

Secure Remote Administration using SSH

LinuXpert Systems

WhatsApp: 98841 65649, 90808 01382

E-mail: baskar@linuxpert.in

TABLE OF CONTENTS

1. ssh 1

2. scp 19

3. sftp 32

4. ssh-keygen 48

5. ssh-copy-id 67

6. ssh-agent 81

7. ssh-add 89

8. ssh-keyscan 100

1. ssh

The **ssh** (Secure Shell) command is a Linux command-line client used to securely connect to and manage remote computers over a network. It provides encrypted communication between the local system and a remote system and is commonly used for remote administration, command execution, file transfers, and secure tunneling.

SSH replaces insecure remote-login utilities such as **telnet** and **rlogin** because authentication credentials and network traffic are encrypted.

Syntax

`ssh [OPTIONS] [USER@]HOST`

- **OPTIONS** – Specify connection, authentication, forwarding, or debugging options.
- **USER** – Username on the remote system.
- **HOST** – Hostname or IP address of the remote system.

Basic SSH Connection

Connect to a remote system

```
ssh 192.168.1.10
```

If the local username is baskar, SSH attempts to log in as baskar on the remote system.

Specify a remote username

```
ssh student@192.168.1.10
```

Connect using a hostname

```
ssh student@server.example.com
```

Connect to localhost

```
ssh localhost
```

This is useful for testing whether the SSH server is running correctly on the local machine.

First-Time Connection

When connecting to a server for the first time, SSH normally displays a message similar to:

```
The authenticity of host '192.168.1.10' can't be established.
ED25519 key fingerprint is SHA256:xxxxxxxxxxxxxxx.
Are you sure you want to continue connecting
(yes/no/[fingerprint])?
```

Enter:

```
yes
```

The server's host key is then normally stored in:

```
~/.ssh/known_hosts
```

On subsequent connections, SSH checks the server's host key against the stored key.

Password Authentication

```
ssh student@192.168.1.10
```

The remote system may prompt:

```
student@192.168.1.10's password:
```

The password is entered on the local terminal but is not displayed while typing.

Specify a Port

SSH normally uses TCP port **22**.

Connect to a non-standard SSH port

```
ssh -p 2222 student@192.168.1.10
```

This connects to TCP port 2222 instead of port 22.

Check an SSH server using a specific port

```
ssh -p 2222 student@server.example.com
```

Execute a Remote Command

SSH does not require an interactive shell.

Execute one command

```
ssh student@192.168.1.10 hostname
```

Example output:

```
server01
```

Display the remote kernel version

```
ssh student@192.168.1.10 uname -r
```

Check remote uptime

```
ssh student@192.168.1.10 uptime
```

Check disk usage

```
ssh student@192.168.1.10 df -h
```

Execute multiple commands

```
ssh student@192.168.1.10 "hostname; uptime; df -h"
```

Interactive Remote Shell

```
ssh student@192.168.1.10
```

After successful authentication, you receive a shell on the remote computer:

```
student@server01:~$
```

Commands entered at this point execute on the remote system.

Exit the remote shell with:

```
exit
```

or:

```
Ctrl+D
```

SSH Keys

SSH supports public-key authentication, which is generally preferable to repeatedly entering passwords.

The user's SSH keys are normally stored under:

```
~/.ssh/
```

Common key files include:

```
id_ed25519
id_ed25519.pub
```

or:

```
id_rsa
id_rsa.pub
```

The private key must be protected and should **never be shared**.

Generate an SSH Key

Generate an Ed25519 key

```
ssh-keygen -t ed25519
```

You may see:

```
Enter file in which to save the key:
```

Press **Enter** to use the default location.

You can optionally protect the private key with a passphrase.

Generate an RSA key

```
ssh-keygen -t rsa -b 4096
```

Ed25519 is generally a good modern default when supported.

Copy Public Key to a Server

The `ssh-copy-id` command can install your public key on a remote account.

```
ssh-copy-id student@192.168.1.10
```

After entering the remote account password, the public key is normally added to:

```
~/.ssh/authorized_keys
```

You can then connect using:

```
ssh student@192.168.1.10
```

Specify an SSH Private Key

`-i`

```
ssh -i ~/.ssh/id_ed25519 student@192.168.1.10
```

For another key:

```
ssh -i ~/.ssh/server_key student@192.168.1.10
```

This is particularly useful when managing multiple servers with different SSH keys.

SSH Agent

The SSH agent can hold decrypted private keys in memory so that you do not have to repeatedly enter the key passphrase.

Start the agent

```
eval "$(ssh-agent -s)"
```

Add a key

```
ssh-add ~/.ssh/id_ed25519
```

List keys loaded into the agent

```
ssh-add -l
```

Remove all keys

```
ssh-add -D
```

SSH Configuration File

SSH supports a per-user configuration file:

```
~/.ssh/config
```

For example:

```
Host myserver
    HostName 192.168.1.10
    User student
    Port 22
    IdentityFile ~/.ssh/id_ed25519
```

You can then connect simply with:

```
ssh myserver
```

This is especially useful when administering many Linux servers.

System-Wide SSH Configuration

The system-wide SSH client configuration is normally:

```
/etc/ssh/ssh_config
```

The user's configuration:

```
~/.ssh/config
```

generally takes precedence for user-specific settings.

Verbose Mode

SSH provides several levels of debugging output.

`-v`

```
ssh -v student@192.168.1.10
```

Useful for troubleshooting authentication and connection problems.

`-vv`

```
ssh -vv student@192.168.1.10
```

Provides more detailed debugging information.

`-vvv`

```
ssh -vvv student@192.168.1.10
```

Provides the highest level of SSH client debugging.

This is extremely useful when diagnosing problems involving:

- DNS
- TCP connectivity
- SSH protocol negotiation
- host keys
- authentication
- public keys
- configuration files

Disable Host Key Checking

SSH normally verifies the server's host key.

Options such as:

```
-o StrictHostKeyChecking=no
```

can alter this behavior.

For example:

```
ssh -o StrictHostKeyChecking=no student@192.168.1.10
```

Use this carefully. Disabling host-key verification can make it easier for a man-in-the-middle attack to go unnoticed. It should generally not be used casually on production systems.

Keep the Connection Alive

Some networks disconnect idle SSH sessions.

You can send periodic keepalive messages:

```
ssh -o ServerAliveInterval=60 student@192.168.1.10
```

This asks the SSH client to send a keepalive approximately every 60 seconds.

You can also specify:

```
ssh -o ServerAliveInterval=60 -o ServerAliveCountMax=3
student@192.168.1.10
```

Connection Timeout

`ConnectTimeout`

```
ssh -o ConnectTimeout=10 student@192.168.1.10
```

The SSH client will wait up to approximately 10 seconds while establishing the connection.

This is useful in scripts where an unavailable server should not cause an indefinite wait.

SSH Compression

`-C`

```
ssh -C student@192.168.1.10
```

Enables compression.

It can be useful when transferring compressible data over a relatively slow connection, although compression may not help much for already-compressed files.

SSH X11 Forwarding

SSH can forward X11 graphical applications.

`-X`

```
ssh -X student@192.168.1.10
```

After logging in, an X11 application can potentially be launched:

```
xclock
```

or:

```
gedit
```

depending on the remote system and X11 configuration.

Trusted X11 forwarding

```
ssh -Y student@192.168.1.10
```

`-Y` enables trusted X11 forwarding and should be used with appropriate caution.

SSH Port Forwarding

SSH can create encrypted tunnels between systems.

Local Port Forwarding

`-L`

```
ssh -L 8080:localhost:80 student@192.168.1.10
```

This creates a local port:

`localhost:8080`

that forwards through the SSH server to:

`localhost:80`

as viewed from the remote server.

A common example is securely accessing an internal service through an SSH server.

Remote Port Forwarding

`-R`

```
ssh -R 8080:localhost:80 student@192.168.1.10
```

This creates a listening port on the remote side that forwards traffic through the SSH connection.

Remote forwarding requires appropriate SSH server configuration.

Dynamic Port Forwarding

`-D`

```
ssh -D 1080 student@192.168.1.10
```

Creates a SOCKS proxy on local port `1080`.

Applications configured to use the SOCKS proxy can send their traffic through the SSH connection.

Run SSH in the Background

`-f`

```
ssh -f -N student@192.168.1.10
```

`-N` means that no remote command is executed.

This combination is useful when SSH is being used primarily to establish a tunnel.

SSH Jump Host

SSH can connect to a destination through an intermediate server.

`-J`

```
ssh -J gateway.example.com student@192.168.10.20
```

Here:

```
Local Computer
      |
      v
Gateway
      |
      v
Target Server
```

This is particularly useful when the destination server is not directly reachable from the user's network.

Execute a Command with sudo

You can execute a remote command using `sudo`:

```
ssh student@192.168.1.10 "sudo systemctl status ssh"
```

If the remote user has appropriate `sudo` privileges, the remote system may request the user's password.

Run a Shell Script Remotely

A local script can be piped to SSH:

```
ssh student@192.168.1.10 'bash -s' < script.sh
```

This allows the script to execute on the remote system without first manually copying the file.

Run Commands on Multiple Servers

For example:

```
for host in server1 server2 server3
do
    ssh student@$host hostname
done
```

This is useful for simple administrative tasks across multiple authorized servers.

File Transfer Using SSH

Although `ssh` itself is primarily a remote-login client, it forms the foundation for other secure file-transfer utilities.

Copy a file using `scp`

```
scp file.txt student@192.168.1.10:/home/student/
```

Copy a directory

```
scp -r mydirectory student@192.168.1.10:/home/student/
```

Copy a remote file to the local system

```
scp student@192.168.1.10:/home/student/file.txt .
```

For more advanced transfers, `sftp` can also use the SSH protocol.

Common SSH Errors

Connection refused

```
ssh: connect to host 192.168.1.10 port 22: Connection refused
```

Possible causes:

- SSH server is not running.
- SSH is listening on another port.
- Firewall is rejecting the connection.
- The target system is reachable but SSH is unavailable.

On a system using systemd, check:

```
sudo systemctl status ssh
```

On some distributions the service is named:

```
sudo systemctl status sshd
```

Connection timed out

```
ssh: connect to host 192.168.1.10 port 22: Connection timed out
```

Possible causes include:

- Network connectivity problem
- Firewall filtering
- Incorrect IP address
- Server unavailable

Permission denied

```
Permission denied (publickey,password).
```

Possible causes include:

- Incorrect username
- Incorrect password
- Public key not installed
- Incorrect private-key permissions
- SSH server authentication configuration

Test SSH Connectivity

Test whether port 22 is reachable

```
nc -zv 192.168.1.10 22
```

Alternatively:

```
nmap -p 22 192.168.1.10
```

If the port is open, SSH may be available, although this does not by itself prove that authentication will succeed.

Useful SSH Options

Option	Purpose
`-p PORT`	Specify SSH port
`-l USER`	Specify login username
`-i FILE`	Specify private key
`-v`	Verbose debugging
`-vv`	More detailed debugging
`-vvv`	Maximum debugging
`-C`	Enable compression
`-X`	Enable X11 forwarding
`-Y`	Trusted X11 forwarding
`-L`	Local port forwarding
`-R`	Remote port forwarding
`-D`	Dynamic SOCKS forwarding
`-J`	Use a jump host
`-N`	Do not execute remote command
`-f`	Run SSH in background
`-n`	Redirect stdin from /dev/null
`-4`	Use IPv4 only
`-6`	Use IPv6 only
`-F FILE`	Specify SSH configuration file
`-o OPTION`	Specify configuration option
`-T`	Disable pseudo-terminal allocation
`-t`	Force pseudo-terminal allocation

Practical Examples

1. Connect to a Linux server

```
ssh student@192.168.1.10
```

---

2. Connect using a non-standard port

```
ssh -p 2222 student@192.168.1.10
```

---

3. Use an SSH key

```
ssh -i ~/.ssh/id_ed25519 student@192.168.1.10
```

---

4. Execute a remote command

```
ssh student@192.168.1.10 hostname
```

---

5. Check remote disk space

```
ssh student@192.168.1.10 df -h
```

---

6. Debug a failed connection

```
ssh -vvv student@192.168.1.10
```

---

7. Use a jump server

```
ssh -J gateway.example.com student@192.168.10.20
```

8. Create an encrypted local tunnel

```
ssh -L 8080:localhost:80 student@192.168.1.10
```

---

9. Create a SOCKS proxy

```
ssh -D 1080 student@192.168.1.10
```

---

10. Execute a local script remotely

```
ssh student@192.168.1.10 'bash -s' < script.sh
```

SSH Client Configuration Example

A convenient configuration for a server could be:

```
Host linuxserver
    HostName 192.168.1.10
    User student
    Port 22
    IdentityFile ~/.ssh/id_ed25519
    ServerAliveInterval 60
```

You can then simply use:

```
ssh linuxserver
```

instead of repeatedly typing the complete connection parameters.

