
# The Unix Shell

Software Carpentry Workshop – Day 1 - 17th November 2020

# Welcome

- Based on **The Unix Shell** Software Carpentry lesson
  - http://swcarpentry.github.io/shell-novice/
- Goals:
  - Explain what Unix is why you'd want/need to use it
  - Get experience with some of the most common Unix commands
  - Get comfortable finding your way around your files on Unix systems
  - Teach you enough to be able to do cool stuff (e.g. use Eddie / supercomputers...)
  - Show you that the Unix Shell is less scary than it might seem!
  - Learn a bit about Unix Philosophy

# Session outline

**We'll do a mix of:**

- Live coding
- Short expositional talks
- Exercises & feedback
- Random nonsense

**Topics**

- Introducing Unix & Shell
- Navigating Files & Directories
- Working with Files & Directories
- Handy Unix commands
- Pipes & Filters
- Loops & Variables
- Shell Scripts
- Finding Things

# Preparation

# Preparation

**Please open up the software you'll be using today...**

- **Windows**: Run **Git Bash** (installed with **Git for Windows**)
- **Mac / Linux**: Run the **Terminal** application

Alternatives:

- Eddie users: can work on Eddie
- Physics & Astronomy folks: can work on School SSH gateway

**Ask for help now if you have problems!**

# Windows to fit on your screen(s)

**Necessary**

1. Zoom
2. The Etherpad
3. Your Terminal / Git Bash window
4. My Terminal: https://shellshare.net/r/unix-shell

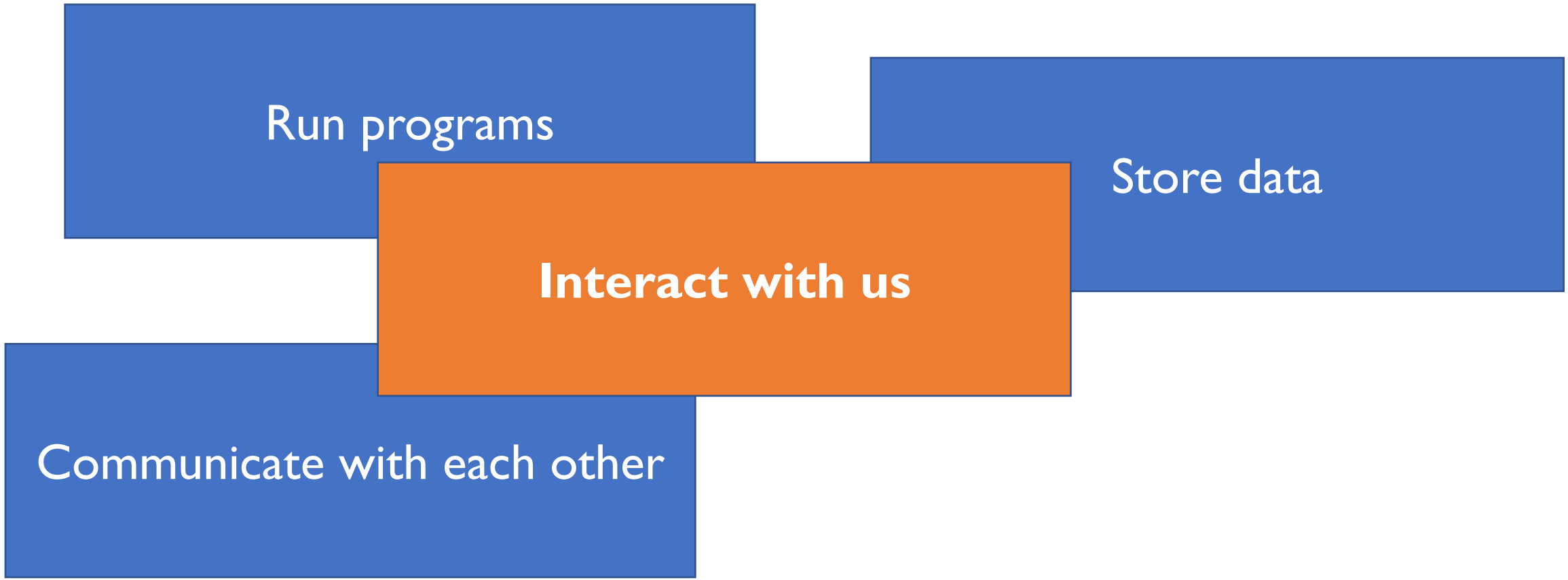
**Optional**

- The Lesson Content: swcarpentry.github.io/shell-novice

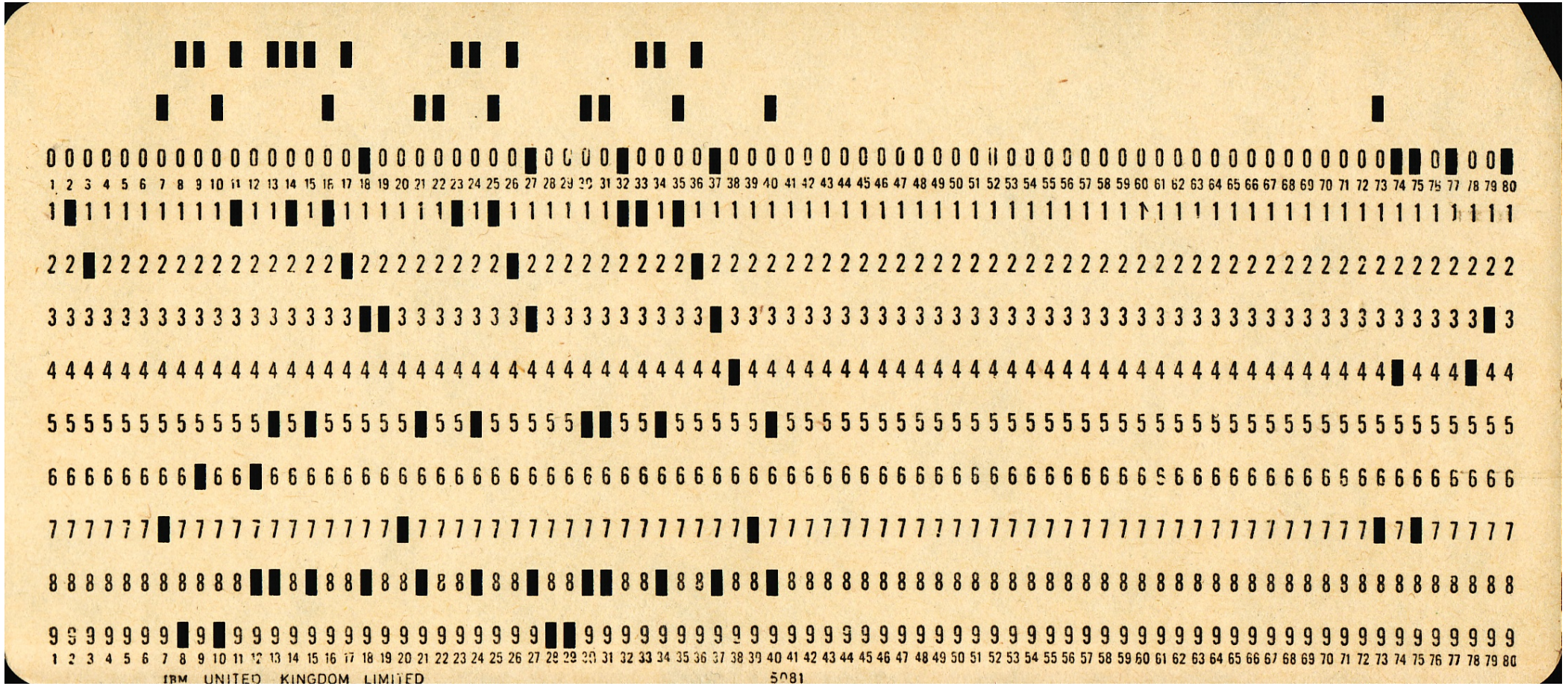
# 1. Introducing Unix & Shell

# Introduction

Computers do 4 basic things:

