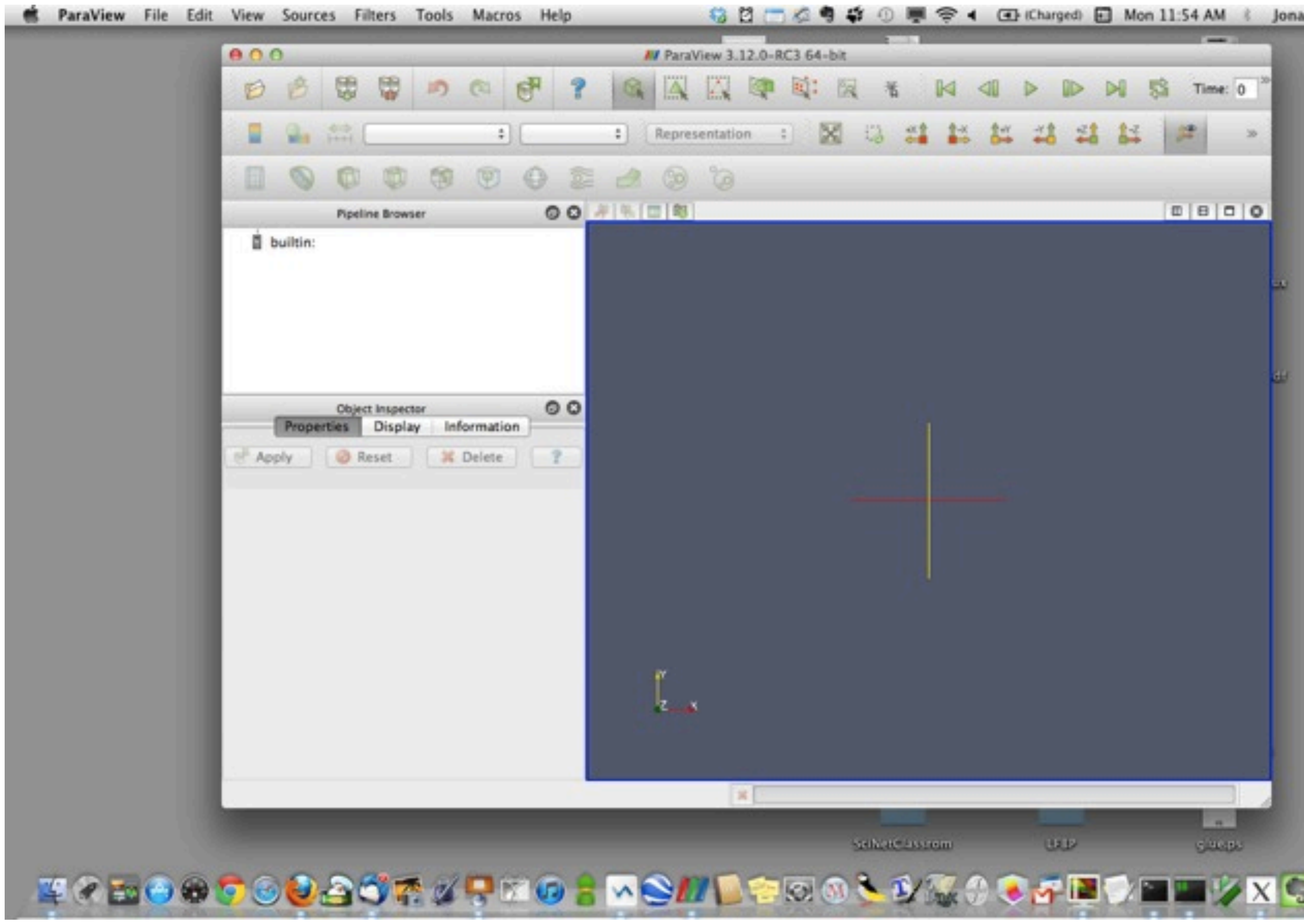
GUI: Operating

- Very good at **operating** an existing system.
- Click on *existing* controls, use *existing* functionality.



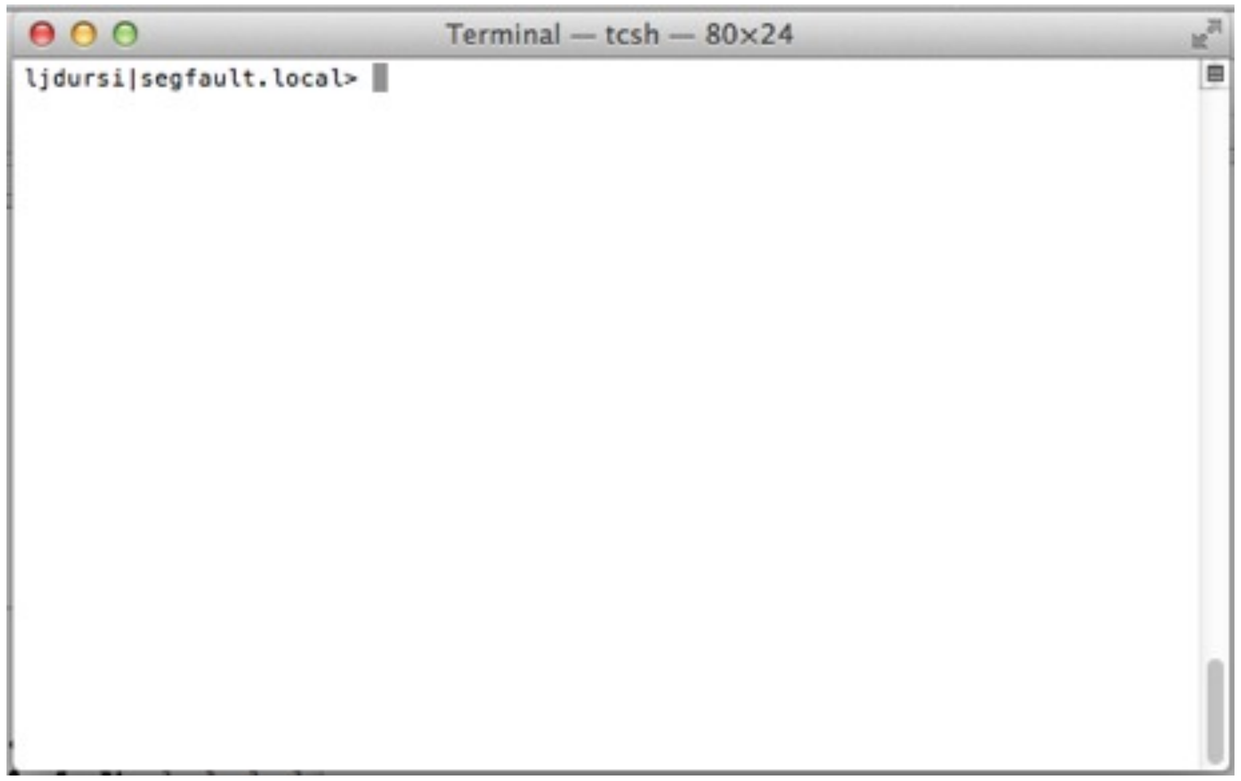
GUI: Operating

- Useful for basic computer operations,
- Operating existing software packages.


ParaView File Edit View Sources Filters Tools Macros Help
ParaView 3.12.0-RC3 64-bit
Representation
Pipeline Browser
builtin:
Object Inspector
Properties Display Information
Apply Reset Delete

CLI - creating

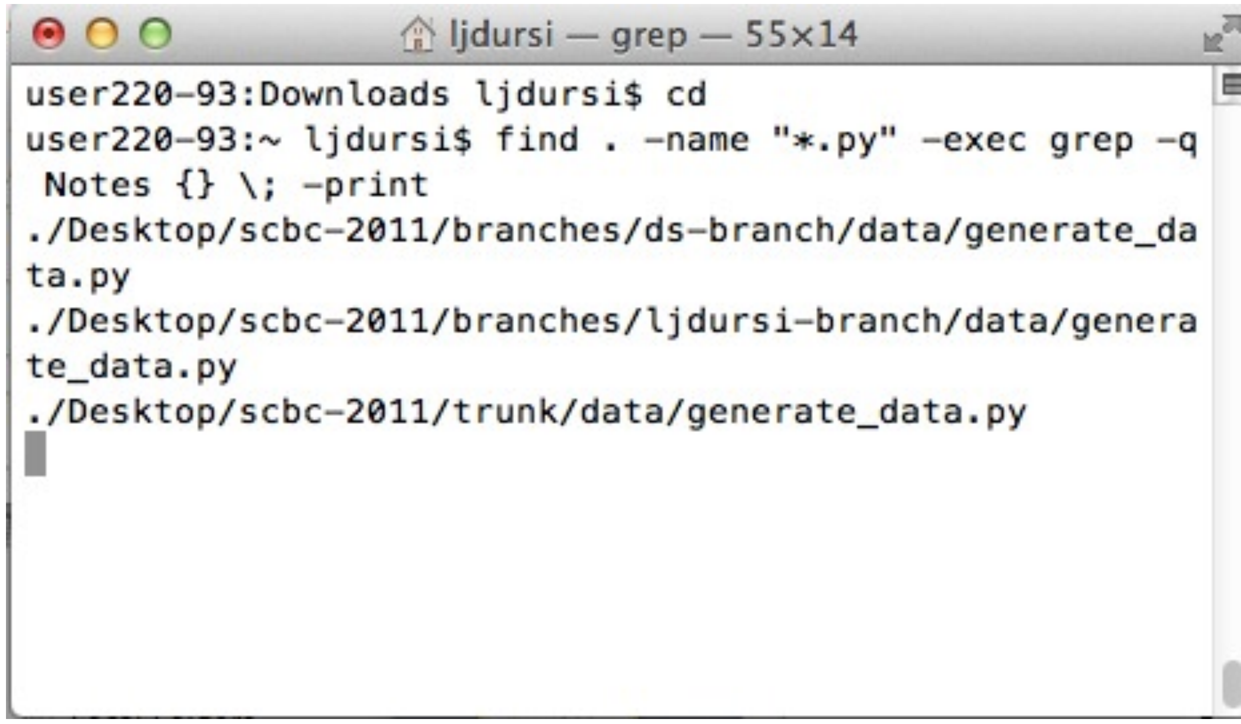
- For better or worse, a blank canvas
- Good for creating/ expressing new things.
- Programming in a GUI hard (but not impossible; Mac OSX Automator)



```
ljdursi|segfault.local>
```

CLI - reproducable

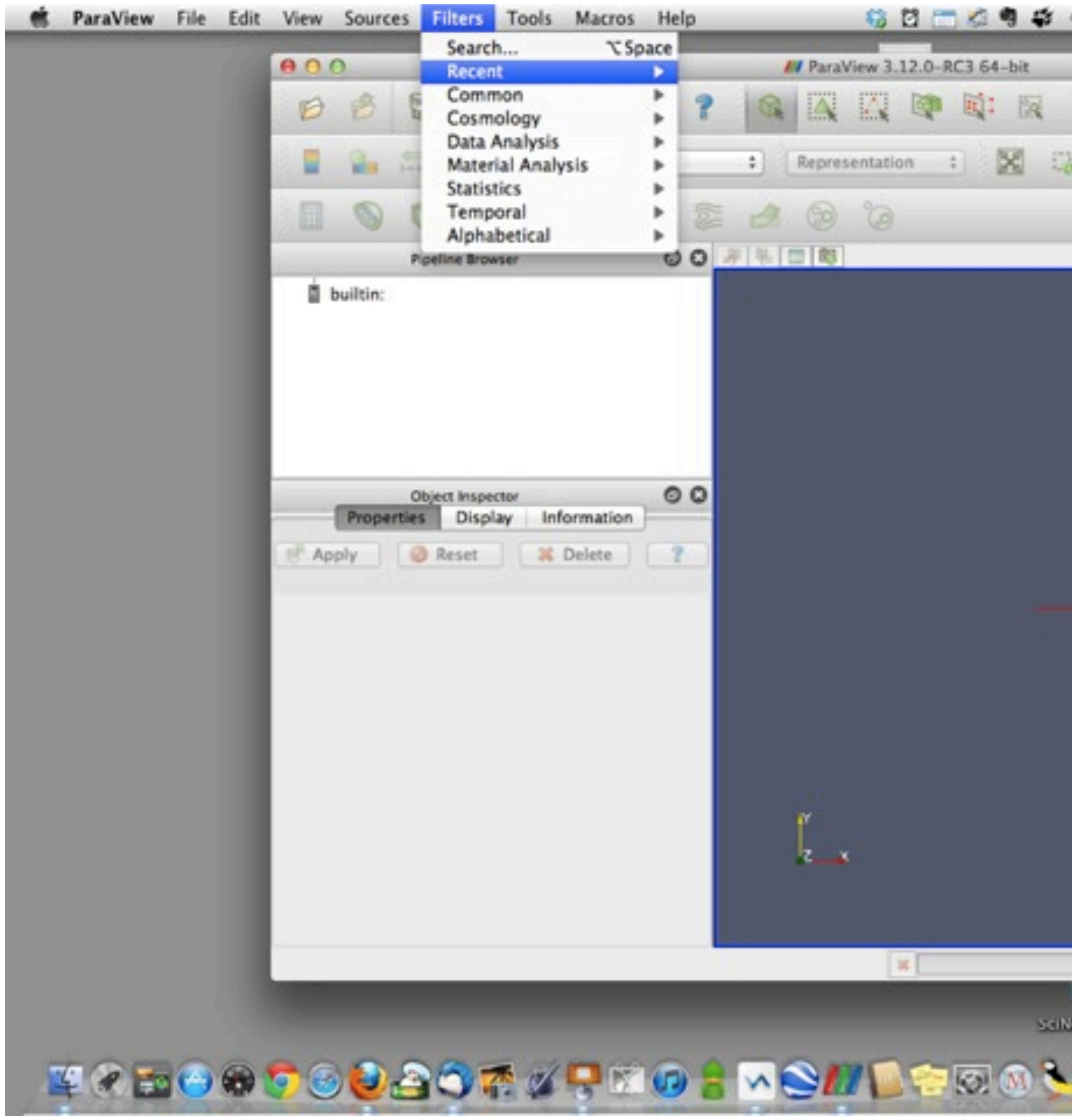
- Command lines can be cryptic to learn, but once you have the command, you can communicate it to others exactly.



```
user220-93:Downloads ljdursi$ cd
user220-93:~ ljdursi$ find . -name "*.py" -exec grep -q
 Notes {} \; -print
./Desktop/scbc-2011/branches/ds-branch/data/generate_da
ta.py
./Desktop/scbc-2011/branches/ljdursi-branch/data/genera
te_data.py
./Desktop/scbc-2011/trunk/data/generate_data.py
```