Important Security Practices

When using SSH:

1. **Protect private keys.** Never share files such as `id_ed25519` or `id_rsa`.
2. Use **public-key authentication** where practical.
3. Protect private keys with a strong passphrase.
4. Verify server host keys, especially when connecting to a server for the first time.

5. Keep the SSH client and server software updated.
6. Disable unnecessary SSH authentication methods on servers.
7. Avoid disabling `StrictHostKeyChecking` unless there is a specific, understood reason.
8. Use a VPN, firewall, or restricted access controls where appropriate for administrative SSH access.

Quick Reference

Command	Purpose
`ssh host`	Connect to a remote host
`ssh user@host`	Connect as a specific user
`ssh -p 2222 user@host`	Connect using port 2222
`ssh -i key user@host`	Use a specific private key
`ssh host command`	Execute a remote command
`ssh -v user@host`	Debug connection
`ssh-keygen -t ed25519`	Generate an SSH key
`ssh-copy-id user@host`	Install public key
`ssh-add key`	Add key to SSH agent
`ssh -J gateway user@host`	Connect through jump host
`ssh -L ...`	Local port forwarding
`ssh -R ...`	Remote port forwarding
`ssh -D 1080 user@host`	Dynamic SOCKS forwarding
`ssh -X user@host`	X11 forwarding
`ssh -C user@host`	Enable compression
`ssh -4 user@host`	Force IPv4
`ssh -6 user@host`	Force IPv6
`ssh -F config user@host`	Use alternate configuration
`ssh -N host`	Do not execute remote command
`ssh -T host`	Disable pseudo-terminal

Related SSH utilities

The SSH ecosystem also includes:

ssh	→	Secure remote login and command execution
scp	→	Secure file copying
sftp	→	Interactive secure file transfer
ssh-keygen	→	Generate and manage SSH keys
ssh-copy-id	→	Install public keys on remote systems
ssh-agent	→	Manage private keys in memory
ssh-add	→	Add keys to ssh-agent
ssh-keyscan	→	Retrieve SSH host keys

2. scp

The **`scp`** (Secure Copy) command is a Linux command-line utility used to **securely copy files and directories between local and remote systems** over an SSH connection. It provides encrypted data transfer and is commonly used for transferring files between Linux servers, workstations, and other SSH-enabled systems.

`scp` is particularly useful for system administration, remote server management, software deployment, backup operations, and transferring configuration or log files.

Note: Modern OpenSSH also recommends considering **`sftp`** or **`rsync`** for many file-transfer tasks. Nevertheless, **`scp`** remains widely available and useful for straightforward secure file copying.

Syntax

```
scp [OPTIONS] SOURCE DESTINATION
```

- **SOURCE** – The local or remote file/directory to copy.
- **DESTINATION** – The local or remote location where the file/directory should be copied.
- **OPTIONS** – Optional parameters that control authentication, ports, compression, recursion, verbosity, and other transfer behavior.

The general format for a remote location is:

```
username@hostname:/path/to/file
```

---

2. Copy a Local File to a Remote System

To copy a file from the local computer to a remote server:

```
scp file.txt user@192.168.1.10:/home/user/
```

This copies `file.txt` to `/home/user/` on the remote system.

If SSH authentication is enabled, `scp` will ask for the remote user's password.

Example

```
scp report.pdf baskar@192.168.1.20:/home/baskar/Documents/
```

3. Copy a Remote File to the Local System

To download a file from a remote system:

```
scp user@192.168.1.10:/home/user/file.txt .
```

The `.` represents the current directory.

Example

```
scp baskar@192.168.1.20:/var/log/syslog .
```

This downloads `/var/log/syslog` from the remote system to the current directory.

---

4. Copy a Remote File to a Specific Local Directory

```
scp user@192.168.1.10:/home/user/report.pdf /home/baskar/Documents/
```

The remote file is copied into `/home/baskar/Documents/`.

---

5. Copy a File Between Two Remote Systems

`scp` can also be used to copy files between remote systems.

```
scp user1@server1:/home/user1/file.txt user2@server2:/home/user2/
```

Depending on the OpenSSH implementation and configuration, this may involve the local machine as an intermediary or use a direct remote-to-remote transfer mode.

---

6. Copy Multiple Files

Multiple files can be specified as the source:

```
scp file1.txt file2.txt file3.txt user@192.168.1.10:/home/user/
```

Example:

```
scp *.pdf baskar@192.168.1.20:/home/baskar/Documents/
```

This copies all PDF files in the current directory.

7. Copy a Directory

Use the `-r` option to recursively copy a directory and its contents.

```
scp -r directory/ user@192.168.1.10:/home/user/
```

Example:

```
scp -r project/ baskar@192.168.1.20:/home/baskar/
```

This copies the entire `project` directory.

---

8. Copy a Directory from Remote to Local

```
scp -r user@192.168.1.10:/home/user/project/ .
```

This downloads the complete `project` directory into the current directory.

---

9. Specify an SSH Port

By default, SSH uses TCP port **22**.

If the remote SSH server uses a different port, use `-P`:

```
scp -P 2222 file.txt user@192.168.1.10:/home/user/
```

Important: `scp` uses uppercase `-P` for the SSH port. Lowercase `-p` has a different purpose.

---

10. Use an SSH Private Key

The `-i` option specifies an SSH private key.

```
scp -i ~/.ssh/id_rsa file.txt user@192.168.1.10:/home/user/
```

For example:

```
scp -i ~/.ssh/server_key report.pdf baskar@192.168.1.20:/home/baskar/
```

This is particularly useful when password authentication is disabled.

11. Copy Using an SSH Key with a Different Name

```
scp -i ~/.ssh/myserver_key backup.tar.gz user@server:/backup/
```

The specified private key is used for authentication.

---

12. Enable Compression

The `-C` option enables SSH compression during the transfer.

```
scp -C largefile.txt user@192.168.1.10:/home/user/
```

Compression can be useful when transferring highly compressible text files over a relatively slow network.

For already compressed files such as `.zip`, `.gz`, `.jpg`, and `.mp4`, compression generally provides little benefit.

---

13. Preserve File Attributes

Use `-p` to preserve modification times, access times, and modes from the source file.

```
scp -p file.txt user@192.168.1.10:/home/user/
```

This is useful when copying files where their original timestamps and permissions are important.

---

14. Display Detailed Transfer Information

Use `-v` for verbose output:

```
scp -v file.txt user@192.168.1.10:/home/user/
```

This displays detailed information about:

- SSH connection establishment
- Authentication
- Key exchange
- Cipher selection
- File transfer
- Connection termination

Verbose mode is particularly useful for troubleshooting connection problems.

15. Specify the SSH Protocol Configuration

`scp` uses SSH and therefore inherits many SSH options.

For example:

```
scp -o ConnectTimeout=10 file.txt user@192.168.1.10:/home/user/
```

This limits the connection establishment time.

Another example:

```
scp -o StrictHostKeyChecking=no file.txt user@192.168.1.10:/home/user/
```

However, disabling strict host-key checking can weaken SSH security and should generally be avoided unless there is a controlled reason to do so.

---

16. Copy Files Using a Hostname

An IP address is not required.

```
scp report.pdf user@server.example.com:/home/user/
```

If the hostname is defined in `/etc/hosts`:

```
scp report.pdf user@linuxserver:/home/user/
```

---

17. Copy a File to the Remote User's Home Directory

The remote user's home directory can be represented using `~`:

```
scp file.txt user@192.168.1.10:~
or:

scp file.txt user@192.168.1.10:~/Documents/
```

---

18. Copy Files with Spaces in Their Names

Use quotes around the filename:

```
scp "my report.pdf" user@192.168.1.10:/home/user/
```

Alternatively, escape the spaces:

```
scp my\ report.pdf user@192.168.1.10:/home/user/
```

19. Copy a File from a Remote Server Using a Non-Standard Port

```
scp -P 2200 user@server:/var/log/application.log .
```

This connects to SSH port `2200`.

---

20. Copy an Entire Website Directory

For example:

```
scp -r website/ user@192.168.1.20:/var/www/
```

If the remote user does not have permission to write directly to `/var/www/`, first copy the directory to the user's home directory and then move it using `sudo` on the server.

---

21. Copy a Backup File to a Remote Server

```
scp backup.tar.gz user@192.168.1.20:/backup/
```

A typical backup workflow might be:

```
tar -czf backup.tar.gz /etc
```

```
scp backup.tar.gz user@192.168.1.20:/backup/
```

---

22. Copy a File from a Remote Server to Another Local Directory

```
scp user@192.168.1.20:/backup/backup.tar.gz /home/baskar/backups/
```

---

23. Copy Files Using an IPv6 Address

IPv6 addresses should normally be enclosed in square brackets:

```
scp file.txt user@[2001:db8::10]:/home/user/
```

24. Limit the Transfer Bandwidth

The `-l` option can limit the bandwidth used by `scp`.

```
scp -l 1000 file.iso user@192.168.1.20:/home/user/
```

The value is specified in **Kbit/s**.

For example:

```
-l 1000
```

limits the transfer to approximately **1 Mbit/s**.

---

25. Use a Specific SSH Cipher

The `-c` option can specify an SSH cipher supported by the installed OpenSSH version:

```
scp -c aes256-gcm@openssh.com file.txt user@server:/home/user/
```

Available ciphers can be examined with:

```
ssh -Q cipher
```

The exact set of available ciphers depends on the OpenSSH version and system configuration.

---

26. Use an SSH Configuration Alias

Instead of repeatedly specifying the username, hostname, port, and key, configure the server in:

```
~/.ssh/config
```

For example:

```
Host myserver
    HostName 192.168.1.20
    User baskar
    Port 2222
    IdentityFile ~/.ssh/server_key
```

Then the file can be copied simply using:

```
scp report.pdf myserver:/home/baskar/
```

This is particularly convenient when administering multiple Linux servers.

27. Copy Files from a Server Running SSH on Port 2222

```
scp -P 2222 backup.tar.gz user@server:/backup/
```

To test the same SSH connection:

```
ssh -p 2222 user@server
```

---

28. Check the `scp` Version

```
scp -V
```

On many OpenSSH installations, version information may instead be displayed through:

```
ssh -V
```

because `scp` is part of the OpenSSH suite.

