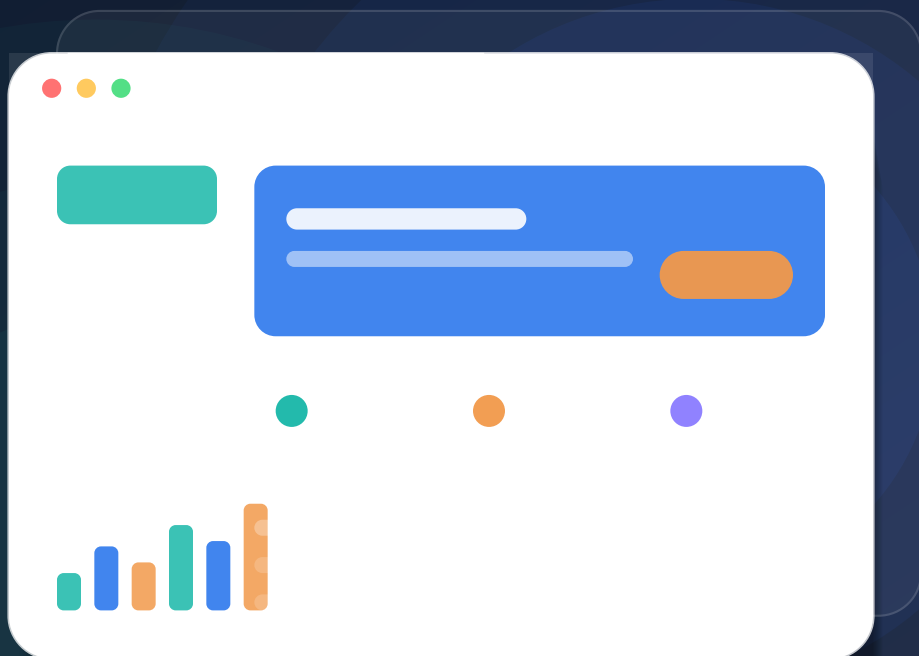


# SSH CHEAT SHEET

A QUICK REFERENCE GUIDE



The Quick Reference for  
Secure Shell (SSH)

CONNECT • AUTHENTICATE • TUNNEL • SECURE

# SSH Cheat Sheet

A Quick Reference for Secure Shell

This guide is provided for general information and describes OpenSSH. Options and defaults vary between versions and systems, so check the manual pages (`man ssh`, `man sshd_config`) before relying on any detail. Only connect to systems you are authorised to use, and test server changes carefully.

# Contents

---

- Introduction . . . . . 4
- 1. SSH Basics . . . . . 5
- 2. Connecting: Syntax and Common Options . . . . . 8
- 3. Keys and Authentication . . . . . 11
- 4. The Client Config File . . . . . 14
- 5. Copying Files . . . . . 17
- 6. Port Forwarding and Tunnels . . . . . 20
- 7. Securing the SSH Server . . . . . 23
- 8. Working in Long Sessions . . . . . 26
- 9. Troubleshooting Errors . . . . . 29
- 10. Security Best Practices . . . . . 32
- Appendix A: Quick Reference . . . . . 34
- Appendix B: Glossary . . . . . 36

# Introduction

SSH (Secure Shell) is the standard way to log in to, manage and transfer files to remote computers safely. System administrators, developers and hobbyists use it every day, and a handful of commands cover most of what they need.

This cheat sheet collects those commands in one place, with short explanations of what each one does and when to use it. It is written for Linux, macOS and Windows users, and it favours safe, modern defaults such as key-based login and Ed25519 keys.

## How to use this book

- **Need a command now?** Go to Appendix A, the one-page quick reference.
- **New to SSH?** Read Chapters 1 to 3 in order.
- **Something is failing?** Chapter 9 decodes the most common error messages.
- **Setting up a server?** Chapter 7 covers hardening, and Chapter 10 gives a security checklist.

In the examples, `user` means your username on the remote machine, `host` means its hostname or IP address, and lines beginning with `$` are typed in your terminal (do not type the dollar sign itself).

### TIP

Commands here are for OpenSSH, the implementation that ships with Linux, macOS and Windows 10 and 11. Options can vary slightly between versions, so run `man ssh` to confirm details on your own system.

