SCP

SCP (Secure Copy Protocol) allows users to securely transfer files between systems. Because it uses SSH, all transferred data is encrypted, making it a safe choice for transmitting sensitive files.

It's a command-line tool that allows transferring files securely between hosts over a network built on the SSH. Whether copying files from your local machine to a remote server or transferring files from one Linux machine to another, SCP ensures that the process is safe and fast.

Syntax

scp [options] [[user@]host1:]source_file_or_directory ... [[user@]host2:]destination

Ex:

To copy a file named **file.tx** from your local machine to a remote server:

scp file.txt username@192.168.1.2:/home/username/

Here...

- **file.txt** is the local file
- **username@192.168.1.2** is the **remote server** and **user**
- **/home/username/** is the destination directory on the remote server.

The options are

-P

It is used to Securely Copy File to a Remote Machine on a Non-Standard SSH Port and specify the port to connect on the remote host. It is useful when our SSH server is listening on a non-standard port.

-p

This option is used when we want the original metadata of the file that has been transferred. Basically, it preserves modification time, access time, and modes from the original file.

-q

It Securely Copy File with Quiet Mode - Disabling Progress Meter .This option hides the progress of the file transfer on the terminal.

-r

This option is used when we want to copy an entire directory and its contents. Which basically means copying entire directory recursively.

To **copy a directory** recursively with SCP:

scp -r /path/to/local/directory username@192.168.1.2:/home/username/

How to Securely Copy Files from Local to Remote Machine

Syntax:

scp [file_name] remoteuser@remotehost:/remote/directory

Here,

- **file_name** = The name of the file that needs to be copied.
- **remoteuser** =The username of the remote host.
- **remotehost** = The IP address or hostname of the remote host.
- **/remote/directory** = The directory where the file should be copied on the remote machine.

RSYNC

rsync or remote synchronization is a software utility for Unix-Like systems that efficiently sync files and directories between two hosts or machines.

One is the source or the local-host from which the files will be synced, the other is the remote-host, on which synchronization will take place.

There are basically two ways in which *rsync* can copy/sync data:

- Copying/syncing to/from another host over any remote shell like *ssh*, *rsh*.
- Copying/Syncing through rsync daemon using TCP.

Rsync is famous for its **delta-transfer algorithm**, in which it copies only the differences between the source files present in the local-host and the existing files in the destination or the remote host.

Example:

rsync local-file user@remote-host:remote-file

What Happens Here:

Rsync will first use SSH to connect as the user to remote-host and will ask for user's password. Once connected, it will invoke the remote host's rsync, and then the two programs will determine what parts of the local-file need to be copied so that the remote file matches the local one. Please note the following behavior of rsync:

- Files that do not exist on the remote-host are copied.
- Files that have been updated will be synced, rsync will copy only the changed parts of files to the remote host.
- files that are exactly the same are not copied to the remote host.

Syntax of `rsync` command in Linux

rsync [options] source [destination]

The Options are

-a (archive mode):Archive mode includes all the necessary options like copying files recursively, preserving almost everything .

- -r: Recursively copy directories.
- -l: Copy symlinks as symlinks.
- -p: Preserve permissions.
- -t: Preserve modification times.
- -g: Preserve group.
- -o: Preserve owner.
- -D: Preserve device files and special files.

-v: Verbose output, showing details of the transfer.

-z: Compress file data during the transfer.

-h: Human-readable output.

CMD → **rsync -avz /home/myuser/data/** user@backupserver.com:/mnt/backup/

This command will copy data directory and its contents to the remote backup server, preserving all attributes, compressing data, and showing verbose output. If you run it again, it will only transfer changed files.

CMD → **rsync -avz --delete user@webserver.com:/var/www/html/ /var/www/html/**

This command ensures your local /var/www/html/ directory is an exact copy of the remote one, removing any files locally that are no longer on the server.

SSH

SSH, or Secure Shell, is a network protocol that allows secure communication between two computers over an unsecured network.

In Linux, it's primarily used for remote access, secure file transfer, and executing commands on a remote server. SSH encrypts the data transmitted between the client and server, ensuring the confidentiality and integrity of the communication.

Syntax

ssh [username]@[hostname or IP address]

[**username**] → remote server username

[**hostname or IP address**] → server's hostname or IP address.