GUIs - not as reproducable

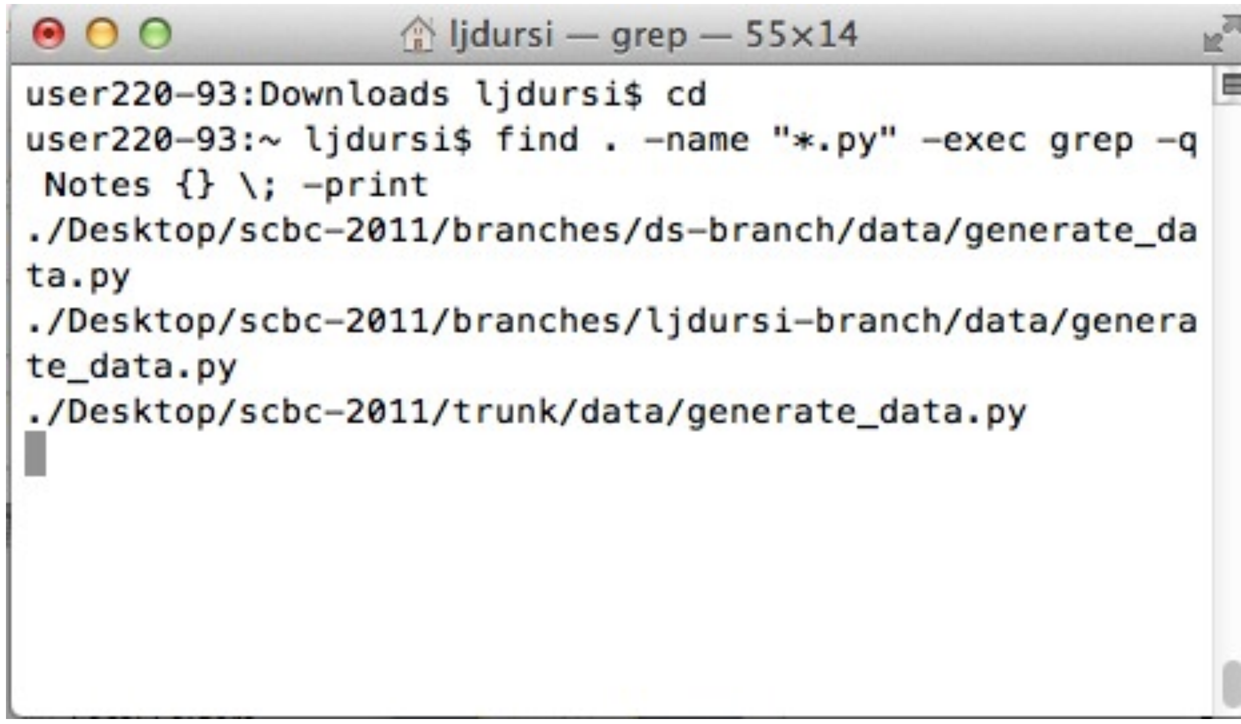
- "Click on Filters, then 'Recent'"
- "Then drag the green arrow down to the big grey box.."
- "... No, the other one.."
- "... Not there!"
- "Ok, let's start again..."



SciNet

CLI makes you more productive

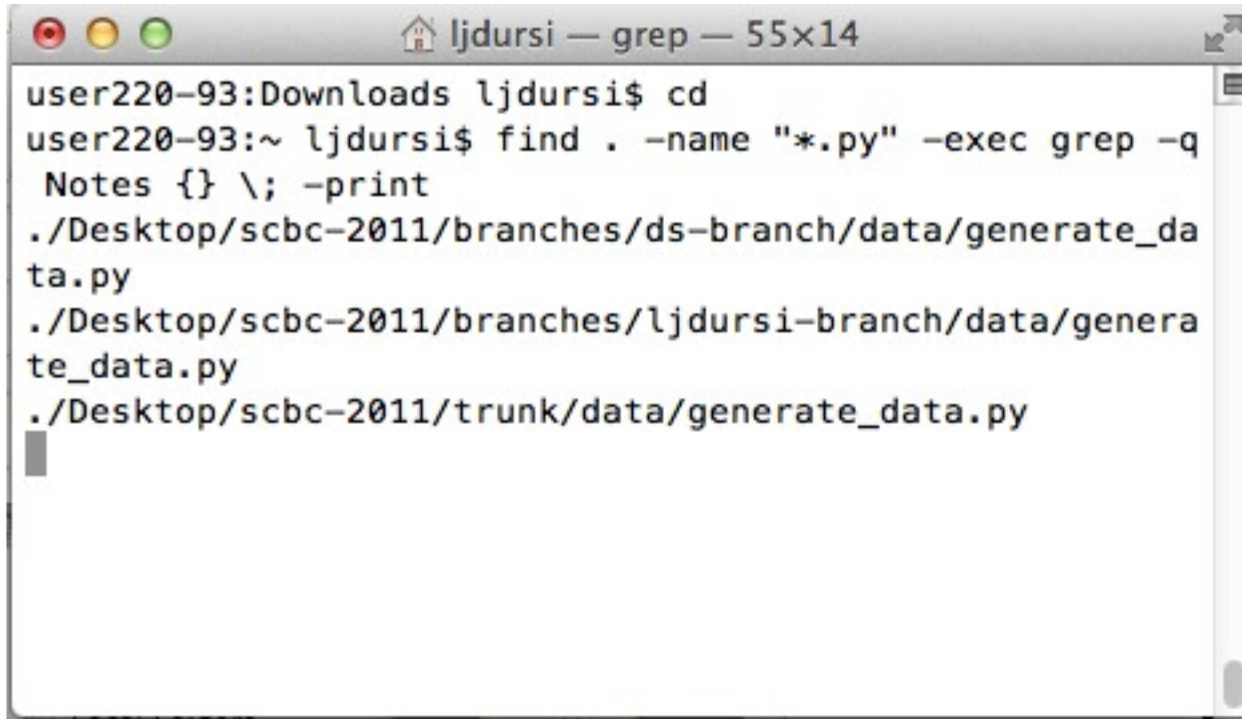
- Reproducable - stop wasting time re-discovering how to do things
- Automatable - can do the same thing hundreds of times easily without wasting time
- More time doing research



```
user220-93:Downloads ljdursi$ cd
user220-93:~ ljdursi$ find . -name "*.py" -exec grep -q
 Notes {} \; -print
./Desktop/scbc-2011/branches/ds-branch/data/generate_da
ta.py
./Desktop/scbc-2011/branches/ljdursi-branch/data/genera
te_data.py
./Desktop/scbc-2011/trunk/data/generate_data.py
```

CLI makes you more productive

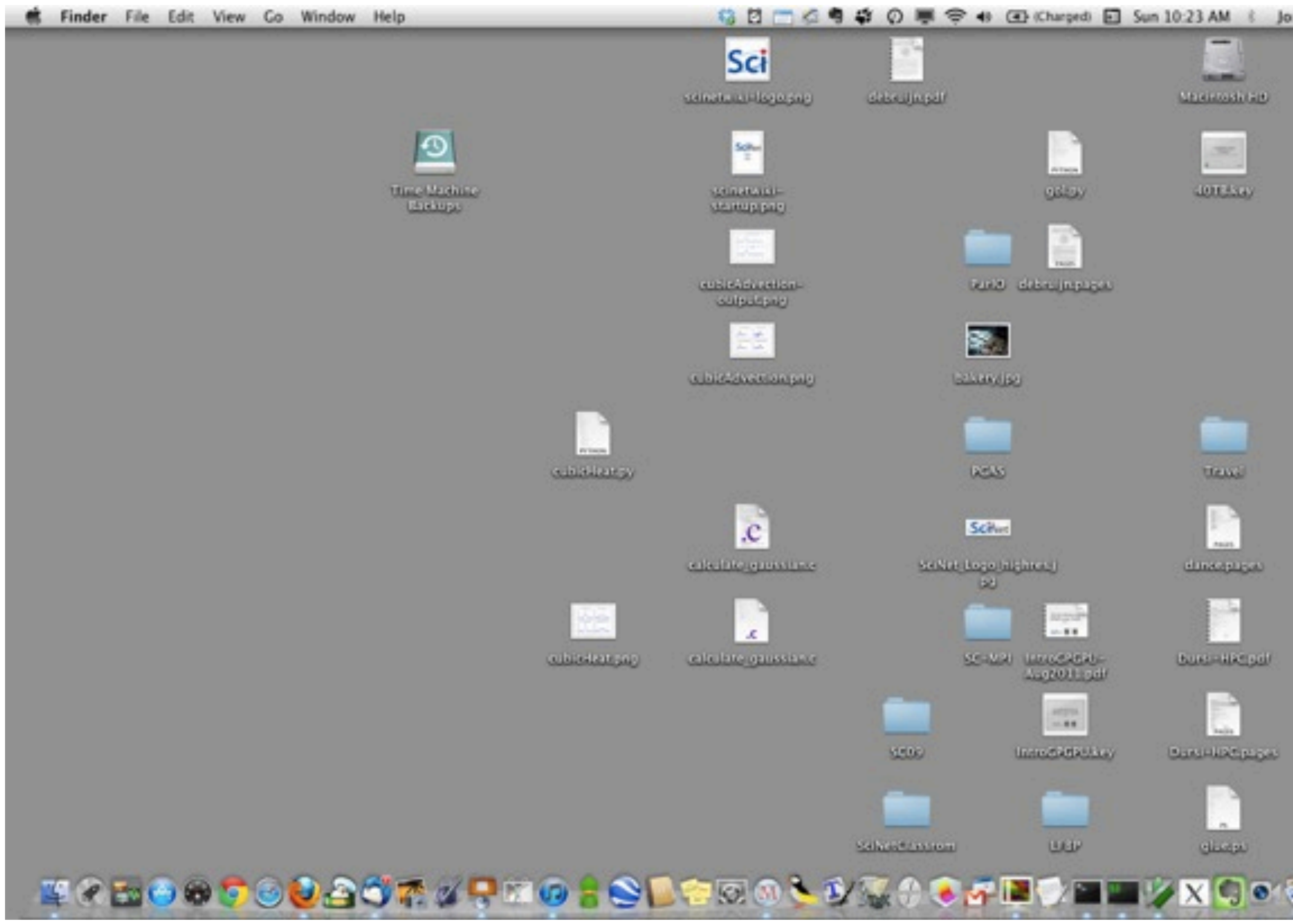
- But there's a learning curve.
- Investment in future productivity.



```
user220-93:Downloads ljdursi$ cd
user220-93:~ ljdursi$ find . -name "*.py" -exec grep -q
 Notes {} \; -print
./Desktop/scbc-2011/branches/ds-branch/data/generate_da
ta.py
./Desktop/scbc-2011/branches/ljdursi-branch/data/genera
te_data.py
./Desktop/scbc-2011/trunk/data/generate_data.py
```

