Week 2-4: Unix, Git, Containers & Scientific Python

Chapter 2: The Unix Operating System

What is Unix?

- Fundamental operating system used to control how computers execute programs
- Originated in the 60s
- Important for data analysis, programming and system management
- Designed to produce output that can be used as input to other programs
- Enables the creation of pipelines using small tools

The Shell

- A command line interface (CLI)
- Allows users to:
    - Run programs
    - Control files and folders
    - Receive output in text form
- Operated as a RELP
    - Read Evaluate Print Loop
- Common in modern system (macOS, Linux, Windows via Git Bash)

Basic Commands

Command	Description
`ls`	List the contents of the current directory.
`cd`	Change the current directory.
`pwd`	Print the path of the current directory.
`mkdir`	Create a new directory.
`touch`	Create a new file.
`cp`	Copy a file or directory.
`mv`	Move or rename a file or directory.
`rm`	Remove a file or directory.

`cat`	Print the contents of a file to the terminal.
`less`	View the contents of a file one page at a time.
`grep`	Search for a pattern in a file or files.
`sort`	Sort the lines of a file.
`find`	Search for files based on their name, size, or other attributes.
`wc`	Print the number of lines, words, and bytes in a file.
`chmod`	Change the permissions of a file or directory.
`chown`	Change the ownership of a file or directory.
`head`	Print the first few lines of a file.
`tail`	Print the last few lines of a file.
`diff`	Compare two files and show the differences between them.

Exploring the Filesystem

- When shell starts, it begins in home directory
- Adding flags changes command behavior
    - Example: ls -F adds / to folders
- Absolute path:
    - Always starts from root /

    - Example: cd /Users/arokem
- Relative path:
    - Depends on current directory
    - Example: cd Documents

Path Shortcuts

- .. – parent directory
    - Example: cd .. goes up one level
- ~ - home directory
    - Example: cd ~
    - Example: cd ~/Documents

Pipe Operator (|)

- Connects commands so output of one becomes input of another

Why Unix Matters

- Provides powerful control over:
    - Files and folders
    - Program execution
    - Automation through pipelines
- Becoming comfortable with Unix makes data work faster and more efficient

Chapter 3: Version Control (Git)

What is Git?

- A widely-used version control tool
- Works via a command line

Initialize a Repository

- Creating a Project
    - Mkdir my_project
    - Cd my_project
    - Get init
- Add a file
    - Touch my_file.txt.
    - Git add my_file.txt
- Check status

    - Git status
- Commit Changes
    - Git commit -m “Statement here”
- Check commit history
    - Git log
- Important concepts
    - Commit: saves a snapshot of your project
    - SHA: unique identifier for each commit
    - HEAD: current state of the repository

Tracking Changes

- Stages
    - Unstaged
    - Stages
    - Committed changes
- View changes
    - Git diff
- Workflow
    - Make changes
    - Git add
    - Git commit
- Undoing changes
    - To revert a file to a previous commit
        - Git checkout <SHA> myfile.txt

Branching and Merging

- Enables experimenting without affecting main code
- Keep main branch stable
- Merge only when readu
- Create and switch branches
    - Git branch feature_x
    - Git checkout feature_x
- Merge branch into main
    - Git checnkout main
    - Git merge feature_x
- Delete branch
    - Git branch -d feature_x

Collaborating with Github

- Remote repository
    - A copy of your respo stored online
    - GitHub is the most common remote
- Add remote and push
    - Git remote add origin <URL>
    - Git push -u origin main
- Authentication
    - GitHub requires
        - Personal Access Token (PAT) OR
        - SSH Keys

Collaboration Workflow

- Cloning
    - Git clone <URL>
- Pulling changes
    - Git pull origin main
- Pushing changes
    - Git push origin main

Common Issues

- Push rejected
    - Caused by remote has changes you don't have locally
- Merge conflict
    - Occurs when two people edit the same lines

Pull Requests

- Used for review before merging into main
- Create a PR after pushing a feature branch
- Allows collaborators to:
    - Review changes
    - Comment
    - Approve merge

Advanced Collaboration

- Larger projects use branches & PRs & code review
- Helps prevents bugs and maintains project history

Version Control for Data: Datalad

- Git is not ideal for large data files
- Datalad is a git-like tool for data visioning
- Useful for tracking:
    - Code
    - Derived datasets
    - Analysis outputs

Chapter 4: Computational Environments & Containers

Why Comp Enviro Matter

- Data Science projects combine many software components
    - Python
    - Multiple Python libraries
    - OS-level dependencies
- Problems arise when:
    - Different projects require different library versions
    - You move work between computers, collaborators, clusters, or cloud
- Goal: reproducibility and portability
- Two main solutions:
    - Virtual environments (conda)
    - Containerization (Docker)

Virtual Environments with Conda

- Virtual Environment: a directory containing:
    - A specific Python version
    - Project-specific libraries and dependencies
- Keeps projects isolated from each other
- Why Conda
    - Popular package manager for scientific Python
    - Manages
        - Python versions
        - Libraries
        - Virtual environments
    - Works on Mac, Linux, Windows
- Base Environment Rule
    - Conda starts in the base environment

    - Best Practice: Do NOT work in base
    - Create one environment per project
- Creating an Environment
    - Conda create -n my_env python=3.8
    - Add packages at creation (jupyter)
- Activating/deactivating
    - Conda activate my_env
    - Conda deactivate
    - Prompt changes to show active enviorment
- Installing Packages
    - Conda install numpy
    - Installs into currently active environment
- Exporting & Sharing Enviroments
    - Exporting dependencies
        - Conda env export > environment.yml
            - Environment.yml
                - Lists exact package versions
                - Can be shared via GitHub
- Limitations of Conda
    - Does not capture:
        - Operating system
        - System level software
        - File system state
    - Leads to containerization

<u>Containerization with Docker</u>

- What is it?
    - Packages
        - OS
        - Software
        - Libraries
        - Data
    - Produces fully reproducible environments
- Docker Concepts
    - Image: Blueprint/recipe
    - Container: Running instance of an image
    - Host: Your local machine
    - Registry: Collection of images (DockerHub)

- Docker Images
    - Identified by:
        - SHA hashes
        - Tags (latest)
    - Latest changes over time -> not full reproducible

Getting Started with Docker

- Pull an Image
    - Docker pull hello-world
- Run a Container
    - Docker run hello-world
- Confirms Docker is working
- Container exists immediately
- Interactive Containers
    - Docker run -it ubuntu bash
    - Flags:
        - -i -> interactive
        - -t -> terminal
        - Bash -> Unix shell inside container
    - Exit container
        - Exit
- Persistence with Volumes
    - Without volumes -> files deleted when container stops

Creating Docker Images (Dockerfile)

- Basics
    - Text file name Dockerfile
    - Defines how to build an image
- Commands
    - FROM: base image
    - RUN: execute shell commands
    - COPY: add files into image
- Building an Image
    - Docker build -t arokem/niabel-notebook:0.1 .
    - Naming convention:
        - <username>/<image-name>:<tag>

Sharing Docker Images

- NeuroDocker
    - Tool for building neuroscience containers
    - Supports:
        - FreeSurfer
        - AFNI
        - FSL
        - ANTs
    - Simplifies complex neuroimaging installs