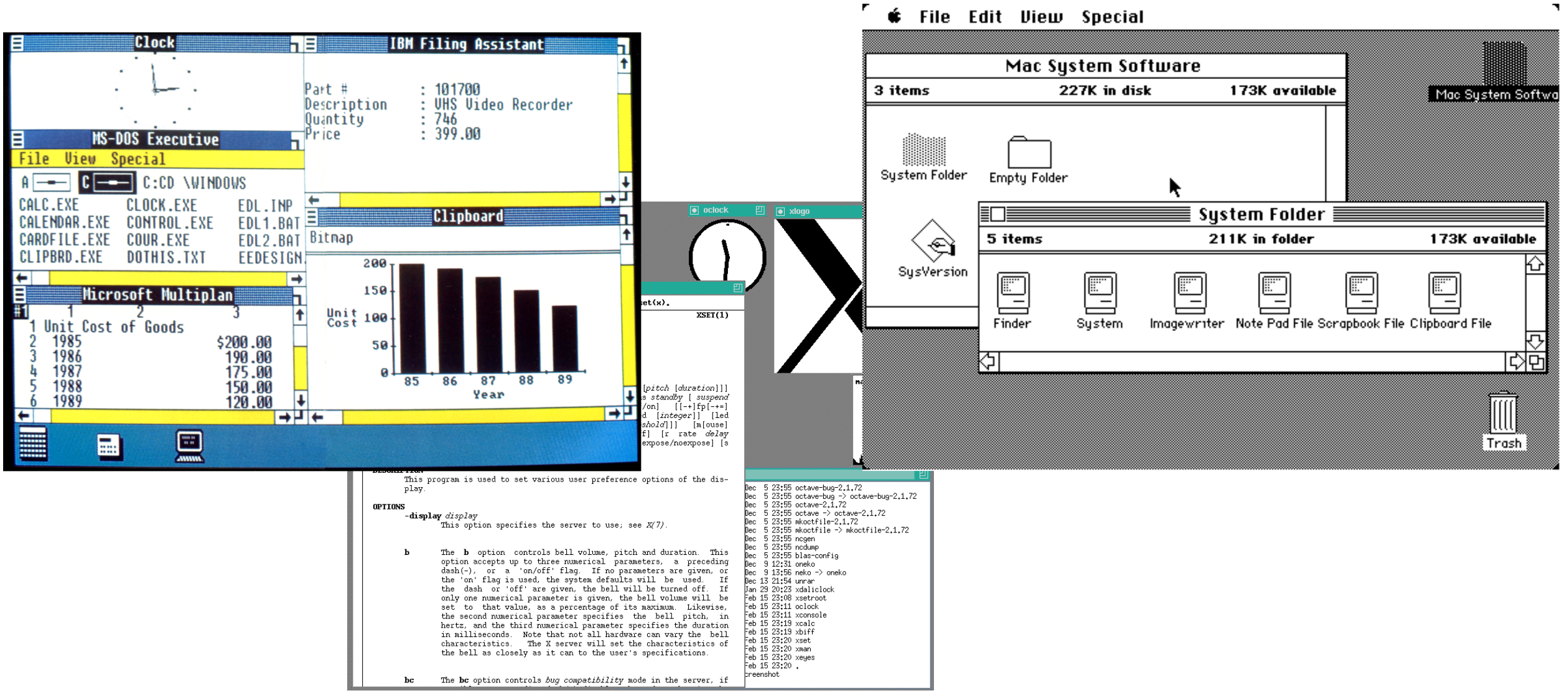


# Interaction in the 1960s: punched cards



# 1970s: Command Line Interfaces (CLI)

# 1980s: Graphical User Interfaces (GUI)



# CLI vs GUI

# CLI now co-exists with GUI

**GUI**

- Easier to pick up at first
- But can become limiting & repetitive

**CLI**

- Harder to learn at first
- But becomes very efficient & versatile

# What is Unix?

- A family of Operating Systems (like Windows)
- Originally developed in the 1970s
- Historically:
  - Operating Systems for big and expensive computers...
  - ...usually used by lots of different people at once
- Modern Unix systems:
  - Still big computers, e.g. Eddie, most supercomputers.
  - But also Apple Mac, Linux computers, Android phones (sort of)