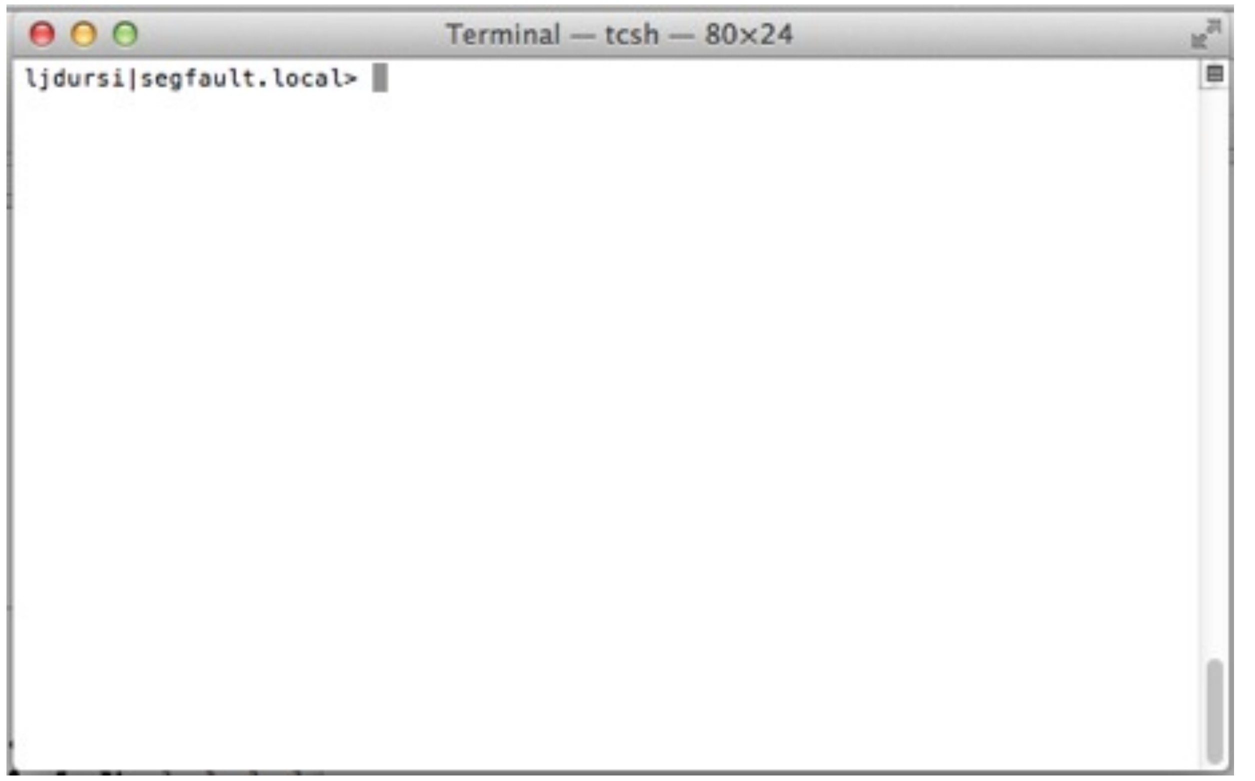
GUI - Easy / Hard

- Easy to *learn/ discover*
- Hard to *use* for big tasks productively.



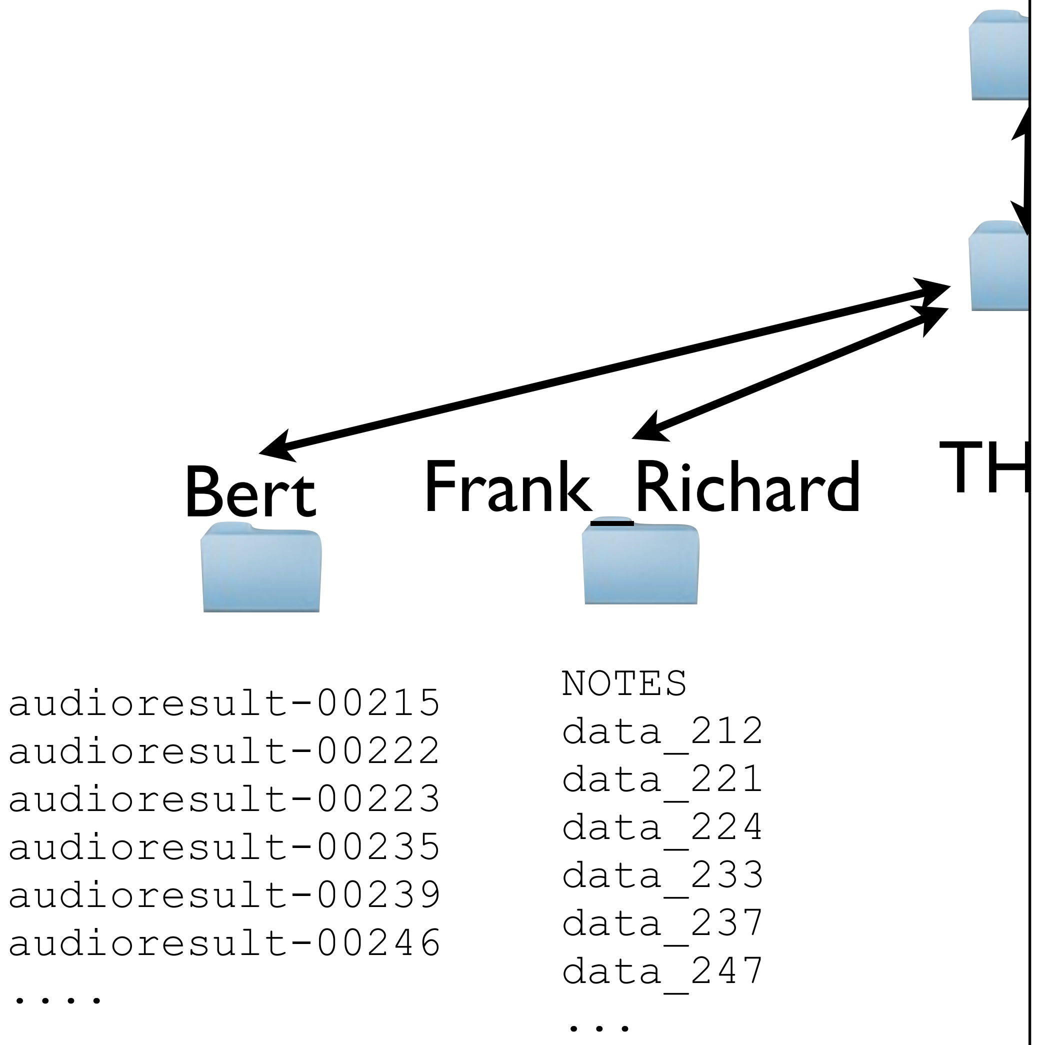
CLI - Hard/Easy

- Hard to *learn/ discover*
- Easy to *use* for big tasks productively.



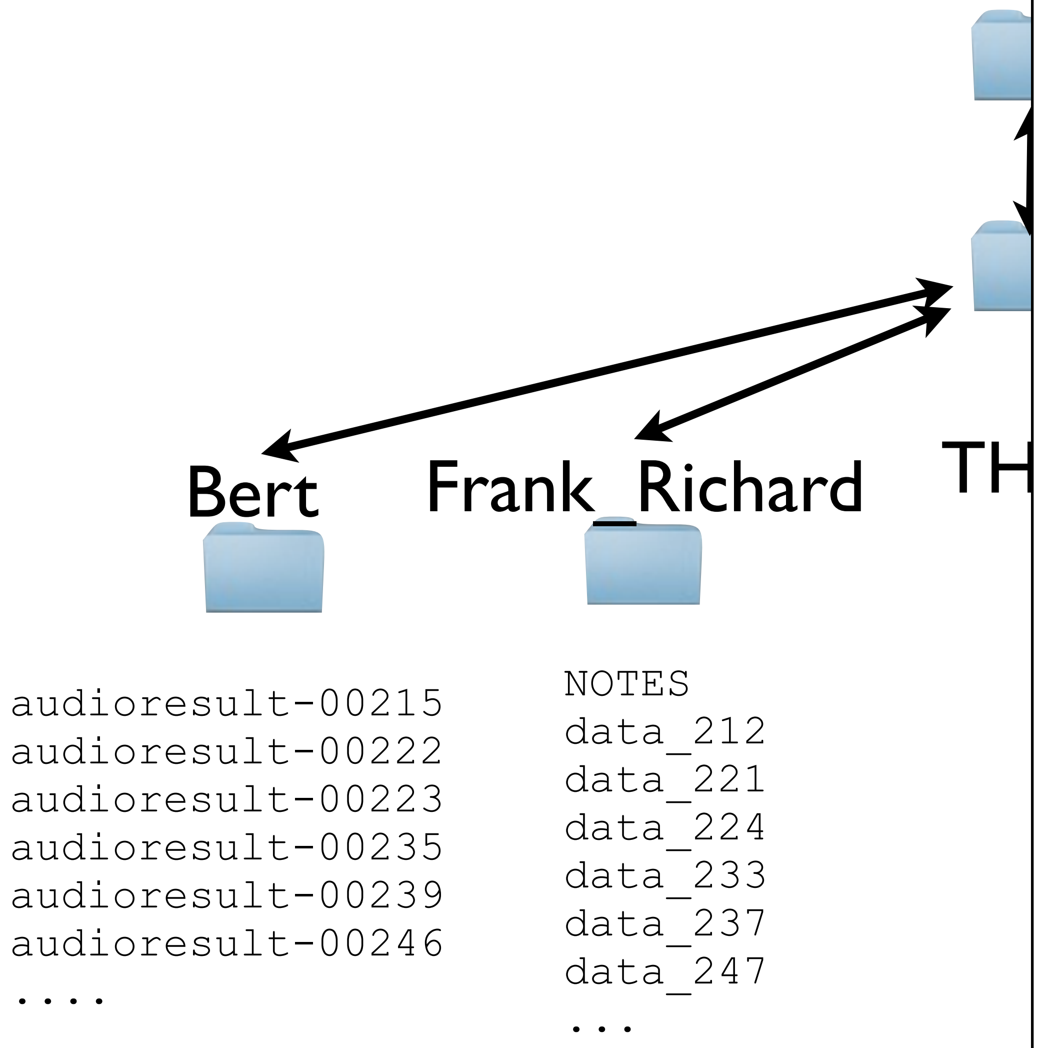
```
ljdursi|segfault.local>
```

GUI vs CLI



- With GUI, we could (painfully) do this one file at a time.
- But in two months, when there's another 350 files, have to do it exactly again.
- No further ahead.

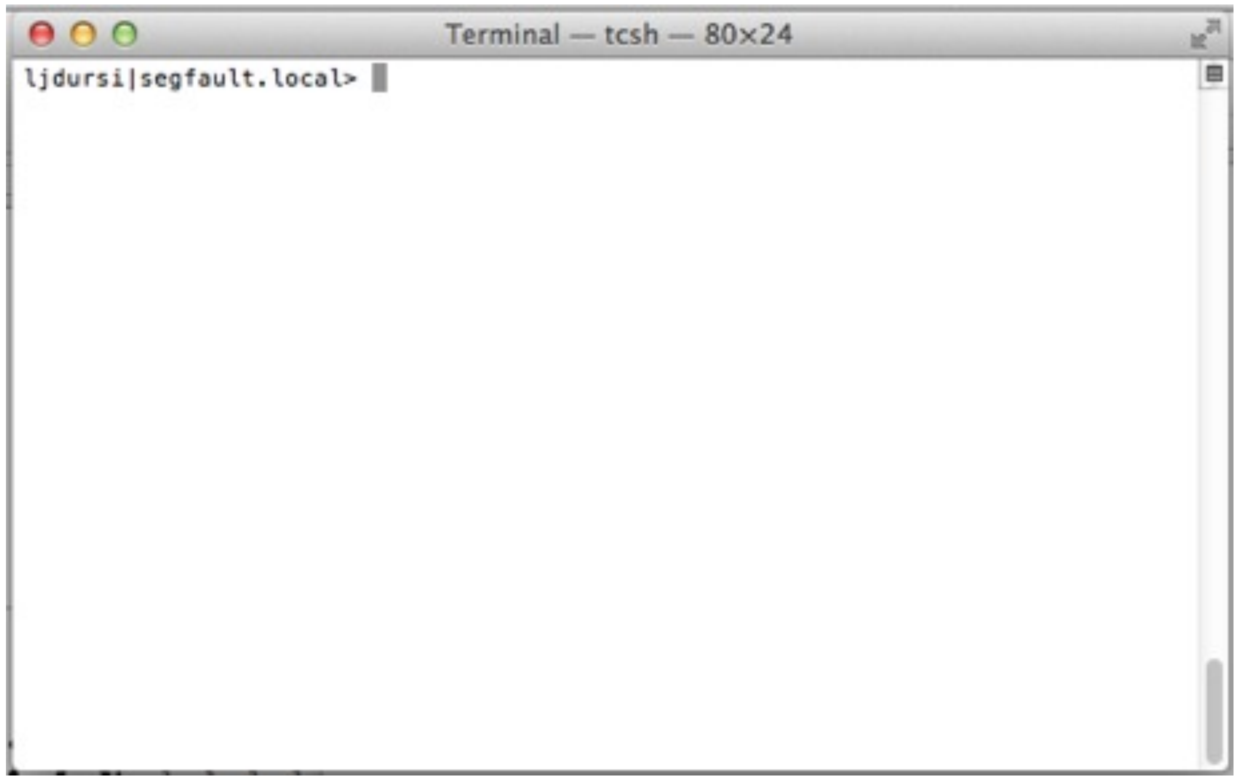
GUI vs CLI



- We’re going to spend a lot of time learning the shell today, towards doing this.
- But doing it the **next** time will be much faster.

Open a Terminal

- Mac: Applications/ Utilities/Terminal. (May as well drag this to the dock)
- Windows: Click on the cygwin icon.
- Linux: Various.

