Welcome to LINUX World

Operating System (OS) Overview

- An Operating System (OS) is software that acts as an interface between computer hardware components and users.
- Every computer must have at least one operating system to run other programs.
- The operating system provides a platform/environment to run applications, such as a browser, notepad, MS Paint, calculator, etc.
- The operating system helps us communicate with the computer without knowing how to code in a computer language.
- Note: We can't use a computer without having an OS.

History of OS

- Operating systems were developed in the 1950s. General Motors' research lab implemented the first OS in the early 1950s for their IBM 701.
- In the mid-1960s, operating systems started using disks. In the late 1960s, the UNIX operating system was developed.
- The first OS developed by Microsoft was MS-DOS, released in 1981.
- In 1985, the first operating system with a GUI (Graphical User Interface) was released.
- We commonly use the Windows operating system on our computers and laptops.
- Windows OS is recommended for personal use.
- Windows is a single-user-based operating system.
- In real-time environments, we often use the Linux OS to run our applications.

Advantages of LINUX

- Linux is free and open source.
- Linux is a multi-user-based operating system.
- Linux is a highly secure operating system.

Explanation of the Linux User Scenario

- Servers might be present in the USA, but a developer can access those servers' OS from India at any time due to Linux's capabilities.
- Multiple users can connect to a Linux machine simultaneously, which is why it is called a multi-user-based operating system.
- AWS engineers or cloud engineers will set up Linux machines in the cloud.
- DevOps engineers will deploy our applications on the servers running on Linux OS.
- Developers will connect to Linux machines to check server/application logs.

History of LINUX

- In 1991, a student named Linus Torvalds developed the Linux OS.
- Linus Torvalds identified some challenges in the UNIX OS and suggested some changes, but the UNIX OS team rejected his suggestions.
- Linus Torvalds used Minix OS to develop Linux:
  - Linus + Minix
  - The first two letters from his name and the last three letters from Minix OS:
    - LI + NUX = LINUX
- Linus Torvalds released Linux OS with its source code to the market, allowing anyone to modify Linux OS. That's why it is called an open-source operating system.
- As Linux OS is open source, many people and companies have taken Linux OS, modified it according to their requirements, and released it into the market with different names called Linux distributions:
  - RHEL (Red Hat Enterprise Linux)
  - Ubuntu
  - CentOS
  - Fedora
  - OpenSUSE
  - Kali Linux
  - Debian

Environment Setup of Linux by AWS EC2 Using MobaXterm

Setting up a Linux environment on AWS EC2 using MobaXterm involves several steps, from creating an AWS account to launching and accessing your EC2 instance. Here is a detailed guide:

Step 1: Create an AWS Account

1. **Visit AWS**: Go to the AWS website and click on "Create an AWS Account."
2. **Sign Up**: Provide your email address and choose a password. Follow the prompts to complete the registration, including entering payment information. AWS offers a free tier for new users for the first 12 months.
3. **Verify Account**: AWS will send a verification email. Verify your account to activate it.

Step 2: Access the AWS Management Console

1. **Login**: Go to the AWS Management Console and sign in with your AWS credentials.
2. **Navigate to EC2**: In the search bar, type "EC2" and select it from the dropdown. This will take you to the EC2 Dashboard.

Step 3: Launch an EC2 Instance

1. **Launch Instance**: Click on the "Launch Instance" button on the EC2 Dashboard.
2. **Choose an Amazon Machine Image (AMI)**: Select an AMI that is Linux-based. The most common choice is "Amazon Linux 2 AMI."
3. **Choose an Instance Type**: For basic setups, the "t2.micro" instance is suitable and falls under the free tier.
4. **Configure Instance Details**:
   - Set the number of instances to 1.
   - Ensure the default network settings are selected.
   - You can leave other settings as default for basic setups.
5. **Add Storage**: The default settings usually suffice. The free tier includes 30 GB of EBS (Elastic Block Store) storage.
6. **Add Tags**: Optionally, add tags to your instance for better management. For example, add a "Name" tag with the value "MyLinuxInstance."
7. **Configure Security Group**:
   - Create a new security group or select an existing one.
   - Add a rule to allow SSH (port 22) from your IP address. You can find your IP address by searching "What is my IP" in a browser.
8. **Review and Launch**: Review all settings and click "Launch."
9. **Key Pair**: Create a new key pair or use an existing one. Download the private key file (.pem) and keep it secure. This file is used to access your instance.

