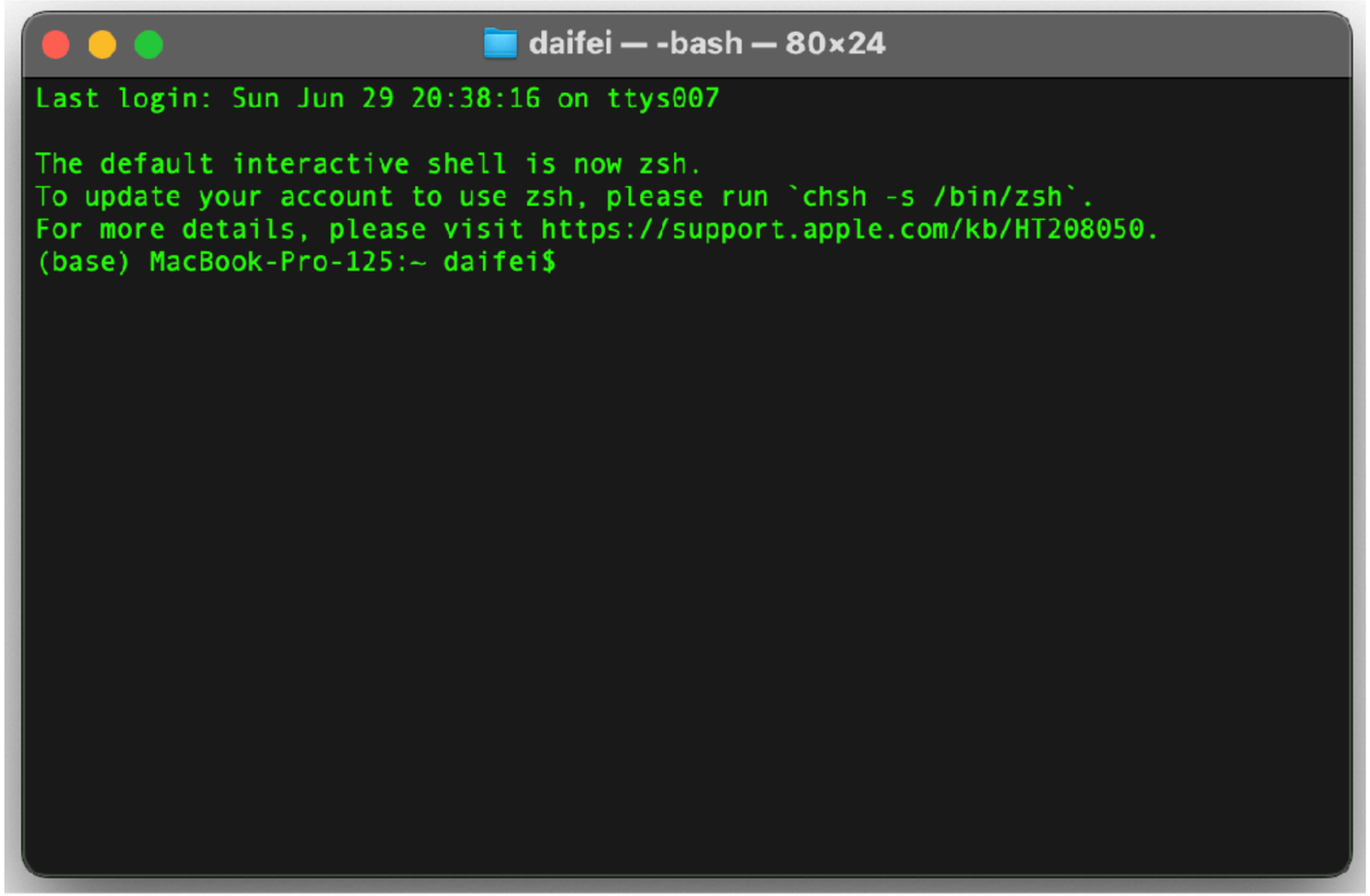
Why you should know about Command Line Interface:

Unix, Shell & Shell Scripts

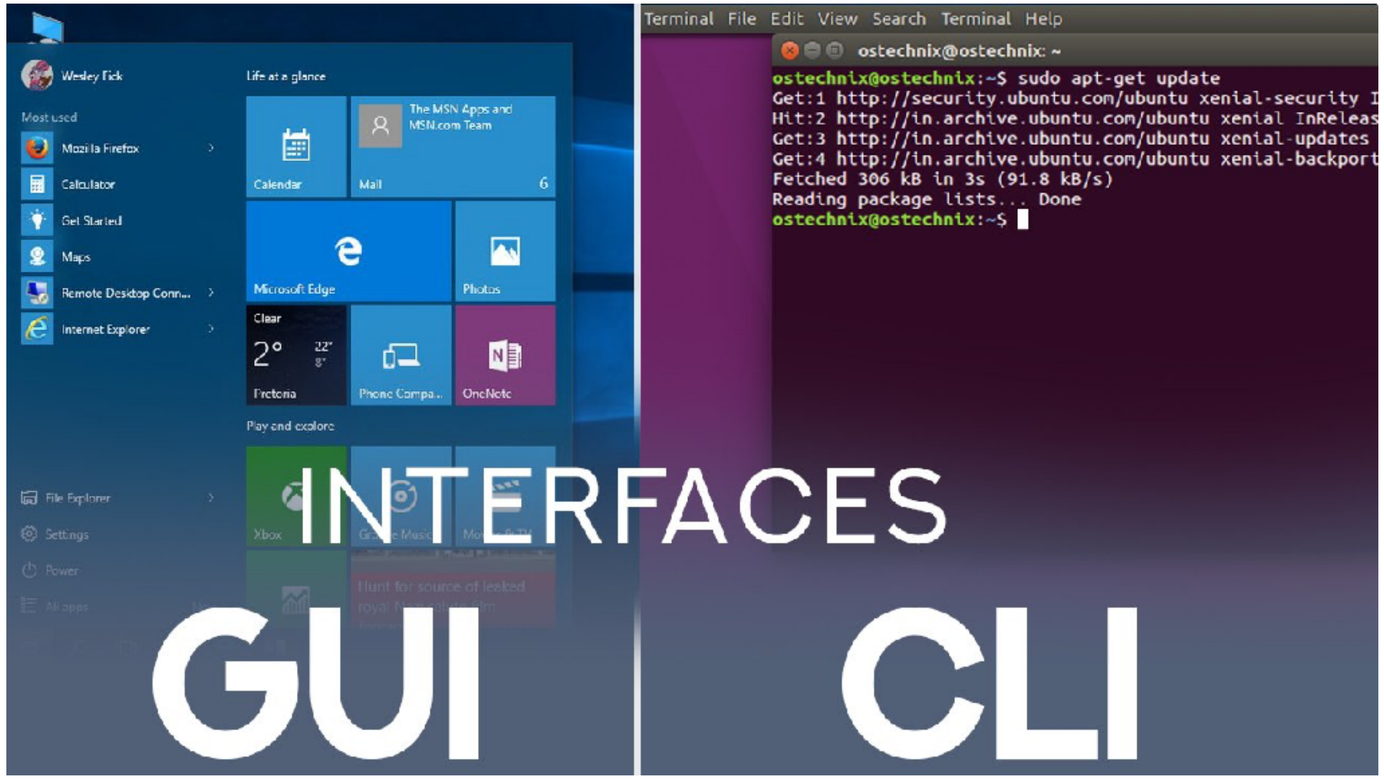
References:
previous year mentors Chetan Chawla (ZS, ASIAA), Joey Murphy (UCSC), The Unix Workbench, and Unix Software Carpentry