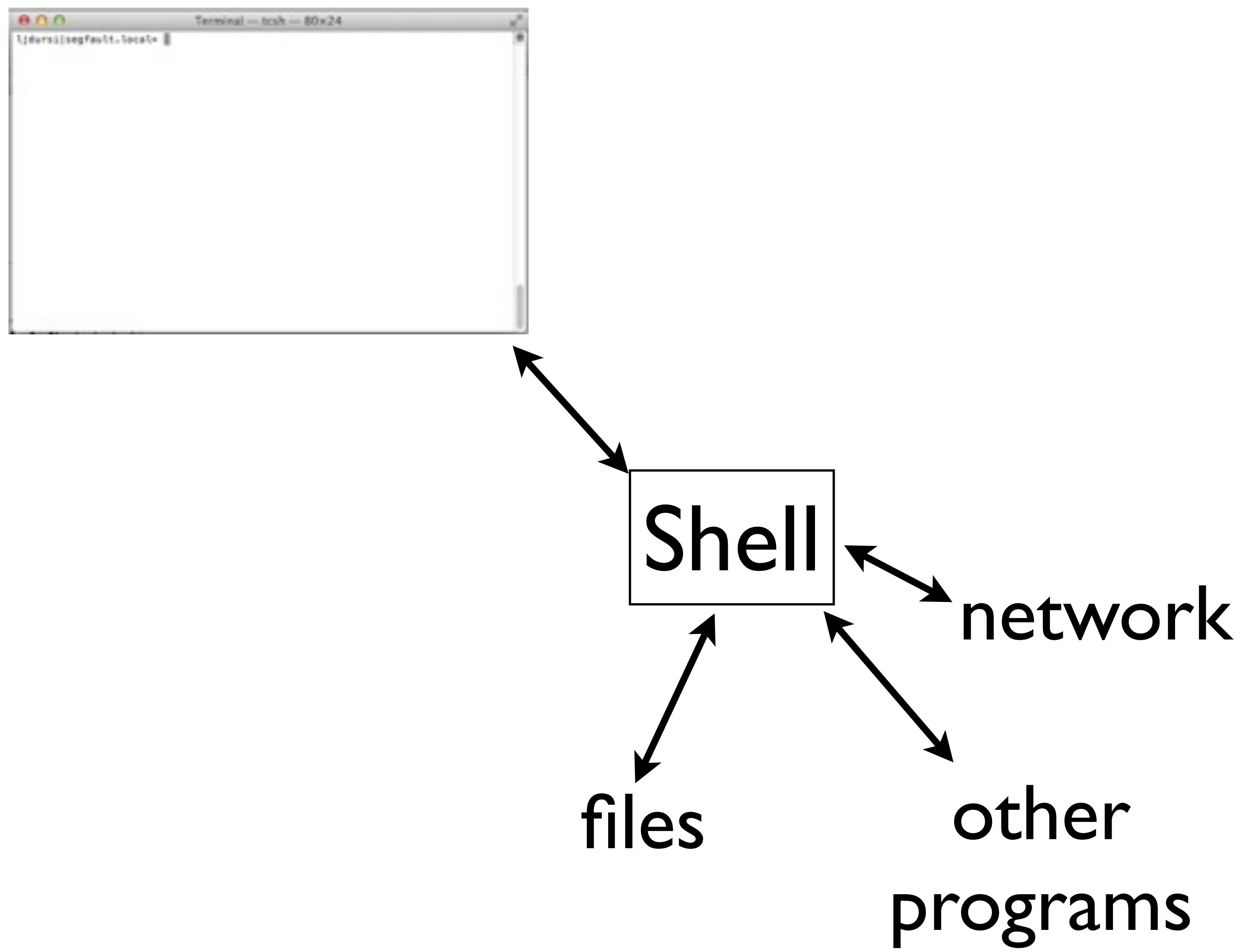


```
ljdursi|segfault.local> 
```

SciNet

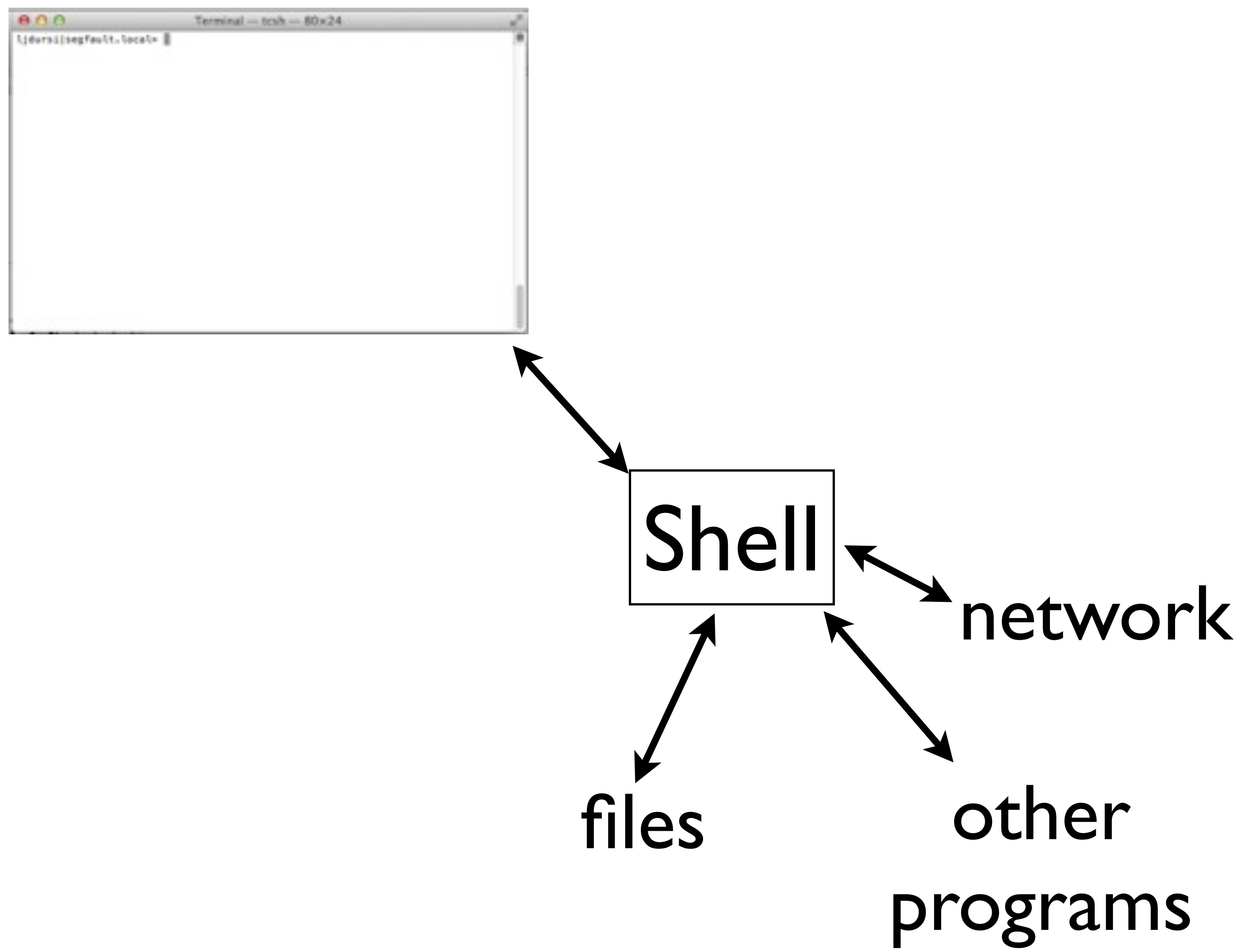
Terminal launches a shell

- When you use a terminal, you're interacting with the shell
- A program provides access to files, network, other programs.



Terminal launches a shell

- You type in commands
- Shell interprets them
- Performs actions on its own, or (more often) launches other programs
- Like ipython



```
ljdursi|segfault.local>
```

“The” shell

- The shell most commonly used is bash (Bourne-Again SHell).
- There are others; mostly the same but some syntax is different.
- Type `hello=”world”` (no spaces).
- If you get an error about no command you’re probably running tcsh. Type “bash” to start a bash shell and try again.

Basics - echo

- Let’s start by having the shell greet you:

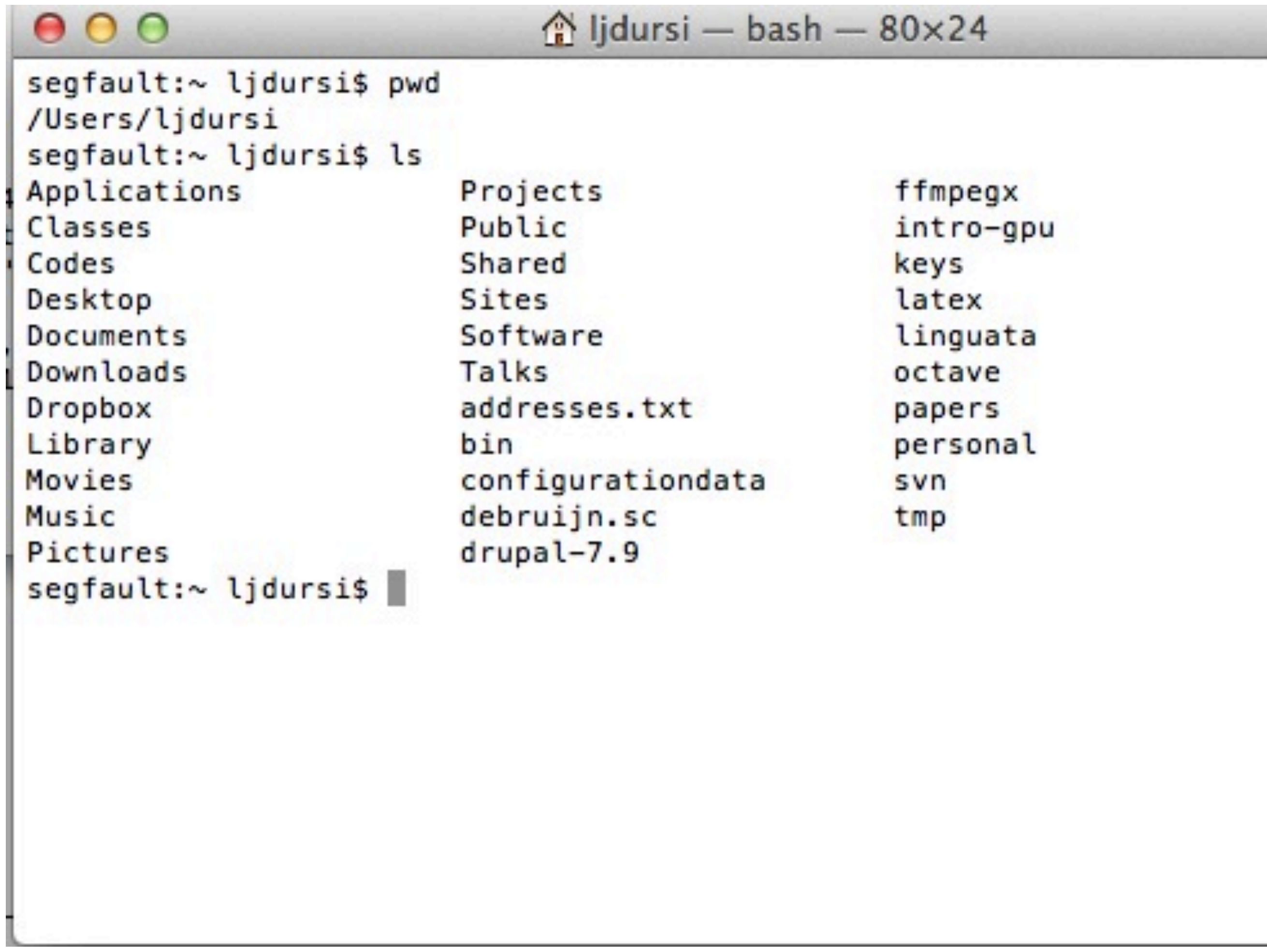
```
segfault:~ ljdursi$ hello=”world”

segfault:~ ljdursi$ echo Hello, world
Hello, world

segfault:~ ljdursi$ echo Hello, $hello
Hello, world
```

Basics - File system

- Now let’s learn how to start moving around amongst our files and directories.
- This is easy to do in a GUI (click on folders), harder here, but you get very fast at it in the shell...