Step 4: Access Your EC2 Instance Using MobaXterm

1. **Download MobaXterm**: If you haven't already, download and install MobaXterm from the official website.
2. **Launch MobaXterm**: Open MobaXterm on your local machine.
3. **Create a New Session**:
   - Click on the "Session" button in the top left corner.
   - Select "SSH" as the session type.
4. **Configure SSH Session**:
   - In the "Remote host" field, enter the Public DNS (IPv4) of your EC2 instance. You can find this in the EC2 Dashboard under "Instances."
   - Ensure the "Specify username" option is checked and enter `ec2-user` (or the appropriate username for your AMI).
   - Click on the "Advanced SSH settings" tab.
   - In the "Use private key" field, click the file browser button and select your downloaded `.pem` file.
5. **Start the Session**: Click "OK" to start the session. MobaXterm will use the provided information to connect to your EC2 instance.
6. **Accept Key Fingerprint**: The first time you connect, you'll be asked to accept the server's key fingerprint. Type `yes` and press Enter.
7. **Connected**: You should now be connected to your EC2 instance via SSH using MobaXterm. You can start running Linux commands and managing your instance.

Explanation of EC2 Service

Amazon EC2 (Elastic Compute Cloud) is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers.

Key Features of EC2

- **Scalability**: Easily scale up or down based on your requirements.
- **Flexibility**: Choose from a wide variety of instance types optimized for different use cases.
- **Pay-As-You-Go**: Pay only for the compute time you use, making it cost-effective.
- **Security**: Provides secure login information for instances using key pairs. Security groups act as a firewall to control traffic to instances.
- **Variety of Operating Systems**: Supports multiple operating systems including various Linux distributions and Windows Server.

Common Use Cases

- **Web Hosting**: Hosting websites and web applications.
- **Big Data Processing**: Running big data applications like Hadoop and Spark.
- **Development and Testing**: Setting up development and test environments quickly.
- **High-Performance Computing (HPC)**: Running high-performance computing applications.

With these steps and explanations, you should be able to set up and understand the basics of a Linux environment on AWS EC2, leveraging the powerful features of EC2 to meet your computing needs.

Summary

This guide has covered the essential aspects of setting up a Linux environment on AWS EC2, including:

- Creating an AWS account.
- Launching an EC2 instance.
- Accessing your instance using MobaXterm.
- Understanding the features and use cases of Amazon EC2.
- By following these steps, you'll be well on your way to effectively using AWS EC2 for your Linux-based projects and applications.

Comprehensive List of Linux Commands for DevOps

Basic Linux Commands

1. **$ pwd**
   - **Description**: Print Working Directory.
   - **Usage**: `pwd`
   - **Example**: Displays the current directory path.
2. **$ ls**
   - **Description**: List directory contents.
   - **Usage**: `ls [options] [directory]`
   - **Options**:
     - `-a`: Include hidden files.
     - `-l`: Long listing format.
     - `-h`: Human-readable sizes.
     - `-r`: Reverse order.
     - `-t`: Sort by modification time.
     - `-R`: Recursively list subdirectories.
   - **Examples**:
     - `ls -l`: Long listing format.
     - `ls -r`: List in reverse order.
     - `ls -t`: Sort by modification time.
     - `ls -lt`: Long listing, sorted by modification time.
     - `ls -rt`: Reverse order, sorted by modification time.
     - `ls -a`: Include hidden files.
     - `ls -li`: List with inode number.
     - `ls -R`: Recursively list subdirectories.
     - `ls -lR`: Long listing format, recursively.
3. **$ cd**
   - **Description**: Change Directory.
   - **Usage**: `cd [directory]`
   - **Example**: `cd /var/log`
4. **$ cp**
   - **Description**: Copy files and directories.
   - **Usage**: `cp [options] source destination`
   - **Options**:
     - `-r`: Recursive copy (for directories).
   - **Example**: `cp file1.txt /tmp`
5. **$ mv**
   - **Description**: Move/rename files and directories.
   - **Usage**: `mv [options] source destination`
   - **Example**: `mv oldname.txt newname.txt`
6. **$ rm**

