Linux Basics
Lesson 4

Linux System Management

Importance of System Management

- Efficient system management is essential for optimal performance, stability, and security.
- Proper system management helps prevent unauthorized access, data breaches, and system failures.
- It ensures the availability of resources, efficient resource allocation, and effective troubleshooting.
- System management is crucial for maintaining compliance with security standards and regulatory requirements.

Key Components of Linux System Management

a) Users and Groups

- Users and groups are fundamental entities in Linux system management.
- Users represent individuals who interact with the system, while groups help organize and manage users.
- User and group management involves creating, modifying, and removing user accounts, assigning proper permissions, and enforcing security policies.

b) Permissions

- Permissions control access to files and directories in Linux.
- Understanding and managing permissions is critical for maintaining data integrity and enforcing security.
- Permissions determine who can read, write, or execute files and directories, and they play a role in maintaining proper separation of privileges.

Key Components of Linux System Management

c) Package Management

- Package management involves the installation, upgrading, and removal of software packages in a Linux system.
- Package managers handle dependencies, ensure software integrity, and provide centralized management of software packages.
- Effective package management streamlines software installations, updates, and maintenance tasks.

d) Log Management

- Logs contain valuable information about system events, errors, and activities.
- Log management involves collecting, storing, analyzing, and interpreting logs to understand system behavior, troubleshoot issues, and ensure compliance.
- Proper log management includes log rotation, setting up centralized log servers, and utilizing log analysis tools.

Understanding Users & Groups

Users & Groups in Linux

Ownership

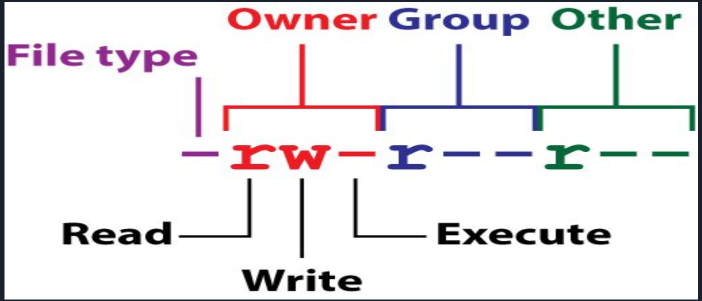
- Every file and process in Linux is associated with a specific user and group
- User ID (UID) and Group ID (GID) are numerical representations of users and groups

Permissions and Access Control

- Linux utilizes users and groups to manage permissions and access control for files and directories.
- File permissions determine what actions can be performed on a file, such as reading, writing, or executing.
- Permissions can be set separately for the owner (user), the group, and others (everyone else).

Understanding file permissions

- Every File permissions are represented by a combination of read (r), write (w), and execute (x) flags.
- Symbolic notation (rwx) and octal notation (0-7) are used to represent the permissions.
- For example, "rw-r--r--" indicates that the owner has read and write permissions, while the group and others have only read permissions.



- 0: No permission (---)
- 1: Execute permission (--x)
- 2: Write permission (-w-)
- 3: Write and execute permissions (-wx)
- 4: Read permission (r--)
- 5: Read and execute permissions (r-x)
- 6: Read and write permissions (rw-)
- 7: Read, write, and execute permissions (rwx)

Managing User and Group Permissions

- The **chmod** command is used to modify file permissions, allowing users to grant or revoke certain rights.
- The **chown** command is used to change the ownership of a file or directory, transferring it to a different user or group.

Default Permissions and umask

- Linux has default permissions for newly created files and directories.
- The **umask** command is used to set the default permissions for new files and directories.
- The umask value subtracts permission bits from the default permissions, allowing for customization.

User and Group Management

- System administrators can create and manage user accounts using tools like useradd and usermod.
- Groups can be created and managed using commands like groupadd and groupmod.
- Assigning users to specific groups enables effective access control and resource sharing.

Useradd Utility

- The useradd utility is used to create new user accounts in Linux.
- It creates a new entry in the system's user database, assigns a unique User ID (UID) to the user, and sets default configurations.
- Common options used with useradd include -m (create home directory), -s (specify login shell), and -G (assign additional groups).

Example: Creating a New User

- To create a new user named "john" with a home directory and default settings, use the command:
  - useradd -m john

Groupadd Utility

- The `groupadd` utility is used to create new groups in Linux.
- It creates a new entry in the system's group database, assigns a unique Group ID (GID) to the group, and sets default configurations.
- Groupadd can be used to create primary groups or additional groups for users.