Basics - File system

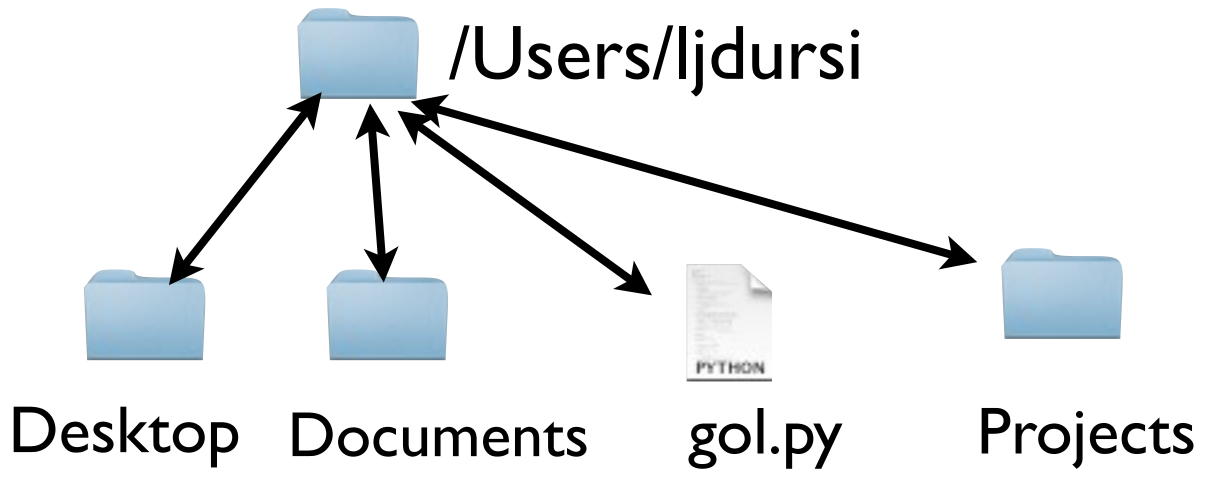
- Let's start poking around.
- Type `pwd`. Prints current "working" directory - where you are in the file structure.
- Type `ls` - that will list the files in that directory



```
segfault:~ ljdursi$ pwd
/Users/ljdursi
segfault:~ ljdursi$ ls
Applications            Projects                ffmpegx
Classes                 Public                  intro-gpu
Codes                   Shared                  keys
Desktop                 Sites                   latex
Documents               Software                linguata
Downloads               Talks                   octave
Dropbox                 addresses.txt           papers
Library                 bin                     personal
Movies                  configurationdata       svn
Music                   debruijn.sc             tmp
Pictures                drupal-7.9
segfault:~ ljdursi$ 
```

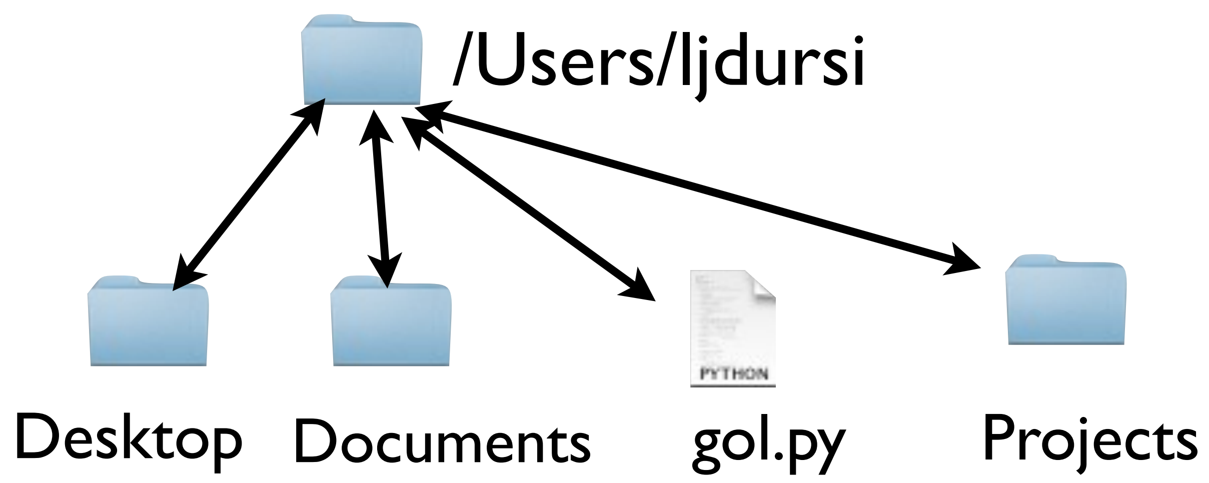
Directories = folders

- Often called folders because of how they're represented in GUIs
- Directories are listings of files - can contain files or other directories



Start at Home

- When you launch a shell, it starts in your home directory
- /Users/[username] or /home/[username] or something
- Top directory of all your stuff



File types

- Would like to know which entries are directories, which are plain files

- ls -F : labels directories with '/', executables with '*', etc.

```
segfault:~ ljdursi$ ls -F
Applications/            addresses.txt
Classes/                 bin/
Codes/                   configurationdata/
Desktop/                 debruijn.sc
Documents/               drupal-7.9/
Downloads/               gol.py*
...
```

Changing Directories: cd

- Choose one of the directories in your home directory and type `cd [dir]`
- And then `ls -F`
- Listing of contents of new directory
- `cd` without arguments will return to home dir

```
segfault:~ ljdursi$ cd Desktop

segfault:Desktop ljdursi$ ls -F
40TB.key            cubicAdvection.png
Dursi-HPC.pages     cubicAdvection.py
Dursi-HPC.pdf       cubicHeat.png
IntroGPGPU.key      cubicHeat.py
LFBP/               dance.pages
...

segfault:~ ljdursi$ cd

segfault:~ ljdursi$ pwd
/Users/ljdursi
```

Commands so far

- A couple things to observe:
- Commands designed to be fast/easy to *use*.
- Pretty cryptic to *learn*.

`echo`	**Prints output**
`pwd`	**Print current directory**
`cd [directory]`	**Change directory**
`cd`	**Change directory to home**
`ls`	**Directory LiSting**
`ls -F`	**LiSting with Filetypes**

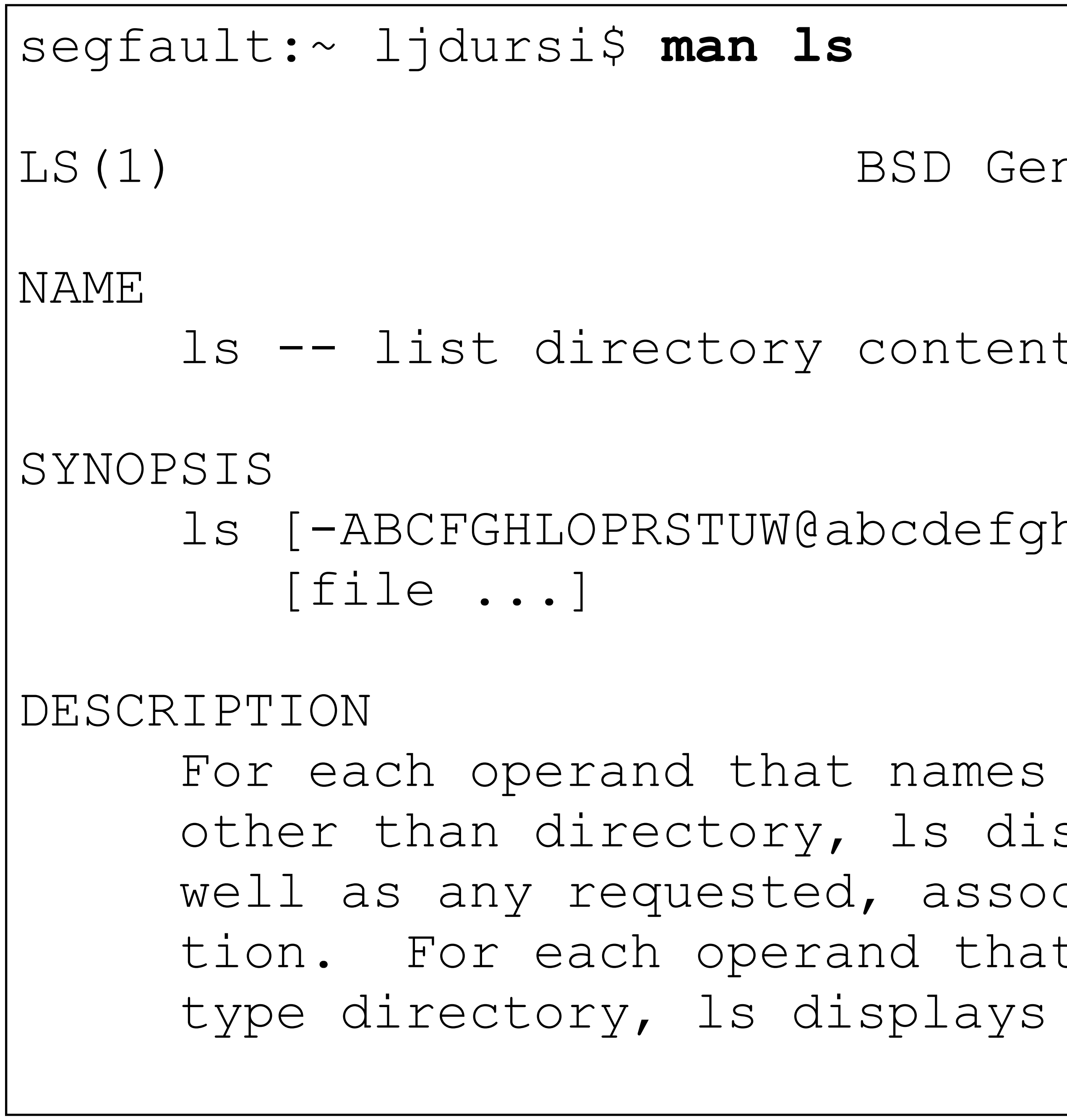
Options: -something

- ls (and it turns out lots of others) have options
- eg, -F
- or --help
- How do we know what the options are?

`echo`	**Prints output**
`pwd`	**Print current directory**
`cd [directory]`	**Change directory**
`cd`	**Change directory to home**
`ls`	**Directory LiSting**
`ls -F`	**LiSting with Filetypes**

Manual: man pages

- Most programs have a manual page describing its use and the options.
- Good for finding out more about a command you already use;
- Less good for learning what a command does.



```
segfault:~ ljdursi$ man ls

LS(1)                        BSD Ger

NAME
     ls -- list directory content

SYNOPSIS
     ls [-ABCFGHLOPRSTUW@abcdefgh
        [file ...]

DESCRIPTION
     For each operand that names
     other than directory, ls dis
     well as any requested, assoc
     tion.  For each operand that
     type directory, ls displays
```

Manual: man pages

- Many programs have gazillions of options.
- No human being who has ever lived has known all the options to 'ls' at same time.
- Over time you find a few that you find useful for your favourite commands.

```
segfault:~ ljdursi$ man ls

LS(1)                         BSD Ger

NAME
     ls -- list directory content

SYNOPSIS
     ls [-ABCFGHLOPRSTUW@abcdefgh
        [file ...]

DESCRIPTION
     For each operand that names
     other than directory, ls dis
     well as any requested, assoc
     tion.  For each operand that
     type directory, ls displays
```

Using ls on other directories

```
segfault:~ ljdursi$ pwd
/Users/ljdursi/Desktop

segfault:Desktop ljdursi$ ls -F /Users/ljdursi
Applications/     addresses.txt
Classes/          bin/
Codes/            configurationdata/
Desktop/          debruijn.sc
Documents/        drupal-7.9/
Downloads/        gol.py*
...
```

- If you give ls an argument, it will do the listing of that directory...

Using ls on other directories

```
segfault:~ ljdursi$ pwd
/Users/ljdursi/Desktop

segfault:Desktop ljdursi$ ls -F /Users/ljdursi/codes
FLASH2.5/              athena3.1/
Gadget-2.0.3-SP.tgz vine1.01.tar.gz

segfault:Desktop ljdursi$
```

- If you give ls an argument, it will do the listing of that directory...

Using ls on other directories

```
segfault:~ ljdursi$ pwd
/Users/ljdursi/Desktop

segfault:Desktop ljdursi$ ls *.py
cubicAdvection.py  gol.py
cubicHeat.py

segfault:Desktop ljdursi$ ls /Users/ljdursi/*.py
/Users/ljdursi/gol.py
```

- ...or those files.

The shell interprets arguments

- The shell takes my line "`ls *.py`"
- It looks for all files that are of the form [anything].py,
- and passes them as arguments to the ls command (`/bin/ls`).

```
segfault:~ ljdursi$ pwd
/Users/ljdursi/Desktop

segfault:Desktop ljdursi$ ls *.py
cubicAdvection.py   gol.py
cubicHeat.py

segfault:Desktop ljdursi$ ls /Users/ljdursi/*.py
/Users/ljdursi/gol.py
```

The shell interprets arguments

- `echo *.py` works just as well;
- Shell generates list of .py files, puts them as arguments to `echo`
- `echo` echos them to screen.

```
segfault:~ ljdursi$ pwd
/Users/ljdursi/Desktop

segfault:Desktop ljdursi$ ls *.py
cubicAdvection.py   gol.py
cubicHeat.py

segfault:Desktop ljdursi$ ls /Users/ljdursi/*.py
/Users/ljdursi/gol.py
```