Fei Dai
Assistant Professor
Institute for Astronomy, University of Hawaii



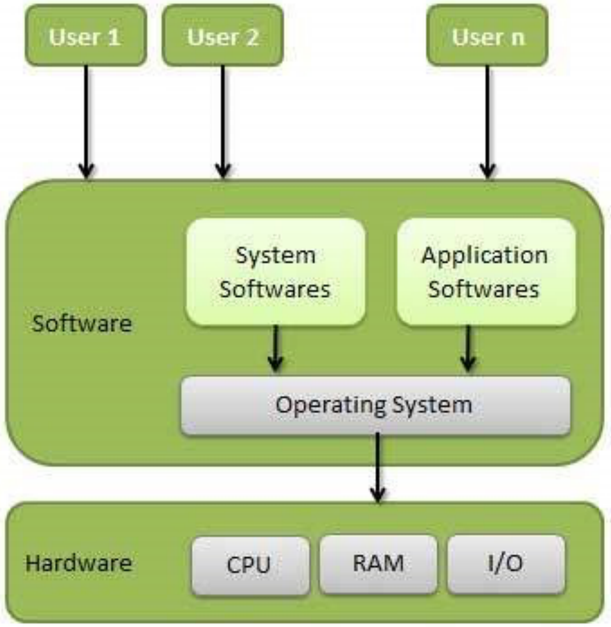
From Graphic User Interface to Command-Line Interface

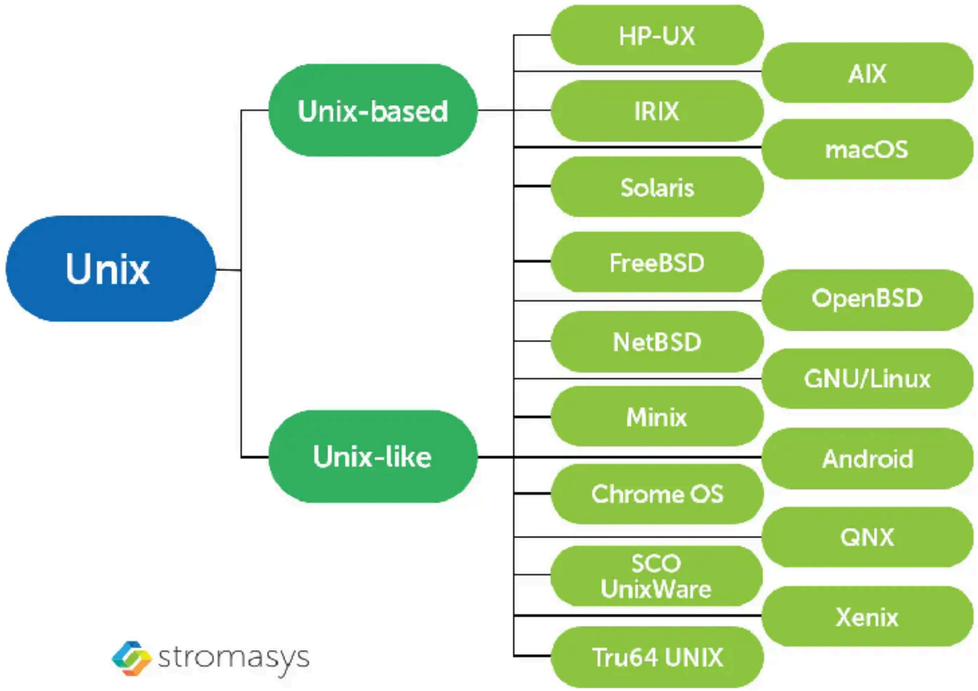
- Most people interact with their computers using a **Graphic User Interface (GUI)** i.e. clicking with a mouse or tapping on a screen.
- Most scientific computing (simulations, large datasets) are performed with a **Command-Line Interface (CLI)**: i.e. typing commands in a terminal.
- CLI is crucial for **automating, standardizing**, and repeating **large-scale** computing tasks **reliably**.
- CLI also makes it easier for a whole team of scientists to work on the same project. Version control (Git)
- With CLI, one can easily remote-control a computer with only limited internet bandwidth.



What is **Unix**?

- Unix is the first portable **operating system (OS)**
- An OS facilitates the interface between hardware and software on your machine, and lets the user interact with the machine
- It was written in 1971, entirely in the C programming language
- It is extremely fast for repetitive tasks and is widely used for distributed applications
- MacOS works on Unix. Linux is also a Unix-like alternative





What then is **Unix Shell**?

