NJIT
New Jersey Institute of Technology

High Performance Computing

Intro to Linux

Table of Contents

What is Linux?

- **Linux** is an open-source operating system kernel that powers servers, desktops, mobile devices, and embedded systems.
- Built on **Unix principles**, it is **secure**, **stable**, and **highly customizable**.
- Originally created by **Linus Torvalds** in 1991, Linux has grown into the core of many operating systems.
- Used in various distributions. **Ubuntu** is popular for desktops, **Fedora** for developers, and **Rocky Linux** is a community-driven enterprise-grade distribution.
- Our **NJIT's HPC Wulver** is running on **Red Hat Enterprise Linux Version 8**
- Supports **multitasking** and **multi-user environments.**
- Powers a vast majority of **web servers**, **supercomputers**, and **Android devices.**
- Linux is free and open-source, with strong community support and contributions.

What is Shell? Bash?

- **Shell** is a program that acts as an interface between the user and the operating system. It allows you to communicate with the system by entering commands.
- **Think of it like a pharmacist:**—you request medicine, and they retrieve it for you. You can't access the storage directly, but you can provide a prescription (commands) to get what you need.
- The shell is like a **command interpreter**: it interprets the commands you type and executes them to perform tasks like navigating files, running programs, and much more.
- **Bash (Bourne Again Shell)** is one of the most widely used Linux shells. It interprets and executes commands typed by users.
  - Bash is both a command-line interface (CLI) and a scripting language that allows you to automate tasks.

Types of Shell

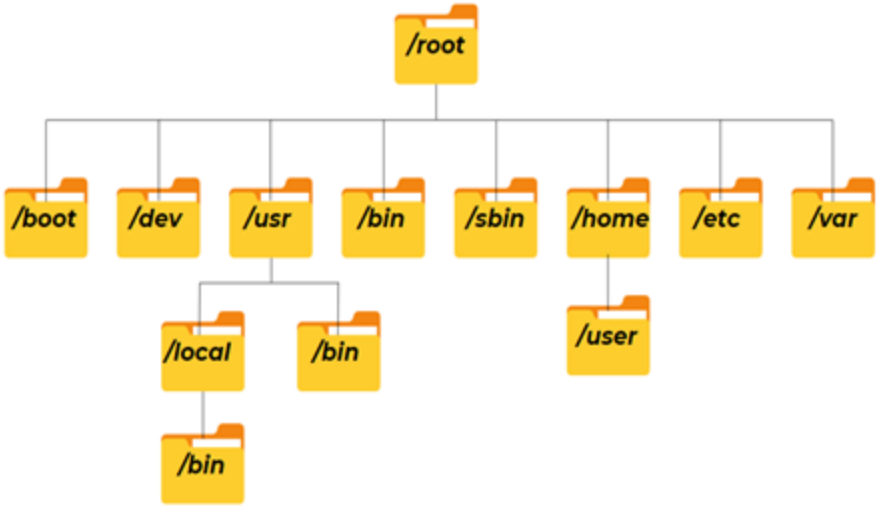
Types of Shell	sh	csh	tcsh	ksh	bash	zsh
Programming Language	Yes	Yes	Yes	Yes	Yes	Yes
Shell Variables	Yes	Yes	Yes	Yes	Yes	Yes
Command alias	No	Yes	Yes	Yes	Yes	Yes
Command history	No	Yes	Yes	Yes	Yes	Yes
Filename completion	No	Yes	Yes	Yes	Yes	Yes
Command line editing	No	No	Yes	Yes	Yes	Yes
Job control	No	Yes	Yes	Yes	Yes	Yes

CLI vs GUI

- **GUI (Graphical User Interface):** Interact by clicking or touching the screen. User-friendly, great for smartphones and desktops.
- **CLI (Command Line Interface):** Interact by typing commands. Preferred in servers, HPC environments (like NJIT HPC), and for automation.
- **Why is CLI Preferred in HPC?**
  - We use smartphones daily, clicking on apps, which internally translates into commands that execute actions. The CLI skips this graphical interpretation process, making it faster.
  - **Automation:** In GUI, copying 1000 files from 20 directories manually is tedious. Using CLI, you can write a script to automate this task, saving you time and effort.