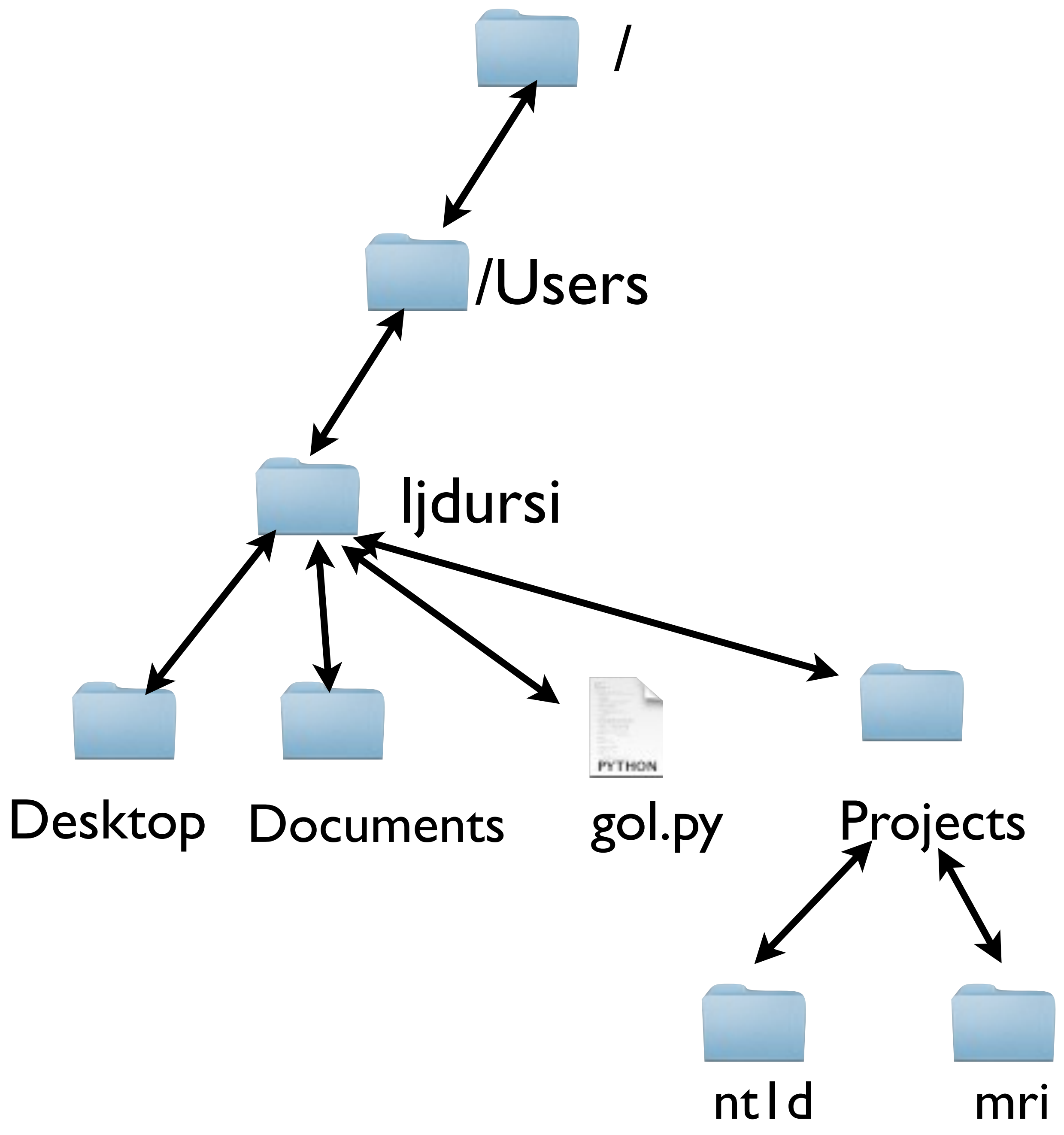
The shell interprets arguments

- If the argument is a directory (or a file name), there's no processing to be done
- Passes it to 'ls'

```
segfault:~ ljdursi$ pwd
/Users/ljdursi/Desktop

segfault:Desktop ljdursi$ ls -F /Users/ljdursi/codes
FLASH2.5/            athena3.1/
Gadget-2.0.3-SP.tgz  vine1.01.tar.gz
```

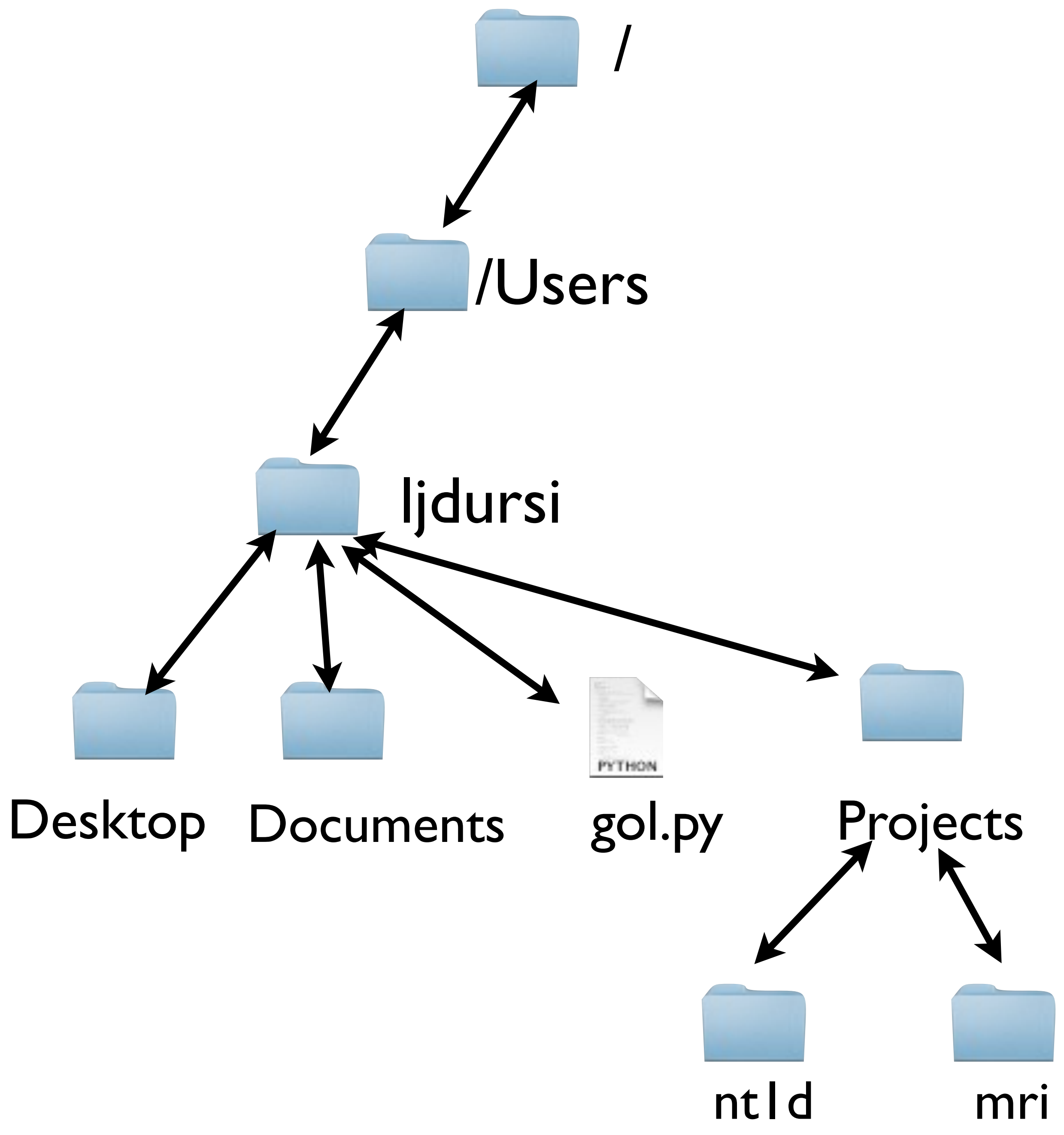
Directories in the shell

- A couple things to observe:
- Directories in bash separated by "/". (Windows - by "\").
- The top directory is "/"; under that, Users, under that, ljdursi, etc.



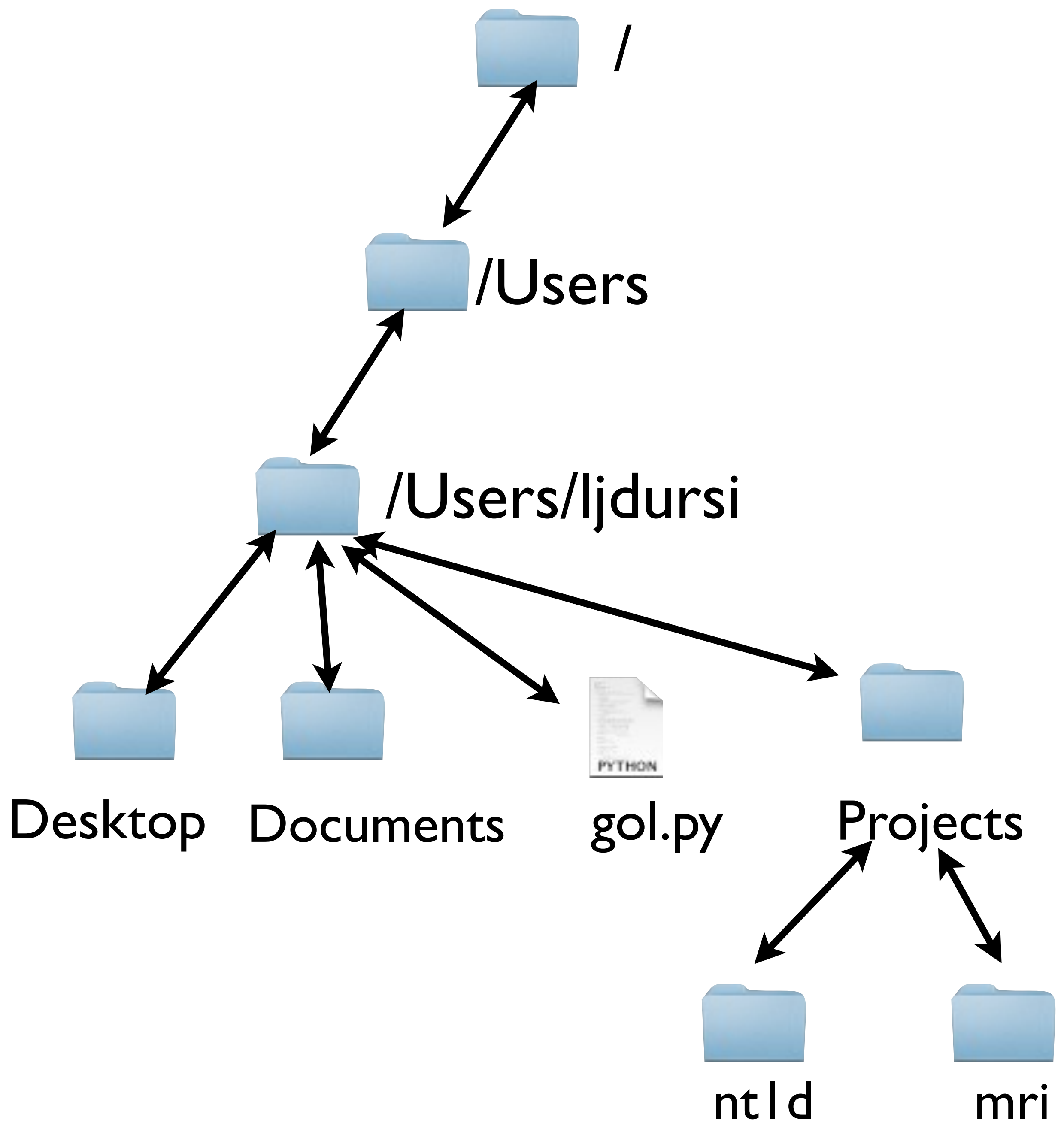
Directories in the shell

- Can always specify a file by it’s full “name”, eg `/Users/ljdursi/Projects/mri/README.txt`
- If you are in that directory, can just say `README.txt`



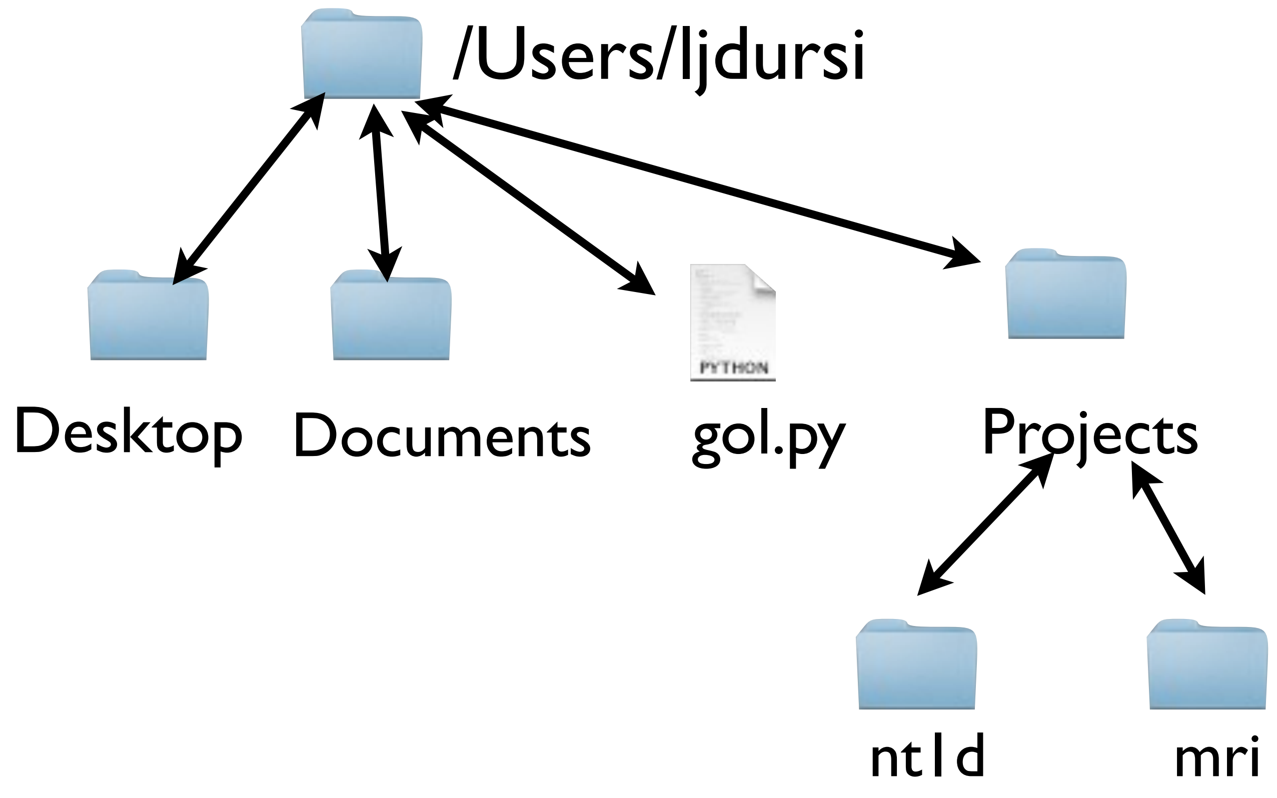
Directories in the shell

- But can also specify relative paths; if you're in Projects, `mri/README.txt` is enough.