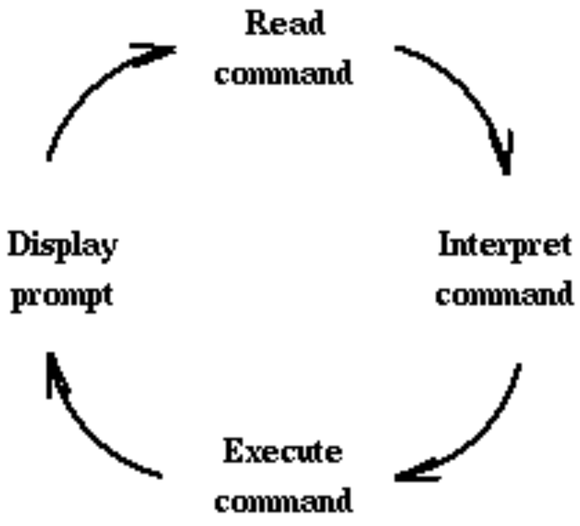
- As Unix is an OS, the **Unix Shell** is the corresponding command-line interface which the users can interact with.

- **Shell** in the command the users typed, interpret it, execute the command, before displaying the output in the terminal.

- **BASH** (aka. the Bourne Again Shell) is the most population implementation of the Unix shell.

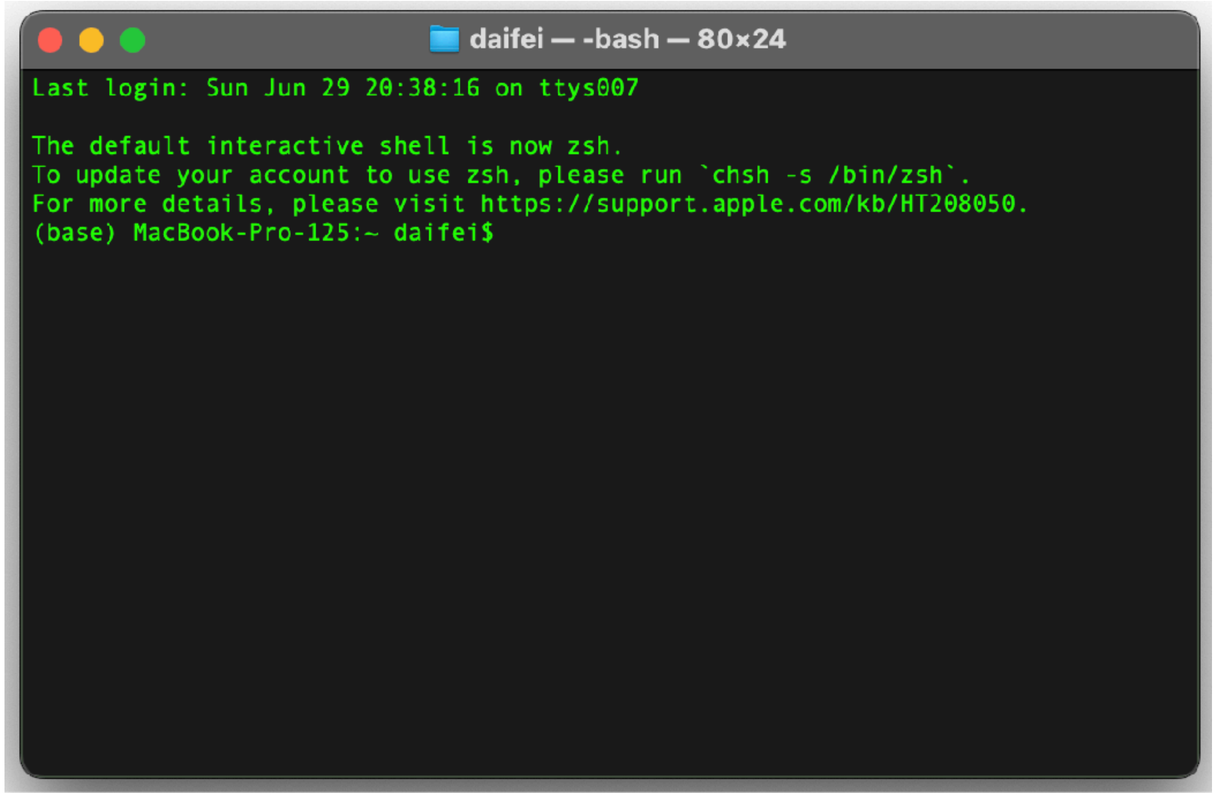
- MacOS and Linux both the *Terminal* built in. For windows users, you should install the *Windows Subsystem for Linux.*