Install SSH Component on Linux

Setting up SSH on Linux may be necessary, as some distributions don't come with it pre-installed.

Installing OpenSSH, a widely used SSH implementation, or opting for a graphical user interface (GUI) solution like the PuTTY client for Ubuntu can address this.

For Debian/Ubuntu-based Systems, open the terminal and run:

sudo apt install openssh-client openssh-server

For Red Hat-based systems like CentOS or Fedora:

sudo dnf install openssh-clients openssh-server

How to Use SSH to Connect to a Remote Server in Linux

COMMAND→ **ssh [options] username@remote_host**

OPTIONS are

A	Authentication agent connection forwarding is enabled.
-a	Authentication agent connection forwarding is disabled.
-C	Compresses all data (including stdin, stdout, stderr, and data for forwarded X11 and TCP connections) for a faster transfer of data.
-f	Requests ssh to go to background just before command execution.
-g	Allows remote hosts to connect to local forwarded ports.
-n	Prevents reading from stdin.
-p	Port to connect to on the remote host.
-q	Suppresses all errors and warnings
-V	Display the version number.
-v	Verbose mode. It echoes everything it is doing while establishing a connection

Encryption Techniques Used by SSH

- **Symmetrical encryption:** This encryption works on the principle of the generation of a single key for encrypting as well as decrypting the data. The secret key generated is distributed among the clients and the hosts for a secure connection.
- **Asymmetrical encryption:** This encryption is more secure because it generates two different keys: Public and Private key. A public key is distributed to different host machines while the private key is kept securely on the client machine. A secure connection is established using this public-private key pair.
- **Hashing:** One-way hashing is an authentication technique which ensures that the received data is unaltered and comes from a genuine sender. A hash function is used to generate a hash code from the data. It is impossible to regenerate the data from the hash value. The hash value is calculated at the sender as well as the receiver's end. If the hash values match, the data is authentic.

TELNET

Telnet is an application protocol or older network protocol that **Telnet** enables the remote access to another machine with the use of telnet client, to connect to and execute commands on a remote machine that's hosting a telnet server.

The telnet client will establish a connection with the server. The client will then become a virtual terminal- allowing you to interact with the remote host. It does not encrypt data, making it vulnerable to **man-in-the-middle attacks**.

Unlike SSH, telnet transmits data, including usernames and passwords, in **plaintext**, making it highly insecure for remote logins over public networks.

It is generally recommended to use SSH instead of Telnet for remote access due to security vulnerabilities. Its primary use today is often for testing network connectivity to specific ports.

Syntax:

telnet [options] host [port]

Options:

- -l user: Specify the username to log in as.
- -E: Stop the escape character from working.

Purpose?

testing open network ports,

troubleshooting almost any network problem,

gaining access to very old legacy systems

To Install Telnet on Ubuntu/Debian

sudo apt update
sudo apt install telnet

Telnet Usage

telnet 192.168.1.1 → This command will attempt to access the device at the specified **IP**

telnet example.com 80 → To check if **port 80** (HTTP) is open on a website

telnet user@remote-host → **Telnet login** is enabled on a remote server

To add a new user for **Telnet login**:

```
sudo adduser telnetuser
sudo passwd telnetuser
```

This creates a **Telnet user** named telnetuser with a secure password.

Package Manager

A software tool that automates the process of installing, updating, configuring, and removing software packages. Think of it as an app store for your operating system, but usually with a powerful command-line interface.

A **Package** is a compressed archive file that contains:

- **Pre-compiled binary software:** The actual executable programs.
- **Libraries:** Shared code that programs depend on.
- **Configuration files:** Settings and preferences for the software.
- **Documentation:** Man pages, READMEs, etc.
- **Metadata:** Information about the package, such as its name, version, description, and, most importantly, its **dependencies**

These packages are typically stored in **repositories**, which are centralized servers accessible via the internet. Each Linux distribution usually maintains its own set of official repositories, ensuring that the software available is compatible, stable, and secure for that specific distribution.

Why We Use Package Managers?

Dependency Resolution →

Package managers automate this entire process. When you tell a package manager to install a program, it automatically fetches and installs all its necessary dependencies, ensuring compatibility and preventing errors.

System Updates and Security →

Package managers make it incredibly easy to keep your entire system up-to-date. With a single command, you can check for and install updates for all installed software.