# What is Unix?

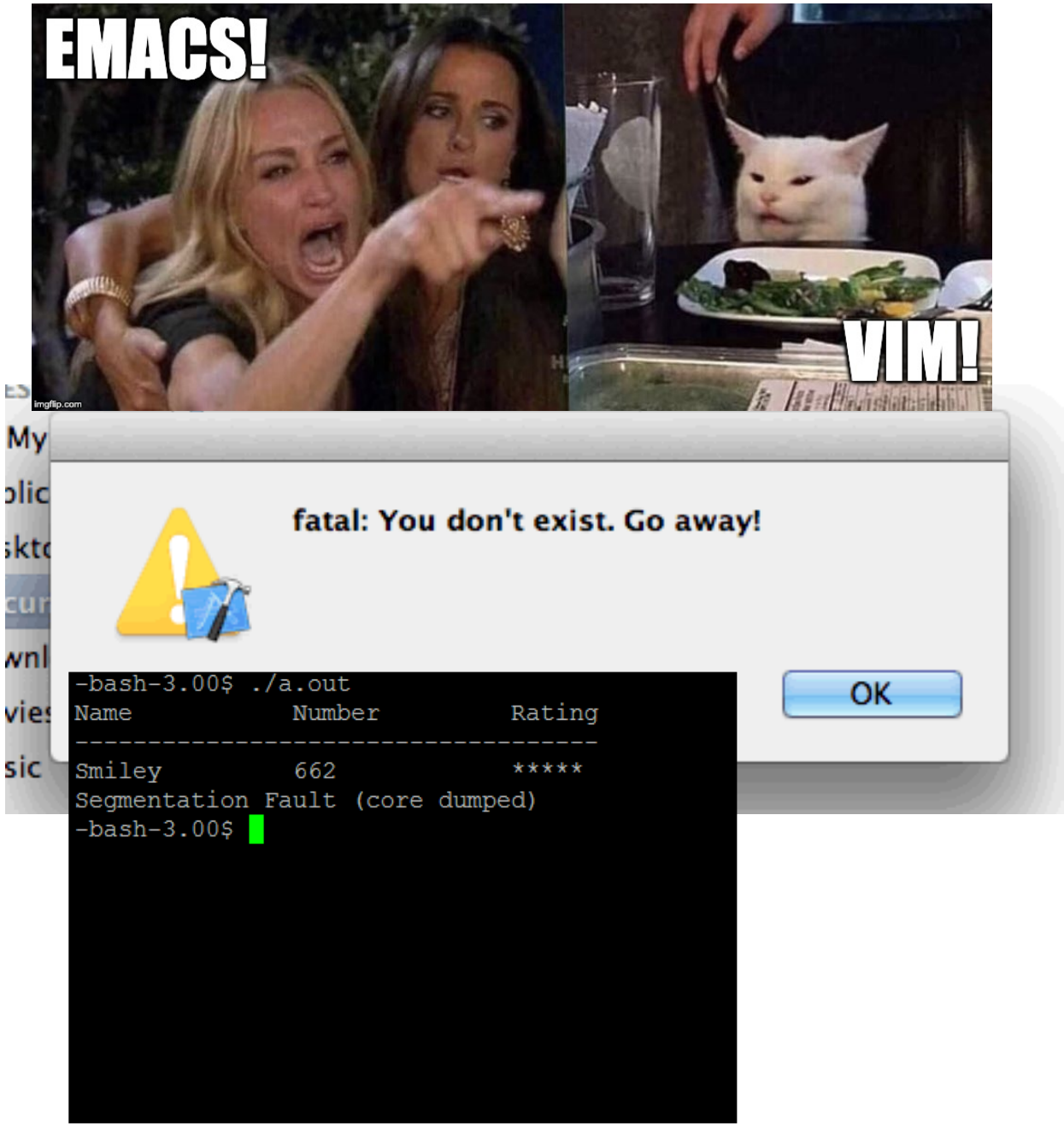
## Unix philosophy

- Modular design
- Many small tools doing one job well
- Joining things up
- The importance of textual data
- Choice - lots of ways to do the same thing

# What is Unix?

## Unix is fun!?

- Long history
- Terse & cryptic commands
- Terrible humour
- Religious wars
- Whimsical/scary error messages



# CLI + Unix = The Unix Shell

- The **Unix Shell** is a CLI for communicating with Unix systems
- (Unix systems do also have GUIs)
- Actually there are lots of different shells available for Unix!
- The shell we'll be learning is called **bash** (**B**ourne **A**gain **Sh**ell... ha!)
- Bash tends to be the default shell on most Unix systems

# Why learn/use the Unix Shell?

- Lets you interact with pretty much **any** Unix system in a uniform way
- Sometimes the only way you can interact with a Unix system!
- Pretty much essential for using a “big” computer

# Why learn/use the Unix Shell?

- As a researcher, knowing a bit of shell can help with:
  - Getting your data & code from A to B
  - Checking & reporting on your data
  - Basic data wrangling
- Learning how to write **Shell Scripts** can:
  - Let you automate, record and document tasks that might be complex, repetitive, error prone etc.
  - Join disparate processes together
  - Most Unix systems are driven by Shell Scripts!

# How do we communicate using a shell?

- Shell provides a **read** → **evaluate** → **print** (REPL) loop.
- We say what we want to do by typing in **commands**.
- Commands typically run programs installed on the system
  - Though sometimes they're special "builtin" commands provided by the shell itself
  - It's also possible to create your own commands

**Analogy: some similarities with issuing commands in English...**

# English analogy: Donald Trump's TODO list

- **Inject** bleach
- **Drink** covfefe noisily
- **Bomb** hurricane
- **Eat** hamburgers in bed

- **Verbs** say what you're **doing**
- Nouns say **what** is involved
- Adverbs provide additional information

# How do we communicate using a shell?

- Shell commands are kind of similar to English
- But:
  - They need to be written precisely
  - Commands are often cryptic / obscure / silly
  - We'll see lots of examples today!
- We can record a series of commands together as a **script**

# Introducing the Shell – Live Coding

**Finally!** We're going to start coding...

I'll share my own Terminal for easy copy & paste.