---

29. Display the Manual Page

```
man scp
```

Search for a specific option:

```
man scp
```

Then press `/` and search for:

```
recursive
```

30. Commonly Used `scp` Options

Option	Purpose
`-r`	Recursively copy directories
`-p`	Preserve modification/access times and modes
`-P`	Specify SSH port
`-i`	Specify private SSH key
`-C`	Enable compression
`-v`	Verbose/debug output
`-l`	Limit bandwidth
`-c`	Specify SSH cipher
`-o`	Specify an SSH configuration option
`-q`	Suppress progress and warning messages
`-4`	Use IPv4 only
`-6`	Use IPv6 only
`-F`	Specify an alternative SSH configuration file
`-S`	Specify an alternative SSH program

---

31. Practical Examples

Upload a document

```
scp document.pdf user@server:/home/user/
```

Download a document

```
scp user@server:/home/user/document.pdf .
```

Upload a directory

```
scp -r project/ user@server:/home/user/
```

Download a directory

```
scp -r user@server:/home/user/project/ .
```

Use a different SSH port

```
scp -P 2222 file.txt user@server:/home/user/
```

Use an SSH key

```
scp -i ~/.ssh/id_ed25519 file.txt user@server:/home/user/
```

Upload several files

```
scp file1.txt file2.txt file3.txt user@server:/home/user/
```

Upload all `.log` files

```
scp *.log user@server:/home/user/logs/
```

Copy while preserving attributes

```
scp -p config.conf user@server:/home/user/
```

Copy with compression

```
scp -C database.sql user@server:/backup/
```

Troubleshoot an SSH connection

```
scp -v file.txt user@server:/home/user/
```

32. `scp` vs `sftp` vs `rsync`

Feature	scp	sftp	rsync
Secure transfer	Yes	Yes	Yes, normally over SSH
Copy files	Yes	Yes	Yes
Copy directories	Yes	Yes	Yes
Interactive file management	No	Yes	No
Incremental synchronization	No	No	**Yes**
Resume/efficient repeated transfers	Limited	Better suited	**Excellent**
Simple one-time copy	**Excellent**	Good	Good
Backup/synchronization	Basic	Basic	**Excellent**

For a simple one-time transfer, **`scp`** is often the easiest choice. For repeated synchronization of large directories, **`rsync`** is generally more efficient because it can transfer only changed portions/files.

---

33. Security Considerations

`scp` transfers data through **SSH**, providing encryption and server authentication. However, good SSH security practices should still be followed:

- Use **SSH keys** instead of passwords where practical.
- Protect private keys with appropriate permissions.
- Verify the server's SSH host key.
- Avoid disabling `StrictHostKeyChecking` unnecessarily.
- Use a non-root account for routine file transfers.
- Restrict SSH access using firewall and SSH configuration policies.
- Use `sudo` on the remote system only when administrative privileges are actually required.

Private SSH keys should normally have restrictive permissions:

```
chmod 600 ~/.ssh/id_ed25519
```

34. Important Difference Between `scp` and `cp`

The local `cp` command:

```
cp file.txt /home/user/
```

copies files **within the same system**.

The `scp` command:

```
scp file.txt user@192.168.1.20:/home/user/
```

copies files **between systems using SSH**.

Therefore:

```
cp
↓
Local file copying
```

```
scp
↓
Secure file copying over SSH
```

---

35. Summary

The `scp` command is an important Linux networking and system-administration utility for **securely transferring files and directories between computers**.

The most important forms to remember are:

```
scp file user@server:/path/
```

Local → Remote

```
scp user@server:/path/file .
```

Remote → Local

```
scp -r directory user@server:/path/
```

Directory → Remote

```
scp -P 2222 file user@server:/path/
```

Using a non-standard SSH port

```
scp -i ~/.ssh/id_ed25519 file user@server:/path/
```

Using an SSH private key

`scp` is particularly useful because it connects the concepts of **SSH, remote login, authentication, secure communication, file transfer, and Linux system administration** into a single practical command.

3. SFTP

The **`sftp` (Secure File Transfer Protocol)** command is a secure, interactive file-transfer utility in Linux used to transfer files between a local system and a remote system over an encrypted **SSH connection**. It provides functionality similar to traditional `ftp`, but unlike FTP, authentication credentials and file data are protected by SSH encryption.

`SFTP` is commonly used by system administrators, developers, and users to upload, download, manage, and transfer files securely between Linux systems.

Syntax

```
sftp [OPTIONS] [USER@]HOST
```

- **USER** – Username on the remote system.
- **HOST** – Hostname or IP address of the remote system.
- **OPTIONS** – Optional parameters that modify the behavior of `sftp`.

For example:

```
sftp student@192.168.1.20
```

The command connects to the remote system using the username `student`.

---

1. Connect to a Remote System

```
sftp user@192.168.1.10
```

Example:

```
sftp baskar@192.168.1.20
```

After successful authentication, an interactive SFTP prompt is displayed:

```
Connected to 192.168.1.20.
sftp>
```

---

2. Connect Using an IP Address

```
sftp user@192.168.1.100
```

Example:

```
sftp student@10.10.10.50
```

---

3. Connect Using a Hostname

```
sftp user@server.example.com
```

Example:

```
sftp admin@linuxserver.example.com
```

---

4. Specify an SSH Port

SFTP normally uses the SSH port **22**.

If the SSH server is configured to use another port:

```
sftp -P 2222 user@192.168.1.10
```

> Note that SFTP uses uppercase `-P` for specifying the port.

---

5. Display SFTP Help

From the Linux shell:

```
sftp -h
```

Inside an SFTP session:

```
sftp> help
```

or:

```
sftp> ?
```

Working with Remote Files

6. Display the Remote Working Directory

```
sftp> pwd
```

Example output:

```
Remote working directory: /home/student
```

pwd displays the current directory on the remote system.

---

7. Display the Local Working Directory

```
sftp> lpwd
```

Example:

```
Local working directory: /home/baskar
```

---

8. List Remote Files

```
sftp> ls
```

Example:

```
document.pdf
report.txt
linux/
```

---

9. List Remote Files with Details

```
sftp> ls -l
```

Example:

```
-rw-r--r--    1 student student  24576 report.txt
drwxr-xr-x    2 student student   4096 linux
```

10. List Local Files

```
sftp> lls
```

This is equivalent to running `ls` on the local system.

```
sftp> lls -l
```

---

11. Change Remote Directory

```
sftp> cd /var/www
```

Example:

```
sftp> cd /home/student/documents
```

---

12. Change to the Remote Home Directory

```
sftp> cd ~
```

---

13. Change Local Directory

Use `lcd`:

```
sftp> lcd /home/baskar/Documents
```

Example:

```
sftp> lcd /tmp
```

Downloading Files

14. Download a File

```
sftp> get report.txt
```

The file is downloaded to the current local directory.

---

15. Download and Rename a File

```
sftp> get report.txt local-report.txt
```

The remote file `report.txt` is saved locally as `local-report.txt`.

---

16. Download a File to a Specific Directory

```
sftp> get report.txt /home/baskar/Documents/
```

---

17. Download Multiple Files

```
sftp> mget *.txt
```

This downloads all `.txt` files from the remote directory.

Example:

```
sftp> mget *.pdf
```

---

18. Download an Entire Directory

Use the recursive option:

```
sftp> get -r linux/
```

This downloads the entire `linux` directory and its contents.

Uploading Files

19. Upload a File

```
sftp> put report.txt
```

This uploads `report.txt` from the local system to the current remote directory.

---

20. Upload and Rename a File

```
sftp> put report.txt server-report.txt
```

---

21. Upload a File to a Specific Remote Directory

```
sftp> put report.txt /home/student/documents/
```

---

22. Upload Multiple Files

```
sftp> mput *.txt
```

Example:

```
sftp> mput *.pdf
```

---

23. Upload an Entire Directory

```
sftp> put -r linux/
```

This recursively uploads the directory and its contents.

Creating and Managing Directories

24. Create a Remote Directory

```
sftp> mkdir backups
```

---

25. Create a Directory Using an Absolute Path

```
sftp> mkdir /home/student/backups
```

---

26. Remove a Remote File

```
sftp> rm oldfile.txt
```

---

27. Remove an Empty Remote Directory

```
sftp> rmdir old-directory
```

---

28. Remove a Directory Recursively

Depending on the OpenSSH SFTP implementation, `rm -r` can be used:

```
sftp> rm -r old-directory
```

Use this carefully because it can remove many files.

File Permissions and Ownership

29. Change File Permissions

```
sftp> chmod 644 report.txt
```

For an executable file:

```
sftp> chmod 755 script.sh
```

---

30. Change File Owner

If permitted by the remote server:

```
sftp> chown 1001 report.txt
```

---

31. Change Group Ownership

```
sftp> chgrp 1001 report.txt
```

These operations require appropriate permissions on the remote system.

SFTP Interactive Commands

Some useful SFTP commands are:

Command	Purpose
`pwd`	Display remote working directory
`lpwd`	Display local working directory
`ls`	List remote files
`lls`	List local files
`cd`	Change remote directory
`lcd`	Change local directory
`get`	Download a file
`mget`	Download multiple files
`put`	Upload a file
`mput`	Upload multiple files
`mkdir`	Create remote directory
`rmdir`	Remove remote directory
`rm`	Remove remote file
`chmod`	Change permissions
`chown`	Change ownership
`chgrp`	Change group
`rename`	Rename remote file
`df`	Display remote filesystem usage
`stat`	Display file information
`help`	Display SFTP commands
`quit`	Exit SFTP

Renaming Files

32. Rename a Remote File

```
sftp> rename old.txt new.txt
```

Example:

```
sftp> rename report.txt final-report.txt
```

File Information

33. Display File Information

```
sftp> stat report.txt
```

This can display information such as:

- File size
- Permissions
- Ownership
- Modification time
- File type

---

34. Display Remote Filesystem Information

```
sftp> df -h
```

Example:

```
sftp> df -h
```

This displays filesystem capacity and available space on the remote system.

Preserving File Attributes

35. Preserve File Modification Times and Permissions

Use `-p` with `get` or `put`:

```
sftp> get -p report.txt
```

or:

```
sftp> put -p report.txt
```

This attempts to preserve file attributes such as modification time and permissions.

Non-Interactive SFTP

SFTP can also be used directly from the Linux command line.

---

36. Download a File

```
sftp user@192.168.1.10:/home/user/report.txt
```

However, for automated downloads, the `-b` batch option is often more appropriate.

---

37. Use an SFTP Batch File

Create a file:

```
nano sftp_commands.txt
```

Add:

```
cd /home/student/documents
lcd /home/baskar/downloads
get report.pdf
bye
```

Run:

```
sftp -b sftp_commands.txt student@192.168.1.10
```

This allows several SFTP commands to be executed automatically.

Using SSH Keys

SFTP supports SSH public-key authentication.

38. Specify a Private Key

```
sftp -i ~/.ssh/id_ed25519 student@192.168.1.10
```

Another example:

```
sftp -i ~/.ssh/id_rsa student@server.example.com
```

This is particularly useful for automated transfers.

Using a Different SSH Configuration

39. Use an SSH Configuration File

If `~/.ssh/config` contains:

```
Host training-server
    HostName 192.168.1.20
    User student
    Port 2222
    IdentityFile ~/.ssh/id_ed25519
```

You can simply connect using:

```
sftp training-server
```

This is convenient when connecting regularly to the same server.

Verbose Mode

40. Troubleshoot an SFTP Connection

Use:

```
sftp -v user@192.168.1.10
```

For more detailed debugging:

```
sftp -vv user@192.168.1.10
```

Maximum verbosity:

```
sftp -vvv user@192.168.1.10
```

This is useful for diagnosing:

- Authentication failures
- SSH connection problems
- Key problems
- Port problems
- SFTP subsystem problems

Common Examples

41. Upload a Website Backup

```
sftp admin@192.168.1.20
```

Then:

```
sftp> cd /backup
sftp> lcd /home/admin/backups
sftp> put website.tar.gz
sftp> bye
```

---

42. Download a Server Log

```
sftp admin@192.168.1.20
```

Then:

```
sftp> cd /var/log
sftp> lcd /home/admin/logs
sftp> get syslog
sftp> bye
```

---

43. Upload Multiple Documents

```
sftp student@192.168.1.20
```

Then:

```
sftp> cd /home/student/assignments
sftp> lcd /home/baskar/assignments
sftp> mput *.pdf
sftp> bye
```

44. Download an Entire Course Directory

```
sftp student@192.168.1.20
```

Then:

```
sftp> lcd /home/baskar/courses
sftp> cd /home/student/linux-course
sftp> get -r lectures
sftp> bye
```

Important SFTP Options

Option	Purpose
`-B buffer_size`	Specify buffer size
`-b batchfile`	Execute commands from a batch file
`-C`	Enable compression
`-F configfile`	Specify SSH configuration file
`-i identity_file`	Specify private key
`-o option`	Specify an SSH option
`-P port`	Specify SSH port
`-q`	Quiet mode
`-R num_requests`	Specify number of outstanding requests
`-v`	Verbose mode
`-vv`	More verbose debugging
`-vvv`	Maximum debugging information

SFTP vs SCP

Both `sftp` and `scp` use SSH for secure file transfer, but they are designed somewhat differently.

