Basics of Linux and Web Server

Linux

Command Line Interface (CLI)

Linux primarily uses a command line interface to interact with the user. The most commonly used CLI (Command Line Interface) commands in Linux include:

1. **ls**: lists the files and directories in a directory
2. **cd**: changes the current working directory
3. **pwd**: displays the current working directory
4. **cat**: concatenates and displays files
5. **cp**: copies files and directories
    1. cp [...file/directory-sources] [destination]
6. **mv**: moves or renames files and directories
7. **rm**: deletes files and directories
8. **mkdir**: creates a new directory
9. **rmdir**: removes an empty directory
10. **touch**: creates a new file or updates the modification time of an existing file
11. **echo**: displays a message or the contents of an environment variable
12. **ps**: displays information about the currently running processes
13. **top**: provides real-time information about system resource usage
14. **ping**: tests connectivity to a network host
15. **ssh**: opens a secure shell connection to a remote host
16. **scp**: securely copies files between hosts
17. **tar**: creates or extracts archive files
18. **grep**: searches for a pattern in text files
19. **find**: searches for files and directories based on certain criteria
20. **chmod**: changes the permissions of files and directories.

File System Hierarchy

Linux uses a hierarchical file system, starting from the root directory "/" and branching out to subdirectories.

The File System Hierarchy in Linux is a standardized way of organizing and storing files and directories on a Linux system. Some key points to know about the Linux File System Hierarchy include:

1. **Root Directory**: The top-level directory in the Linux file system hierarchy is the root directory, represented by a forward slash "/".
2. **Subdirectories**: The root directory branches out into several subdirectories, each serving a specific purpose.
3. **Standard Directories**: Some of the standard directories in the Linux file system hierarchy include:

- **/bin**: contains essential user command binaries
- **/boot**: contains boot loader files and the Linux kernel
- **/dev**: contains device files that represent system devices
- **/etc**: contains configuration files for the system and applications
- **/home**: contains user home directories
- **/lib**: contains libraries required by the system and applications
- **/media**: contains mount points for removable media devices
- **/mnt**: used as a mount point for temporarily mounting file systems
- **/opt**: contains optional software packages
- **/root**: contains the home directory of the root user
- **/srv**: contains data for services provided by the system
- **/tmp**: contains temporary files
- **/usr**: contains user-related data and applications
- **/var**: contains variable data, such as log files and spool directories

4. **File Permissions**: In Linux, every file and directory has permissions that determine who can access and modify them. These permissions can be viewed and modified using the "**ls -l**" and "**chmod**" commands.
5. **Hidden Files**: In Linux, files and directories that start with a dot (.) are considered hidden and are not displayed by default when using the "**ls**" command.
6. **Symbolic Links**: Symbolic links, also known as symlinks, are special files that point to other files or directories. They can be created using the "ln -s" command.
7. **Mount Points**: In Linux, different file systems can be mounted at various mount points in the file system hierarchy to make them accessible. The "**mount**" and "**umount**" commands can be used to mount and unmount file systems, respectively.

Directory Navigation

The "**cd**" command is used to navigate through directories in Linux.

Directory navigation in Linux involves moving around the file system hierarchy to access and manipulate files and directories. Here are some key points to know about directory navigation in Linux:

1. **Current Directory**: The current working directory is the directory that you are currently in and is indicated by the prompt in the terminal.
2. **Absolute Path**: An absolute path is the full path to a file or directory, starting from the root directory "/".
3. **Relative Path**: A relative path is a path to a file or directory relative to the current working directory.
4. cd Command: The "**cd**" (change directory) command is used to navigate to different directories in the file system hierarchy.
5. **.. and .** : The "**..**" (dot dot) symbol represents the parent directory of the current working directory, while the "." (dot) symbol represents the current working directory itself.
6. **Tab Completion**: In the terminal, you can use tab completion to automatically complete the names of files and directories.
7. **~ (Tilde)**: The "**~**" symbol represents the home directory of the current user.
8. **Shortcuts**: Some common directory navigation shortcuts include:
   - **cd ~:** goes to the current user's home directory
   - **cd /**: goes to the root directory
   - **cd ..**: goes to the parent directory of the current working directory
9. **Pushd** and **Popd**: The "**pushd**" and "**popd**" commands can be used to save and switch between directories, respectively. They are part of the dirstack, a stack of directories that you can navigate.

By understanding these basic concepts and commands, you can efficiently navigate the file system hierarchy in Linux.