# SSH Basics

SSH creates an encrypted connection between two computers. Everything sent across it, including passwords, commands and file contents, is protected from eavesdroppers and tampering.

## Client and server

- The **SSH client** (ssh) runs on the computer you are sitting at.
- The **SSH server** (sshd) runs on the remote computer and listens, by default, on **TCP port 22**.

## Check that SSH is installed

```
$ ssh -V
OpenSSH_9.6p1, OpenSSL 3.0.13 30 Jan 2024
```

If you see a version number, the client is installed. To install the client or server if needed:

| System          | Command                                                                     |
|-----------------|-----------------------------------------------------------------------------|
| Debian / Ubuntu | sudo apt install openssh-client openssh-server                              |
| Fedora / RHEL   | sudo dnf install openssh-clients openssh-server                             |
| macOS           | Client built in. Server: System Settings > General > Sharing > Remote Login |
| Windows 10/11   | Client built in. Add the server under Settings > Optional features          |

# Your first connection

```
$ ssh user@host
```

The first time you connect to a machine, SSH shows its **host key fingerprint** and asks you to confirm:

```
The authenticity of host 'host (203.0.113.10)' can't be
established. ED25519 key fingerprint is SHA256:abc123...
Are you sure you want to continue connecting (yes/no)?
```

Type **yes** only if the fingerprint matches the one given to you by the server's owner or shown in the provider's console. SSH then saves the key in `~/.ssh/known_hosts`, and warns you loudly if it ever changes.

# Run one command and exit

```
$ ssh user@host 'uptime'
$ ssh user@host 'df -h /'
```

# Log out

Type `exit` or press **Ctrl + D**.

# Where SSH keeps things

| Path                                | Purpose                                       |
|-------------------------------------|-----------------------------------------------|
| <code>~/.ssh/config</code>          | Your personal client settings (Chapter 4)     |
| <code>~/.ssh/known_hosts</code>     | Fingerprints of servers you have trusted      |
| <code>~/.ssh/id_ed25519</code>      | Your <b>private</b> key. Never share it       |
| <code>~/.ssh/id_ed25519.pub</code>  | Your <b>public</b> key. Safe to share         |
| <code>~/.ssh/authorized_keys</code> | (On the server) public keys allowed to log in |

| Path                 | Purpose                               |
|----------------------|---------------------------------------|
| /etc/ssh/sshd_config | (On the server) the server's settings |

## Notes for Windows users

- Open **PowerShell** or **Windows Terminal** and use the same ssh commands as on Linux and macOS.
- Your files live in C:\Users\YourName\.ssh\.
- To keep keys loaded, enable the **OpenSSH Authentication Agent** service in Windows Services.
- **PuTTY** is a popular graphical alternative. It uses its own key format (.ppk), which PuTTYgen can convert.

# Connecting: Syntax and Common Options

Every SSH command follows the same pattern:

```
ssh [options] [user@]host [command]
```

## Everyday examples

```
$ ssh user@host # default port 22
$ ssh -p 2222 user@host # custom port
$ ssh -i ~/.ssh/mykey user@host # choose a key
$ ssh user@192.168.1.20 # use an IP address
$ ssh -l user host # same as user@host
$ ssh host # same username as local
```

## Option reference

| Option        | What it does                                        |
|---------------|-----------------------------------------------------|
| -p PORT       | Connect to a port other than 22                     |
| -i FILE       | Use this private key (identity file)                |
| -l USER       | Log in as this user                                 |
| -v, -vv, -vvv | Verbose output for debugging; more v's, more detail |
| -q            | Quiet mode; suppress most messages                  |
| -t            | Force a terminal; needed for interactive commands   |
| -T            | Disable the terminal; useful in scripts             |
| -N            | Do not run a command (for tunnels only)             |
| -f            | Send SSH to the background after login              |

| Option       | What it does                                         |
|--------------|------------------------------------------------------|
| -A           | Forward your SSH agent (use only with trusted hosts) |
| -X / -Y      | Forward graphical (X11) applications                 |
| -C           | Compress data; helps on slow links                   |
| -J JUMP      | Connect through a jump (bastion) host                |
| -L / -R / -D | Port forwarding (Chapter 6)                          |
| -o KEY=VALUE | Set any config option on the command line            |
| -F FILE      | Use a different config file                          |
| -4 / -6      | Force IPv4 or IPv6                                   |