- One can also interact with Shell in a Python Notebook with the command %%bash.



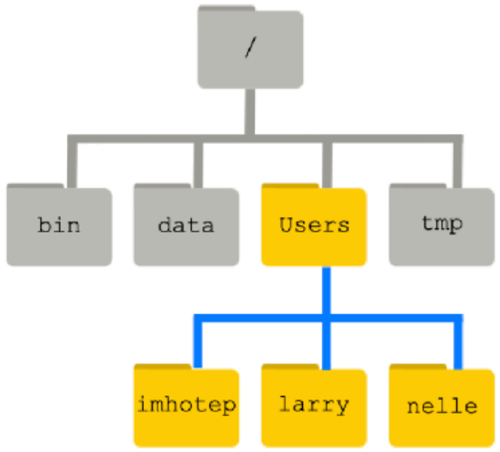
Interacting with Shell

- When the shell is first opened, you are presented with a prompt (in most cases), indicating that the shell is waiting for input
- We write a command and press enter to execute it
- `pwd` is the command used to tell us the present working directory (folder), i.e., where the shell is currently operating in the file system



Navigating the Unix File System

- Unix (and linux) file system has a tree like filesystem, in which, `/` directory represents the root of the file system
- It treats files and direcetories likewise in the file system
- The root directory has a bunch of directories and files, which we can navigate to using `/Users/nelle/` (for nelle directory) or `/Users/nelle/filename.txt` (for a file)
- `~` represents the user's home directory. The terminal opens up at home directory by default



Working with directories using commands

- We can move around our working directory by using the command `cd [path]` (change directory)
- Let us first go to the home directory using `cd ~`
- We can now check the present working directory to see if we are in the home directory

In [1]:

```
%%bash
cd ~ # Move to the home directory
pwd
```

Navigating the Unix File System

- We can use the command `ls` to see contents of the current working directory we are in

In [2]:

```
%%bash
pwd
ls # List the contents of the current directory
```

```
/Users/chetanchawla/Astrophysics/Mentoring/Intro2Astro2024/Intro-to-Astro2
024/Week1_Unix_Python_Papers
PaperWritingGuide.pdf
Python Tutorial.ipynb
Python and Jupyter Intro.html
Python and Jupyter Intro.ipynb
README.md
Unix_Git_Tutorial.ipynb
Unix_Git_Tutorial.slides.html
data
how_to_read_scientific_papers.md
images
notebook_names.txt
```

Some more directory commands

Changing Directories: Relative paths

- `cd ..` : to up one directory in the filesystem (the parent durectory of current directory)
- `cd .` : change directory to the current directory (no action)
- `cd /` : change directory to root
- `cd ~` or `cd` : change directory to user's home directory
- `cd -` : change directory to the last directory you were using
- `cd <directory>` : change directory to the name of directory passed as an argument to the command

Changing Directories: Absolute paths

- `cd /Users/chetanchawla/Astrophysics/` : Here, an absolute path to the directory is given

Listing the Contents of a Directory

- Flags are options that we pass on for a command to behave in a particular way
- Single-letter flags are passed using `-` while multi-letter flags are passed using `--`
- We can look for complete documentations and the information about flags for each command by using the command `man command` (this opens up a manual in an editor. use :q to quit), or `command --help` (used in linux)
- `ls -l`: lists the directories and files in current directory in a list format
- `ls -t`: lists the directories and files in current directory sorted by the time they were created/modified
- We can also combine these flags. `ls -ltrh`: lists the directories and files in current directory in a **l**ist format, sorted by **t**ime in **r**everse order, in a **h**uman readable format