File Permissions

In Linux, files and directories have permissions that determine who can access and modify them.

File permissions in Linux determine who can access and modify files and directories. Here are some key points to know about file permissions in Linux:

1. **User Types**: In Linux, there are three types of users for file permissions: **owner**, **group**, and **others**.
2. Read, Write, and Execute Permissions: For each file or directory, there are three types of permissions: read, write, and execute.
   - **Read** (r) permission allows the user to view the contents of the file or list the contents of the directory.
   - **Write** (w) permission allows the user to modify the contents of the file or add/remove files in the directory.
   - **Execute** (x) permission allows the user to run the file as a program or enter the directory as the current working directory.
3. **Permission Representation**: File permissions are represented by a string of 9 characters, in the form of **rwxrwxrwx**. The first three characters represent the owner's permissions, the next three characters represent the group's permissions, and the last three characters represent others' permissions.
4. **Octal Representation**: File permissions can also be represented using octal (base-8) notation, where each digit represents a set of permissions. For example, 755 represents read, write, and execute permissions for the owner and read and execute permissions for group and others.
5. **chmod Command**: The "chmod" (change mode) command is used to change the permissions of a file or directory. The permissions can be changed using either the symbolic or the octal representation.
6. **umask**: The umask is a default permission mask that is applied to new files and directories created by the user. It can be viewed and changed using the "**umask**" command.

By understanding these basic concepts and commands, you can effectively manage file permissions in Linux to ensure the security and accessibility of your files and directories.

Package Management

Linux distributions use package management systems such as **apt**, **yum**, or **pacman** to install, update and manage software packages.

Package management is an essential aspect of Linux systems, used to **install**, **update**, and manage software packages. Here are some key points to know about package management in Linux:

1. **Package**: A package is a collection of files, scripts, and metadata that make up a software program. Packages are typically distributed in a pre-compiled and compressed format, such as a .**deb** or .**rpm** file.
2. **Package Manager**: A package manager is a program that handles the installation, removal, and updating of packages. Some popular package managers in Linux include **apt** (for Debian-based systems), **yum** (for Red Hat-based systems), and **dnf** (for Fedora-based systems).
3. **Repositories**: Repositories are centralized collections of packages that can be accessed and installed through the package manager. Official repositories are typically maintained by the Linux distribution, while third-party repositories may be created and maintained by individuals or organizations.
4. **Searching and Installing Packages:** The package manager allows you to search for packages in the repositories, view package information, and install packages. The specific commands for searching and installing packages vary depending on the package manager in use.

5. **Updating and Removing Packages**: The package manager also provides commands for updating packages to the latest version and removing packages that are no longer needed.
6. **Package Dependencies**: Packages often have dependencies, which are other packages that must be installed for the package to work correctly. The package manager handles the installation of dependencies automatically.
7. **Package Management in Virtual Environments**: Package management can also be performed in virtual environments, such as virtualenv for Python or conda for data science. This allows you to manage packages and dependencies for individual projects, rather than system-wide.

By understanding these basic concepts, you can effectively use package management to install and manage software packages on your Linux system.

Text Editors

Linux provides several text editors such as **nano**, **vi**, and **emacs** for editing files in the terminal.

Text editors are a crucial tool for writing and editing text-based files in Linux. Here are some key points to know about text editors in Linux:

1. **Types of Text Editors**: There are two main types of text editors in Linux: graphical and command-line. Graphical text editors, such as Gedit and Sublime Text, provide a graphical user interface for editing text. Command-line text editors, such as nano and vi, are text-based and run in the terminal.
2. **nano**: nano is a simple, easy-to-use text editor that is often used as a beginner-friendly alternative to vi. nano has a menu-based interface and provides basic editing features, such as cut and paste.
3. **vi**: vi is a powerful, but more complex, text editor that is included with most Linux distributions. vi operates in two modes: command mode and insert mode. In command mode, you can navigate and manipulate the text, while in insert mode, you can insert and edit the text.
4. **Syntax Highlighting**: Many text editors, including **nano** and **vi**, support syntax highlighting, which displays different elements of the code, such as keywords and comments, in different colors. This can make the code easier to read and understand.
5. **Customization**: Many text editors, including vi, can be customized with plugins and configurations to add additional functionality and tailor the editor to your needs.
6. **Choosing a Text Editor**: The text editor you choose will depend on your needs and preferences. If you're new to Linux, **nano** may be a good place to start. If you're looking for a more powerful and customizable option, **vi** may be a better choice.