File & Directory Management

- File & Directory Management is about organizing, navigating, and manipulating files and directories in a Linux system.
- Understanding this is crucial because files are the core of how Linux stores and interacts with data.
- Linux file system is hierarchical, starting from the root directory (/).
- Directories organize files into structures, and files contain data (text, programs, etc.).



File & Directory Management

- **ls** – Lists files and directories.

  $ ls Documents

  - **List with details:** Shows additional details like permissions, owner, size, etc.

    $ ls -l

- **cd** – Changes the current directory.

  $ cd Documents

  - **Navigate up one level:** Moves up to the parent directory

    $ cd ..

- **pwd** – Displays the present working directory.

  $ pwd

File & Directory Management

- **mkdir** – Creates a new directory.
  - Make sure you **don't add space** in your folder name.

    $ mkdir NewFolder
- **rmdir** – Removes an **empty directory**

    $ rmdir OldFolder
- **rm** – Deletes files or directories. To remove a directory that is not empty, you need to use the **-r** flag with the rm command.

    $ rm -r Documents

File Operations & Viewing

- A file operation involves performing tasks like viewing, editing, or searching the content of files on your system. These operations help you manage and interact with files.
- **Viewing Files :** You can open and view the content of files. This is helpful when you want to see what's inside a document, log file, or text file.
- **Editing Files :** Editing involves making changes to the content of a file. You might want to update information or fix something in a document.
- **Searching Within Files :** Searching allows you to find specific words or phrases within files, so you don't have to scroll through everything manually.

File Operations & Viewing

- **cat** – Displays the content of a file

  $ cat file.txt

  - **For example:** If you wanna see output of your job on Wulver

    $ cat fileName.out

- **grep** – Search for specific text within files

  $ grep "search_term" file.txt

  - Search recursively in all files within a directory

    $ grep -r "search_term" /path/to/directory

- **less** – View content of large files with more control (scrolling).

  - Use the up and down arrows to scroll, type q to quit.

    $ less file.txt

File Compression & Storage management

- HPC users deal with large datasets, logs, and results, which can quickly consume **disk space** and **quota limits**.
- Compressing files **reduces storage usage** and speeds up file transfers.
- Some applications on HPC require files in compressed formats to save memory.
- Best Practices for HPC users:
  - Regularly compress large files to free up space.
  - Use scratch storage for temporary files instead of home directories and then transfer these files to your home or project directory because scratch is deleted after 30 days.
  - Delete unnecessary files and check disk usage with **du -sh ***.

File Compression & Storage management

- **Checking Disk Usage**
  - **du -sh *** : Shows the space usage of each file and folder in the current directory.

    $ du -sh *
  - **quota_info** : Shows the used storage of home, project, scratch. Also displays Service Units.

    $ quota_info
  - **homespace** : Shows the used storage of the home directory with a detailed breakdown of file storage for each subdirectory

    $ homespace
- **Compressing files**
  - To save storage space on HPC, compress your files using efficient algorithms. Compression reduces file size while maintaining data integrity.

    $ xz myfile.txt

    $ bzip2 myfile.txt

    $ gzip myfile.txt

File Compression & Storage management

- To uncompress the files, use xz -d command

  $ xz -d myfile.txt.xz

- **Compressing directories**
  - Creates a compressed archive of the ***myfolder/***.

    $ tar -czf myfolder.tar.gz myfolder/

- **Extracting Compressed Files**

  $ tar -xzf myfolder.tar.gz

User Information and History

- **whoami** – Displays the current logged-in user

  $ whoami

- **man** – Displays the manual for a command

  $ man ls

- **history** – Shows previously used commands

  $ history

- **id** – Displays user ID (UID) and group ID (GID)
  - **For example :** You use this command to check your group and pi's ucid on Wulver

    $ id

- **groups** – Lists the groups a user belongs to

  $ groups

User Information and History

- **exit** – Closes the terminal session

  $ exit

- **clear** – Clears the terminal screen

  $ clear

- **alias** – Creates shortcuts for commands.
  - **For example :** Create a shortcut to clear the terminal screen (instead of typing clear every time):

    $ alias cls = "clear"