So please open:

https://shellshare.net/r/unix-shell

# 2. Navigating Files & Directories

# Objectives

- Understand **files** vs **directories**
- Learn how to move around on a Unix system
- Experience how **options** fine tune how Unix commands work
- Learn some handy typing shortcuts
- Learn how to get out of things

# Key concepts: Files & directories

## Files

Files store information/data

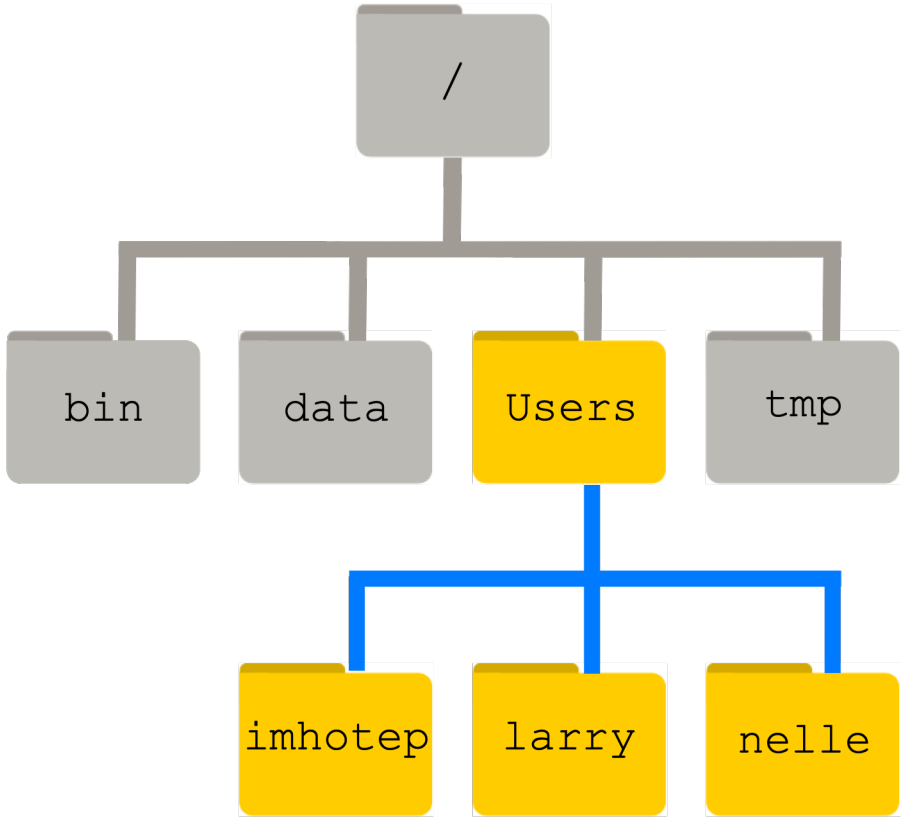
## Directories

Directories contain other files and/or directories

Directories are often called **Folders**

# Key concepts: The filesystem

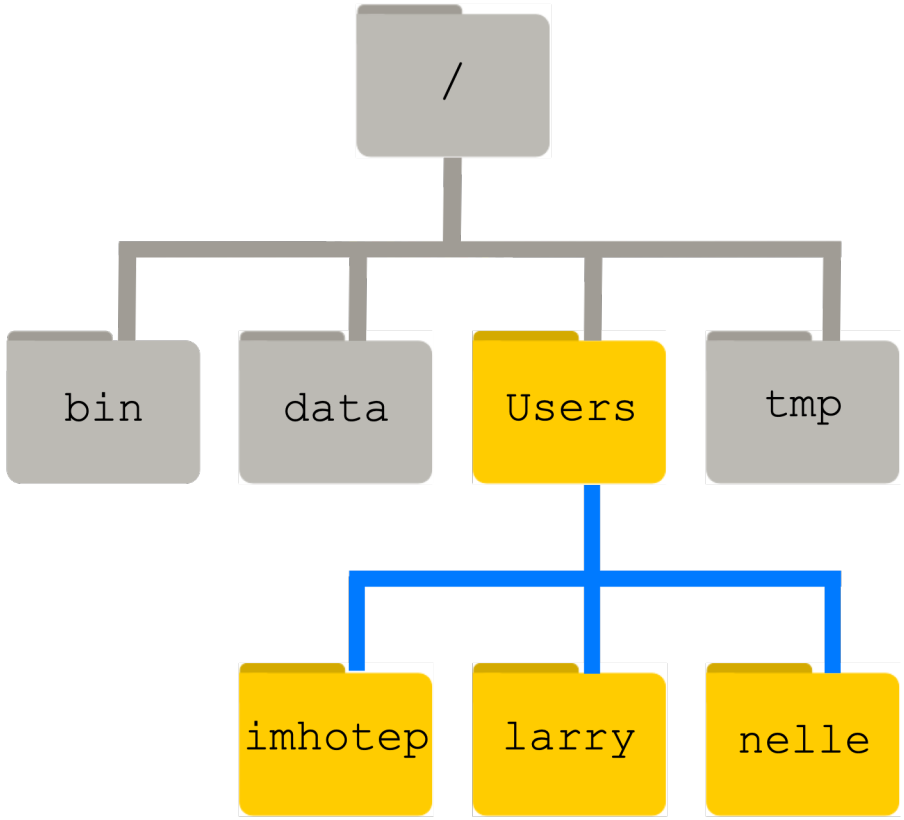
- Directories & files make a hierarchical (**tree**) structure called a **filesystem**
- Unix systems have a single **root directory** at the **top** of the tree
  - Windows has multiple roots, one for each drive
- The **root directory** is denoted **/**