By understanding these basic concepts, you can effectively use text editors in Linux to create and edit text-based files.

Environment Variables

Environment variables in Linux provide information about the system and user settings.

Environment variables are a fundamental aspect of the Linux operating system, used to store information about the system and user configuration. Here are some key points to know about environment variables in Linux:

1. **Purpose**: Environment variables store information such as system paths, user settings, and program configuration options. They can be accessed and used by both the operating system and various

programs and scripts.
2. **Setting Environment Variables**: Environment variables can be set in various ways, including in shell profiles, shell scripts, and program configuration files. The syntax for setting environment variables varies depending on the shell in use.
3. **Viewing Environment Variables**: The command `echo $VAR_NAME` can be used to view the value of an environment variable in the terminal, where `VAR_NAME` is the name of the environment variable. The `printenv` command can be used to display all environment variables and their values.
4. **Common Environment Variables**: Some common environment variables include `PATH`, which stores the directories where the system should search for executable programs, and `HOME`, which stores the path to the user's home directory.
5. **Persistence**: Environment variables are typically set for a specific session, and their values will be lost when the session is closed. To persist environment variable values across sessions, they must be set in a shell profile or configuration file that is executed every time the system starts.
6. **Modifying Environment Variables**: Environment variables can be modified, added, or removed as needed. This can be done through the terminal or by editing configuration files.

By understanding these basic concepts, you can effectively use environment variables in Linux to store and manage information about the system and user configuration.

Process Management

The "**ps**" and "**top**" commands can be used to view processes running on a Linux system, while "kill" and "killall" can be used to terminate them.

Process management is a key aspect of the Linux operating system, as it determines how programs and tasks run on the system. Here are some key points to know about process management in Linux:

1. **What is a Process**: A process is an instance of a running program, including the program's code and data in memory. Each process has a unique process ID (**PID**) that is assigned by the system.
2. **ps Command**: The `ps` command displays information about the processes running on the system. It can be used to view the PID, status, and other information about processes.
3. **top Command**: The `top` command provides an interactive interface for viewing information about processes running on the system, including the CPU and memory usage of each process.
4. **kill Command**: The `kill` command can be used to send signals to processes, such as the `SIGTERM` signal, which terminates the process. The `kill` command is often used to stop unresponsive or rogue processes.
5. **nohup Command**: The `nohup` command can be used to run a process in the background, even if the terminal is closed. This is useful for running long-running processes that should continue running even if the user logs out.
6. **& Operator**: The `&` operator can be used to run a process in the background from the terminal. For example, `command &` will run `command` in the background.
7. **fg and bg Commands**: The `fg` and `bg` commands can be used to bring a background process to the foreground or to move a foreground process to the background, respectively.

By understanding these basic concepts, you can effectively manage processes in Linux and ensure that your programs and tasks run smoothly on the system.

Web Server

A web server is a software that provides access to web pages over the internet. Here are some key points to know about web servers. By understanding these basic concepts, you can effectively set up and manage a web server to serve web pages over the internet:

Types of Web Servers

Some of the most popular web servers include Apache, Nginx, and Microsoft IIS. Each web server has its own strengths and weaknesses, and the choice of web server depends on the needs of the particular application.

There are several types of web servers, each with its own strengths and weaknesses. Here are some of the most popular web servers:

1. **Apache**: Apache is the most widely-used web server software, with a large user community and a wealth of documentation and support resources. Apache supports a wide range of operating systems and is known for its flexibility and stability.
2. **Nginx**: Nginx is a high-performance web server that is often used to serve static content, such as images and videos. Nginx is known for its efficiency and can handle a large number of concurrent connections with a low memory footprint.
3. **Microsoft IIS**: Microsoft IIS is a web server designed to run on Windows-based servers. IIS is often used in enterprise environments and integrates well with other Microsoft products, such as Windows Active Directory.
4. **Lighttpd**: Lighttpd is a lightweight web server that is often used for smaller websites and applications. Lighttpd is known for its low resource usage and high performance.
5. **Node.js**: Node.js is a JavaScript runtime that can be used as a web server. Node.js is known for its fast performance and event-driven architecture, which makes it well-suited for real-time applications and web sockets.
6. **Tomcat**: Tomcat is a web server and servlet container, designed to run Java applications. Tomcat is widely used for Java-based web applications and integrates well with other Java technologies, such as JSP and JSF.