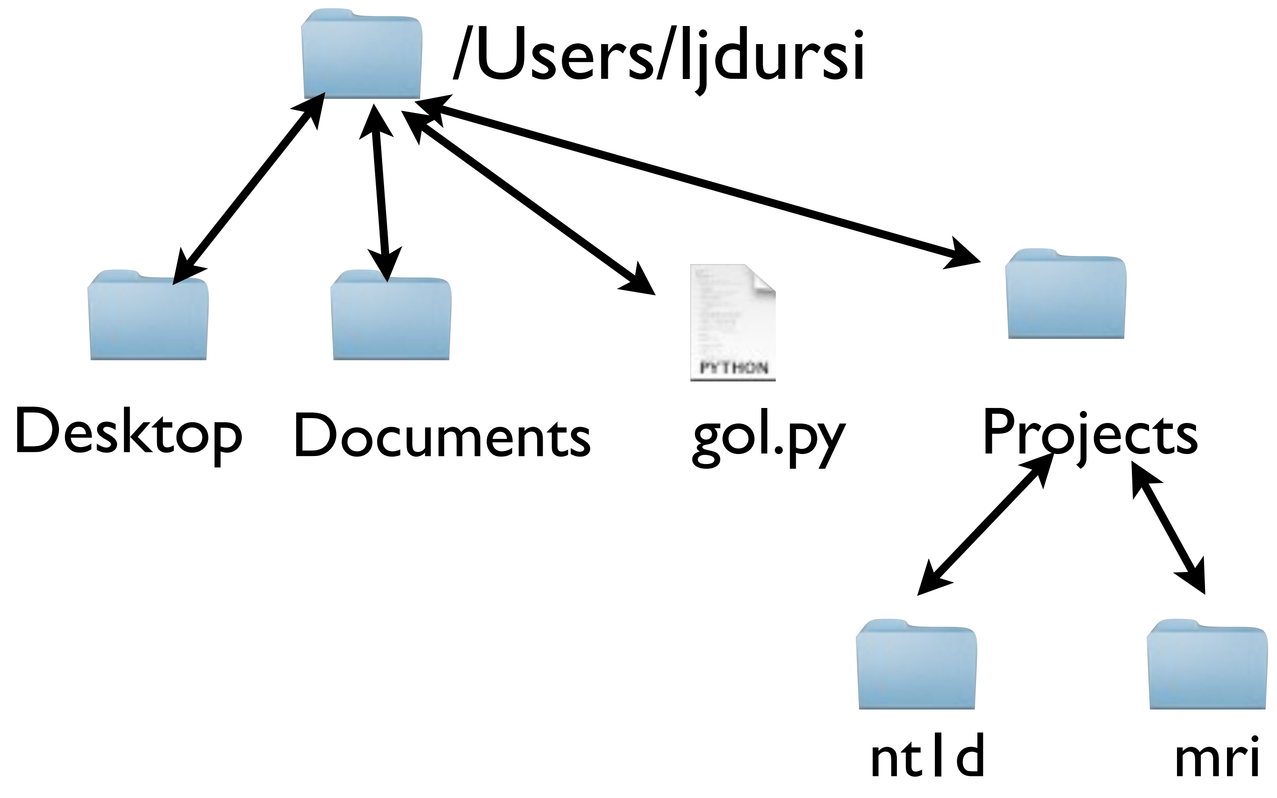
Shortcuts for moving around directories:

- A shortcut for "one directory up" is `..`
- If I'm in Desktop, `ls ..` does an ls of home directory;
- and `ls ../Projects` looks in my Projects directory.



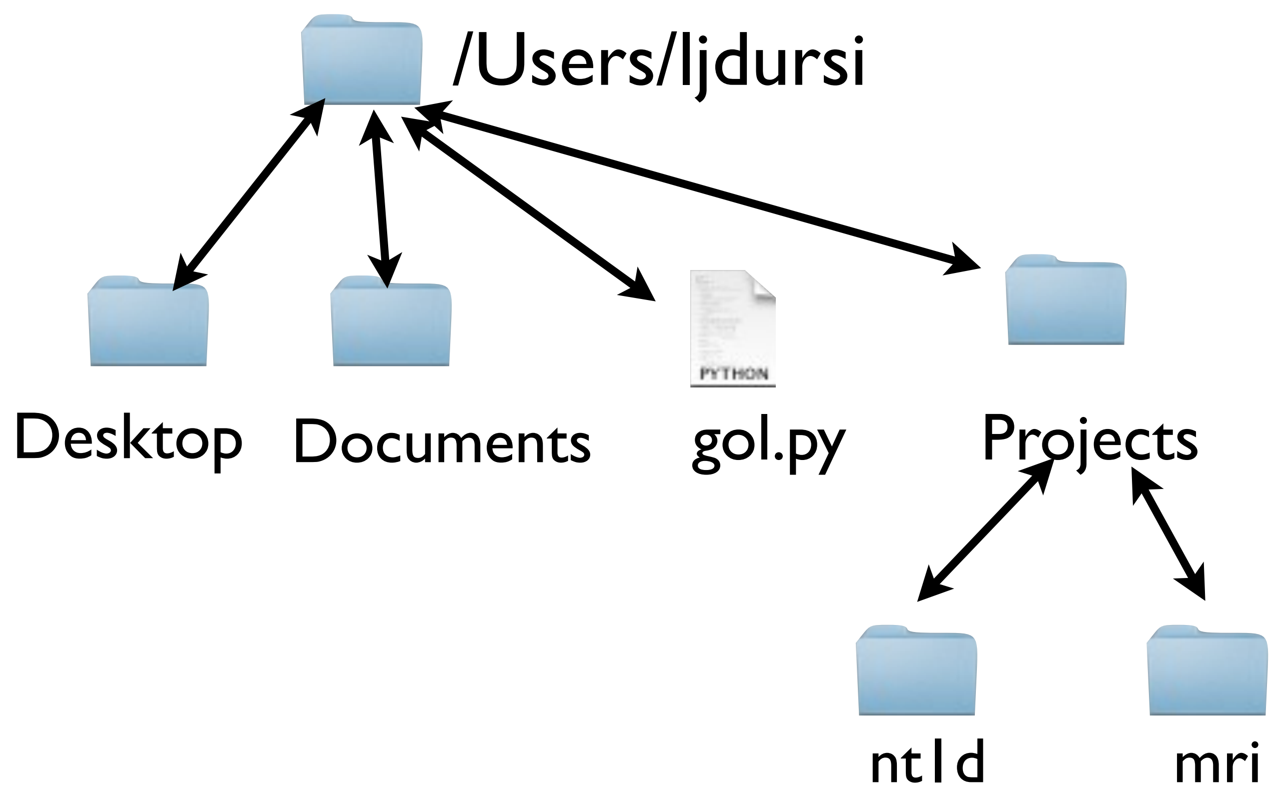
Shortcuts for moving around directories:

- One dot means the current directory: .
- If I'm in my home directory, `ls ./gol.py` just lists the gol.py there.



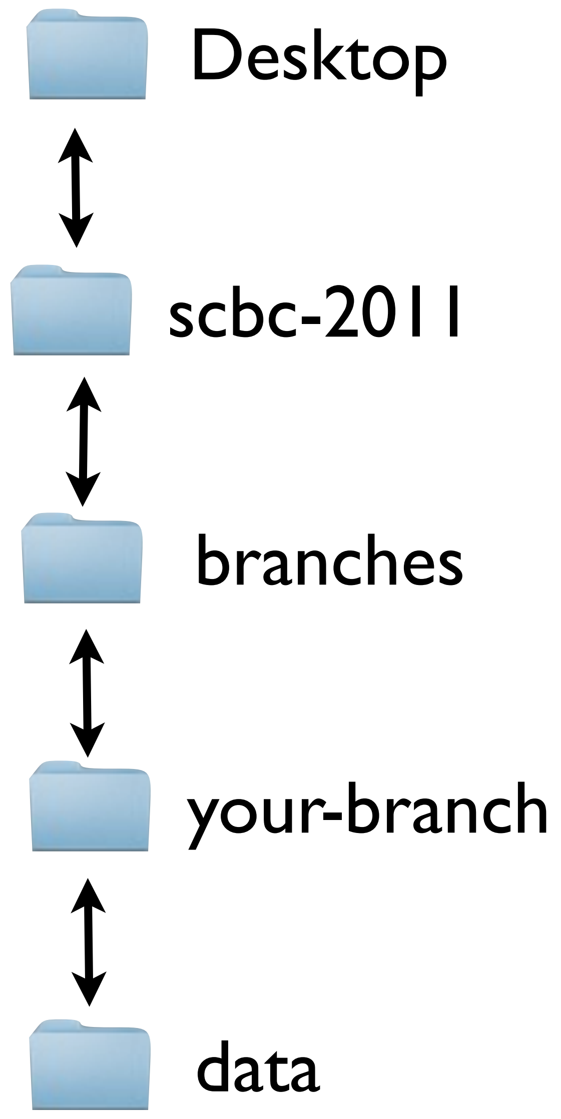
Shortcuts for moving around directories:

- A shortcut for your home directory is `~`
- Wherever I am, `ls ~` does a listing of `/Users/ljdursi`
- `ls ~/Desktop` does a listing of `/Users/ljdursi/Desktop`.



Looking at files

- Let's go into the data directory from svn:



Your svn location

Your branch

```
segfault:~ ljdursi$ cd ~/Desktop/scbc-2011/branches/ljdursi-branch/data

segfault:data ljdursi$ ls -F
data/          ex_data.txt       generate_data.py

segfault:data ljdursi$ cd data

segfault:data ljdursi$ ls -F
Bert/              Lawrence/      alexander/ jamesm/
Frank_Richard/     THOMAS/         gerdal/

segfault:data ljdursi$ cd Bert

segfault:Bert ljdursi$ ls
audioresult-00215 audioresult-00332 audioresult-00451
audioresult-00222 audioresult-00350 audioresult-00453
audioresult-00223 audioresult-00353 audioresult-00460
audioresult-00235 audioresult-00355 audioresult-00466
...

segfault:Bert ljdursi$
```

Looking at a file

```
segfault:~ ljdursi$ file audioresult-00215
audioresult-00215: ASCII text
```

`echo`	**Prints output**
`pwd`	**Print current directory**
`cd [directory]`	**Change directory**
`cd`	**Change directory to home**
`ls`	**Directory LiSting**
`ls -F`	**LiSting with Filetypes**
`man [cmd]`	**MANual page for [cmd]**
`file [filename]`	**What is in [filename]?**

Looking at a file

```
segfault:Bert ljdursi$ file audioresult-00215
audioresult-00215: ASCII text

segfault:Bert ljdursi$ file au<TAB>
segfault:Bert ljdursi$ file audioresult-00
```

Tab completion!
If you hit <TAB> when typing a filename, shell will complete what you're typing (as much as possible)

Looking at a file

```
segfault:Bert ljdursi$ file audioresult-00215
audioresult-00215: ASCII text

segfault:Bert ljdursi$ file au<TAB>
segfault:Bert ljdursi$ file audioresult-00
```

Other handy tip -
Up arrow lets you preview previous commands; can edit and/ or press <Return>

Looking at a file

```
segfault:Bert ljdursi$ file audioresult-00215
audioresult-00215: ASCII text

segfault:Bert ljdursi$ more audioresult-00215
#
Reported: Sun Jun 26 14:56:54 2011
Subject: beyonceLennon177
Year/month of birth: 1993/09
Sex: N
CI type: 20
Volume: 8
Range: 5
Discrimination: 7

segfault:Bert ljdursi$
```

Looking at a file

```
segfault:Bert ljdursi$ file audioresult-00215
audioresult-00215: ASCII text

segfault:Bert ljdursi$ more audioresult-00215
#
Reported: Sun Jun 26 14:56:54 2011
Subject: beyonceLennon177
Year/month of birth: 1993/0
Sex: N
CI type: 20
Volume: 8
Range: 5
Discrimination: 7

segfault:Bert ljdursi$
```

`echo`	**Prints output**
`pwd`	**Print current directory**
`cd [directory]`	**Change directory**
`cd`	**Change directory to home**
`ls`	**Directory LiSting**
`ls -F`	**LiSting with Filetypes**
`man [cmd]`	**MANual page for [cmd]**
`file [filename]`	**What is in [filename]?**
`more [filename]`	**Prints out filename(s) by page**
`cat [filename]`	**Dumps out filename(s)**

More vs Cat:

- try `more au*`
- and then `cat au*`
- What's the difference?

cat’ing files together

- Dumping all the files together is how ‘cat’ got its name - short for concatenate.
- Try cat’ing all the files together into a new file:

```
segfault:~ ljdursi$ cat au* > all-results
segfault:~ ljdursi$ more all-results
```

Redirection

- `[cmd] > [filename]` takes what would have gone to the screen, creates a new file `[filename]`, and redirects output to that file.
- Blows away previous contents of file if it had existed.