File Transfers & Synchronization

- When working on NJIT HPC, you often need to move files between your local computer and the cluster for processing.
- Common use cases include uploading datasets, scripts, software, and configurations, and downloading results or logs from your HPC jobs.
- **Large File Sizes:** Research datasets and simulation outputs can be massive, making efficient transfer methods essential.
- **Network Limitations:** Transferring files over SSH ensures security but can be slower over poor network connections.
- Key transfer methods include scp (Secure Copy Protocol), rsync (Remote Synchronization), sftp (Secure FTP), wget & curl

File Transfers & Synchronization

- **scp –** Securely copy files between your computer and NJIT HPC
  - Upload a file to HPC: $ scp my_script.py *ucid*@wulver.njit.edu:/home/*ucid*/
  - Download a file from HPC: $ scp *ucid*@wulver.njit.edu:/home/ucid/output.txt /path/to/local
- **rsync –** Efficient synchronization between local and HPC
  - Upload files while preserving timestamps and compressing data:

    $ rsync -avz my_data *ucid*@hpc.njit.edu:/home/ucid/
  - Download a file from HPC:

    $ rsync -avz *ucid*@hpc.njit.edu:/home/ucid/results/ ./results/
- **wget –** Download files from external sources to HPC

  $ wget https://example.com/dataset.csv

File Transfers & Synchronization

- **cp –** Copies files and directories.

```
$ cp file.txt Documents/Backup
```

- **mv –** Moves or renames files.

```
$ mv file.txt Documents/Archive
# moves files

$ mv file.txt new_name.txt
# rename file
```

Permissions & Ownership

- Linux was built to be secure and have permission functionality. Permissions determine who can read, write, or execute a file.
- Suppose there are multiple users on a machine, and only some users should be allowed to access a certain directory or file. In such cases, permissions are applied.
- Linux machines can have multiple users, and they can be grouped together. The system also has one administrator (root), who has the power to do anything.
- Files have a specific format for permissions:
  - r → Read
  - w → Write
  - x → Execute

Permissions & Ownership

- Permissions are grouped into three sets of three:
  - Owner of the file
  - Group associated with the file
  - Others (everyone else)
- For example, **rwx --w --r** means:
  - rwx → The owner can read, write, and execute the file.
  - --w → The group can write, but not read or execute.
  - --r → Others can only read the file.
- Permissions are also represented as numbers:
  - Read → 4
  - Write → 2
  - Execute → 1
- For example, rwx --w --r is represented as **724**
- The **d** at the beginning indicates a directory **(e.g., drwx---r--).**

Permissions & Ownership

- **ls -l :** Displays the detailed permissions for files and directories.

  $ ls -l myfolder

  drwxrwx--- 2 user1 group1 4096 Feb 21 12:00 myfolder

- **chmod :** Changes the permissions of a file or directory.

  $ chmod 700 myfolder

  # Now, only the owner (user1) can access the folder.

- **chmod :** Changes the permissions recursively of a directory.

  $ chmod -R 700 myfolder

  # Now, only the owner (user1) can access the folder.

  $ chmod u=rwx,g=rx,o= myfolder

  # Now, only the owner (user1) can access the folder.

Permissions & Ownership

- **chown :** Changes the owner and group of a file.

  $ chown user2:group2 myfolder

  # Changes ownership to user2 and group2.

- **chgrp :** Changes the group associated with a file or directory.

  $ chgrp group2 myfolder

  # Changes the group to group2.

Process Management

- In Linux, a process is simply a program that is running. Process management refers to how the system handles these programs, ensuring they run efficiently and without conflicts.
- Processes in Linux are assigned a unique PID (Process ID), and they can run in the **foreground** or **background**:
  - **Foreground processes** run in the terminal and block other tasks until they finish.
  - **Background processes** run independently, allowing you to do other work in the terminal.
- You can control processes using commands to check their status, kill unwanted processes, or even prioritize them.