- **Description**: Remove files or directories.
- **Usage**: `rm [options] file`
- **Options**:
  - `-r`: Recursive (for directories).
  - `-f`: Force deletion.
- **Example**: `rm -rf /tmp/testdir`

7. **$ mkdir**
   - **Description**: Create directories.
   - **Usage**: `mkdir [options] directory`
   - **Options**:
     - `-p`: Create parent directories as needed.
   - **Example**: `mkdir -p /tmp/testdir`
8. **$ rmdir**
   - **Description**: Remove empty directories.
   - **Usage**: `rmdir directory`
   - **Example**: `rmdir /tmp/testdir`
9. **$ touch**
   - **Description**: Create empty files or update the timestamp of existing files.
   - **Usage**: `touch filename`
   - **Example**: `touch newfile.txt`
10. **$ cat**
    - **Description**: Concatenate and display files.
    - **Usage**: `cat [options] file`
    - **Example**: `cat /etc/passwd`
11. **$ tac**
    - **Description**: Concatenate and display files in reverse.
    - **Usage**: `tac file`
    - **Example**: `tac /etc/passwd`
12. **$ rev**
    - **Description**: Reverse lines characterwise.
    - **Usage**: `rev file`
    - **Example**: `rev /etc/passwd`
13. **$ more**
    - **Description**: View file contents one page at a time.
    - **Usage**: `more filename`
    - **Example**: `more /etc/passwd`
14. **$ less**
    - **Description**: View file contents with backward/forward navigation.
    - **Usage**: `less filename`
    - **Example**: `less /var/log/syslog`
15. **$ head**
    - **Description**: Display the first lines of a file.
    - **Usage**: `head [options] file`

- **Options**:
  - `-n N`: Show the first N lines.
- **Example**: `head -n 10 /var/log/syslog`

16. **$ tail**
    - **Description**: Display the last lines of a file.
    - **Usage**: `tail [options] file`
    - **Options**:
      - `-n N`: Show the last N lines.
      - `-f`: Follow the file (useful for logs).
    - **Example**: `tail -f /var/log/syslog`
17. **$ chmod**
    - **Description**: Change file modes or Access Control Lists (ACLs).
    - **Usage**: `chmod [options] mode file`
    - **Options**:
      - `u`: User.
      - `g`: Group.
      - `o`: Others.
    - **Example**: `chmod 755`
18. **$ chown**
    - **Description**: Change file owner and group.
    - **Usage**: `chown [options] owner[:group] file`
    - **Example**: `chown root:root /var/www/html`
19. **$ df**
    - **Description**: Report file system disk space usage.
    - **Usage**: `df [options]`
    - **Options**:
      - `-h`: Human-readable format.
    - **Example**: `df -h`
20. **$ du**
    - **Description**: Estimate file space usage.
    - **Usage**: `du [options] [directory]`
    - **Options**:
      - `-h`: Human-readable format.
      - `-s`: Summary of directory usage.
    - **Example**: `du -sh /var/log`
21. **$ find**
    - **Description**: Search for files in a directory hierarchy.
    - **Usage**: `find [path] [expression]`
    - **Example**: `find /var -name "*.log"`
22. **$ grep**
    - **Description**: Print lines matching a pattern.
    - **Usage**: `grep [options] pattern [file]`
    - **Options**:

        - `-i`: Ignore case.
        - `-R`: Recursive search.
        - `-c`: Count the number of matches.
        - `-n`: Show line numbers with output lines.
        - `-l`: List filenames containing the pattern.
        - `-h`: Suppress filename in output.
    - **Examples**:
        - `grep -i "error" /var/log/syslog`: Ignore case.
        - `grep -R "error" /var/log`: Recursive search.
        - `grep -c "error" /var/log/syslog`: Count matches.
        - `grep -n "error" /var/log/syslog`: Show line numbers.
        - `grep -l "error" /var/log/*`: List filenames containing the pattern.
        - `grep -h "error" /var/log/syslog`: Suppress filename in output.

