First lecture

What we'll cover

- General course structure
- What is programming?
- Why use programming?
- The Unix environment.

General course structure

- First six weeks are the fundamentals of programming
- Second six are special topics
- We will have homework and in-class work
- There will be cheat sheets posted online

General course structure

- You must:
  - Have a laptop with a running Unix/Linux terminal
  - Have a good text editor
  - Do the homework and read the cheat sheets
  - Give us feedback and participate

General course structure

- You must:
  - Have a laptop with a running Unix/Linux terminal
  - Have a good text editor
  - Do the homework and read the cheat sheets
  - Give us feedback and participate
  - **Send us**
    - **A.) Description of what you envision using programming for in your research and,**
    - **B.) Sample dataset from your research. This should be a small version of some data you wish to manipulate using programming.**
    - **Please send this over the weekend**

Additional Tip

- Make friends or a study group

Additional Tip

- Make friends or a study group
- Make connections between concepts in class and actions you'd like to do with your data
- Seek out practice

Literature

- "Practical Computing for Biologists" (Haddock and Dunn)
  - Great for beginners
  - Mostly geared towards text editing

- O'Reilly Books
  - "Bioinformatics Programming in Python"
    - In Python 3, but good methodology

- http://greenteapress.com/thinkpython/thinkpython.pdf
  - Free!

Course Philosophy

- Establish a community
- Help people who don’t know where to start
- Provide resources to get you from 0 to not 0
- Most learning will be done on your own
  - No one learns without trial and error (and error)
  - This means you must be proactive and program a lot

What is a program?

How to get *inside* your computer

What is a program?

- A series of commands for your computer
- Written in a human-readable language
  - These are translated to binary by an assembler language that is in-between your script and the computer itself.

What is programming?

- One or more scripts saved in text files
  - Must be accessible to the operating system

- Creating software and scripts is the goal.
  - Your operating system itself is just a collection of scripts that interoperate

Why learn programming?

- Repeatability of tasks
  - Paper trail!
- Speed of execution
- Automation
  - No more click, type, click, type, click, type, snooze.

Languages!

- There are many, many computer programming languages.

Languages!

- There are many, many computer programming languages.
- Things to consider:
  - Speed versus readability
  - Documentation

Languages!

- There are many, many computer programming languages.
- Things to consider:
  - Speed versus readability
  - Documentation
- What are people in your field are using?
  - Stats - R
  - Dense computation - C & C++
  - Next-Gen - Perl & Python & Unix
  - Unix is often used as "glue" in workflows

Why Python?

- Readable
- Popular
- Well-documented

Why Unix?

- Almost universally used in computers, supercomputers and file systems
  - This is how most programmers manage and organize files

A taste of Unix

- Commands are small programs
  - Type name of command and hit "enter"
  - Unix searches for the program's text file, and executes it.

- They interact with files that are in a specified directory
  - If no directory specified, UNIX looks in your **working directory (the directory you are in)**

File systems

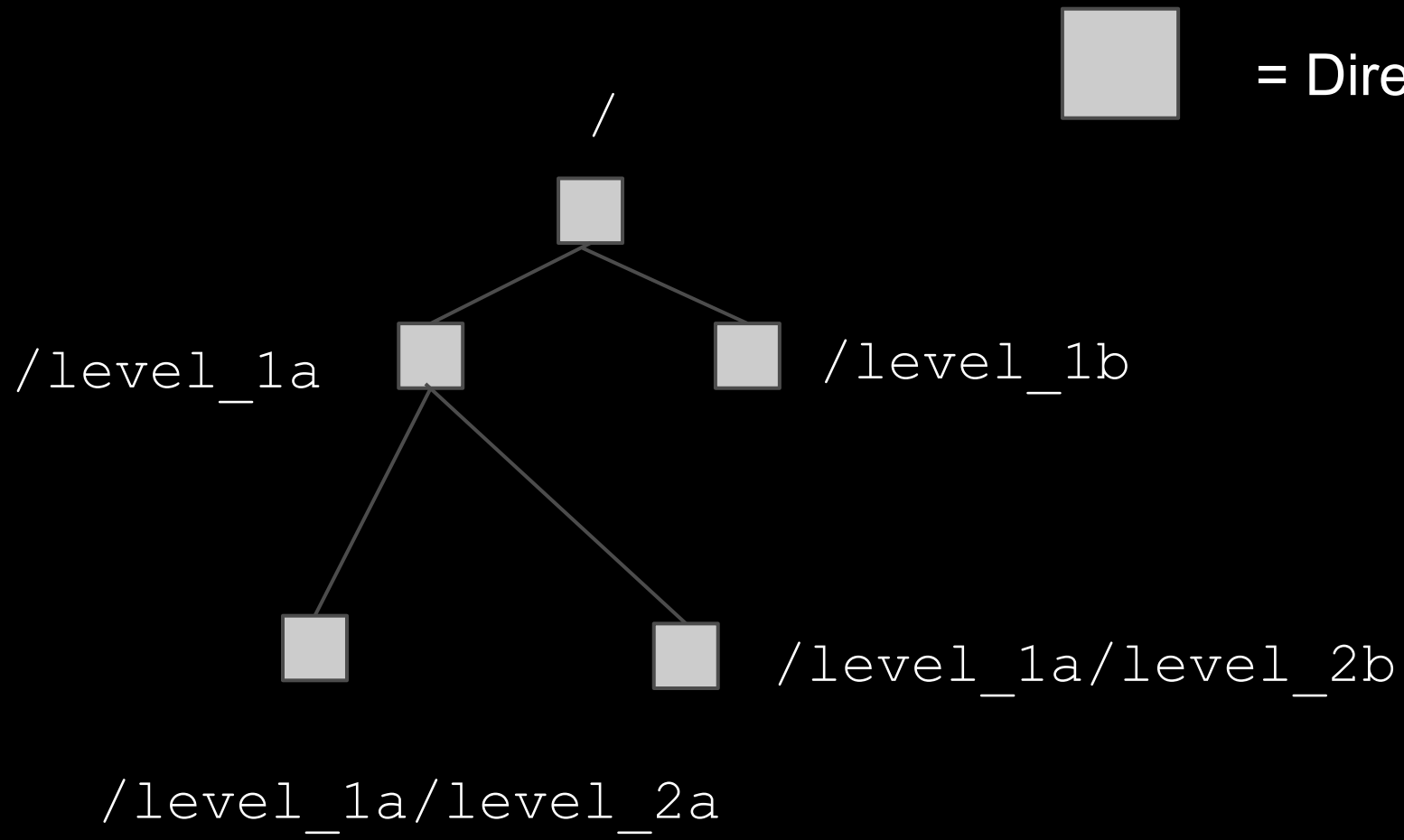
- Your computer contains a nested hierarchy of directories.
  - In Unix, no bird's eye view of directory structure
    - Wherever you go, there you are
  - No visible "folders."
  - Harder to keep track of where you and your programs are.
  - You need to learn a new way of doing things.