## Useful -o settings

```
$ ssh -o ServerAliveInterval=30 user@host
$ ssh -o StrictHostKeyChecking=ask user@host
$ ssh -o PreferredAuthentications=password user@host
$ ssh -o IdentitiesOnly=yes -i ~/.ssh/key user@host
```

## Run commands remotely

```
$ ssh user@host 'ls -l /var/www'
$ ssh user@host 'cat /etc/os-release'
$ ssh -t user@host 'sudo systemctl restart nginx'
$ ssh user@host 'bash -s' < local_script.sh
```

The last command runs a script that lives on *your* machine on the remote one, with no need to copy it first. The `-t` flag in the third is required whenever the remote command asks for input, such as a `sudo` password.

**TIP**

Quote remote commands in single quotes. Without quotes, your local shell expands variables and wildcards *before* sending the command, which is rarely what you want.

CHAPTER 3

# Keys and Authentication

Passwords can be guessed, stolen or reused. **SSH keys** are far stronger and more convenient. A key pair has two halves: a **private key** that stays secret on your computer, and a **public key** that you place on servers. Only the holder of the private key can log in.

## Generate a key pair

```
$ ssh-keygen -t ed25519 -C "you@example.com"
```

Accept the default location and **set a passphrase** when prompted. The passphrase encrypts the private key on disk, so a stolen laptop does not mean a stolen key. If a system does not support Ed25519, use RSA with at least 3072 bits:

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

| Key type         | Notes                                         |
|------------------|-----------------------------------------------|
| ed25519          | Modern, fast, short keys. <b>Recommended</b>  |
| rsa (3072+ bits) | Widely compatible. Use 4096 for extra margin  |
| ecdsa            | Supported, but Ed25519 is generally preferred |
| dsa              | Obsolete and insecure. Do not use             |

## Install your public key on a server

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

This appends your public key to `~/.ssh/authorized_keys` on the server. Without `ssh-copy-id`, do it by hand:

```
$ cat ~/.ssh/id_ed25519.pub | ssh user@host \  
'mkdir -p ~/.ssh && chmod 700 ~/.ssh && \  
cat >> ~/.ssh/authorized_keys && \  
chmod 600 ~/.ssh/authorized_keys'
```

## Correct file permissions

SSH refuses to use keys when permissions are too open. These are the right values:

```
$ chmod 700 ~/.ssh  
$ chmod 600 ~/.ssh/id_ed25519  
$ chmod 644 ~/.ssh/id_ed25519.pub  
$ chmod 600 ~/.ssh/authorized_keys ~/.ssh/config
```

## Use the SSH agent

The **agent** remembers your decrypted key for the session, so you type the passphrase once instead of every time.

```
$ eval "$(ssh-agent -s)"      # start the agent  
$ ssh-add ~/.ssh/id_ed25519  # load a key  
$ ssh-add -l                 # list loaded keys  
$ ssh-add -d ~/.ssh/id_ed25519 # remove one key  
$ ssh-add -D                 # remove all keys
```

On macOS, store the passphrase in the Keychain with `ssh-add --apple-use-keychain ~/.ssh/id_ed25519`, and add `AddKeysToAgent yes` and `UseKeychain yes` to your config.

## Key housekeeping

```
$ ssh-keygen -p -f ~/.ssh/id_ed25519      # new passphrase
$ ssh-keygen -lf ~/.ssh/id_ed25519.pub    # fingerprint
$ ssh-keygen -y -f ~/.ssh/id_ed25519     # print public key
$ ssh-keygen -R host                      # forget a host
```

### WARNING

Never share, email or commit your **private** key (the file without .pub). If one is ever exposed, remove its public half from every authorized\_keys file and generate a new pair.

### TIP

Use a separate key for each purpose, such as one for work, one for personal servers and one for GitHub. If one is compromised, the others are unaffected.

## CHAPTER 4

# The Client Config File

Typing long commands gets tedious. The file `~/.ssh/config` lets you give each server a short name and store its settings, so that `ssh user@203.0.113.10 -p 2222 -i ~/.ssh/work_key` becomes simply `ssh work`.