Verification and Authenticity →

Packages in official repositories are typically signed with digital signatures. Package managers verify these signatures, ensuring that the packages you download are authentic and haven't been tampered with. This adds a crucial layer of security.

Popular Package Managers

APT (Advanced Package Tool) - Debian/Ubuntu Family

Distributions: Debian, Ubuntu, Linux Mint, Kali Linux, Raspbian

Package Format: .deb (Debian package).

Underlying Tool: dpkg (Debian Package management system) is the low-level tool that actually handles the installation and removal of individual .deb files. apt is a higher-level, more user-friendly interface to dpkg and handles dependency resolution and repository management.

Commands→

sudo apt update: This command refreshes the local package index. It downloads the latest information about available packages and their versions from the configured repositories.

sudo apt upgrade: After apt update, this command installs the newer versions of already installed packages. It will prompt you for confirmation before proceeding.

sudo apt install <package-name>: Installs a new software package and all its necessary dependencies.

DNF / YUM (Dandified YUM / Yellowdog Updater, Modified) - Red Hat Family

Distributions: Fedora, Red Hat Enterprise Linux (RHEL), CentOS, Rocky Linux, AlmaLinux.

Package Format: .rpm (Red Hat Package Manager).

Evolution: yum was the traditional package manager for RHEL and its derivatives. dnf is its modernized successor, aiming to resolve some of yum's limitations (like dependency resolution performance) while largely maintaining a similar command syntax. For newer systems, dnf is the default.

Commands →

sudo dnf check-update (or sudo yum check-update): Checks for available updates without installing them.

Pacman (Package Manager) - Arch Linux Family

Distributions: Arch Linux, Manjaro, EndeavourOS.

Package Format: .pkg.tar.zst (or .pkg.tar.xz).

Philosophy: Arch Linux is known for its "rolling release" model and its simplicity, and Pacman reflects that with its fast and efficient operations.

sudo pacman -Syu: Synchronizes package databases *and* updates all installed packages. This is the most common command for a full system update on Arch.

SU

The su command (short for **"substitute user"** or sometimes **"switch user"**) allows you to switch to another user account within the current terminal session.

By default, if no username is specified, it attempts to switch to the **root** user.

The core idea behind su is to change your user identity for the remainder of the session (or until you exit the new user's shell)

Syntax:

su [options] [username]

su → switch to the root user

su - → full login to the root enivronments

su john → switch to the specific user

Why ?

Full Root Environment: When you genuinely need to operate as the root user with their complete environment, home directory, and full set of administrative privileges.

Testing as Another User: For developers or administrators to test software or configurations from the perspective of another user, without logging out and back in.

Legacy Systems: On some older or minimalist Unix/Linux systems, su might be the primary way to gain root privileges, as sudo may not be installed by default or as widely adopted.

Where to Use su

Single-user administrative tasks

Debugging environment issues

Non-graphical environments

SUDO

The sudo command allows a permitted user to execute a command as another user (by default, the root user) without actually switching their entire shell environment.

It grants temporary, command-specific elevated privileges.

The core idea behind sudo is to execute *one specific command* with elevated privileges, rather than switching to a new user environment entirely.

Syntax:

sudo [options] command

sudo apt update → Execute a command as root

sudo -u john touch /home/john/new_file.txt → Execute a command as a different user

sudo -I → Run a shell as root

Why ?

Principle of Least Privilege → sudo allows users to perform administrative tasks without logging in as root.

Accountability → sudo logs every command executed by users with elevated privileges

Where ?

Everyday administrative tasks

Multi-user environments

When you only need root privileges for a single command

Feature	su	sudo
Password Prompt	Asks for the target user's password (e.g., root's password)	Asks for your own user's password
Scope	Changes your entire user identity for the current shell session	Executes a single command with elevated privileges
Environment	Can load the target user's full environment (su -) or retain original's (su)	Executes command within the current user's environment (unless sudo -i or sudo -s is used for a shell)
Configuration	No specific configuration file (relies on user passwords)	Configured via /etc/sudoers file (using visudo)
Security/Audit	Less granular control; harder to track who did what if root password is shared	Highly granular control; logs all sudo commands for accountability
Principle	"Become" the target user	"Do this command" as the target user
Best Practice	Less common for daily admin; useful for full environment changes or testing	Recommended for most administrative tasks