Feature	SFTP	SCP
Interactive operation	Yes	No
Upload files	Yes	Yes
Download files	Yes	Yes
Multiple file operations	Yes	Yes
Directory browsing	Yes	No
Remote file management	Yes	Limited
Resume support	Better suited	Limited
Batch operations	Yes	Limited
SSH encryption	Yes	Yes
Uses SSH	Yes	Yes

SFTP is generally preferable when you need to interactively browse and manage files on a remote server. `scp` is convenient for straightforward file-copy operations.

Security Considerations

SFTP provides encryption through SSH, protecting:

- Usernames
- Passwords
- File contents
- File transfers
- Commands exchanged during the session

For regular administrative or automated transfers, **SSH public-key authentication** is preferable to password authentication.

Avoid transferring sensitive files through unencrypted protocols such as traditional FTP when SFTP is available.

Quick Reference

```
# Connect
sftp user@server

# Connect on another SSH port
sftp -P 2222 user@server

# Connect using SSH key
sftp -i ~/.ssh/id_ed25519 user@server

# Remote directory
sftp> pwd

# Local directory
sftp> lpwd

# Remote files
sftp> ls

# Local files
sftp> lls

# Change remote directory
sftp> cd /path

# Change local directory
sftp> lcd /path
```

```
# Download
sftp> get file.txt

# Download multiple files
sftp> mget *.txt

# Download directory
sftp> get -r directory/

# Upload
sftp> put file.txt

# Upload multiple files
sftp> mput *.txt

# Upload directory
sftp> put -r directory/

# Create directory
sftp> mkdir directory

# Delete file
sftp> rm file.txt

# Rename file
sftp> rename old.txt new.txt

# Change permissions
sftp> chmod 644 file.txt

# File information
sftp> stat file.txt

# Filesystem information
sftp> df -h

# Help
sftp> help

# Exit
sftp> bye
```

Key point

`sftp` is the secure, SSH-based replacement for traditional FTP when you need an interactive environment for transferring and managing files on a remote Linux system.

4. ssh-keygen

The **`ssh-keygen`** command is a Linux utility used to **generate, manage, convert, and inspect SSH authentication keys**. It is primarily used with the OpenSSH suite for passwordless and public-key authentication.

SSH keys provide a more secure alternative to password-based authentication and are widely used for **remote server administration, secure file transfer, Git repositories, automation, and system-to-system authentication**.

Syntax

```
ssh-keygen [OPTIONS] [ARGUMENTS]
```

Common options include:

- `-t` – Specify the type of key to create.
- `-b` – Specify the number of bits in the key.
- `-C` – Add a comment to the public key.
- `-f` – Specify the filename for the key.
- `-N` – Specify a passphrase.
- `-p` – Change the passphrase of an existing key.
- `-l` – Display the fingerprint of a key.
- `-y` – Generate a public key from a private key.
- `-i` – Convert an externally generated key into OpenSSH format.
- `-e` – Convert an OpenSSH public key into another format.
- `-R` – Remove a host key from `known_hosts`.
- `-F` – Search for a host in `known_hosts`.

1. Generate an SSH Key

The simplest way to generate an SSH key is:

```
ssh-keygen
```

You will normally be prompted for:

```
Enter file in which to save the key (/home/user/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
```

This creates:

```
~/.ssh/id_ed25519
~/.ssh/id_ed25519.pub
```

The first file is the **private key** and the second is the **public key**.

Never share the private key.

---

2. Generate an Ed25519 Key

Ed25519 is a modern and generally preferred SSH key type.

```
ssh-keygen -t ed25519
```

Specify a comment:

```
ssh-keygen -t ed25519 -C "admin@server"
```

Specify a filename:

```
ssh-keygen -t ed25519 -f ~/.ssh/server_key
```

This produces:

```
~/.ssh/server_key
~/.ssh/server_key.pub
```

3. Generate an RSA Key

RSA keys are still supported and may be required for compatibility with older systems.

```
ssh-keygen -t rsa
```

Generate a 4096-bit RSA key:

```
ssh-keygen -t rsa -b 4096
```

With a comment:

```
ssh-keygen -t rsa -b 4096 -C "admin@example.com"
```

The resulting files may be:

```
~/.ssh/id_rsa
~/.ssh/id_rsa.pub
```

---

4. Specify the Key Filename

Use `-f` to specify where the key should be stored.

```
ssh-keygen -t ed25519 -f ~/.ssh/myserver
```

Files created:

```
~/.ssh/myserver
~/.ssh/myserver.pub
```

This is particularly useful when you use different keys for different servers.

For example:

```
ssh-keygen -t ed25519 -f ~/.ssh/college_server
```

and:

```
ssh-keygen -t ed25519 -f ~/.ssh/cloud_server
```

---

5. Add a Comment to a Key

The `-C` option adds a comment.

```
ssh-keygen -t ed25519 -C "baskar@linuxserver"
```

The comment appears at the end of the public key:

```
ssh-ed25519 AAAAC3...xyz baskar@linuxserver
```

The comment is only an identifier; it does not affect authentication.

---

6. Protect the Private Key with a Passphrase

When generating a key:

```
ssh-keygen -t ed25519
```

enter a passphrase when prompted.

A passphrase protects the private key if somebody obtains the key file.

For example:

```
Enter passphrase (empty for no passphrase):
```

A passphrase is recommended for keys used interactively.

---

7. Generate a Key Without Interactive Prompts

You can specify the filename and passphrase directly:

```
ssh-keygen -t ed25519 -f ~/.ssh/mykey -N "mypassword"
```

However, putting a passphrase directly on the command line can expose it through shell history or process information.

For automated scripts, consider the security implications carefully.

---

8. Generate a Key Without a Passphrase

For situations where a passphrase-less key is specifically required:

```
ssh-keygen -t ed25519 -f ~/.ssh/automation_key -N ""
```

This can be useful for automated services, but the private key must be **strictly protected with filesystem permissions and other controls**.

9. Display the Public Key

Display an existing public key:

```
cat ~/.ssh/id_ed25519.pub
```

Example:

```
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAA... user@hostname
```

The public key can be copied to a remote server.

---

10. Generate a Public Key from a Private Key

If you have the private key but the corresponding `.pub` file is missing:

```
ssh-keygen -y -f ~/.ssh/id_ed25519
```

For example:

```
ssh-keygen -y -f ~/.ssh/myserver
```

This outputs the corresponding public key.

You can save it:

```
ssh-keygen -y -f ~/.ssh/myserver > ~/.ssh/myserver.pub
```

---

11. Display the Key Fingerprint

Use:

```
ssh-keygen -l -f ~/.ssh/id_ed25519.pub
```

Example:

```
256 SHA256:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx user@hostname (ED25519)
```

The fingerprint provides a short representation of the public key.

12. Display Fingerprint in Different Formats

SHA-256 is normally the default:

```
ssh-keygen -l -f ~/.ssh/id_ed25519.pub
```

MD5 format:

```
ssh-keygen -l -E md5 -f ~/.ssh/id_ed25519.pub
```

Bubble Babble format:

```
ssh-keygen -l -E bubblebabble -f ~/.ssh/id_ed25519.pub
```

---

13. Display the Randomart Image

SSH can display a visual representation of a key:

```
ssh-keygen -lv -f ~/.ssh/id_ed25519.pub
```

Example:

```
+--[ED25519 256]--+
|        .+o      |
|       . = .     |
|        o +      |
|       . + .     |
|        o S .    |
|         + *     |
|        . * .    |
|         o .     |
|          .      |
+----[SHA256]-----+
```

This is sometimes called **SSH randomart**.

---

14. Change the Passphrase

To change the passphrase of an existing private key:

```
ssh-keygen -p -f ~/.ssh/id_ed25519
```

You will be prompted for the old and new passphrases.

15. Change the Key Comment

Use:

```
ssh-keygen -c -f ~/.ssh/id_ed25519
```

For example:

```
ssh-keygen -c -C "linux-admin-key" -f ~/.ssh/id_ed25519
```

The `-C` option supplies the new comment.

---

16. Check Whether a Key Already Exists

List the SSH directory:

```
ls -la ~/.ssh/
```

Typical output:

```
id_ed25519
id_ed25519.pub
known_hosts
authorized_keys
config
```

You can inspect the public keys:

```
ls ~/.ssh/*.pub
```

---

17. Copy a Public Key to a Remote Server

Although `ssh-keygen` creates the key, `ssh-copy-id` is normally used to install it on a remote server:

```
ssh-copy-id user@192.168.1.10
```

For a non-default key:

```
ssh-copy-id -i ~/.ssh/myserver.pub user@192.168.1.10
```

The public key is added to:

```
~/.ssh/authorized_keys
```

on the remote system.

You can then connect using:

```
ssh user@192.168.1.10
```

---

18. Use a Specific Private Key

The `ssh` client can be instructed to use a particular key:

```
ssh -i ~/.ssh/myserver user@192.168.1.10
```

For example:

```
ssh -i ~/.ssh/college_server administrator@192.168.1.20
```

---

19. Generate an ECDSA Key

ECDSA keys can be generated with:

```
ssh-keygen -t ecdsa
```

Specify the key size:

```
ssh-keygen -t ecdsa -b 521
```

Common ECDSA sizes include:

```
256
384
521
```

For new deployments, Ed25519 is generally preferable unless compatibility requirements dictate otherwise.

---

20. Generate a DSA Key

Older OpenSSH versions supported:

```
ssh-keygen -t dsa
```

However, **DSA/SSH protocol 2 support has been disabled or removed in modern OpenSSH configurations**.

It should not be used for new deployments.

SSH Key Types

Key Type	Example	Recommendation
Ed25519	`ssh-ed25519`	**Preferred for most new systems**
RSA	`ssh-rsa`	Good compatibility choice
ECDSA	`ecdsa-sha2-nistp256`	Supported
DSA	`ssh-dss`	**Obsolete; avoid**

21. Manage `known_hosts`

`ssh-keygen` can also manage SSH host keys stored in:

`~/.ssh/known_hosts`

Search for a host:

```
ssh-keygen -F server.example.com
```

Search by IP address:

```
ssh-keygen -F 192.168.1.10
```

Remove a host:

```
ssh-keygen -R server.example.com
```

Remove an IP address:

```
ssh-keygen -R 192.168.1.10
```

This is useful when a server's host key has legitimately changed.

22. Hash Hostnames in `known_hosts`

You can hash hostnames in the known-hosts file:

```
ssh-keygen -H
```

This makes the hostnames less obvious if someone obtains the file.

A backup may be created automatically, such as:

```
known_hosts.old
```

---

23. Convert a Private Key

`ssh-keygen` can convert private-key formats.

For example:

```
ssh-keygen -p -m PEM -f ~/.ssh/id_rsa
```

The `-m` option specifies the key format.

This can be useful when working with applications that require a particular private-key format.

---

24. Convert an OpenSSH Public Key

Convert an OpenSSH public key into another supported format:

```
ssh-keygen -e -f ~/.ssh/id_ed25519.pub
```

For example, RFC 4716 format:

```
ssh-keygen -e -m RFC4716 -f ~/.ssh/id_rsa.pub
```

---

25. Import an RFC 4716 Public Key

Convert an RFC 4716 public key into OpenSSH format:

```
ssh-keygen -i -f publickey.txt
```

For RFC 4716:

```
ssh-keygen -i -m RFC4716 -f publickey.txt
```

26. Generate a Host Key

`ssh-keygen` can also generate keys used by an SSH server.