# Key concepts: Present Working Directory (PWD)

- This is the directory you are "in" at any giving time
- You usually start in your special **home** directory
- You move around the filesystem by changing your PWD

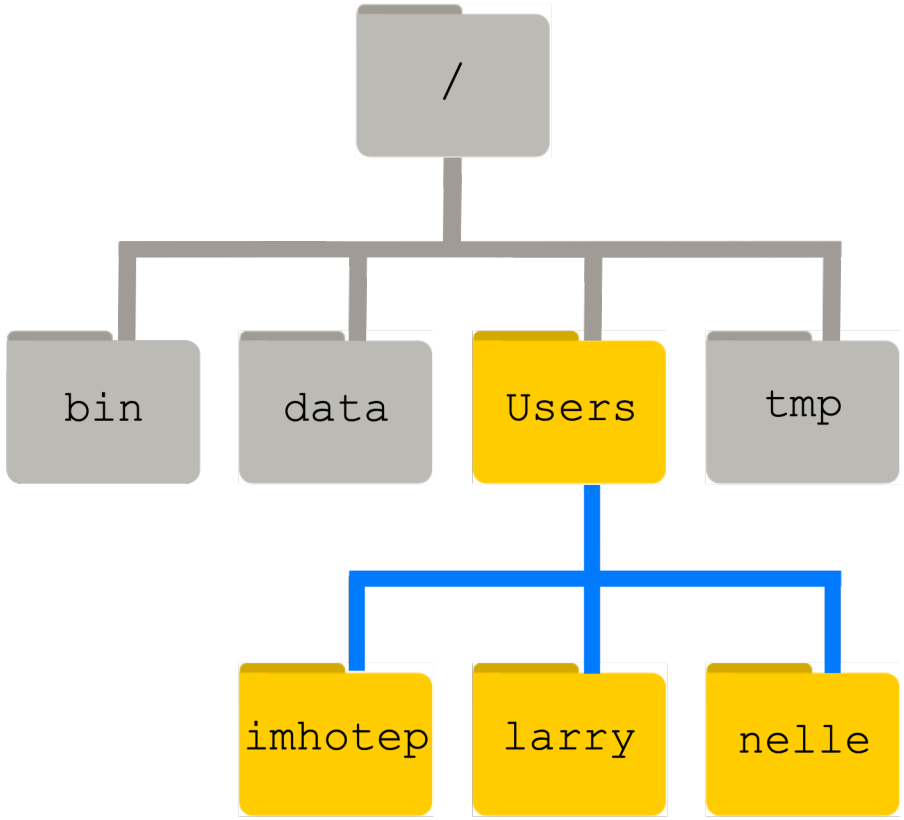
We'll learn commands for showing & changing your PWD



# Key concepts: File Paths

**File paths** tell you where a file lives in the filesystem

- An **absolute path** shows how to get to a file by starting from /
- A **relative path** shows how to get to a file by starting from a chosen directory
- In Unix we make a file path by **joining** the names of each intermediate file or directory with a / character



# Special navigation shortcuts

| Shortcut | What it means |
|---|---|
| . | current directory |
| .. | parent directory (i.e. up one) |
| / | root directory (the top of the tree) |
| ~ | your **home** (default) directory |

# Key Unix commands for navigating

- **pwd** (present working directory) – where am I?
- **ls** (list) – see what's in the present or specified directory
- **cd** (change directory) – navigate to specified directory

Got lost?

- Type **cd** on its own to take you home!

**Let's do some live coding examples now!**

# Recap: Handy keyboard shortcuts

- **Up** and **Down arrow** keys to access typing history
- **Left** and **Right arrow** keys to move within current line
- **Tab** completion to fill in names of commands / files etc.