Process Management

- **ps :** Displays a list of running processes in the current terminal session.

```
$ ps
PID TTY          TIME CMD
1234 pts/0    00:00:01 bash
2345 pts/0    00:00:00 ps
```

- **ps aux :** Lists all processes running on the system, including those from other users.

```
$ ps aux
USER       PID %CPU %MEM    VSZ   RSS      TTY     STAT START   TIME COMMAND
user1     1234  0.1  1.5  25000 1024 pts/0    Ss+   08:00   0:01    bash
user1     2345  0.0  0.1   4567   600   pts/0    R+    08:01   0:00    ps aux
```

- **kill PID:** Kills the process with the specified PID.

```
$ kill 1234
# kills the process with id 1234
$ kill -9 1234
# Forcefully kills a process if the normal kill doesn't work.
```

Environment Variables

- Think of environment variables as sticky notes that the system uses to remember important settings.
- They store information like your username, system paths, and configuration details that programs need to run properly.
- In HPC, environment variables help manage software, compilers, and job scheduling settings.
- They allow users to customize their system without modifying core files.
- Use **printenv** or **env** command to see all the environment variables:

```
$printenv
....
HOME=/home/user1
PATH=/bin:/usr/bin:/usr/local/bin:/usr/local/sbin
SHELL=/bin/bash
....
```

Environment Variables

- You can also check an environment variable by using echo.

  $ echo $PATH

  # This shows all directories where the system looks for commands.

- **Setting a Temporary Environment Variable:**
  - You can create a new variable for the current session. This will set MY_VARIABLE but will disappear when you close the terminal.

    $ export MY_VARIABLE="Hello, NJIT HPC!"
- **Making a Variable Permanent:**
  - To keep a variable across sessions, add it to **~/.bashrc** or **~/.bash_profile**

    $ echo 'export MY_VARIABLE="Hello, NJIT HPC!"' >> ~/.bashrc

    $ source ~/.bashrc
  - Now, the variable is available every time you log in!

Common Environment Variables

- **$USER:** Contains the username of the current user.
- **$HOME:** Refers to the home directory of the current user.
- **$PATH:** A colon-separated list of directories that the shell searches through to find executable files.
- **$SHELL:** Indicates the path to the current shell being used. (It's /bin/bash on Wulver)
- **$PWD:** Stands for "Print Working Directory" and gives the current directory the shell is operating in.
- **$LOGNAME:** The login name of the user. (Display's UCID on Wulver)

Scripting Basics

- A shell script is like a to-do list for your computer. Instead of typing commands one by one, you write them in a file, and the system runs them automatically.
- In NJIT HPC, shell scripts are often used to automate tasks, run programs, or submit jobs to a computing cluster.
- Saves time and reduces manual intervention.
- Shell scripts are written as plain text files that contain a series of commands to be executed by the shell.
- Key Components:
  - **Variables:** Store values for use later in the script.
  - **Loops:** Repeat tasks until a condition is met.
  - **Conditionals:** Execute commands based on conditions.

Example Script

- Let's learn this by taking an example of **Automating File Copying**
- **Why Automate?**
  - **Time Saver:** Instead of copying files one by one, a script does it all at once.
  - **Consistency:** Reduces the chance of human error—every file gets copied reliably.
  - **Efficiency:** In Machine Learning workflows, large datasets are common; automating file transfers ensures data is always in the right place.
- Create the script file using nano editor because its included in all linux versions by default. You can choose other editors as well like emacs or vim. This will open an editor where you can write your script.

  $ nano copy_files.sh
- First line tells shell to use the Bash program to run the script, ensuring the commands are interpreted correctly.

Example Script

- Write the script:

```
#!/bin/bash

# Define source and destination folders
SOURCE_DIR="/home/user1/raw_data"
DEST_DIR="/home/user1/processed_data"

# Create destination folder if it doesn't exist
mkdir -p "$DEST_DIR"

# Copy all CSV files from source to destination
cp "$SOURCE_DIR"/*.csv "$DEST_DIR"

echo "All CSV files have been copied from $SOURCE_DIR to $DEST_DIR"
```

Example Script

- Make the script executable:

  $ chmod +x copy_files.sh
- Finally run the script:

  $ ./copy_files.sh
- Researchers at NJIT HPC often need to move or backup data between folders. Automating this task means more time for actual research and less on repetitive manual work.
- We can further automate many tasks by using advanced scripting, and schedule them with cron to run automatically at regular intervals, eliminating the need for manual intervention.

Thank You

Any Questions?

Our website: https://hpc.njit.edu/
Our email: hpc@njit.edu