For example:

```
ssh-keygen -t ed25519 -f /etc/ssh/ssh_host_ed25519_key
```

The corresponding public key is:

```
/etc/ssh/ssh_host_ed25519_key.pub
```

SSH server host keys are normally generated automatically when OpenSSH is installed.

---

27. Generate All Default Host Keys

On systems where host keys are missing:

```
ssh-keygen -A
```

This generates the host keys that are missing.

For example:

```
/etc/ssh/ssh_host_rsa_key
/etc/ssh/ssh_host_ecdsa_key
/etc/ssh/ssh_host_ed25519_key
```

---

28. Test a Public Key

You can inspect the contents of a public key:

```
ssh-keygen -l -f ~/.ssh/id_ed25519.pub
```

This is useful for verifying:

- Key type
- Key length
- Fingerprint
- Key identity/comment

29. Generate a Certificate

OpenSSH also supports certificates.

A user certificate can be signed using a Certificate Authority (CA) key:

```
ssh-keygen -s ca_key -I user01 -n user01 user01.pub
```

This creates a certificate such as:

```
user01-cert.pub
```

Certificates are useful in large environments where administrators want centralized SSH authentication rather than placing individual public keys into every server's `authorized_keys`.

---

30. Generate a Certificate for a Host

For example:

```
ssh-keygen -s host_ca -I server01 -h -n server01 server01.pub
```

The `-h` option specifies that the certificate is for a host.

---

31. View an SSH Certificate

Use:

```
ssh-keygen -L -f user01-cert.pub
```

This displays certificate information such as:

```
Type
Key ID
Serial
Principals
Valid after
Valid before
Critical Options
Extensions
```

32. Remove a Key from `known_hosts`

If an SSH server's host key has changed:

```
ssh-keygen -R hostname
```

For example:

```
ssh-keygen -R 192.168.1.100
```

You can then reconnect:

```
ssh user@192.168.1.100
```

SSH will ask you to verify the new host key.

Do not blindly remove host keys merely to bypass an SSH warning. A changed host key can indicate a legitimate server rebuild, but it can also indicate a man-in-the-middle attack.

---

33. Specify the Number of Bits

For RSA:

```
ssh-keygen -t rsa -b 4096
```

For ECDSA:

```
ssh-keygen -t ecdsa -b 521
```

For Ed25519, the key size is fixed by the algorithm, so `-b` is not normally used.

---

34. Create Multiple SSH Keys

You can maintain different keys for different purposes:

```
ssh-keygen -t ed25519 -f ~/.ssh/college_server
```

```
ssh-keygen -t ed25519 -f ~/.ssh/github_key
```

```
ssh-keygen -t ed25519 -f ~/.ssh/cloud_server
```

This allows keys to be revoked or replaced independently.

35. SSH Configuration with Multiple Keys

After creating multiple keys, the SSH client configuration file:

```
~/.ssh/config
```

can specify which key should be used for each server.

Example:

```
Host college-server
    HostName 192.168.1.10
    User admin
    IdentityFile ~/.ssh/college_server

Host cloud-server
    HostName 203.0.113.10
    User administrator
    IdentityFile ~/.ssh/cloud_server
```

You can then simply use:

```
ssh college-server
```

or:

```
ssh cloud-server
```

---

36. SSH Key File Permissions

Private keys should be protected:

```
chmod 600 ~/.ssh/id_ed25519
```

Public keys can normally be:

```
chmod 644 ~/.ssh/id_ed25519.pub
```

The SSH directory itself should normally be:

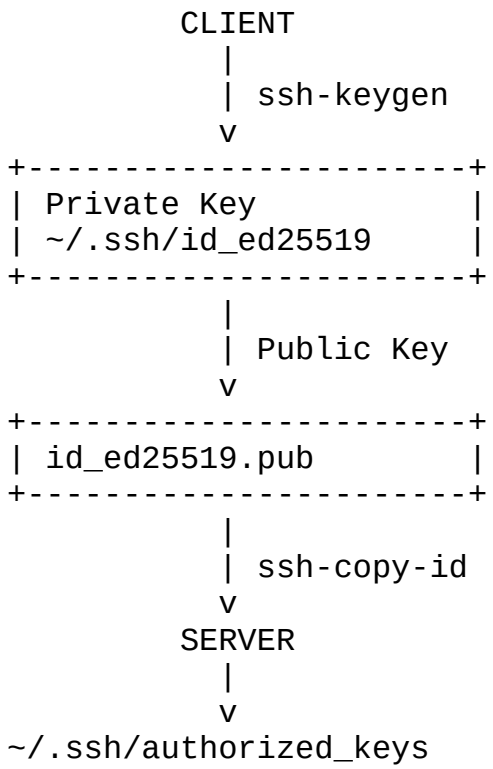
```
chmod 700 ~/.ssh
```

Check permissions:

```
ls -ld ~/.ssh
ls -l ~/.ssh/
```

37. Typical SSH Key Authentication Workflow

A typical workflow is:



When the client connects:

```
ssh user@server
```

the SSH server verifies that the client possesses the private key corresponding to an authorized public key.

The **private key itself is never sent to the server**.

---

38. Important SSH Files

File	Purpose
`~/.ssh/id_ed25519`	Ed25519 private key
`~/.ssh/id_ed25519.pub`	Ed25519 public key
`~/.ssh/id_rsa`	RSA private key
`~/.ssh/id_rsa.pub`	RSA public key
`~/.ssh/authorized_keys`	Public keys authorized for a user
`~/.ssh/known_hosts`	Known server host keys
`~/.ssh/config`	SSH client configuration
`/etc/ssh/ssh_config`	System-wide SSH client configuration
`/etc/ssh/sshd_config`	SSH server configuration
`/etc/ssh/ssh_host_*`	SSH server host keys

39. Useful `ssh-keygen` Options

Option	Description
`-t type`	Specify key type
`-b bits`	Specify key size
`-C comment`	Add/change key comment
`-f filename`	Specify key file
`-N passphrase`	Set new passphrase
`-p`	Change passphrase
`-c`	Change comment
`-l`	Display fingerprint
`-v`	Verbose output
`-y`	Extract public key from private key
`-i`	Import key
`-e`	Export key
`-L`	Display certificate information
`-R hostname`	Remove host from known_hosts
`-F hostname`	Find host in known_hosts
`-H`	Hash hostnames in known_hosts
`-A`	Generate missing host keys
`-s CA_key`	Sign a certificate
`-I identity`	Specify certificate identity
`-n principals`	Specify certificate principals

40. Practical Examples

Generate the recommended type of key for a new user

```
ssh-keygen -t ed25519 -C "student@linuxlab"
```

Generate a 4096-bit RSA key

```
ssh-keygen -t rsa -b 4096 -C "server-admin"
```

Generate a separate key for a server

```
ssh-keygen -t ed25519 -f ~/.ssh/lab_server
```

Display the public key

```
cat ~/.ssh/lab_server.pub
```

Display its fingerprint

```
ssh-keygen -l -f ~/.ssh/lab_server.pub
```

Recreate the public key from the private key

```
ssh-keygen -y -f ~/.ssh/lab_server
```

Change the key passphrase

```
ssh-keygen -p -f ~/.ssh/lab_server
```

Remove an old server from `known_hosts`

```
ssh-keygen -R 192.168.1.50
```

Generate missing SSH server host keys

```
sudo ssh-keygen -A
```

Important Security Practices

1. **Never share your private key.**
2. Use **Ed25519** for new SSH keys unless compatibility requires another type.
3. Protect private keys with a strong passphrase when practical.
4. Use appropriate permissions such as `700` for `~/.ssh` and `600` for private keys.
5. Use separate keys for important environments when appropriate.
6. Remove compromised or retired keys from `authorized_keys`.
7. Verify changed host-key warnings instead of blindly deleting entries.
8. Keep backups of important private keys in a secure location.
9. For large organizations, consider SSH certificates and a dedicated SSH CA.
10. Avoid obsolete **DSA (`ssh-dss`)** keys.

Quick Reference

```
# Generate Ed25519 key
ssh-keygen -t ed25519

# Generate RSA key
ssh-keygen -t rsa -b 4096

# Specify filename
ssh-keygen -t ed25519 -f ~/.ssh/mykey

# Add comment
ssh-keygen -t ed25519 -C "admin@server"

# Display fingerprint
ssh-keygen -l -f ~/.ssh/id_ed25519.pub

# Generate public key from private key
ssh-keygen -y -f ~/.ssh/id_ed25519
```

```
# Change passphrase
ssh-keygen -p -f ~/.ssh/id_ed25519

# Change comment
ssh-keygen -c -C "new-comment" -f ~/.ssh/id_ed25519

# Find host in known_hosts
ssh-keygen -F server.example.com

# Remove host from known_hosts
ssh-keygen -R server.example.com

# Hash known_hosts
ssh-keygen -H

# Generate missing SSH host keys
sudo ssh-keygen -A

# Display certificate information
ssh-keygen -L -f user-cert.pub
```

In short: `ssh-keygen` is the principal OpenSSH utility for **creating and managing SSH keys, fingerprints, host keys, and SSH certificates**. For modern Linux systems, `ssh-keygen -t ed25519` is the most useful starting point for learning SSH public-key authentication.

5. ssh-copy-id

The **`ssh-copy-id`** command is a Linux utility used to **install an SSH public key on a remote server** so that you can log in using SSH keys instead of entering the remote user's password every time. It is commonly used to configure **passwordless SSH authentication**.

`ssh-copy-id` works together with the SSH client and the `~/.ssh/authorized_keys` file on the remote system.

Syntax

```
ssh-copy-id [OPTIONS] [USER@]HOST
```

- **USER** – The username on the remote system.
- **HOST** – The hostname or IP address of the remote system.
- **OPTIONS** – Options that control how the public key is selected or transferred.

Basic Example

```
ssh-copy-id user@192.168.1.20
```

You will normally be asked for the remote user's password once. The public key is then added to:

`~/.ssh/authorized_keys`

After that, you can normally connect using:

```
ssh user@192.168.1.20
```

without entering the account password.

---

1. Copy the Default SSH Public Key

```
ssh-copy-id user@server
```

By default, `ssh-copy-id` looks for a suitable public key in the local user's SSH directory, commonly:

`~/.ssh/id_ed25519.pub`

or:

```
~/.ssh/id_rsa.pub
```

Example:

```
ssh-copy-id baskar@192.168.1.100
```

---

2. Specify a Particular Public Key

Use `-i` to specify the public-key file.

```
ssh-copy-id -i ~/.ssh/id_ed25519.pub user@server
```

Example:

```
ssh-copy-id -i ~/.ssh/id_rsa.pub baskar@192.168.1.100
```

This is particularly useful when you have several SSH keys.

---

3. Copy an Ed25519 Key

If you have generated an Ed25519 key using:

```
ssh-keygen -t ed25519
```

copy it to the server with:

```
ssh-copy-id -i ~/.ssh/id_ed25519.pub user@server
```

For example:

```
ssh-copy-id -i ~/.ssh/id_ed25519.pub root@192.168.1.50
```

---

4. Specify a Non-Standard SSH Port

If the SSH server is listening on a port other than `22`, use `-p`.

```
ssh-copy-id -p 2222 user@server
```

Example:

```
ssh-copy-id -p 2222 baskar@192.168.1.100
```

The corresponding SSH connection would be:

```
ssh -p 2222 baskar@192.168.1.100
```

---

5. Copy the Key to a Remote Host Using an IPv6 Address

For an IPv6 address, use brackets when appropriate:

```
ssh-copy-id user@[2001:db8::10]
```

Example:

```
ssh-copy-id baskar@[2001:db8::25]
```

---

6. Copy a Key to a Different User

The local username and remote username do not have to be the same.

```
ssh-copy-id student@192.168.1.100
```

For example, if your local account is `baskar` but the remote account is `student`:

```
ssh-copy-id student@192.168.1.100
```

The key is installed in the remote user's:

```
/home/student/.ssh/authorized_keys
```

---

7. Copy Multiple Keys

You can execute `ssh-copy-id` multiple times to install several public keys.

```
ssh-copy-id -i ~/.ssh/id_ed25519.pub user@server
ssh-copy-id -i ~/.ssh/id_rsa.pub user@server
```

The remote user can then authenticate using either corresponding private key.