Example: Creating a New Group

To create a new group named "developers," use the command:

```
groupadd developers
```

Usermod Utility

- The usermod utility is used to modify existing user accounts in Linux.
- It allows administrators to change various attributes of a user, such as the username, home directory, login shell, or group membership.
- Usermod is a powerful tool for managing user accounts efficiently.

Example: Modifying User Attributes

- To change the home directory and default shell for the user "john," use the command:
  - usermod -d /home/john_new -s /bin/bash john

Concepts of User ID, Group ID, and Stickiness Bit

User ID & Group ID

- User ID (UID) and Group ID (GID) are numerical representations assigned to users and groups in Linux.
- They play a crucial role in file permissions and access control, ensuring proper security and management.
- Every user in Linux is associated with a unique UID, while each group has a unique GID.
- UIDs and GIDs are used internally by the system to identify and manage user and group ownership.

File Permissions & UIDs/GIDs

- File permissions in Linux are based on the owner, group, and others (world) concept.
- UIDs and GIDs are associated with file ownership, allowing the system to enforce access control.

Examples of File Permissions

- File permissions are represented by a combination of read (r), write (w), and execute (x) permissions.
- They are denoted for the owner, group, and others as a sequence of three characters.

Stickiness Bit

- The stickiness bit (represented by the letter "t" in the permission string) has a special role in file and directory permissions.
- When set on a directory, it ensures that only the owner of a file can delete or rename it, even if other users have write permissions.
- The primary purpose of the stickiness bit is to enable the creation of directories where multiple users or processes can write files but cannot delete or modify files created by other users.
- A typical example is the /tmp directory, which is commonly set with the sticky bit. This ensures that users can create and modify their own temporary files within /tmp, but they cannot delete or tamper with files belonging to other users.

Stickiness Bit

Examples of Stickiness Bit

- To view file and directory permissions along with the stickiness bit, use the ls -l command.
- To set the stickiness bit on a directory named "data," use the command:
    - chmod +t data

Changing UIDs and GIDs

- The chown command is used to change the owner and group of a file or directory.
- To change the owner of a file named "myfile" to user "john," use the command:
    - chown john myfile

Understanding Linux Permissions: chmod, chown

Linux permission model

- The Linux permission model governs access control to files and directories, ensuring proper security and control.
- Permissions are defined for the owner, group, and others, specifying read, write, and execute privileges.

'chmod' Command

- The 'chmod' command is used to change the permissions of files and directories in Linux.
- It allows users to modify the read, write, and execute permissions for the owner, group, and others.

Changing Permissions with 'chmod'

- To change the permissions of a file named 'myfile' to allow read and write access for the owner, use the command: chmod u+rw myfile
- To remove execute permissions for the group and others, use the command: chmod go-x myfile

Changing Ownership with 'chown'

- The 'chown' command is used to change the owner and group of files and directories in Linux.
- It allows users to transfer ownership from one user to another or change the group affiliation.

Changing Ownership with 'chown' (contd.)

- To change the owner of a file named 'myfile' to the user 'john,' use the command:
  - chown john myfile

Symbolic and Numeric Modes in 'chmod'

- 'chmod' supports two modes for specifying permissions: symbolic and numeric.
- Symbolic mode uses letters to represent permissions, while numeric mode uses a three-digit octal value.

Symbolic Mode Examples

- To give the group read and execute permissions for a file named 'myfile,' use the command:
  - chmod g+rx myfile

Numeric Mode Examples

- To set read, write, and execute permissions for the owner, read and execute for the group, and read-only for others, use the command:
  - chmod 754 myfile

Special File Permissions: suid, sgid, and the Sticky Bit

SUID (Set User ID)

In addition to standard file permissions, Linux also provides special permissions: suid, sgid, and the sticky bit. These special permissions allow for additional control and functionality on files and directories.

SUID (Set User ID)

- SUID is a special permission that allows a user to execute a file with the permissions of the file owner.
- When a file with SUID is executed, it runs with the owner's privileges, regardless of the user executing it.
- Common use cases for SUID include allowing users to execute certain programs with elevated privileges.
- To set the SUID permission on a file named 'myprog' owned by 'root,' use the command:
  - chmod u+s myprog

SGID (Set Group ID)

- SGID is a special permission that allows a user to execute a file with the permissions of the file group.
- When a file with SGID is executed, it runs with the group's privileges, regardless of the user executing it.
- Common use cases for SGID include shared directories or files where multiple users need access to a common group.