Creating files and directories

- `touch [file]` : creates [file]
- `mkdir [directory]` : creates a new directory
- `rm [file]` : delete [file]. We can also delete an empty directory using `rm [empty-dir]`
- `rm -r [dir]` : delete a directory and recursively delete all files/subdirectories within that directory. `-i` is used for requesting confirmation before deleting something.

Be careful with `rm`! Especially if you're thinking about using the `-f` (force) or `-r` (recursive) options. Make sure you use `-i` option to confirm before deleting. This is permanent and you may use your important files, or even your complete filesystem (including the os).

Copying and moving files and directories

- `cp [source file] [target file]` : make a copy of [source file] named [target file] in the working directory
- `mv [file] [destination]` : move [file] from the working directory to the location specified by the [destination] path. Also used to rename files
- `cp -r [directory1] [directory2]` : To make a copy/backup of a directory and all its contents recursively
- `mv -r [directory1] [directory2]` : To move a directory and all its contents recursively to another directory

Working with files and directories

Working with files

- `cat [file]` : print the content of the file on the terminal
- `head -n 10 [file]` : display first 10 lines of [file]
- `tail -n 5 [file]` : display last 5 lines of [file]

Operators

They are used to combine the outputs of different commands together to create more powerful commands.

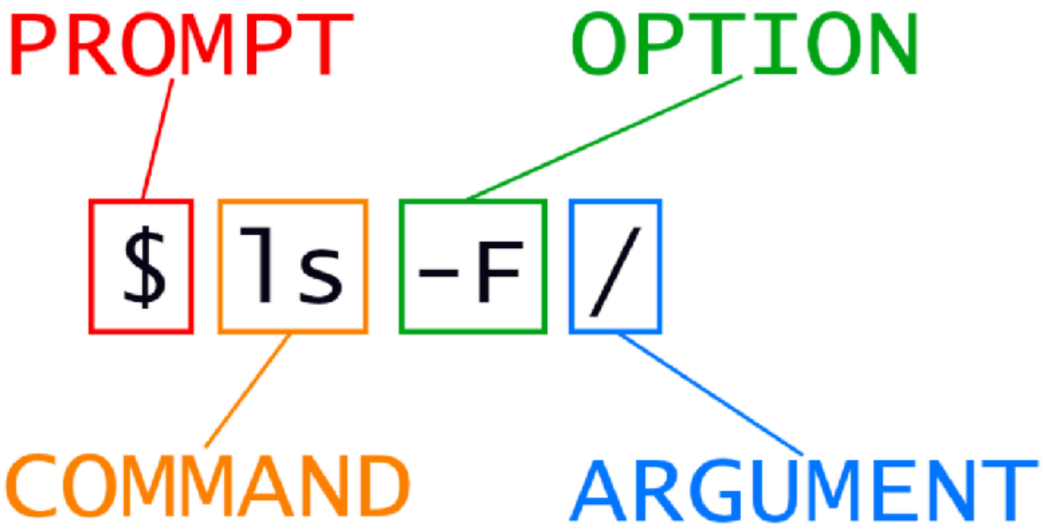
- `cat > [file]` : overwrite the contents of [file] (starting at line 0) with the standard input from keyboard. (Use ctrl+d to end input)
- `cat >> [file]` : append standard input to [file]'s existing contents
- Here `>` is the overwrite operator and `>>` is the append operator. Similarly, `|` is the pipe operator which passes the output of one command as the input to another command.