8. Specify the Remote SSH Port and Key

Both options can be used together:

```
ssh-copy-id -i ~/.ssh/id_ed25519.pub -p 2222 user@server
```

Example:

```
ssh-copy-id -i ~/.ssh/id_ed25519.pub -p 2222 baskar@192.168.1.100
```

---

9. Use SSH Options

`ssh-copy-id` can pass options to the SSH client using `-o`.

For example:

```
ssh-copy-id -o StrictHostKeyChecking=no user@server
```

This disables strict host-key checking for that connection.

Caution: Disabling host-key checking reduces protection against man-in-the-middle attacks and should not normally be used on production systems.

---

10. Copy a Key When the Remote SSH Server Uses a Different Configuration

For example:

```
ssh-copy-id -o ConnectTimeout=10 user@server
```

This limits the connection attempt to approximately 10 seconds.

Multiple SSH options can be supplied:

```
ssh-copy-id -o ConnectTimeout=10 -o ServerAliveInterval=5 user@server
```

---

11. Test Key-Based Authentication

After copying the key:

```
ssh-copy-id user@server
```

test the connection:

```
ssh user@server
```

If the key has been configured correctly, SSH should authenticate using the private key.

To explicitly specify the private key:

```
ssh -i ~/.ssh/id_ed25519 user@server
```

---

12. Display Detailed SSH Authentication Information

Use `ssh -v`:

```
ssh -v user@server
```

For more detailed debugging:

```
ssh -vv user@server
```

or:

```
ssh -vvv user@server
```

This is useful when `ssh-copy-id` succeeds but key-based authentication subsequently fails.

---

13. Specify an SSH Identity File

You can specify the public key to copy:

```
ssh-copy-id -i ~/.ssh/myserver.pub user@server
```

For example:

```
ssh-copy-id -i ~/.ssh/college_server.pub root@192.168.1.50
```

Remember that the corresponding private key might be:

```
~/.ssh/college_server
```

14. Copy the Key to a Server for the First Time

A typical workflow is:

Step 1 – Generate an SSH key

```
ssh-keygen -t ed25519
```

Step 2 – Copy the public key

```
ssh-copy-id user@server
```

Step 3 – Test SSH login

```
ssh user@server
```

Step 4 – Verify the authorized key

On the remote server:

```
cat ~/.ssh/authorized_keys
```

You should see the public key that was copied from the client.

Important SSH Files

Local system

Private key:

```
~/.ssh/id_ed25519
```

Public key:

```
~/.ssh/id_ed25519.pub
```

SSH configuration:

```
~/.ssh/config
```

Remote system

Authorized public keys:

```
~/.ssh/authorized_keys
```

SSH server configuration:

```
/etc/ssh/sshd_config
```

Never copy or expose your private key.

For example, this file:

```
~/.ssh/id_ed25519
```

must remain private.

Only this file should normally be copied to the server:

```
~/.ssh/id_ed25519.pub
```

---

15. Check the Permissions

SSH is sensitive to incorrect permissions.

On the remote server, typical permissions are:

```
chmod 700 ~/.ssh
chmod 600 ~/.ssh/authorized_keys
```

The user's home directory should also not be writable by unauthorized users.

You can check:

```
ls -ld ~/.ssh
ls -l ~/.ssh/authorized_keys
```

Expected permissions are commonly:

```
drwx------ ~/.ssh
-rw------- ~/.ssh/authorized_keys
```

16. Copy the Key to Multiple Servers

If you manage several Linux servers:

```
ssh-copy-id user@server1
ssh-copy-id user@server2
ssh-copy-id user@server3
```

For example:

```
ssh-copy-id baskar@192.168.1.101
ssh-copy-id baskar@192.168.1.102
ssh-copy-id baskar@192.168.1.103
```

Afterward:

```
ssh baskar@192.168.1.101
ssh baskar@192.168.1.102
ssh baskar@192.168.1.103
```

can use key-based authentication.

---

17. Using ssh-copy-id with a Hostname

Instead of an IP address:

```
ssh-copy-id user@linuxserver
```

If the hostname is defined in `/etc/hosts`:

```
192.168.1.100 linuxserver
```

you can use:

```
ssh-copy-id user@linuxserver
```

---

18. Using SSH Configuration

If you have configured an entry in:

```
~/.ssh/config
```

such as:

```
Host myserver
    HostName 192.168.1.100
    User baskar
    Port 2222
```

you can simply use:

```
ssh-copy-id myserver
```

and later:

```
ssh myserver
```

This is convenient when administering many servers.

---

19. What ssh-copy-id Actually Does

Conceptually, the command:

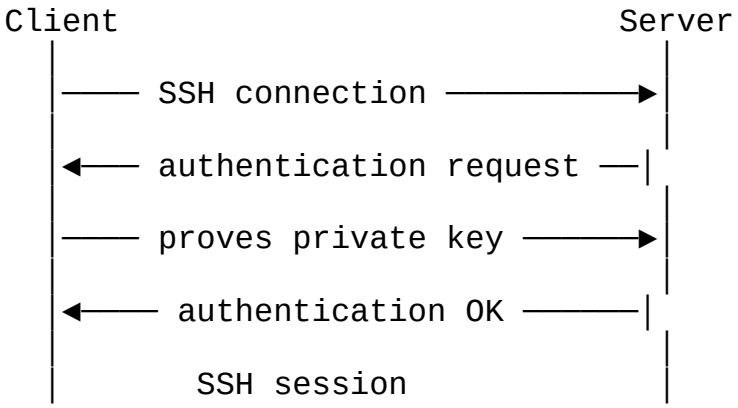
```
ssh-copy-id user@server
```

performs these operations:

```
Local Computer
      │
      │  Public key
      ▼
~/.ssh/id_ed25519.pub
      │
      │  SSH connection
      ▼
Remote Server
      │
      ▼
~/.ssh/authorized_keys
```

The remote SSH server subsequently checks whether the public key corresponding to the client's private key is present in `authorized_keys`.

The authentication process is therefore approximately:



The **private key never needs to be sent to the server**.

---

20. ssh-copy-id vs ssh

These commands have different purposes:

Command	Purpose
`ssh`	Connect to a remote system
`ssh-keygen`	Generate SSH keys
`ssh-copy-id`	Install a public key on a remote account
`scp`	Securely copy files
`sftp`	Secure interactive file transfer

A typical SSH administration sequence is:

```
ssh-keygen

↓

ssh-copy-id user@server

↓

ssh user@server
```

21. ssh-copy-id vs Manual Key Installation

Without `ssh-copy-id`, you could manually install a public key:

```
cat ~/.ssh/id_ed25519.pub
```

Copy the output and append it to the remote server's:

```
~/.ssh/authorized_keys
```

`ssh-copy-id` automates this process:

```
ssh-copy-id user@server
```

It is therefore much less error-prone for normal SSH key deployment.

---

22. Common Errors

Permission denied

```
Permission denied (publickey,password)
```

Check:

```
ls -ld ~/.ssh
ls -l ~/.ssh/authorized_keys
```

and verify the SSH server configuration.

No public key found

You may see an error indicating that no public key was found.

Check:

```
ls -l ~/.ssh/*.pub
```

If necessary, generate one:

```
ssh-keygen -t ed25519
```

Then:

```
ssh-copy-id -i ~/.ssh/id_ed25519.pub user@server
```

Wrong SSH port

If the server uses port 2222:

```
ssh-copy-id -p 2222 user@server
```

Using:

```
ssh-copy-id user@server
```

would attempt to connect to the default SSH port, normally 22.

Key authentication still does not work

Use:

```
ssh -vvv user@server
```

and examine which authentication methods SSH attempts.

Also check the server's SSH log. On Ubuntu/Debian systems, for example:

```
sudo journalctl -u ssh
```

or:

```
sudo tail -f /var/log/auth.log
```

---

23. Useful Options

Option	Purpose
`-i FILE`	Use the specified public-key file
`-p PORT`	Specify the SSH port
`-f`	Force copying the key without checking whether it is already installed
`-n`	Do not actually modify the remote authorized_keys file
`-o OPTION`	Pass an SSH client option
`-h`	Display help

For example:

```
ssh-copy-id -i ~/.ssh/id_ed25519.pub -p 2222 user@server
```

24. Practical Examples

Example 1 – Basic

```
ssh-copy-id user@192.168.1.10
```

Example 2 – Ed25519 key

```
ssh-copy-id -i ~/.ssh/id_ed25519.pub user@192.168.1.10
```

Example 3 – RSA key

```
ssh-copy-id -i ~/.ssh/id_rsa.pub user@192.168.1.10
```

Example 4 – Custom SSH port

```
ssh-copy-id -p 2222 user@192.168.1.10
```

Example 5 – Custom key and port

```
ssh-copy-id -i ~/.ssh/server.pub -p 2222 user@192.168.1.10
```

Example 6 – Multiple servers

```
ssh-copy-id user@server1
ssh-copy-id user@server2
ssh-copy-id user@server3
```

Example 7 – Test the connection

```
ssh user@server
```

Summary

`ssh-copy-id` is primarily used to **install an SSH public key into a remote user's `~/.ssh/authorized_keys` file**. It is one of the most useful commands for setting up secure key-based authentication between Linux systems.

The most important commands to remember are:

```
ssh-keygen -t ed25519

ssh-copy-id user@server

ssh-copy-id -i ~/.ssh/id_ed25519.pub user@server

ssh-copy-id -p 2222 user@server

ssh user@server
```

A particularly important security principle is:

Copy the public key, never the private key.

This makes `ssh-copy-id` an essential command to learn alongside **`ssh`**, **`ssh-keygen`**, **`scp`**, and **`sftp`** for Linux system and network administration.

6. ssh-agent

The **`ssh-agent`** command is an OpenSSH utility used to manage and cache private SSH keys in memory. It allows users to authenticate to remote SSH servers without repeatedly entering the passphrase of their private key.

It is especially useful when connecting to multiple servers, using `ssh`, `scp`, `sftp`, `git`, or other applications that use SSH authentication.

Syntax

```
ssh-agent [OPTIONS] [COMMAND [ARG...]]
```

- **OPTIONS** – Options that control how the SSH authentication agent operates.
- **COMMAND** – Optional command to execute with the agent's environment.
- **ARG** – Arguments passed to the command.

What is an SSH Agent?

Normally, if an SSH private key is protected with a passphrase, you may need to enter that passphrase when the key is used.

`ssh-agent` solves this by:

1. Starting an authentication agent process.
2. Loading the private key into the agent using `ssh-add`.
3. Keeping the decrypted key available in memory.
4. Allowing SSH clients to use the key without repeatedly asking for its passphrase.

The private key itself is **not copied to the remote server**.

Check whether ssh-agent is running

```
echo $SSH_AUTH_SOCK
```

If the agent is running, this normally displays a Unix socket path such as:

```
/tmp/ssh-XXXXXX/agent.1234
```

You can also check:

```
ps aux | grep ssh-agent
```

Start ssh-agent

```
eval "$(ssh-agent)"
```

Example output:

```
Agent pid 12345
```

The `eval` command sets environment variables such as `SSH_AUTH_SOCK` and `SSH_AGENT_PID` in the current shell.

Start ssh-agent with a specific shell

```
ssh-agent bash
```

This starts a new Bash shell with the SSH-agent environment configured.