File path

- Every file has an address on your computer
  - This is the **filepath**

- If you are going to do an operation on a file, you'll need it's address

Slash notation



Symbols for a few important paths

Here	.
One level up	..
Home	~ or $HOME
Root (highest level)	/

.. and . -----> "relative paths"

~ or /usr/bin -----> "absolute paths"

Commands for Getting Around

1.) Common commands

2.) Working on files

3.) Stringing them together

A taste of Unix

- Interact with Unix via a "shell"
  - The shell channels information between the user and the Unix programs through "standard streams"
- Information on screen is called standard output or "stdout"
- Input to programs is "stdin"
- Also, "sterr" - will be useful later

Commands for Getting Around

cd	Change Directory
mkdir	make directory
ls	List
rm	Remove
pwd	Print working directory
man	Manual

Commands for Getting Around

cd	cd : takes you home cd .. : takes you up one level (to the containing directory)
mkdir	mkdir filename
ls	ls -a : shows hidden files ls -l : shows files along with sizes and timestamps
rm	rmdir: remove directory **CAUTION** Not a trashcan! Once it's gone, it's gone.

Getting Comfortable

tab	Auto complete
*	Wildcard
Up arrow	Last command
Ctrl + C	Escape process
Ctrl + L	Clear screen

Getting Comfortable

tab	Enter enough unique characters and press tab. This will complete the filepath or command.
*	Matches every character in a filename.

Tasks

- Create a file and a directory. Create a second directory. Copy the file into it.
- Now, go into the original directory and delete the original file.
- Change back into the second directory and move the file into the original directory.
  - How is this different than copying?

File operations

grep	print line with matching plain text string
cat	Concatenate, stream to "standard out"
head/tail	Print the first or last lines in file
|	Send output of one command or program to another as input
wc	Word count
cp and mv	Copy and move

File operations

grep	grep word filename
cat	cat file1
head/tail	head -n1 file1 tail -n4 file1
|	ls -l | wc -l
wc	wc -l counts number of lines wc filename counts the words in the file
cp and mv	cp file folder makes a copy of a file into a folder mv file folder moves that file, leaving no copy

File operations

touch	create file
nano	open file and write to it

File operations

**Looking at the manual for all the commands we are showing you is worth your while. Typing 'man command name' will show the manual file

Or just Google it!

Redirection

- > versus >>
  - > overwrites file content with whatever is on the left side of the redirect symbol
  - >> appends whatever is on the left side to the file on the right side
- Between the pipe and the redirect, you can write a one-line custom program for text editing
  - "Get all sequence names from a fasta file"
  - grep ">" file1.fas | cut -d ">" -f 2 >> seqs.txt

nano

- nano is Unix's default text editor
- Type 'nano' to access it
- This will open a text editor within your terminal
- Saving, exiting and other file functions are controlled with ctrl + letter keys
- If you create a document and write to it, saving it will add the document to the current directory

Bonus task

- Copy all the tree files to home
- Remove all the tree file in home
- Concatenate all the tree files in a file called trees.txt in home
- How many trees are in this file?
- The second tree is unrooted and has node labels. Make a new file with just the second tree from each of the tree files called trees2.txt

Literature - The internet

- Stack Overflow
  - The answer is there, but it might be snarky

- Software Carpentry
  - Lot's of great free lessons
  - Host lectures - keep an eye out
- LearnPython
- CodeAcademy