23. **$ diff**
    - **Description**: Compare files line by line.
    - **Usage**: `diff [options] file1 file2`
    - **Example**: `diff file1.txt file2.txt`
24. **$ ps**
    - **Description**: Report a snapshot of current processes.
    - **Usage**: `ps [options]`
    - **Options**:
        - `-e`: All processes.
        - `-f`: Full format listing.
    - **Example**: `ps -ef`
25. **$ top**
    - **Description**: Display tasks and system resource usage.
    - **Usage**: `top`
    - **Example**: Shows real-time system processes and resource usage.
26. **$ kill**
    - **Description**: Send a signal to a process.
    - **Usage**: `kill [options] pid`
    - **Example**: `kill 1234`
27. **$ history**
    - **Description**: Show command history.
    - **Usage**: `history`
    - **Example**: Displays previously executed commands.
28. **$ sudo**
    - **Description**: Execute a command as another user (usually root).
    - **Usage**: `sudo command`
    - **Example**: `sudo apt-get update`
29. **$ apt-get**
    - **Description**: APT package handling utility (for Debian-based systems).
    - **Usage**: `apt-get [options] command`

- **Example**: `sudo apt-get install nginx`

30. **$ yum**
    - **Description**: Package manager (for Red Hat-based systems).
    - **Usage**: `yum [options] command`
    - **Example**: `sudo yum install httpd`

Advanced Linux Commands

1. **$ awk**
    - **Description**: Pattern scanning and processing language.
    - **Usage**: `awk 'pattern {action}' file`
    - **Example**: `awk '{print $1}' /etc/passwd`
2. **$ sed**
    - **Description**: Stream editor for filtering and transforming text.
    - **Usage**: `sed [options] 'script' file`
    - **Example**: `sed 's/old/new/g' file.txt`
3. **$ tar**
    - **Description**: Archive files.
    - **Usage**: `tar [options] [archive] [file/directory]`
    - **Options**:
        - `-c`: Create an archive.
        - `-x`: Extract an archive.
        - `-v`: Verbose mode.
        - `-f`: File name.
        - `-z`: Compress with gzip.
        - `-j`: Compress with bzip2.
    - **Examples**:
        - `tar -czvf archive.tar.gz /path/to/directory`
        - `tar -xzvf archive.tar.gz`
4. **$ gzip**
    - **Description**: Compress files.
    - **Usage**: `gzip [options] file`
    - **Example**: `gzip file.txt`
5. **$ gunzip**
    - **Description**: Decompress files.
    - **Usage**: `gunzip [options] file`
    - **Example**: `gunzip file.txt.gz`
6. **$ iptables**
    - **Description**: Administration tool for IPv4 packet filtering and NAT.
    - **Usage**: `iptables [options]`
    - **Example**: `sudo iptables -L`
7. **$ docker**
    - **Description**: Manage Docker containers.

- **Usage**: `docker [command]`
- **Commands**:
  - `docker run`: Run a container.
  - `docker ps`: List running containers.
  - `docker stop`: Stop a running container.
- **Example**: `docker run -d nginx`

8. **$ kubectl**
   - **Description**: Kubernetes command-line tool.
   - **Usage**: `kubectl [command]`
   - **Commands**:
     - `kubectl get`: List resources.
     - `kubectl apply`: Apply a configuration to a resource.
     - `kubectl delete`: Delete resources.
   - **Example**: `kubectl get pods`
9. **$ ansible**
   - **Description**: Automation tool for managing servers.
   - **Usage**: `ansible [command]`
   - **Commands**:
     - `ansible-playbook`: Run an Ansible playbook.
     - `ansible-inventory`: Display or dump the configured inventory.
   - **Example**: `ansible-playbook site.yml`
10. **$ git**
    - **Description**: Version control system.
    - **Usage**: `git [command]`
    - **Commands**:
      - `git clone`: Clone a repository.
      - `git commit`: Commit changes.
      - `git push`: Push changes to a remote repository.
    - **Example**: `git clone`
11. **$ vagrant**
    - **Description**: Tool for building and managing virtualized development environments.
    - **Usage**: `vagrant [command]`
    - **Commands**:
      - `vagrant init`: Initialize a Vagrant environment.
      - `vagrant up`: Start the Vagrant environment.
      - `vagrant halt`: Stop the Vagrant environment.
    - **Example**: `vagrant up`
12. **$ terraform**
    - **Description**: Infrastructure as code tool.
    - **Usage**: `terraform [command]`
    - **Commands**:
      - `terraform init`: Initialize a configuration.