Add a private key to the agent

`ssh-agent` itself does not normally load keys. Use **`ssh-add`**:

```
ssh-add ~/.ssh/id_ed25519
```

You will be prompted for the key's passphrase:

```
Enter passphrase for /home/user/.ssh/id_ed25519:
Identity added: /home/user/.ssh/id_ed25519
```

For an RSA key:

```
ssh-add ~/.ssh/id_rsa
```

List keys currently loaded

```
ssh-add -l
```

Example:

```
256 SHA256:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx user@server
(ED25519)
```

To display the public keys:

```
ssh-add -L
```

Remove a specific key

```
ssh-add -d ~/.ssh/id_ed25519
```

Remove all keys

```
ssh-add -D
```

This removes the keys from the running agent, but **does not delete the private-key files**.

Connect to a server using the agent

Once the key has been added:

```
ssh-add ~/.ssh/id_ed25519
```

Then connect normally:

```
ssh user@192.168.1.10
```

SSH automatically communicates with the agent.

Use ssh-agent with scp

After loading the key:

```
ssh-add ~/.ssh/id_ed25519
```

You can use:

```
scp file.txt user@192.168.1.10:/home/user/
```

You normally do not need to specify the private key again.

Use ssh-agent with sftp

```
sftp user@192.168.1.10
```

The SFTP client can use the key held by the agent.

Add multiple keys

```
ssh-add ~/.ssh/id_ed25519
ssh-add ~/.ssh/id_rsa
```

Check them:

```
ssh-add -l
```

You might see:

```
256 SHA256:AAAA... user@desktop (ED25519)
2048 SHA256:BBBB... user@server (RSA)
```

Remove the agent

If you started an agent manually, you can terminate it with:

```
ssh-agent -k
```

Example:

```
unset SSH_AUTH_SOCK;
unset SSH_AGENT_PID;
echo Agent pid 12345 killed;
```

Start agent and immediately add a key

A common sequence is:

```
eval "$(ssh-agent -s)"
ssh-add ~/.ssh/id_ed25519
```

Then:

```
ssh user@remote-server
```

Use a custom socket

You can specify the agent socket:

```
ssh-agent -a /tmp/my-ssh-agent.sock
```

This is useful in specialized scripts or environments where the default socket location is unsuitable.

Specify an environment variable

The SSH client uses the `SSH_AUTH_SOCK` environment variable to locate the agent:

```
echo $SSH_AUTH_SOCK
```

You can explicitly set it:

```
export SSH_AUTH_SOCK=/tmp/my-ssh-agent.sock
```

Common options

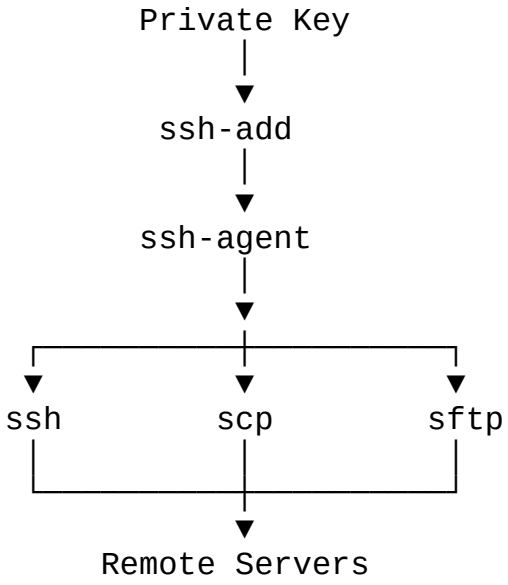
Option	Purpose
`-a path`	Bind the agent to the specified Unix-domain socket
`-c`	Generate C-shell commands
`-s`	Generate Bourne-shell commands; commonly used with eval
`-k`	Kill the currently running agent
`-D`	Run in foreground rather than as a daemon
`-d`	Debug mode

`ssh-agent` vs `ssh-add`

These commands have different responsibilities:

Command	Purpose
`ssh-agent`	Starts and manages the SSH authentication agent
`ssh-add`	Adds or removes private keys from the agent
`ssh`	Connects to the remote SSH server and uses the agent for authentication

A typical workflow is:



Practical Example

Suppose you have an encrypted private key:

```
~/.ssh/id_ed25519
```

Start the agent:

```
eval "$(ssh-agent -s)"
```

Add the key:

```
ssh-add ~/.ssh/id_ed25519
```

Verify:

```
ssh-add -l
```

Connect to the server:

```
ssh student@192.168.1.20
```

Copy a file:

```
scp notes.txt student@192.168.1.20:/home/student/
```

Transfer files using SFTP:

```
sftp student@192.168.1.20
```

All three operations can use the private key already held by `ssh-agent`.

Important Security Considerations

`ssh-agent` keeps private-key material available in memory. Therefore:

- Do not add private keys to an agent on an untrusted machine.
- Use passphrase-protected private keys.
- Remove keys when they are no longer needed:

  ```
  ssh-add -D
  ```

- Avoid forwarding your agent to untrusted servers unless necessary.
- On shared systems, be careful about who can access the agent's Unix socket.

Useful Troubleshooting Commands

Check the agent socket:

```
echo $SSH_AUTH_SOCK
```

List loaded keys:

```
ssh-add -l
```

If no identities are loaded:

```
The agent has no identities.
```

Add a key:

```
ssh-add ~/.ssh/id_ed25519
```

If you get:

```
Could not open a connection to your authentication agent.
```

start an agent in the current shell:

```
eval "$(ssh-agent -s)"
```

Then add the key:

```
ssh-add ~/.ssh/id_ed25519
```

In Short

`ssh-agent` provides a temporary in-memory key-management service for SSH. Instead of repeatedly entering the passphrase for an encrypted private key, you enter it once when adding the key with `ssh-add`, after which `ssh`, `scp`, `sftp`, Git, and other SSH-enabled applications can use the agent for authentication.

7. ssh-add

The **`ssh-add`** command is a Linux utility used to **add private SSH keys to the SSH authentication agent (`ssh-agent`)**. Once a key is loaded into the agent, SSH can use it for authentication without repeatedly asking for the key's passphrase.

It is commonly used together with **`ssh-agent`** and **`ssh-keygen`**.

Syntax

```
ssh-add [OPTIONS] [IDENTITY_FILE...]
```

- **OPTIONS** – Control how keys are added, listed, removed, or managed.
- **IDENTITY_FILE** – The private SSH key file to add to the agent.

Common SSH Key Files

Key file	Description
`~/.ssh/id_ed25519`	Ed25519 private key
`~/.ssh/id_rsa`	RSA private key
`~/.ssh/id_ecdsa`	ECDSA private key
`~/.ssh/id_dsa`	DSA private key (obsolete)

Important: Never add or copy the private key (`id_ed25519`, `id_rsa`, etc.) to another system. Only the public key (`.pub`) is normally copied to the remote server.

---

1. Start the SSH Agent

Before using `ssh-add`, an SSH authentication agent must normally be running.

```
eval "$(ssh-agent -s)"
```

Example:

```
Agent pid 2451
```

2. Add the Default SSH Key

To add the default Ed25519 private key:

```
ssh-add ~/.ssh/id_ed25519
```

If the key is protected with a passphrase:

```
Enter passphrase for /home/user/.ssh/id_ed25519:
```

After successful authentication:

```
Identity added: /home/user/.ssh/id_ed25519
```

---

3. Add an RSA Key

```
ssh-add ~/.ssh/id_rsa
```

Example:

```
Identity added: /home/user/.ssh/id_rsa
```

---

4. Add a Specific Private Key

You can add any private key by specifying its path:

```
ssh-add ~/.ssh/myserver_key
```

For example:

```
ssh-add ~/.ssh/production_key
```

This is particularly useful when you maintain separate keys for different servers or organizations.

---

5. List Keys Currently Loaded in the Agent

Use:

```
ssh-add -l
```

Example:

```
256 SHA256:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
user@desktop (ED25519)
```

The `-l` option displays the fingerprints of the identities currently loaded into the agent.

6. Display Public Keys Loaded in the Agent

Use:

```
ssh-add -L
```

Example:

```
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAI... user@desktop
```

Difference:

```
ssh-add -l
```

Displays **key fingerprints**.

```
ssh-add -L
```

Displays the **complete public keys**.

---

7. Remove a Key from the Agent

To remove a particular key:

```
ssh-add -d ~/.ssh/id_ed25519
```

Example:

```
Identity removed: /home/user/.ssh/id_ed25519
```

The private key itself is **not deleted**. It is only removed from the running SSH agent.

---

8. Remove All Keys

To remove all identities currently loaded into the agent:

```
ssh-add -D
```

Example:

```
All identities removed.
```

This does **not** delete the private key files from `~/.ssh/`.

9. Add All Default Keys

Use:

```
ssh-add
```

With no identity file specified, `ssh-add` attempts to add the default identity files.

Depending on the OpenSSH version and configuration, these may include keys such as:

```
~/.ssh/id_rsa
~/.ssh/id_ecdsa
~/.ssh/id_ecdsa_sk
~/.ssh/id_ed25519
~/.ssh/id_ed25519_sk
```

---

10. Add a Key with a Lifetime

You can tell the SSH agent to automatically remove a key after a specified period.

For example, remove the key after **1 hour**:

```
ssh-add -t 1h ~/.ssh/id_ed25519
```

For 30 minutes:

```
ssh-add -t 30m ~/.ssh/id_ed25519
```

For one day:

```
ssh-add -t 1d ~/.ssh/id_ed25519
```

This is useful on shared or temporary systems.

---

11. Add a Key with a 10-Minute Lifetime

```
ssh-add -t 10m ~/.ssh/id_ed25519
```

After 10 minutes, the agent automatically removes the key.

12. Add a Key Without Changing Its Lifetime

If the agent has a default key lifetime configured, you can use:

```
ssh-add -T
```

However, for normal key loading, simply use:

```
ssh-add ~/.ssh/id_ed25519
```

---

13. Use a Different SSH Agent

The SSH agent is identified through the `SSH_AUTH_SOCK` environment variable.

Check it with:

```
echo $SSH_AUTH_SOCK
```

Example:

```
/tmp/ssh-XXXXXX/agent.2451
```

You can inspect the running agent with:

```
ssh-add -l
```

If no agent is available, you may see:

```
Could not open a connection to your authentication agent.
```

Start one with:

```
eval "$(ssh-agent -s)"
```

Then add your key:

```
ssh-add ~/.ssh/id_ed25519
```

---

14. Use ssh-add with Git

Suppose you use an SSH key to access Git repositories:

```
ssh-add ~/.ssh/github_ed25519
```

After the key is loaded, Git can use the agent for SSH authentication.

For example:

```
git clone git@github.com:user/project.git
```

The private-key passphrase does not need to be entered for every connection while the key remains loaded in the agent.

---

15. Use Multiple SSH Keys

You can load multiple identities:

```
ssh-add ~/.ssh/id_ed25519
ssh-add ~/.ssh/work_key
ssh-add ~/.ssh/github_key
```

Check them:

```
ssh-add -l
```

Example:

```
256 SHA256:AAAA... user@desktop (ED25519)
4096 SHA256:BBBB... work@desktop (RSA)
256 SHA256:CCCC... github@desktop (ED25519)
```

SSH can then use the identities available through the agent.

---

16. Add a FIDO/U2F Security-Key Identity

Modern OpenSSH supports security-key-backed identities such as:

```
ssh-add ~/.ssh/id_ed25519_sk
```

or:

```
ssh-add ~/.ssh/id_ecdsa_sk
```

These keys may require interaction with a hardware security key.

17. Confirm that the Agent Contains a Key

Run:

```
ssh-add -l
```

If a key is loaded:

```
256 SHA256:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx user@host
(ED25519)
```

If no keys are loaded:

```
The agent has no identities.
```

---

18. Test SSH Authentication

After adding a key:

```
ssh-add ~/.ssh/id_ed25519
```

You can test authentication with:

```
ssh user@192.168.1.100
```

SSH will use the key from the authentication agent.