## A basic entry

```
Host work
  HostName 203.0.113.10
  User deploy
  Port 2222
  IdentityFile ~/.ssh/work_key
  IdentitiesOnly yes
```

Now connect with `ssh work`. The same alias works with `scp`, `sftp` and `rsync`.

## Settings for all hosts

A `Host *` block applies to every connection. Put it at the **end** of the file, because SSH uses the first value it finds for each setting.

```
Host *
  ServerAliveInterval 60
  ServerAliveCountMax 3
  AddKeysToAgent yes
  HashKnownHosts yes
```

## Common config options

| Option   | Meaning                         |
|----------|---------------------------------|
| HostName | The real hostname or IP address |

| Option              | Meaning                                         |
|---------------------|-------------------------------------------------|
| User                | The login name                                  |
| Port                | The server's port                               |
| IdentityFile        | Path to the private key to use                  |
| IdentitiesOnly      | Offer only the key named here (avoids lockouts) |
| ProxyJump           | Go through another host first                   |
| ForwardAgent        | Forward your agent (default no)                 |
| LocalForward        | Standing -L port forward                        |
| ServerAliveInterval | Seconds between keep-alive pings                |
| Compression         | Enable compression (yes or no)                  |
| ControlMaster       | Share one connection between sessions           |

## Jump hosts (bastions)

When a server is reachable only through another, use **ProxyJump**:

```
Host bastion
  HostName bastion.example.com
  User admin

Host internal
  HostName 10.0.0.5
  User deploy
  ProxyJump bastion
```

Now `ssh internal` hops through the bastion automatically. On the command line the same idea is `ssh -J admin@bastion.example.com deploy@10.0.0.5`.

## Wildcards and patterns

```
Host web-* db-*  
  User deploy  
  IdentityFile ~/.ssh/prod_key
```

## Faster repeat connections

**Multiplexing** reuses one network connection for many sessions, making the second and later logins instant:

```
Host *  
  ControlMaster auto  
  ControlPath ~/.ssh/cm-%r@%h:%p  
  ControlPersist 10m
```

### TIP

Check what settings SSH will really use for a host, including the effect of every config block, with `ssh -G work`.

## CHAPTER 5

# Copying Files

SSH is not just for shells. The same secure connection can carry files in either direction. Three tools do the job: **scp** for simple copies, **rsync** for efficient synchronisation, and **sftp** for an interactive session.

### scp: secure copy

```
$ scp file.txt user@host:/home/user/      # upload
$ scp user@host:/var/log/app.log .        # download
$ scp -r project/ user@host:~/           # a folder
$ scp -P 2222 file.txt user@host:~/      # custom port
$ scp -i ~/.ssh/key file.txt user@host:~/ # choose a key
$ scp host1:/data/a.txt host2:/backup/    # remote to remote
```

#### WARNING

scp uses a capital **-P** for the port, while ssh uses a lowercase **-p**. Mixing them up is one of the most common mistakes.

### rsync over SSH

rsync copies only what has changed, can resume interrupted transfers and preserves permissions and timestamps. It is the best choice for large folders and backups.

```
$ rsync -avz project/ user@host:/srv/project/
$ rsync -avz user@host:/srv/project/ ./backup/
$ rsync -avz -e "ssh -p 2222" src/ user@host:~/dst/
$ rsync -avz --progress --partial big.iso user@host:~/
$ rsync -avz --delete src/ user@host:/srv/mirror/
```

| Flag | Meaning                                             |
|------|-----------------------------------------------------|
| -a   | Archive: recursive, keeps permissions, times, links |

| Flag          | Meaning                                                       |
|---------------|---------------------------------------------------------------|
| -v            | Verbose                                                       |
| -z            | Compress during transfer                                      |
| -n            | Dry run: show what would happen, change nothing               |
| --progress    | Show per-file progress                                        |
| --partial     | Keep partly transferred files so they can resume              |
| --delete      | Remove files on the target that no longer exist on the source |
| --exclude 'x' | Skip files matching the pattern                               |

#### WARNING

A trailing slash matters. `src/` copies the *contents* of the folder; `src` copies the folder itself. Always test with `-n` first, especially with `--delete`.

