7-Oct-2024

Internship Day – 55 Report:

Managing users and groups:

1. User Management Commands:

- **useradd username:** Creates new user account with default settings like home directory, shell, and adds entry to /etc/passwd file.

```
Input:  useradd newuser
Output: Creates a new user account and updates `/etc/passwd`
-------------------------------------------------------
$ useradd newuser
$ tail -n 1 /etc/passwd
newuser:x:1002:1002::/home/newuser:/bin/bash
```

- **userdel username:** Removes user account and can optionally delete their home directory and mail spool.

```
Input:  userdel -r newuser
Output: Deletes user account and home directory
-------------------------------------------------------
$ userdel -r newuser
$ grep "newuser" /etc/passwd
(no output)
```

- **passwd username:** Sets or changes password for a user account, stores encrypted password in /etc/shadow.

```
Input:  passwd newuser
Output: Prompts to set or change user password
-------------------------------------------------------
$ passwd newuser
New password: ****
Retype new password: ****
passwd: password updated successfully
```

- **usermod:** Modifies existing user account settings including groups, home directory, shell, and account expiry.

```
Input:  usermod -s /bin/zsh newuser
Output: Changes user's default shell to Zsh
-----------------------------------------------------
$ usermod -s /bin/zsh newuser
$ grep "newuser" /etc/passwd
newuser:x:1002:1002::/home/newuser:/bin/zsh
```

2. Group Management:

- **groupadd groupname:** Creates a new group in the system and adds an entry to /etc/group file.

```
Input:  groupadd developers
Output: Creates a new group and updates `/etc/group`
-----------------------------------------------------
$ groupadd developers
$ tail -n 1 /etc/group
developers:x:1003:
```

- **groupdel groupname:** Removes an existing group from the system, fails if it's a primary group of any user.

```
Input:  groupdel developers
Output: Deletes the group
-----------------------------------------------------
$ groupdel developers
$ grep "developers" /etc/group
(no output)
```

- **usermod -aG groupname username:** Adds a user to supplementary groups while preserving existing group memberships.

```
Input:  usermod -aG developers newuser
Output: Adds user to supplementary group without affecting primary group
-----------------------------------------------------
$ usermod -aG developers newuser
$ groups newuser
newuser : newuser developers
```

3. Important Files:

- **/etc/passwd:** Stores essential user account information including username, UID, GID, home directory, and default shell.

```
Input:  cat /etc/passwd
Output: Lists all user accounts and details
-----------------------------------------------------
$ tail -n 1 /etc/passwd
newuser:x:1002:1002::/home/newuser:/bin/bash
```

- **/etc/shadow:** Contains encrypted password information and password policy settings for user accounts.

```
Input:  sudo cat /etc/shadow
Output: Displays encrypted password and policy (requires superuser)
-----------------------------------------------------
$ sudo tail -n 1 /etc/shadow
newuser:$6$randomsalt$encryptedpassword:::::::
```

- **/etc/group:** Maintains group information including group name, GID, and list of group members.

```
Input:  cat /etc/group
Output: Lists all system groups and their members
-----------------------------------------------------
$ tail -n 1 /etc/group
developers:x:1003:newuser
```

4. User Information:

- **id username:** Displays user's UID, GID, and all groups they belong to.

```
Input:  id newuser
Output: Displays UID, GID, and groups of the user
-----------------------------------------------------
$ id newuser
uid=1002(newuser) gid=1002(newuser) groups=1002(newuser),1003(developers)
```

- **whoami:** Shows the current username, useful when switching between users or verifying effective user.

```
Input:  whoami
Output: Displays the current logged-in user
-----------------------------------------------------
$ whoami
newuser
```

- **groups username:** Lists all groups that a specific user belongs to, including primary and supplementary groups.

```
Input:  groups newuser
Output: Lists all groups for a user
-----------------------------------------------------
$ groups newuser
newuser : newuser developers
```

8-Oct-2024

Internship Day – 56 Report:

Types of Files in linux:

Here are the main types of files in Linux with their descriptions:

File type	Description
Ordinary or regular files	Contain data of various content types such as text, script, image, videos, etc.
Directory files	Contain the name and address of other files.
Block or character special files	Represent device files such as hard drives, monitors, etc.
Link files	Point or mirror other files
Socket files	Provide inter-process communication
Named pipe files	Allow processes to send data to other processes or receive data from other processes.

