Week 1

Summer Class Handout

1 Basic Linux Commands

The following commands are essential for navigating and managing files in a Linux environment.

- **ls**: List files and directories in the current directory. Example: `ls -l` for detailed view.
- **cd**: Change directory. Example: `cd /home/user` navigates to `/home/user`.
- **pwd**: Print working directory. Example: `pwd` shows current path.
- **mkdir**: Create a new directory. Example: `mkdir new_folder`.
- **echo**: Write text to the screen or file. Example: `echo "Hello" > file.txt`.
- **touch**: Create a new empty file. Example: `touch newfile.txt`.
- **cat**: View contents of a file. Example: `cat file.txt`.
- »/«: Redirection operators. Example: `echo "Text" » file.txt` appends; `cat « EOF > file.txt` writes until `EOF`.
- |: Pipe operator. Example: `ls | grep .txt` filters for `.txt` files.
- **rm**: Remove files or directories. Example: `rm file.txt`; `rm -r folder` for directories.
- **mv**: Move or rename files/directories. Example: `mv file.txt newfile.txt`.
- **cp**: Copy files/directories. Example: `cp file.txt copy.txt`.
- **chmod**: Change file permissions. Example: `chmod 755 script.sh` for executable permissions.
- **sudo**: Run commands with superuser privileges. Example: `sudo apt-get update`.
- **grep**: Search text in files. Example: `grep "error" log.txt`.
- **find**: Locate files/directories. Example: `find . -name "*.txt"`.
- **df**: Display disk space usage. Example: `df -h` for human-readable format.

2 Basic Shell Scripting

Shell scripting automates tasks in Linux using bash. Below are key constructs and examples.

- **For Loop**:

  ```
  for file in *.csv; do
      echo "Processing␣$file"
  done
  ```

  Loops over all `.csv` files in the current directory.

- **Case Statement**:

  ```
  case $1 in
      start) echo "Starting␣service" ;;
      stop) echo "Stopping␣service" ;;
      *) echo "Unknown␣command" ;;
  esac
  ```

  Matches `$\1` against patterns.

- **Running a Python Script**:

  ```
  #!/bin/bash
  python3 script.py
  ```

  Ensure the script is executable (`chmod +x script.sh`).

- **Checking File Existence**:

  ```
  if [ -f "data.csv" ]; then
      echo "File␣exists"
  else
      echo "File␣not␣found"
  fi
  ```

  Uses `-f` to test for regular files.

- **Looping Over Files**:

  ```
  for file in /path/to/dir/*; do
      if [ -f "$file" ]; then
          echo "Found␣file:␣$file"
      fi
  done
  ```

  Iterates over files in a directory.

- **Argument Handling**:
  - `$\#`: Number of arguments passed. Example: `echo $\#` outputs argument count.
  - `$\@`: All arguments as a list. Example: `echo $\@` prints all arguments.

  - `$\1, $\2, ...`: Positional arguments. Example: `echo $\1` prints the first argument.

- **Downloading Files with wget/curl**:

```
wget https://cdn.wsform.com/wp-content/uploads/2020/06/color_srgb.csv -O
    local_file.csv
curl -o local_file.csv
    https://cdn.wsform.com/wp-content/uploads/2020/06/color_srgb.csv
```

  `wget` or `curl` downloads `file.csv` and saves as `local_file.csv`.

3 Basic Python Scripting

Python scripting is widely used in data engineering for automation and data processing. Below are equivalents to the shell scripting tasks.

- **For Loop**:

```
import os
for file in os.listdir("."):
    if file.endswith(".csv"):
        print(f"Processing␣{file}")
```

  Loops over `.csv` files in the current directory.

- **Conditional Statement**:

```
command = input("Enter␣command:␣")
if command == "start":
    print("Starting␣service")
elif command == "stop":
    print("Stopping␣service")
else:
    print("Unknown␣command")
```

  Handles user input with `if-elif-else`.

- **Running a Python Script**: Save as `script.py` and run with `python3 script.py`. Make executable:

```
#!/usr/bin/env python3
print("Hello,␣World!")
```

  Use `chmod +x script.py` and run with `./script.py`.

- **Checking File Existence**:

```
import os
if os.path.isfile("data.csv"):
    print("File␣exists")
else:
    print("File␣not␣found")
```

Uses `os.path.isfile`.

- **Looping Over Files**:

```
import os
for file in os.listdir("/path/to/dir"):
    if os.path.isfile(file):
        print(f"Found␣file:␣{file}")
```

  Iterates over files in a directory.

- **Argument Handling**:
  - `sys.argv`: List of arguments. Example: `len(sys.argv)` gives argument count.
  - `sys.argv[1], sys.argv[2], ...`: Positional arguments. Example:

```
import sys
print(f"First␣argument:␣{sys.argv[1]}")
```

- **Downloading Files**:

```
import requests
url = "https://cdn.wsform.com/wp-content/uploads/2020/06/color_srgb.csv"
response = requests.get(url)
with open("local_file.csv", "wb") as f:
    f.write(response.content)
```

  Uses `requests` to download and save `file.csv`.

4 Basic Git Commands

Git is a version control system for tracking code changes. Below are essential commands.

- **git init**: Initialize a new Git repository. Example: `git init` creates `.git` directory.
- **git checkout**: Switch branches or restore files. Example: `git checkout main`; `git checkout -b new-branch` creates and switches to `new-branch`.
- **git add**: Stage changes for commit. Example: `git add file.txt`; `git add .` for all changes.
- **git commit**: Persist staged changes. Example: `git commit -m "Add feature"`.
- **git merge**: Combine branches. Example: `git merge feature` merges `feature` into current branch.
- **git clone**: Copy a remote repository to local. Example: `git clone https://github.com/neotheob`
- **git push**: Push local changes to remote. Example: `git push origin main`.
- **git pull**: Fetch and merge remote changes. Example: `git pull origin main`.
- **Note**: Other commands like `git revert`, `git rebase`, `git restore`, and `git stash` exist for advanced workflows. Study these independently.