# Getting out of things!

Try:

- **Ctrl-C**: interrupts most commands
- **q**: quits some interactive commands (e.g. less)

Found yourself in **vim** and can't get out?!

- Press **Escape** key
- Then type **:q!**
- Then press **Return**

# 3. Working with Files & Directories

# Objectives

Learn how to...

- Create new directories and files
- Pick good names for new directories & files
- Edit text files
- Delete, rename, copy or move files & directories
- Use wildcards to pick multiple files at once

# Creating a new directory

**mkdir DIRNAME**

# Creating a new file

- Can use a **text editor** to create a new text file
  - Lesson uses a simple text editor called **nano** for this
  - Other common text editors are **vi(m)** and **emacs**
- Can also create an empty file using the **touch** command

# Good naming for files & directories

- Try to stick to combinations of
  - Alphabetic letters (a-z, A-Z)
  - Numbers (0-9)
  - Dot (.), Underscore (_), Hyphen (-)
- Avoid starting names with hyphens
  - That's because options usually start with hyphens... confusion!
- Avoid using spaces in file names
- Avoid exotic letters, symbols and emojis!

# Copying, moving or renaming things

- Copy:
  **cp SOURCE DESTINATION**
- Rename or move:
  **mv SOURCE DESTINATION**

# Deleting files & directories

- **rm FILENAME**
- Beware! Deleting is forever!
- Risky usage:
  - **rm -r** deletes directory **and all of its contents**
  - **rm -rf** forceful version of the above
- Safer usage:
  - **rm -i** asks for confirmation
  - **rm -ir** safe recursive deletion
  - **rmdir** deletes a directory, but only if it's empty

# Wildcards

Wildcards allow you to specify multiple files/dirs whose names contain (match) patterns of your choice.

**Key wildcards**

| | |
|---|---|
| * | matches any (zero or more) number of characters |
| ? | matches any one character |
| [...] | matches any of the characters inside the square brackets |

# Wildcards & regular expressions

# 3½: Some handy Unix commands

# Outputting

- **echo** – outputs a message

# Peeking into files

- **cat** – concatenate, i.e. show file contents
- **more** – show file contents one page at a time
- **less** – better version of more... ho ho ho!
- **head** – show first few lines of file
- **tail** – show bottom few lines of files
- **wc** – word count... also line & character count

# Extracting and reformatting data in files

These are all great for manipulating text files:

- **head** & **tail**
- **sort** – sorts file contents
- **uniq** – removes duplicates
  - sort & uniq can be combined to extract unique values or do grouping
- **cut** – picks out columns from tabular data
- **grep** – search file contents (covered in Chapter 7)
- More advanced: **sed** & **awk**
- Even more advanced: write some code (e.g. in Python)