Each web server has its own strengths and weaknesses, and the choice of web server depends on the needs of the particular application. When choosing a web server, consider factors such as performance, scalability, ease of use, and compatibility with the technologies you plan to use.

Serving Web Pages

A web server serves web pages to clients, usually in response to an HTTP request. The web server retrieves the requested web page from the file system, processes any dynamic content (such as PHP or Python scripts), and sends the result to the client.

Serving web pages is the main function of a web server. Here are some key points to know about serving web pages:

1. **HTTP Requests**: A web server serves web pages in response to HTTP requests from clients, such as web browsers. The HTTP request specifies the web page the client wants to access, and the web server returns the requested web page in the HTTP response.
2. **Retrieving Web Pages**: The web server retrieves the requested web page from the file system and processes any dynamic content, such as PHP or Python scripts. The result of this processing is a

complete web page, ready to be sent to the client.
3. **Dynamic Content**: Dynamic content refers to web pages that are generated in real-time based on user input or other data. For example, a PHP script might generate a web page that shows the current time, or a Python script might retrieve data from a database and use it to generate a web page.
4. **Caching**: To improve performance, a web server may cache frequently-requested web pages in memory. When a client requests a cached web page, the web server can return the cached page instead of retrieving it from the file system and processing dynamic content, which saves time.
5. **Performance Optimization**: A web server can be optimized for performance in several ways, such as serving static content efficiently, compressing content before sending it to the client, and reducing the number of HTTP round-trips between the client and server.

By understanding these basic concepts, you can effectively serve web pages to clients using a web server.

Handling Requests

A web server can handle multiple requests at the same time, using threads or processes to manage each request. The web server must manage the process of receiving requests, finding the requested web page, and returning the result to the client.

Serving web pages refers to the process of delivering web content (such as HTML, CSS, and JavaScript files) to clients (such as web browsers) in response to HTTP requests. Here are some key points to know about serving web pages:

1. **HTTP Requests**: When a client requests a web page, it sends an HTTP request to the web server. The request specifies the URL of the requested page, as well as any parameters or data that should be sent to the server.
2. **Dynamic Content**: Some web pages contain dynamic content, which is generated on the fly by the server in response to the request. For example, a server-side scripting language such as PHP or Python can be used to generate dynamic content, based on user input or other variables.
3. **Static Content**: Other web pages contain static content, which does not change from one request to the next. For example, images, videos, and CSS files are typically static content that can be served directly by the web server.
4. **File System**: The web server retrieves the requested web page from the file system, based on the URL in the request. The web server must be configured to map URLs to file paths on the file system, so that it knows where to find the requested page.
5. **Processing**: The web server may need to process the requested web page before sending it to the client. For example, if the page contains server-side scripts, the web server must execute those scripts and include their output in the response.
6. **Response**: Once the web server has processed the requested web page, it sends an HTTP response back to the client. The response includes the contents of the requested web page, as well as headers that provide information about the response (such as the content type, encoding, and length).
7. **Caching**: Web browsers often cache web pages, so that they don't need to request the same page multiple times. Web servers can also cache web pages, either on the server itself or in a proxy server. Caching can improve performance by reducing the number of requests that need to be processed by the web server.

By understanding these basic concepts, you can effectively serve web pages to clients over the internet, either by configuring a web server or by writing your own server-side code.

Configuration

A web server is typically configured through configuration files, which specify the server's behavior and settings. The configuration can include settings such as the server's IP address, port number, and the location of the web pages on the file system.

When it comes to configuring a web server, there are several key aspects to understand:

1. **Server Settings**: The web server must be configured with various settings, such as the IP address and port to listen on, the document root (the location on the file system where the web content is stored), and the default file to serve when a directory is requested. These settings can be configured in the web server's configuration file, or through a graphical user interface provided by the web server software.
2. **Virtual Hosts**: Many web servers support virtual hosting, which allows multiple websites to be hosted on a single server. Virtual hosting enables different websites to have different IP addresses, domains, and configurations, even though they are all running on the same physical server.
3. **SSL/TLS Certificates**: If your website needs to support secure connections (such as for online transactions), you'll need to obtain an SSL/TLS certificate and configure the web server to use it. The certificate is used to encrypt traffic between the client and server, and to verify the identity of the website.
4. **Access Control**: Web servers can be configured to restrict access to certain parts of the website, based on criteria such as the IP address of the client, the URL being requested, or the credentials provided by the client. This is typically done through the use of access control rules, which can be specified in the web server's configuration file.
5. **Logging**: Web servers log information about incoming requests and other events, such as errors or warnings. Logs can be useful for debugging and monitoring the performance of the web server. Web server configurations often include settings for logging, such as the location of the log files and the format of the logs.
6. **Performance**: Web server configurations can also be optimized for performance, to handle a high volume of traffic or to improve the response time for clients. This can involve tuning various settings, such as the maximum number of concurrent connections, the buffer size for incoming requests, or the number of worker processes.