        - `terraform apply`: Apply changes.
        - `terraform destroy`: Destroy the infrastructure.
    - **Example**: `terraform apply`

Command Combinations and Utilities

1. **Combining Commands with Pipes (`|`)**
    - **Example**: `ps -ef | grep nginx`
    - **Description**: List all processes and filter for nginx.
2. **Using Redirection (>, >>, <)**
    - **Example**: `ls -l > filelist.txt`
    - **Description**: Save the output of `ls -l` to `filelist.txt`.
    - **Example**: `cat file1.txt >> file2.txt`
    - **Description**: Append the contents of `file1.txt` to `file2.txt`.
3. **Chaining Commands with `&&`**
    - **Example**: `mkdir newdir && cd newdir`
    - **Description**: Create a new directory and change into it.
4. **Using Subshells with `()`**
    - **Example**: `(cd /tmp && ls)`
    - **Description**: Change to `/tmp`, list files, and return to the original directory.
5. **Command Substitution with `$()`**
    - **Example**: `echo "The date is $(date)"`
    - **Description**: Insert the output of `date` into the echo statement.

By understanding these commands and their possibilities, you'll be well-equipped to perform a wide range of tasks in a Linux environment, from basic file operations to advanced system administration and DevOps workflows.

Working with Text Editors in Linux

VI Editor

The VI editor is a powerful text editor available on most Unix-like systems. It operates in different modes, each serving a specific purpose. Here's a detailed guide to using VI:

VI Modes

1. **Normal Mode**: This is the default mode when you open VI. In this mode, you can navigate the file, delete text, and perform other editing tasks.
2. **Insert Mode**: In this mode, you can insert text into the file. You switch to this mode from Normal mode.
3. **Visual Mode**: This mode allows you to select text.

4. **Command-Line Mode**: Used for executing commands like saving and quitting.

Switching Between Modes

- **Normal Mode to Insert Mode**:
  - Press `i` to insert text before the cursor.
  - Press `I` to insert text at the beginning of the line.
  - Press `a` to append text after the cursor.
  - Press `A` to append text at the end of the line.
- **Insert Mode to Normal Mode**:
  - Press `Esc` to return to Normal mode.
- **Normal Mode to Visual Mode**:
  - Press `v` to start visual selection.
  - Press `V` for line-wise visual selection.
- **Normal Mode to Command-Line Mode**:
  - Press `:` to enter Command-Line mode.

Example: Creating and Editing a File with VI

1. **Create a File**:
2. **Insert Text**:
   - Press `i` to enter Insert mode.
   - Type your text.
   - Press `Esc` to return to Normal mode.
3. **Save and Exit**:
   - Press `:` to enter Command-Line mode.
   - Type `wq` (write and quit) and press `Enter`.
4. **Navigating Text**:
   - `k`: Move up one line.
   - `j`: Move down one line.
   - `h`: Move left one character.
   - `l`: Move right one character.
   - `$`: Move to the end of the line.

Using `sed` Command

`sed` is a stream editor used for parsing and transforming text. It processes input line by line and applies specified commands to each line.

Basic Syntax

```
sed [options] 'command' file
```

Common Commands and Options

1. **Substitute (`s`)**
   - **Description**: Replaces the first occurrence of `pattern` with `replacement` in each line.
   - **Example**: Replace `apple` with `orange`.

     ```
     sed 's/apple/orange/' file
     ```

   - **Global Replacement (`g`)**: Replaces all occurrences in each line.

     ```
     sed 's/apple/orange/g' file
     ```

   - **Case-Insensitive Replacement (`I`)**: Makes the search case-insensitive.

     ```
     sed 's/apple/orange/I' file
     ```

   - **Only the Nth Occurrence (`N`)**: Replaces only the Nth occurrence.

     ```
     sed 's/apple/orange/2' file
     ```

2. **Delete (`d`)**
   - **Description**: Deletes lines that match the pattern.
   - **Example**: Delete lines containing `error`.

     ```
     sed '/error/d' file
     ```

   - **Delete Specific Line**: Deletes a specific line number.

     ```
     sed '3d' file
     ```

3. **Print (`p`)**
   - **Description**: Prints lines that match the pattern.
   - **Example**: Print lines containing `warning`.