Working with files and directories

Inspecting files

- `grep [string] [file]` : search for and return all lines containing [string] in [file]
- `ls | grep foo` : look for and return all file or sub-directory names in the working directory that contain "foo"
- `find . -name *.txt` : Command to find files and directories by their names in the current directory (.). The argument passed to the name flag, `*.txt` implies that the command will find all files that ends in the extension *.txt*.
- `*` is a wildcard, which matches zero or more characters. `?` is also a wildcard, but it matches exactly one character. For example
    - `a*` means all names starting with letter a.
    - `*a` means all names ending with a
    - `*a*` means all names having the letter a
    - `?a?` means a 3 letter word having a as the second word, and any two letters, one on each side of it

The General Command




```
pwd            # Print current directory
ls             # List files
cd folder      # Change directory
mkdir new      # Make a new folder
cp file.txt backup.txt    # Copy
mv file.txt data/file.txt   # Move/rename
rm file.txt # Remove
```

Shell Scripts

- A **shell script** is a file that stores a sequence of commands — essentially a **small program**.
- Scripts let you **automate repetitive tasks**, making your work faster and less error-prone.
- They improve **accuracy** (fewer typos) and ensure **reproducibility** of your work.
- You (or others) can **rerun entire workflows** later with a single command.

Imagine you have 100 FITS files from Keck. This script loops over them and runs your Python pipeline.

```
for file in *.fits; do
  echo "Processing $file"
  python process_image.py $file
done
```

Executing a Shell Script

- You first need to grant the shell script permission with *chmod +x *.sh*

- The *chmod* command in Unix/Linux stands for **"change mode"**, and it is used to **change the permissions** (i.e., who can read, write, or execute) of files and directories.

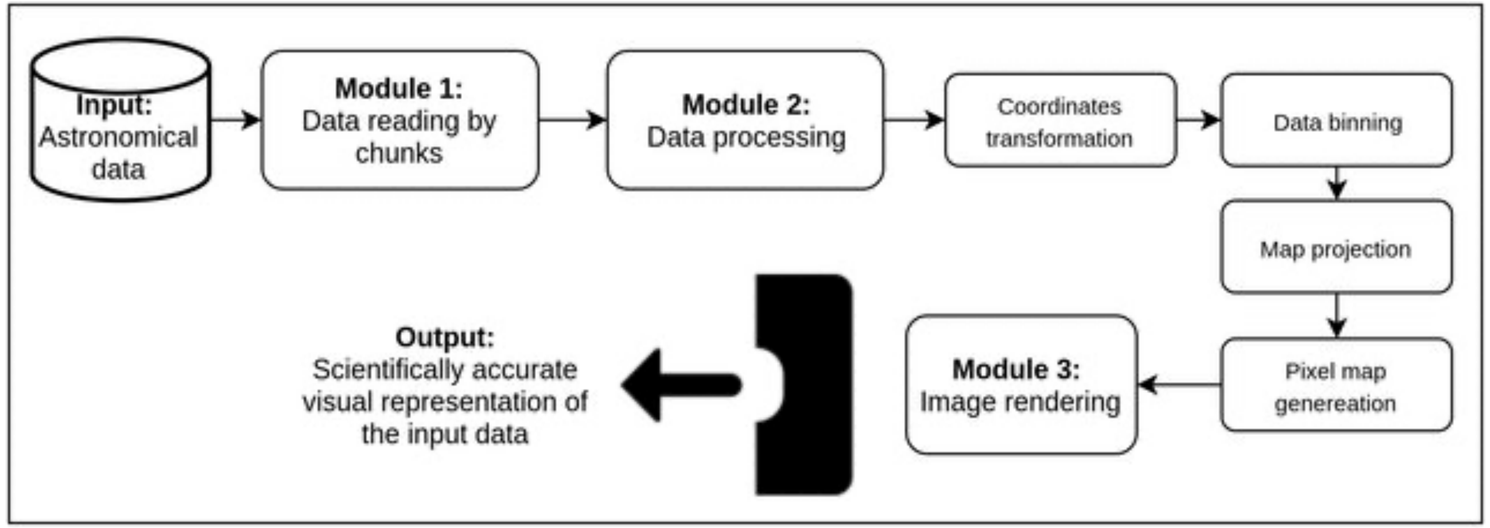
- Then execute the shell script with *./*.sh*

```
for file in *.fits; do
  echo "Processing $file"
  python process_image.py $file
done
```