By understanding these configuration concepts, you can effectively set up and manage a web server to meet the needs of your website or application.

Security

A web server must be secured to prevent unauthorized access and protect sensitive information. This can include implementing encryption (such as SSL/TLS), limiting access to the server with firewalls, and applying patches and security updates.

Security is a critical aspect of web server administration, and there are several key things to know in order to keep your web server and the data it handles secure:

1. **Patch Management**: Keeping your web server software up to date is one of the most important steps in maintaining security. Software vendors frequently release updates that fix security vulnerabilities, so it's important to apply these updates in a timely manner.
2. **Firewall Configuration**: A firewall can be used to block incoming traffic that is not necessary for the functioning of the web server. This helps to reduce the attack surface of the server and makes it more difficult for attackers to exploit vulnerabilities.

3. **Secure Configuration**: The web server software should be configured securely to minimize the risk of attack. For example, the software should be configured to run with the minimum privileges necessary, to disable unnecessary services and features, and to use strong authentication and encryption mechanisms.
4. **Input Validation**: Web servers must be designed to validate all incoming data to prevent attacks such as SQL injection or cross-site scripting (XSS). This involves checking the format and content of incoming requests, and rejecting requests that contain unexpected or malicious data.
5. **Access Control**: The web server should be configured to restrict access to sensitive resources, such as configuration files, system logs, and user data. This can be achieved through the use of authentication and authorization mechanisms, such as passwords, certificates, or tokens.
6. **Monitoring**: Regularly monitoring the web server for signs of attack or compromise is an important aspect of security. This includes monitoring logs for unusual activity, using intrusion detection systems (IDS) or intrusion prevention systems (IPS) to detect and block attacks, and performing regular security audits.

By following these security best practices, you can help to protect your web server and the data it handles against potential security threats.

Scalability

As the number of clients accessing a web server increases, the server may become overwhelmed. Web servers can be scaled horizontally (by adding more servers) or vertically (by upgrading the hardware of a single server) to handle increased traffic.

Scalability refers to the ability of a system to handle an increasing load as the demand for its services grows. When it comes to web servers, scalability is important because a single web server may not be able to handle the traffic of a high-traffic website.

Here are some key concepts to understand when it comes to scalability in web servers:

1. **Load Balancing**: Load balancing is a technique for distributing incoming traffic across multiple web servers. This helps to ensure that no single web server becomes overwhelmed, and that the overall system can handle a larger volume of traffic. Load balancing can be achieved through hardware devices such as load balancers or through software solutions such as reverse proxies.
2. **Horizontal Scaling**: Horizontal scaling refers to the practice of adding more web servers to the system as the demand for its services increases. This allows the system to handle a growing volume of traffic without sacrificing performance. Horizontal scaling is often achieved by adding more nodes to a cluster of web servers, which can then be load balanced.
3. **Vertical Scaling**: Vertical scaling involves upgrading the hardware of a single web server, such as by increasing the amount of memory or processing power, in order to handle a growing load. Vertical scaling can be effective in the short term, but has limitations in the long term, as there are physical limits to how much hardware can be added to a single server.
4. **Caching**: Caching is a technique for storing frequently-used data in memory, so that it can be served more quickly in response to client requests. Caching can help to improve the performance and scalability of a web server, as it reduces the number of requests that need to be processed by the server.
5. **Content Delivery Networks (CDN)**: A content delivery network (CDN) is a network of servers that distribute content to clients based on their geographic location. CDNs can help to improve the scalability and performance of a web server by caching content closer to the clients and reducing the load on the primary web server.

By understanding these scalability concepts and techniques, you can design and build web servers that are capable of handling increasing traffic and delivering high performance, even under heavy load.

Revision #8
Created 8 February 2023 07:04:46 by Daniel Azimi
Updated 3 June 2023 16:29:32 by Daniel Azimi