Setting SGID Permission

- To set the SGID permission on a directory named 'shared' owned by 'group1,' use the command:
- chmod g+s shared

Sticky Bit

- The sticky bit is a special permission that can be applied to directories.
- When the sticky bit is set on a directory, only the owner of a file within that directory can delete or rename the file.
- Common use cases for the sticky bit include shared directories where users need to be able to create files but not delete or modify others' files.

Setting Sticky Bit Permission

- To set the sticky bit on a directory named 'public,' use the command: chmod +t public

Viewing Special Permissions

- To view the special permissions of a file or directory, use the 'ls' command with the '-l' option: ls -l myfile

Understanding Umask Value and Its Role in File Permissions

Umask

- In Linux, the umask is a value that determines the default permissions assigned to newly created files and directories.
- Understanding umask is crucial for controlling the default permissions and ensuring proper security and access control.

Impact of umask on file and directory permissions

- The umask value affects the default file and directory permissions for all users on the system.
- It is important to set the umask value appropriately to ensure the security and accessibility of data.
- The umask value can be set in the user's shell startup files, such as ~/.bashrc or ~/.bash_profile.

Umask calculation

- The umask value is a three-digit octal number that represents the permissions that will be removed from the default file permission.
- It is calculated by subtracting the umask value from the maximum permission value (777 for files, 666 for directories).
- For example, if the umask value is 022, then the default file permission will be 644 (666-022), and the default directory permission will be 755 (777-022).

Changing the umask value

- The umask value can be changed using the 'umask' command, followed by the desired value.
- For example, to set the umask value to 077, use the command: umask 077.
- The new umask value will take effect for newly created files and directories.

Extended Rights Standard: Introduction to POSIX Access Control Lists (ACL)

POSIX Access Control Lists

- POSIX Access Control Lists (ACLs) are an extension to the traditional Unix/Linux file permissions.
- They provide a more fine-grained control over file and directory access by allowing additional access permissions and specifying access rights for specific users and groups.

Purpose of ACLs

- ACLs are used to grant or restrict permissions beyond the basic owner-group-other permissions.
- They allow administrators to define permissions for individual users or groups, enabling more precise access control.
- ACLs are especially useful in scenarios where complex access control requirements exist, such as shared directories or multi-user environments.

Difference between traditional permissions and ACLs

- Traditional permissions consist of three sets (owner-group-other), while ACLs can define permissions for multiple users and groups.
- Traditional permissions are limited to a fixed set of permissions (read, write, execute), while ACLs offer more granular control with additional permissions.
- ACLs can be used alongside traditional permissions, enhancing the existing permission model.

Viewing and modifying ACLs

- The 'getfacl' command is used to view ACLs for files and directories. It displays both the traditional permission settings and any ACL entries.
- The 'setfacl' command is used to modify ACLs. It allows adding, modifying, or removing ACL entries for specific users or groups.

Examples

- To view ACLs for a file: getfacl filename
- To modify ACLs for a file: setfacl -m u:user:permissions filename

Working with POSIX ACL: setfacl and getfacl Utilities

setfacl Command

- The 'setfacl' command is used to set or modify ACLs for files and directories.
- It allows adding or removing specific entries for users or groups, granting or revoking permissions.

Syntax: setfacl -m [user/group]:[permissions] [file/directory]

- Example 1: Grant read and write access to a specific user
  - setfacl -m u:john:rw file.txt
- Example 2: Revoke execute permission for a group
  - setfacl -m g:admins:-x directory

getfacl Command

- The 'getfacl' command is used to retrieve and display the ACLs of a file or directory.
- It provides detailed information about the existing ACL entries, including the traditional permissions.

Syntax: getfacl [file/directory]

- Example: View ACLs for a directory
  - getfacl /path/to/directory

Package Management Basics: Introduction to dpkg, apt, apt-get, rpm, yum

Package managers

Package managers are essential tools for managing software packages in Linux distributions. They streamline the installation, updating, and removal of software, ensuring system stability and security.

Package Managers in Linux

There are several package managers commonly used in Linux distributions:

- dpkg: The package manager for Debian-based distributions.
- apt (Advanced Package Tool): A high-level package management system built on top of dpkg.
- apt-get: A command-line tool for package management in Debian-based distributions.
- rpm (Red Hat Package Manager): The package manager for Red Hat-based distributions.
- yum (Yellowdog Updater, Modified): A high-level package management utility for RPM-based distributions.