# 4. Pipes & Filters

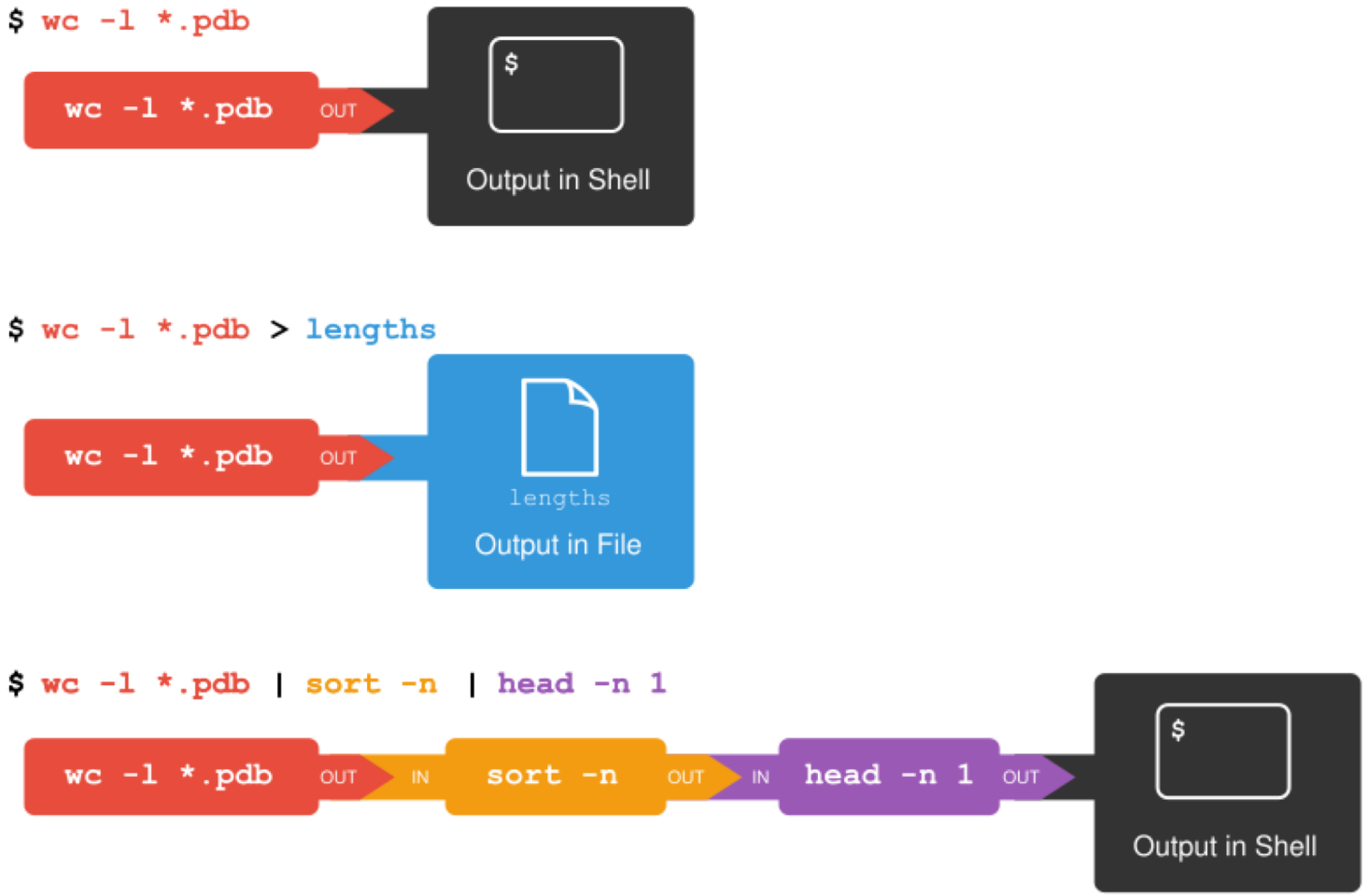
# Objectives

- Learn how to **redirect** (save) a command's output to a file
- Learn how to chain commands together into a **pipeline**
- Construct some basic pipelines and **solve problems** using them
- Explore Unix's Lego brick philosophy

# Redirecting output & making pipes

- **command > file**
  Redirects a command's output to a file
  Overwrites any existing content!
- **command >> file**
  Appends a command's output to a file
- **first | second**
  Creates a pipeline: the output of the first command is used as the input to the second.

# Redirecting output & making pipes



# 5. Loops

# Loops: Motivation

Sometimes you need to perform the same set of commands to a bunch of files

Manually handling each file is tedious and error prone!

**Loops** provide a nice solution to this

Loops come up in lots of computing contexts so good reusable skill!

# Loops: Objectives

- Learn how to write loops in the Unix shell
- Understand the basics about variables
- Demonstrate how to see what commands you've recently executed
- Learn more handy keyboard shortcuts

# 6. Shell Scripts

# Shell Scripts: Motivation

Shell Scripts let you "record" a sequence of commands

- This lets you reuse, repeat & automate your workflows
- Can save you a lot time in the long run
- Reduces risk of making errors
- Documents your workflow... handy when you revisit it later
- You can build up a personal library of useful scripts
- Scripts are used to submit work to "big" computers (e.g. Eddie)

# 7. Finding Things

# Objectives

- Learn how to use **grep** to find content within files
- Learn how to use **find** to search for files
- Learn about **$(...)** to generate a list of files from the output of a command
- Learn how to combine all of these for more complex searching

# Grep exercise

- Go back to the **creatures** directory
- Remember how we earlier extracted the CLASSIFICATION line from one of these files?
  - E.g. head -n 2 basilisk.dat | tail -n 1
- Can you use grep to do the same thing?

# Searching for chemical elements

1. Go to the top of **data-shell**
2. Write a command to find all *.pdb files
   These all represent various chemical compounds
3. Pick **one** file and look at it using the **less** command
4. Note the ATOM lines - the 3rd column is a chemical element present in the compound
5. Can you write a command to find all *.pdb files for elements containing Chlorine (Cl)?

Try to make your command as reliable as you can!

# Possible decent solution

grep -wi Cl $(find . -name “*.pdb”) | grep ATOM

- Using grep -i as some files say CL but others say Cl
- grepping ATOM ensures we’re only looking at the ATOM lines
- This lists more than just the matching file names though.

# Wildcards & regular expressions

# End of Day 1

# End of Day 1

- Please give feedback on today's lesson on Etherpad
  - List **one good thing** and **one thing that could be improved**
- See you all tomorrow for Python!