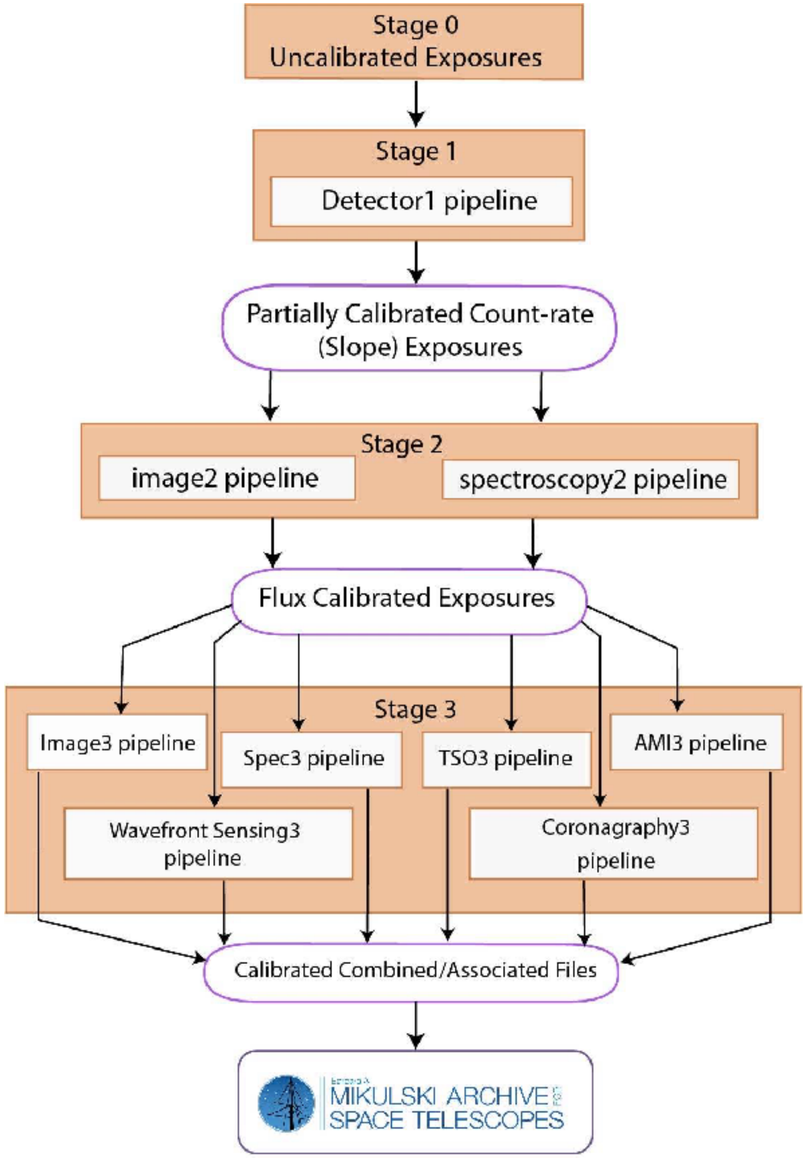
From Graphic User Interface to Command-Line Interface

- Often a “Pipeline” (a series of operations) is created that allows you to work faster, and to make whatever tasks you are doing reproducible.

- The JWST Pipeline is shown on the right.



Barros et al. 2016



Unix, Shell, and Shell Script

- You don’t have to memorize commands — you’ll learn by doing. Or simply look them up.

- A good analogy to understand Unix, Shell, and Shell Script is that: If your computer is a laboratory, Unix is the lab bench, and shell scripts are the lab protocols — you can rerun experiments exactly the same way every time.

Assignment 1

- Create a new directory named foo_dir
- Change directories to foo_dir
- Write "Hello, world" to a text file
- Display the contents of the text file
- Make a copy of the text file
- Place the copy in a new sub-directory, foo_sub_dir

Assignments

- Complete The Unix Workbench, and Unix Software Carpentry