---

19. Remove All Keys Before Leaving a System

On a temporary or shared workstation, it can be useful to remove all identities:

```
ssh-add -D
```

Then verify:

```
ssh-add -l
```

Output:

```
The agent has no identities.
```

20. Common ssh-add Options

Option	Purpose
`-l`	List fingerprints of keys loaded in the agent
`-L`	List public keys loaded in the agent
`-d`	Delete a specified identity
`-D`	Delete all identities
`-t life`	Set maximum lifetime of an added key
`-c`	Require confirmation before using the key
`-C`	When adding keys, use smart-card/PKCS#11-related behavior depending on OpenSSH version
`-K`	Load resident keys from a FIDO authenticator, where supported
`-k`	Remove keys from a smartcard, where supported
`-q`	Quiet mode

---

21. Require Confirmation Before Using a Key

The `-c` option causes the agent to ask for confirmation whenever the identity is used.

```
ssh-add -c ~/.ssh/id_ed25519
```

This can provide an additional layer of protection for sensitive keys.

When SSH attempts to use the key, the agent asks the user for confirmation through the available confirmation mechanism.

---

22. Combine Key Lifetime and Confirmation

For example:

```
ssh-add -c -t 1h ~/.ssh/production_key
```

This means:

- the key requires confirmation when used
- the key is automatically removed after one hour

This can be useful for administrative or production-server access.

23. Troubleshooting: Agent Not Running

Command:

```
ssh-add ~/.ssh/id_ed25519
```

Error:

```
Could not open a connection to your authentication agent.
```

Solution:

```
eval "$(ssh-agent -s)"
```

Then:

```
ssh-add ~/.ssh/id_ed25519
```

---

24. Troubleshooting: Wrong Key Permissions

Check:

```
ls -l ~/.ssh/id_ed25519
```

A private key should normally be accessible only by its owner.

Correct the permissions with:

```
chmod 600 ~/.ssh/id_ed25519
```

Then:

```
ssh-add ~/.ssh/id_ed25519
```

---

25. Troubleshooting: Key Does Not Exist

If you run:

```
ssh-add ~/.ssh/id_ed25519
```

and the file does not exist, check:

```
ls -la ~/.ssh/
```

You may find:

```
id_rsa
id_rsa.pub
```

In that case:

```
ssh-add ~/.ssh/id_rsa
```

---

26. Complete Example

A typical workflow is:

```
ssh-keygen -t ed25519
```

Start the agent:

```
eval "$(ssh-agent -s)"
```

Add the private key:

```
ssh-add ~/.ssh/id_ed25519
```

Verify:

```
ssh-add -l
```

Then connect:

```
ssh user@server
```

The sequence is:

```
ssh-keygen
     ↓
Creates private/public key pair
     ↓
ssh-agent
     ↓
Keeps private key available in memory
     ↓
ssh-add
     ↓
Loads private key into agent
     ↓
ssh
     ↓
Uses agent for authentication
```

`ssh-add` vs `ssh-agent` vs `ssh-keygen`

Command	Main purpose
`ssh-keygen`	Creates and manages SSH keys
`ssh-agent`	Runs an authentication agent
`ssh-add`	Loads/removes keys from the agent
`ssh-copy-id`	Installs a public key on a remote account
`ssh`	Connects to a remote system

In simple terms: `ssh-keygen` **creates** the key, `ssh-agent` **holds** the key, and `ssh-add` **puts the key into the agent**.

8. ssh-keyscan

The **`ssh-keyscan`** command is a Linux utility used to collect the **public SSH host keys** of remote servers. It is commonly used to obtain host keys before establishing an SSH connection, populate the `~/.ssh/known_hosts` file, and automate SSH host-key management in scripts and deployment environments.

Unlike **`ssh`**, **`ssh-keyscan` does not authenticate or log in to the remote server**. It simply connects to the SSH service and retrieves the server's host key.

Syntax

```
ssh-keyscan [OPTIONS] [HOST | ADDRESS | CIDR_RANGE]
```

- **HOST** – Hostname of the remote SSH server.
- **ADDRESS** – IPv4 or IPv6 address of the SSH server.
- **CIDR_RANGE** – A network range from which SSH host keys can be scanned.
- **OPTIONS** – Control the key types, port, timeout, hashing, and output format.

---

1. Scan a Single Host

```
ssh-keyscan server1.example.com
```

Example output:

```
server1.example.com ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAI...
```

This retrieves the SSH host key without logging into the server.

---

2. Scan an IP Address

```
ssh-keyscan 192.168.1.10
```

Example:

```
192.168.1.10 ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAI...
```

3. Scan Multiple Hosts

```
ssh-keyscan server1.example.com server2.example.com
server3.example.com
```

The host keys of all specified servers are displayed.

---

4. Specify the SSH Port

If the SSH server is running on a non-standard port:

```
ssh-keyscan -p 2222 server1.example.com
```

Here, `2222` is the TCP port used by the SSH service.

---

5. Specify the Key Type

Use `-t` to request particular host-key types.

```
ssh-keyscan -t ed25519 server1.example.com
```

Other commonly encountered types include:

```
ssh-keyscan -t rsa server1.example.com
```

```
ssh-keyscan -t ecdsa server1.example.com
```

Multiple types can be specified:

```
ssh-keyscan -t ed25519,rsa,ecdsa server1.example.com
```

---

6. Save the Host Key to a File

```
ssh-keyscan server1.example.com > hostkeys.txt
```

The retrieved key is saved in `hostkeys.txt`.

You can examine it with:

```
cat hostkeys.txt
```

7. Add a Host Key to `known_hosts`

A common use of `ssh-keyscan` is adding a server's host key to the SSH `known_hosts` file.

```
ssh-keyscan server1.example.com >> ~/.ssh/known_hosts
```

After this, SSH can recognize the server's host key when connecting.

Important: `ssh-keyscan` retrieves whatever key the server presents. For security-sensitive use, you should **verify the fingerprint through a trusted channel before adding the key**. Blindly accepting a scanned key can undermine SSH's protection against man-in-the-middle attacks.

---

8. Scan a Host and Remove Duplicate Entries

```
ssh-keyscan server1.example.com >> ~/.ssh/known_hosts
```

To avoid repeatedly adding the same entry, you can use:

```
ssh-keygen -F server1.example.com
```

to check whether the host already exists.

---

9. Scan an Entire Network

`ssh-keyscan` can scan a CIDR network range.

```
ssh-keyscan 192.168.1.0/24
```

This attempts to retrieve SSH host keys from hosts in the specified network range.

This can be useful for discovering SSH-enabled systems, but scanning networks that you do not own or have permission to test may be inappropriate.

---

10. Scan Only IPv4 Hosts

Use `-4`:

```
ssh-keyscan -4 server1.example.com
```

---

11. Scan Only IPv6 Hosts

Use `-6`:

```
ssh-keyscan -6 server1.example.com
```

12. Set a Connection Timeout

Use `-T` to specify the timeout in seconds:

```
ssh-keyscan -T 5 server1.example.com
```

The command waits up to 5 seconds when attempting to obtain the host key.

This is particularly useful in scripts where you do not want an unreachable host to cause a long delay.

---

13. Hash Hostnames in the Output

Use `-H`:

```
ssh-keyscan -H server1.example.com
```

Instead of displaying:

```
server1.example.com ssh-ed25519 AAAA...
```

the hostname is hashed.

This provides some privacy if the resulting `known_hosts` file is exposed.

---

14. Scan Several Servers and Store Their Keys

```
ssh-keyscan server1.example.com server2.example.com
server3.example.com > known_hosts
```

The resulting file can then be used as an SSH known-hosts file.

For example:

```
ssh -o UserKnownHostsFile=known_hosts user@server1.example.com
```

---

15. Scan a Server Running on a Custom Port and Save the Result

```
ssh-keyscan -p 2222 server1.example.com > server1_known_hosts
```

This is useful when servers use custom SSH ports.

16. Display Only the Host Key

You can request a specific key type:

```
ssh-keyscan -t ed25519 server1.example.com
```

This is useful when you only want the Ed25519 host key instead of all supported host-key types.

---

17. Check a Server's SSH Host Key Fingerprint

First obtain the key:

```
ssh-keyscan -t ed25519 server1.example.com > /tmp/server_key
```

Then calculate its fingerprint:

```
ssh-keygen -lf /tmp/server_key
```

Example:

```
256 SHA256:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
server1.example.com (ED25519)
```

You should compare this fingerprint with a **trusted fingerprint obtained from the server administrator or another trusted source**.

---

18. Use `ssh-keyscan` in a Script

For example:

```
#!/bin/bash

HOST="server1.example.com"

if ssh-keyscan -T 5 "$HOST" > /tmp/hostkey 2>/dev/null; then
    echo "SSH host key retrieved successfully"
else
    echo "Unable to retrieve SSH host key"
fi
```

This can be useful for automated deployment or provisioning.

19. Use with Configuration Management

For example:

```
ssh-keyscan server1.example.com >> ~/.ssh/known_hosts
```

This can be used before an automated SSH operation:

```
ssh user@server1.example.com
```

It is commonly encountered in automated environments such as CI/CD pipelines, provisioning scripts, and configuration-management systems.

However, automated workflows should preferably use **pre-verified host keys** rather than blindly trusting the output of `ssh-keyscan`.

Important Options

Option	Purpose
`-4`	Use IPv4 only
`-6`	Use IPv6 only
`-p PORT`	Specify SSH port
`-T TIMEOUT`	Set connection timeout
`-t TYPE`	Specify host-key types
`-H`	Hash all hostnames in output
`-f FILE`	Read hosts from a file
`-v`	Verbose output
`-c`	Request certificates rather than normal host keys
`-D`	Search for SSHFP DNS records
`-q`	Quiet mode

Reading Hosts from a File

Suppose `servers.txt` contains:

```
server1.example.com
server2.example.com
server3.example.com
```

You can scan all of them with:

```
ssh-keyscan -f servers.txt
```

Save the results:

```
ssh-keyscan -f servers.txt > known_hosts
```

Using `ssh-keyscan` with `ssh-keygen`

A useful combination is:

```
ssh-keyscan server1.example.com > /tmp/key
ssh-keygen -lf /tmp/key
```

The first command obtains the host key, while the second displays its fingerprint.

This illustrates an important distinction:

```
ssh-keyscan
     │
     └── Retrieves the server's public host key

ssh-keygen
     │
     └── Examines/converts/manages SSH keys and fingerprints

ssh
     │
     └── Establishes an SSH connection
```

`ssh-keyscan` vs `ssh`

Feature	ssh-keyscan	ssh
Connects to SSH service	Yes	Yes
Logs into server	No	Yes
Retrieves host key	Yes	Yes, as part of connection
Executes commands	No	Yes
Authenticates user	No	Yes
Updates known_hosts automatically	No	Yes, depending on configuration
Useful for automation	Yes	Yes

`ssh-keyscan` vs `ssh-keygen`

These commands serve different purposes:

```
ssh-keyscan
     ↓
Retrieve public host keys from SSH servers
```

```
ssh-keygen
     ↓
Generate, inspect, convert and manage SSH keys
```

For example:

```
ssh-keyscan server1.example.com
```

retrieves a server's host key, whereas:

```
ssh-keygen -t ed25519
```

generates a **new SSH key pair** for a user.

Practical Example

Suppose you are configuring a new server:

```
ssh-keyscan -t ed25519 192.168.1.50
```

You receive:

```
192.168.1.50 ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAI...
```

Verify the fingerprint through a trusted source, then add the key:

```
ssh-keyscan -t ed25519 192.168.1.50 >> ~/.ssh/known_hosts
```

Now you can connect:

```
ssh user@192.168.1.50
```

Key Points to Remember

- **`ssh-keyscan` retrieves SSH server host keys.**
- It **does not log in** to the remote server.
- It can retrieve keys from **one or many hosts**.
- `-p` specifies a non-standard SSH port.
- `-t` selects the host-key type.
- `-4` and `-6` select IPv4 or IPv6.
- `-H` hashes hostnames in the output.
- `-T` controls the connection timeout.
- `-f` reads a list of hosts from a file.
- It is particularly useful for **automation and `known_hosts` management**.
- **Do not blindly trust `ssh-keyscan` output** for security-sensitive connections; verify the host-key fingerprint through a trusted source first.