     ```
     sed -n '/warning/p' file
     ```

4. **Append (`a`)**
   - **Description**: Appends `text` after lines that match the pattern.
   - **Example**: Append `End of file` after lines containing `EOF`.

     ```
     sed '/EOF/a\End of file' file
     ```

5. **Insert (`i`)**

- **Description**: Inserts `text` before lines that match the pattern.
- **Example**: Insert `Start of file` before lines containing `HEADER`.

```
sed '/HEADER/i\Start of file' file
```

6. **Change (`c`)**
   - **Description**: Replaces lines that match the pattern with `text`.
   - **Example**: Replace lines containing `old_text` with `new_text`.

   ```
   sed '/old_text/c\new_text' file
   ```

7. **Transform (`y`)**
   - **Description**: Replaces characters in `old_chars` with corresponding characters in `new_chars`.
   - **Example**: Replace characters `abc` with `123`.

   ```
   sed 'y/abc/123/' file
   ```

8. **Print Line Numbers (`=`)**
   - **Description**: Prints line numbers along with lines.
   - **Example**: Print line numbers for all lines.

   ```
   sed '=' file
   ```

9. **Replace and Save (`-i`)**
   - **Description**: Edits files in place (saves changes to the file).
   - **Example**: Replace `foo` with `bar` and save the changes.

   ```
   sed -i 's/foo/bar/g' file
   ```

Advanced Usage

1. **Multiple Commands**
   - **Description**: Applies multiple `sed` commands to each line.
   - **Example**: Replace `foo` with `bar`, then delete lines containing `baz`.

   ```
   sed -e 's/foo/bar/g' -e '/baz/d' file
   ```

2. **Address Ranges**
   - **Description**: Applies commands only within the specified range of lines.
   - **Example**: Replace `apple` with `orange` from lines 2 to 5.

   ```
   sed '2,5s/apple/orange/g' file
   ```

3. **Using Patterns**
   - **Description**: Applies commands only to lines matching `pattern`.

- **Example**: Replace `foo` with `bar` only on lines containing `baz`.

  ```
  sed '/baz/s/foo/bar/g' file
  ```

4. **In-place Editing with Backup**
   - **Description**: Edits files in place with a backup of the original file.
   - **Example**: Replace `foo` with `bar` and keep a backup of the original file.

     ```
     sed -i.bak 's/foo/bar/g' file
     ```

Summary

- `sed` is a powerful tool for text processing, offering various commands to search, replace, delete, and manipulate text.
- Understanding `sed` commands can simplify text processing tasks and automate transformations.

File Permissions in Linux

File permissions determine who can read, write, or execute a file. Permissions are represented by the characters `r`, `w`, and `x`, and are applied to the file owner, group, and others. Each permission allows different types of access and actions on the file.

Permission Symbols

1. **`r` (Read)**
   - **Description:** Grants the ability to view the contents of the file.
   - **Access:** When read permission is granted, users can open and view the file's content but cannot modify it.
   - **Usage Example:** To list the contents of a file or to view a configuration file, the read permission is necessary.
2. **`w` (Write)**
   - **Description:** Allows modification of the file's content.
   - **Access:** Users with write permission can change or overwrite the file's content. This includes adding or deleting data within the file.
   - **Usage Example:** To edit a text file or update a configuration file, write permission is required.
3. **`x` (Execute)**
   - **Description:** Enables running the file as a program or script.
   - **Access:** When execute permission is granted, users can run the file if it is a script or binary executable. This does not grant permission to modify or view the file's content.
   - **Usage Example:** To run a shell script or a compiled program, execute permission is needed.

Permission Combinations

Permissions are applied to three categories:

- **Owner (user):** The file's owner, who can be granted any combination of `r`, `w`, and `x` permissions.
- **Group:** Users who are members of the file's group, with permissions that can differ from the owner.
- **Others:** All other users, with permissions that can be set separately from the owner and group.

Examples of Permission Combinations

- **`r--` (Read Only)**
  - Grants read access to the file but not write or execute permissions.
  - Users can open and view the file’s content but cannot modify or run it.
- **`rw-` (Read and Write)**
  - Grants read and write access to the file, but not execute permission.
  - Users can view and modify the file’s content but cannot run it as a script or program.
- **`r-x` (Read and Execute)**
  - Grants read and execute access but not write permission.
  - Users can view the file’s content and execute it (if it is a script or program) but cannot modify it.
- **`rw-x` (Read, Write, and Execute)**
  - Grants full access: read, write, and execute.
  - Users can view, modify, and run the file.