1. Regular Files (-) :

- Most common file type that contains data, text, or program instructions
- Examples include text files, images, scripts, and binary executables

```
Input:  ls -l file.txt
Output: Shows file type as a regular file (-)
-----------------------------------------------
$ ls -l file.txt
-rw-r--r--  1 user group 1234 Nov 16 12:00 file.txt
```

2. Directory (d) :

- Special file that contains a list of filenames and related information
- Acts as a container for other files and directories, creating the filesystem hierarchy

```
Input:  ls -ld my_directory
Output: Shows file type as a directory (d)
-----------------------------------------------
$ ls -ld my_directory
drwxr-xr-x  2 user group 4096 Nov 16 12:00 my_directory
```

3. **Link Files (l) :**

- Symbolic links (soft links) that point to other files in the filesystem
- Acts like shortcuts in Windows, containing the path to the target file

```
Input:  ln -s file.txt link_file && ls -l link_file
Output: Creates a symbolic link and identifies it as a link (l)
-----------------------------------------------
$ ln -s file.txt link_file
$ ls -l link_file
lrwxrwxrwx  1 user group 8 Nov 16 12:00 link_file -> file.txt
```

4. **Character Device Files (c) :**

- Provide interface for serial I/O access to hardware devices
- Examples include terminals, keyboard, mouse, and serial ports

```
Input:  ls -l /dev/tty
Output: Identifies the terminal device as a character file (c)
-----------------------------------------------
$ ls -l /dev/tty
crw-rw-rw-  1 root tty 5, 0 Nov 16 12:00 /dev/tty
```

5. **Block Device Files (b) :**

- Provide interface for block I/O access to hardware devices
- Used for devices that store or hold data like hard drives and USB drives

```
Input:  ls -l /dev/sda
Output: Identifies the hard disk as a block device file (b)
-----------------------------------------------
$ ls -l /dev/sda
brw-rw----  1 root disk 8, 0 Nov 16 12:00 /dev/sda
```

6. **Socket Files (s) :**

- Enable communication between processes on the same machine

- Used for inter-process communication and networking

```
Input:  ls -l /run/docker.sock
Output: Identifies a socket file (s)
-----------------------------------------------
$ ls -l /run/docker.sock
srw-rw----  1 root docker 0 Nov 16 12:00 /run/docker.sock
```

7. **Named Pipes/FIFO (p) :**

- Special files that allow processes to communicate with each other
- Acts as a connection between two processes for data transfer

```
Input:  mkfifo my_pipe && ls -l my_pipe
Output: Creates a named pipe and identifies it as FIFO (p)
-----------------------------------------------
$ mkfifo my_pipe
$ ls -l my_pipe
prw-r--r--  1 user group 0 Nov 16 12:00 my_pipe
```

9-Oct-2024

Internship Day – 57 Report:

Introduction to Python

Python is a high-level, interpreted programming language that has gained immense popularity since its inception in the late 1980s. Developed by Guido van Rossum and first released in 1991, Python was designed with code readability and simplicity in mind. Its syntax is clear and intuitive, making it an excellent choice for both beginners and experienced programmers.

Why We Need Python

Ease of Learning:

- Python's straightforward syntax and structure allow beginners to grasp programming concepts quickly. This makes it an ideal first language for new programmers.

Versatility:

- Python is a multiparadigm language, supporting object-oriented, imperative, and functional programming styles. This versatility allows developers to choose the best approach for their projects.

Wide Range of Applications:

Python is used in various domains, including:

- **Web Development:** Frameworks like Django and Flask make it easy to build robust web applications.
- **Data Science and Machine Learning:** Libraries like Pandas, NumPy, and scikit-learn provide powerful tools for data analysis and machine learning.
- **Automation and Scripting:** Python is often used for writing scripts to automate repetitive tasks.
- **Game Development:** Libraries such as Pygame allow developers to create games.
- **Scientific Computing:** Python is widely used in scientific research and simulations with libraries like SciPy and Matplotlib.

Python has a large and active community of developers who contribute to its growth. This means abundant resources, including documentation, tutorials, and forums, are available to help learners and developers troubleshoot issues.

Rich Ecosystem of Libraries and Frameworks:

- Python has a vast ecosystem of libraries and frameworks that extend its functionality. This allows developers to leverage existing tools to accelerate development and focus on solving specific problems rather than reinventing the wheel.

Cross-Platform Compatibility:

- Python is compatible with various operating systems, including Windows, macOS, and Linux. This cross-platform nature allows developers to write code once and run it anywhere.

Integration Capabilities:

- Python can easily integrate with other languages like C, C++, and Java, allowing developers to use Python as a scripting language within larger applications.

Career Opportunities:

- With the rise of data science, machine learning, and web development, Python has become one of the most sought-after programming languages in the job market. Proficiency in Python can open doors to numerous career opportunities.

What is JSON?

JSON (JavaScript Object Notation) is a lightweight data interchange format that is easy for humans to read and write and easy for machines to parse and generate. It is primarily used for transmitting data between a server and a web application as an alternative to XML.

Key Characteristics of JSON:

- **Text-Based:** JSON is a text format that is completely language-independent but uses conventions that are familiar to programmers of the C family of languages (C, C++, Java, JavaScript, etc.).

- **Data Structure:** JSON supports simple data structures like objects (key-value pairs) and arrays (ordered lists), making it suitable for representing complex data.

- **Format:** JSON data is represented in a key-value pair format, where keys are strings and values can be strings, numbers, objects, arrays, booleans, or null.

What is YAML?

YAML (YAML Ain't Markup Language) is a human-readable data serialization format that is often used for configuration files and data exchange between languages with different data structures. It is designed to be easy to read and write, making it a popular choice for configuration files and data representation.

Key Characteristics of YAML:

- **Human-Readable:** YAML is designed to be easy for humans to read and write, with a focus on clarity and simplicity.