Redirection

- `[cmd] >> [filename]` *appends* to `[filename]` if it exists.
- `[cmd] < [filename]` - program's input comes from file, as if you were typing.

cat - echos input

- If cat isn't given filenames, it just dumps its input to the screen.

```
segfault:Bert ljdursi$ cat
hello
hello
there
there
^D
```

mv, cp

- We’ve created our first file from the shell!
- We can make copies, or move the file around:

mv, cp

```
segfault:Bert ljdursi$ cp all-results all-results-2
segfault:Bert ljdursi$ ls all*
???

segfault:Bert ljdursi$ mv all-results all-results-3
segfault:Bert ljdursi$ ls all*
???

segfault:Bert ljdursi$ mv all-results3 ..
???
```

mv, cp - move, copy

```
segfault:Bert ljdursi$ cp all-results all-results-2
segfault:Bert ljdursi$ ls all*
all-results all-results-2

segfault:Bert ljdursi$ mv all-results all-results-3
segfault:Bert ljdursi$ ls all*
all-results all-results-3

segfault:Bert ljdursi$ mv all-results3 ..
segfault:Bert ljdursi$ ls all*
all-results

segfault:Bert ljdursi$ ls ..
Bert           Lawrence  alexander      gerdal
Frank_Richard  THOMAS    all-results-3  jamesm
```

rm - remove

- Deletes (ReMoves) file.
- Does *not* move it to trash; deletes it.
- No safety net!
- (But - if the file is in version control, can recover it).

```
segfault:Bert ljdursi$ ls ..
Bert               Lawrence  alexander        gerdal
Frank_Richard      THOMAS    all-results-3    jamesm
```

rm

```
segfault:Bert ljdursi$ ls -F ..
Bert/            Lawrence/ alexander/      gerdal/
Frank_Richard/   THOMAS/   all-results-3   jamesm/

segfault:Bert ljdursi$ rm ../all-results-3

segfault:Bert ljdursi$ ls -F ..
Bert/            Lawrence/ alexander/      gerdal/
Frank_Richard/   THOMAS/   jamesm/
```

mkdir, rmdir

- To create and delete directories, use mkdir and rmdir.
- Uncharacteristically, rmdir protects you - you can't delete a directory with files in it
- Have to delete them first

mkdir, rmdir

```
segfault:Bert ljdursi$ mkdir foo

segfault:Bert ljdursi$ ls foo

segfault:Bert ljdursi$ cp all-results foo

segfault:Bert ljdursi$ ls foo
all-results

segfault:Bert ljdursi$ rmdir foo
rmdir: foo: Directory not empty

segfault:Bert ljdursi$ rm foo/all-results

segfault:Bert ljdursi$ rmdir foo
```

wc - word count of text files

- `wc [filename]` prints the lines, words, and characters (non-spaces) in a text file
- `wc -l`, `wc -w`, and `wc -c` print just the # of lines, words, and characters of the file
- try `wc all-results` (tab completion will work after the 'al')

wc

- We've just `wc`'ed a `cat`'ed file
- Should have same as totals of all files
- Let's try that: `wc au*`

wc

```
segfault:Bert ljdursi$ wc all-results
     423     1124     6916 all-results

segfault:Bert ljdursi$ wc au*
...
       9       24      147 audioresult-00521
       9       24      146 audioresult-00532
       9       24      147 audioresult-00534
       9       24      151 audioresult-00535
       9       24      148 audioresult-00557
     423     1124     6916 total
```

Dealing with too much output

- `wc au*` printed out results for each file, and total - handy.
- But it provided too much output; couldn’t see it all.
- How are we going to fix that (using just what we know so far)?

wc, more

```
segfault:Bert ljdursi$ wc all-results
     423     1124     6916 all-results

segfault:Bert ljdursi$ wc au* > all-wcs

segfault:Bert ljdursi$ more all-wcs
```

head, tail

```
segfault:Bert ljdursi$ head all-wcs
???

segfault:Bert ljdursi$ tail all-wcs
???
```

`head, tail` prints start, end of file

- Useful options to head/tail:
  - `-n [number]` : only first/last n lines. (default = 10)

Pipeline of commands

- This idea of chaining commands together - the output from one becomes the input of another - is part of what makes the shell (and programming generally) so powerful.

Pipeline of commands

- So far we've done

```
segfault:Bert ljdursi$ wc au* > all-wcs

segfault:Bert ljdursi$ more all-wcs
```

- Creates a temporary file we don't really care about; we just want to page through all the wc results.

Pipeline of commands

- Interesting (honest, you'll see) fact - like cat, if more isn't given a filename, it also reads from input:
- So this would also work:

```
segfault:Bert ljdursi$ wc au* > all-wcs

segfault:Bert ljdursi$ more < all-wcs
```

Pipeline of commands

```
segfault:Bert ljdursi$ wc au* > all-wcs

segfault:Bert ljdursi$ more < all-wcs
```

- This combination of actions - output of one command goes straight into another - so common that shell has special facilities for this:

```
segfault:Bert ljdursi$ wc au* | more
```

Pipeline of commands

- Allows you to chain together small pieces into a very powerful analysis pipeline.
- Let’s look at another example:

`sort` sorts lines in a file

- Let’s create a short file and have `sort` sort it.
- Can write file in editor, but let’s use our new cat-and-redirection skills:

```
segfault:Bert ljdursi$ cat > toBeSorted
Ernie
Bert
Oscar
Big Bird
segfault:Bert ljdursi$
```

`sort` sorts lines in a file

```
segfault:Bert ljdursi$ cat toBeSorted
Ernie
Bert
Oscar
Big Bird

segfault:Bert ljdursi$ sort toBeSorted
Bert
Big Bird
Ernie
Oscar
```

`sort` sorts lines in a file

- Useful options to sort:
  - `-n` : sort in numerical order (not lexicographic; eg, 101 < 30 without -n.)
  - `-k [number]` : sort by the k'th column.
  - `-r` : reverses order (decreasing, not increasing)

`sort` the data files by size (in characters)

```
segfault:Bert ljdursi$ sort -n -k 3 all-wcs
...
       9      24     151 audioresult-00535
       9      24     152 audioresult-00286
       9      24     152 audioresult-00353
     423    1124    6916 total

segfault:Bert ljdursi$ sort -n -k 3 -r all-wcs
..
       9      24     144 audioresult-00239
       9      23     144 audioresult-00453
       9      24     143 audioresult-00393
       9      24     142 audioresult-00493
```

`sort` the data files by size (in characters)

```
segfault:Bert ljdursi$ wc au* | sort -n -k 3
...
       9      24      151 audioresult-00535
       9      24      152 audioresult-00286
       9      24      152 audioresult-00353
     423    1124     6916 total

segfault:Bert ljdursi$ wc au* | sort -n -k 3  | more
??
```

Pop quiz!

Modify this to print only smallest, then only largest, data file.

```
segfault:Bert ljdursi$ wc au* | sort -n -k 3
...
       9      24     151 audioresult-00535
       9      24     152 audioresult-00286
       9      24     152 audioresult-00353
     423    1124    6916 total

segfault:Bert ljdursi$ wc au* | sort -n -k 3  | more
??
```

Our first shell script

- So this is useful enough that we are going to write a script that contains this line.
- Will be a program that prints largest (say) data file in the directory.
- First, clean up:

```
segfault:Bert ljdursi$ rm all-wcs all-results toBeSorted
```

Our first shell script

- Create the following file, called “biggest”.
- More complex than toBeSorted: use an editor

```
#!/bin/bash
wc * | sort -n -k 3 | tail -2 | head -1
```

- Now run it with

```
segfault:Bert ljdursi$ source biggest
```

- what do you get?

Our first shell script

- To make this into a "real" program, we're going to tell the OS that this file is executable.
- Then the `#!/bin/bash` line will tell the OS to run this program with our shell, bash

```
segfault:Bert ljdursi$ chmod a+x biggest
segfault:Bert ljdursi$ ./biggest
```

Largest range - `grep`

- Largest number of characters in data file - probably not super important for our analysis.
- How about experiment with largest range?
- Data files all have line "Range: [Number]"

```
segfault:Bert ljdursi$ grep Range audioresult-00557
Range: 2
```

- grep outputs lines containing the first input string in all of the files given.

```
segfault:Bert ljdursi$ grep Range *
???
```

Pop Quiz

- Modify biggest to print out which experiment has the biggest Range.
- Quick tip - what column needs to be sorted?
- (And do we need the head/tail trick?)

Pop Quiz

- Modify biggest to print out which experiment has the biggest Range.
- Quick tip - what column needs to be sorted?
- (And do we need the head/tail trick?)

```
segfault:Bert ljdursi$ more biggestRange
#!/bin/bash
grep Range * | sort -n -k 2 | tail -1
```

Arguments in bash scripts

- We'd like to use this for each directory, but we don't want one copy in each directory.
- Let's move it up one level in directory, and modify it so it would work on any directory's files

```
segfault:data ljdursi$ more biggestRange
#!/bin/bash
grep Range $1/* | sort -n -k 2 | tail -1
```

Arguments in bash scripts

- When you run a command in the shell, it’s name is put in argument 0 ($0)
- Any other arguments are $1, $2...

```
segfault:data ljdursi$ more biggestRange
#!/bin/bash
grep Range ${1}/* | sort -n -k 2 | tail -1
```

Arguments in bash scripts

```
segfault:data ljdursi$ ./biggestRange Bert
Bert/audioresult-00384:Range: 10

segfault:data ljdursi$ ./biggestRange THOMAS
THOMAS/0336:Range: 10
```

For loops in bash

- Bash has for loops much like python does.
- We can use this to run our program on several directories:

For loops in bash

```
segfault:data ljdursi$ for dir in Bert gerdal jamesm
> do
> echo "The biggest range in directory " ${dir} " is:"
> ./biggestRange ${dir}
> done
The biggest range in directory  Bert  is:
Bert/audioresult-00384:Range: 10
The biggest range in directory  gerdal  is:
gerdal/Data0559:Range: 10
The biggest range in directory  jamesm  is:
jamesm/data_517.txt:Range: 10

segfault:data ljdursi$
```

find

- Wildcards are very powerful:
- From data/data directory, type: `ls */*00*`
- Finds files with ‘00’ in name in any subdirectory
- Similarly: `echo */*00*`
- or

```
for i in */*00* ; do echo ${i}; done
```

find

- But can only match if you know the path (how many levels of dirs down)
- And can only match by filename.
- `find` is a tool which lets you find files *anywhere* below a given directory, based on *arbitrary* criteria.

`find`: do the following

```
segfault:data ljdursi$ find . -print | more
```

directory to start

What to do to the file

`find`: can execute arbitrary commands

```
segfault:data ljdursi$ find . -exec echo {} \; | more
```

directory to start

What to do. {} gets filled in with filename; command ends with \;

`find`: can execute arbitrary commands

```
segfault:data ljdursi$ find . -exec echo {} \; | more
```

directory to start

What to do.
{} gets filled in with filename; command ends with \;

`find`: can choose files by type

```
segfault:data ljdursi$ find . -type f -print | more
```

directory to start

Only files (type f) get printed; directories are excluded

`find`: can choose files by type, name

```
segfault:data ljdursi$ find . -type f -name "*00*" -print | more
```

directory to start

Only files with 00 in their names; can chain together conditions

`find`: can choose files by contents

```
find . -type f -exec grep "Volume" {} \; -print | more
```

Only search files

If grep returns true (eg, contains "Volume"), then matches

Assignment

- Copy all of the data files from data/data/.. to a new directory, 'cleaneddata'.
- All data files must end in .txt
- Get rid of the NOTES files.
- Need to have this done before databases section (next)

Assignment

- Do it manually: that works.
- Try to find a solution which will work next time it needs to be done, too.
- Play with things on the command line..
- Many ways to do this!
- "Bonus points": put it in a script!

`echo`	**Prints output**
`pwd`	**Print current directory**
`cd [directory]`	**Change directory**
`cd`	**Change directory to home**
`ls`	**Directory LiSting**
`ls -F`	**LiSting with Filetypes**
`man [cmd]`	**MANual page for [cmd]**
`file [filename]`	**What is in [filename]?**
`more [filename]`	**Prints out filename(s) by page**
`cat [filename]`	**Dumps out filename(s)**
`wc [filename]`	**Line/word/char count of file**
`mv [src] [dest]`	**Move file**
`cp [src] [dest]`	**Copy file**
`rm [filename]`	**Delete file**
`head [filename]`	**First lines of file**
`tail [filename]`	**Last lines of file**
`sort [filename]`	**Sort lines of file**
`mkdir [filename]`	**Create directory**
`rmdir [filename]`	**Remove directory**
`grep`	**Searches input for text**
`for..do..done`	**for loops in bash**
`find`	**Searches for files**