Numeric Permissions and Their Symbolic Equivalents

In Linux, file permissions are also represented numerically using octal values (0-7), which correspond to the combination of `r`, `w`, and `x` permissions:

- **0** - No Permissions (`---`): No access to the file.
- **1** - Execute Only (`--x`): Only execute permission granted.
- **2** - Write Only (`-w-`): Only write permission granted.
- **3** - Write and Execute (`-wx`): Both write and execute permissions granted.
- **4** - Read Only (`r--`): Only read permission granted.

- **5** - Read and Execute (`r-x`): Both read and execute permissions granted.
- **6** - Read and Write (`rw-`): Both read and write permissions granted.
- **7** - Read, Write, and Execute (`rwx`): All permissions granted.

Combining Permissions

Permissions are specified for three categories (owner, group, others) and are combined into a single octal number. For example:

- **`chmod 755 filename`** translates to `rwxr-xr-x`:
  - **Owner**: `rwx` (Read, Write, Execute)
  - **Group**: `r-x` (Read, Execute)
  - **Others**: `r-x` (Read, Execute)
- **`chmod 644 filename`** translates to `rw-r--r--`:
  - **Owner**: `rw-` (Read, Write)
  - **Group**: `r--` (Read)
  - **Others**: `r--` (Read)

Understanding these permissions helps in correctly setting file access controls to ensure appropriate access levels are granted to different users.

Working with User Accounts in Linux

Creating a User Account to create a new user account,
use: `sudo useradd username`
Options:

- `-m`: Create a home directory.
- `-s`: Specify the shell (e.g., `/bin/bash`).
- `-G`: Add the user to additional groups.

Example: `sudo useradd -m -s /bin/bash -G sudo username`

Deleting a User Account to delete a user account,
use: `sudo userdel username`
Options:

- `-r`: Remove the home directory and mail spool.

Example: `sudo userdel -r username`

Renaming a User Account to rename a user account,
use: `sudo usermod -l newusername oldusername`
Options:

- `-l`: Specify the new login name.

Changing a User's Password To change a user's password,
use: `sudo passwd username`

Listing User Account to list all user accounts,
view the `/etc/passwd` file:`cat /etc/passwd` Or
use: `cut -d: -f1 /etc/passwd`

Working with Groups in Linux

Creating a Group to create a new group,
use: `sudo groupadd groupname`

Deleting a Group to delete a group,
use: `sudo groupdel groupname`

Renaming a Group to rename a group,
use: `sudo groupmod -n newgroupname oldgroupname`
Options:

- `-n`: Specify the new group name.

Adding a User to a Group to add a user to a group,
use: `sudo usermod -aG groupname username`
Options:

- `-aG`: Append the user to the specified group.

Removing a User from a Group to remove a user from a group,
use: `sudo gpasswd -d username groupname`

Listing Group Members to list the members of a group,
use: `getent group groupname`

Working with Find and Locate Commands

Find Command Searches for files and directories within a directory hierarchy.
Basic Syntax: `find [path] [options] [expression]`Examples:

- Find files by name: `find /path/to/search -name filename`
- Find files by type (e.g., directories): `find /path/to/search -type d`
- Find files by size: `find /path/to/search -size +1G`
- Find files modified in the last 7 days: `find /path/to/search -mtime -7`

Locate Command Finds files by name using a pre-built index database.
Basic Syntax: `locate [filename]`
Examples:

- Locate a file: `locate filename`
- Update the database: `sudo update db`

Common Network and System Commands

Ping Command Checks the reachability of a host on a network.
Basic Syntax: `ping [hostname or IP address]`
Example: `ping`
Man Command Displays the manual pages for commands.
Basic Syntax: `man [command]`
Example: `man ls`
Curl Command Transfers data from or to a server using various protocols.
Basic Syntax: `curl [options] [URL]`
Examples:

- Fetch a webpage: `curl`
- Download a file: `curl -O`

Wget Command Retrieves files from the web.
Basic Syntax: `wget [options] [URL]`
Examples:

- Download a file: `wget`
- Download a file and save it under a different name: `wget -O newfile.txt`