- **Data Structure:** YAML supports complex data structures, including scalars (strings, integers), sequences (lists), and mappings (key-value pairs).

- **Indentation-Based:** YAML uses indentation to represent structure, which makes it visually intuitive but requires careful attention to spacing.

What is a Variable?

A variable in programming is a named storage location in memory that can hold a value. It acts as a container for data, allowing you to store, modify, and retrieve information during the execution of a program. Variables are fundamental to programming because they enable you to work with dynamic data.

```
1 name = "Chandan"
2 print(name)  # Output: Chandan
```

10-Oct-2024

Internship Day – 58 Report:

What is an Operator?

An operator is a symbol or keyword in programming that performs a specific operation on one, two, or more operands (variables or values). Operators are fundamental to programming as they allow you to manipulate data and perform calculations.

Types of Operators in Python

Python supports several types of operators, which can be categorized as follows:

1. **Arithmetic Operators**
   Used to perform mathematical operations.

   Examples:
   - Addition (+)
   - Subtraction (-)
   - Multiplication (*)
   - Division (/)
   - Modulus (%)
   - Exponentiation (**)
   - Floor Division (//)

```python
a = 10
b = 3
print(a + b)  # Output: 13 (Addition)
print(a - b)  # Output: 7 (Subtraction)
print(a * b)  # Output: 30 (Multiplication)
print(a / b)  # Output: 3.3333 (Division)
print(a % b)  # Output: 1 (Modulus)
print(a ** b) # Output: 1000 (Exponentiation)
print(a // b) # Output: 3 (Floor Division)
```

2. **Comparison Operators**
   Used to compare two values and return a Boolean result (True or False).

   Examples:
   - Equal to (==)
   - Not equal to (!=)
   - Greater than (>)

- Less than (<)
- Greater than or equal to (>=)
- Less than or equal to (<=)

```
x = 5
y = 10
print(x == y)  # Output: False (Equal to)
print(x ≠ y)  # Output: True (Not equal to)
print(x > y)   # Output: False (Greater than)
print(x < y)   # Output: True (Less than)
print(x ⩾ y)  # Output: False (Greater than or equal to)
print(x ⩽ y)  # Output: True (Less than or equal to)
```

3. **Logical Operators:**
   Used to combine conditional statements.

   Examples:
   - Logical AND (and)
   - Logical OR (or)
   - Logical NOT (not)

```
a = True
b = False
print(a and b)  # Output: False (Logical AND)
print(a or b)   # Output: True (Logical OR)
print(not a)    # Output: False (Logical NOT)
```

4. **Assignment Operators:**
   Used to assign values to variables.

   Examples:
   - Assignment (=)
   - Add and assign (+=)
   - Subtract and assign (-=)
   - Multiply and assign (*=)
   - Divide and assign (/=)

- Modulus and assign (%=)

```
c = 10
c += 5    # Equivalent to c = c + 5
print(c)  # Output: 15
c -= 3    # Equivalent to c = c - 3
print(c)  # Output: 12
```

11-Oct-2024

Internship Day – 59 Report:

Lists in Python

A list in Python is a built-in data structure that allows you to store an ordered collection of items. Lists are versatile and can hold a variety of data types, including numbers, strings, and even other lists. They are mutable, meaning you can modify them after their creation (e.g., adding, removing, or changing elements).

Characteristics of Lists

1. **Ordered:** The items in a list have a specific order, and that order is preserved. You can access items by their index, which starts at 0.

2. **Mutable:** Lists can be changed in place. You can add, remove, or modify items in a list.

3. **Heterogeneous:** Lists can contain items of different data types.

4. **Dynamic:** The size of a list can change as you add or remove items.

Creating a List

You can create a list by placing comma-separated values inside square brackets [].

```
# Creating a list with various data types
name_list = ["Chandan", 25, 5.9, True]
print(name_list)  # Output: ['Chandan', 25, 5.9, True]
```

Accessing List Elements

You can access elements in a list using their index.

```
# Accessing elements
print(name_list[0])  # Output: Chandan
print(name_list[1])  # Output: 25
print(name_list[-1]) # Output: True (negative index counts from the end
```

Modifying Lists

You can modify lists using various methods:

Adding Elements:

- **append():** Adds an item to the end of the list.
- **insert():** Inserts an item at a specified index.
- **extend():** Adds multiple items to the end of the list.

```
# Adding elements
name_list.append("Developer")
print(name_list)  # Output: ['Chandan', 25, 5.9, True, 'Developer']

name_list.insert(1, "Age")
print(name_list)  # Output: ['Chandan', 'Age', 25, 5.9, True, 'Developer']
```

Slicing Lists

You can access a subset of a list using slicing.

```
# Slicing
name_list = ["Chandan", "Alice", "Bob", "Eve"]
print(name_list[1:3])  # Output: ['Alice', 'Bob'] (from index 1 to 2)
print(name_list[:2])    # Output: ['Chandan', 'Alice'] (from start to index 1)
print(name_list[2:])    # Output: ['Bob', 'Eve'] (from index 2 to end)
```