Differences between package managers

- dpkg:
  - Handles individual .deb packages.
  - Manages software at a lower level.
- apt and apt-get:
  - Provide higher-level package management features.
  - Resolve dependencies automatically.
  - apt is the recommended command-line tool in newer Debian-based distributions.
- rpm:
  - Manages individual .rpm packages.
  - Lower-level package management compared to yum.
- yum:
  - Higher-level package management utility for RPM-based distributions.
  - Resolves dependencies and handles package installation, updates, and removal.

Differences between package managers

- dpkg:
  - Handles individual .deb packages.
  - Manages software at a lower level.
- apt and apt-get:
  - Provide higher-level package management features.
  - Resolve dependencies automatically.
  - apt is the recommended command-line tool in newer Debian-based distributions.
- rpm:
  - Manages individual .rpm packages.
  - Lower-level package management compared to yum.
- yum:
  - Higher-level package management utility for RPM-based distributions.
  - Resolves dependencies and handles package installation, updates, and removal.

Understanding Repositories and Compilation of Own Lists

Repositories

- Software repositories are centralized collections of software packages for a specific Linux distribution.
- They provide a convenient and secure way to access and install software on a Linux system.

Structure and Contents of Repositories

- Repositories contain a vast range of software packages organized into categories.
- Each package in a repository has metadata that includes its name, version, dependencies, and other relevant information.
- Repositories are typically maintained and updated by the Linux distribution's community or vendor.

Compiling Your Own List of Repositories

- Users can compile their own list of repositories to expand the available software options.
- Adding additional repositories can provide access to software not available in the default repositories of a Linux distribution.
- It allows users to leverage a wider range of software packages and stay up to date with the latest releases.

Adding Repositories

Example 1: Ubuntu-based distributions (e.g., Ubuntu, Linux Mint)

Use the 'add-apt-repository' command to add a repository:

- Syntax: add-apt-repository repository_name
- Example: add-apt-repository ppa:example/repository

Example 2: Red Hat-based distributions (e.g., Fedora, CentOS)

Use the 'dnf' command to add a repository:

- Syntax: dnf config-manager --add-repo repository_url
- Example: dnf config-manager --add-repo https://example.com/repo.repo

Removing Repositories

Example 1: Ubuntu-based distributions

Use the 'add-apt-repository' command with the '--remove' flag:

- Syntax: add-apt-repository --remove repository_name
- Example: add-apt-repository --remove ppa:example/repository

Example 2: Red Hat-based distributions

Use the 'dnf' command to remove a repository:

- Syntax: dnf config-manager --remove repository_id
- Example: dnf config-manager --remove example-repo

Manually Building Packages from Source Codes

Building Packages from Source in Linux

- Building packages from source involves compiling software directly from its source code.
- Source code refers to the human-readable code written by developers that can be compiled into executable programs.

Reasons to Build from Source

- Gain access to the latest features or bug fixes not yet available in pre-built packages.
- Customize software by enabling or disabling specific features during the build process.
- Improve performance by optimizing the code specifically for the target system.

General Process of Building from Source

1. Downloading the Source Code

   Source code can usually be obtained from the software project's website or repository. It is often distributed as compressed archives

2. Unpacking the Source Code

   Use the appropriate tool to extract the contents of the compressed archive (e.g., tar).

3. Configuring the Build

   Run the configuration script included in the source code to adapt the build process to the system. This step may involve specifying installation directories, enabling optional features, or setting compilation flags.

4. Compiling the Source Code

   Use a compiler such as GCC (GNU Compiler Collection) to convert the source code into executable binaries. The 'make' utility is commonly used to automate the compilation process.

5. Installing the Compiled Software

   After successful compilation, the software can be installed using the 'make install' command. The installed files are typically placed in the appropriate system directories.

Building Packages from Source in Linux

Building Tools Used in the Build Process

- 'make': A build automation tool that reads a Makefile to execute compilation tasks.
- GCC: The GNU Compiler Collection, a suite of compilers supporting various programming languages.

Importance of Reading README and INSTALL Files

- README and INSTALL files often accompany the source code and provide important instructions and prerequisites.
- They contain information on system requirements, dependencies, build options, and installation steps.
- Reading these files helps ensure a successful build and proper usage of the software.

Q&A