## sftp: interactive transfer

```
$ sftp user@host
sftp> ls
sftp> get report.pdf
sftp> put notes.txt
sftp> exit
```

| sftp command | Action                                |
|--------------|---------------------------------------|
| ls / lls     | List remote / local files             |
| cd / lcd     | Change remote / local directory       |
| pwd / lpwd   | Show remote / local directory         |
| get FILE     | Download a file (get -r for a folder) |
| put FILE     | Upload a file (put -r for a folder)   |

| sftp command       | Action                          |
|--------------------|---------------------------------|
| mkdir / rm / rmdir | Manage remote files and folders |
| bye / exit         | Leave sftp                      |

## Mount a remote folder

With **sshfs** installed, a remote folder can appear as if it were a local one:

```
$ mkdir ~/remote
$ sshfs user@host:/srv/data ~/remote
$ fusermount -u ~/remote      # unmount (Linux)
$ umount ~/remote            # unmount (macOS)
```

# Port Forwarding and Tunnels

SSH can carry other network traffic through its encrypted connection. This is called **tunnelling** or **port forwarding**, and it lets you reach services that are not exposed to the internet, or protect traffic that is not encrypted.

## The three types at a glance

| Type    | Flag | Use it to                                                       |
|---------|------|-----------------------------------------------------------------|
| Local   | -L   | Reach a remote service as if it were on your own machine        |
| Remote  | -R   | Expose a service on your machine to the remote machine          |
| Dynamic | -D   | Create a SOCKS proxy that routes all traffic through the server |

## Local forwarding (-L)

```
ssh -L LOCAL_PORT:DEST_HOST:DEST_PORT user@server
```

Traffic sent to your local port travels through server and emerges at the destination. Examples:

```
# Reach a remote web server at localhost:8080
$ ssh -L 8080:localhost:80 user@server

# Reach a database that only the server can see
$ ssh -L 5433:db.internal:5432 user@server

# Tunnel only, no shell, in the background
$ ssh -N -f -L 5433:db.internal:5432 user@server
```

After the second command, point your database tool at localhost:5433.

## Remote forwarding (-R)

```
ssh -R REMOTE_PORT:DEST_HOST:DEST_PORT user@server
```

```
# Let the server reach your local app on port 3000
$ ssh -R 9000:localhost:3000 user@server
```

On the server, localhost:9000 now leads to your machine's port 3000. By default the forwarded port listens only on the server's loopback address. To allow outside access, the server needs GatewayPorts enabled, which should be done with care.

## Dynamic forwarding (-D)

```
$ ssh -D 1080 -N -f user@server
```

This starts a SOCKS5 proxy on localhost:1080. Set your browser or application to use it, and its traffic exits from the server. This is handy on untrusted networks.

## Managing background tunnels

```
$ ps aux | grep 'ssh -N'      # find them
$ pkill -f 'ssh -N -f -L 5433' # stop one
$ ssh -O exit work           # close a multiplexed master
```

## Tunnels that survive drops

Add keep-alive options so a silent connection is noticed and closed, and consider the `autossh` tool, which restarts a tunnel automatically:

```
$ autossh -M 0 -N -f \
    -o ServerAliveInterval=30 -o ServerAliveCountMax=3 \
    -L 5433:db.internal:5432 user@server
```

### TIP

Remember the direction: **-L** brings a remote service *to you*, while **-R** sends a local service *to the remote side*.

# Securing the SSH Server

An SSH server on the internet is attacked within minutes of going online, so settings matter. The server's configuration lives in `/etc/ssh/sshd_config`. Always keep a second session open while you make changes.

**WARNING**

A mistake in `sshd_config` can lock you out of a remote machine. Keep your current session open, test the configuration, and confirm a **new** login works before closing anything.

## Recommended settings

```
PubkeyAuthentication yes
PasswordAuthentication no
KbdInteractiveAuthentication no
PermitRootLogin no
MaxAuthTries 3
LoginGraceTime 30
X11Forwarding no
AllowUsers alice bob
ClientAliveInterval 300
ClientAliveCountMax 2
```

| Setting                   | Why it matters                               |
|---------------------------|----------------------------------------------|
| PasswordAuthentication no | Blocks password guessing entirely; keys only |
| PermitRootLogin no        | Forces login as a normal user, then sudo     |
| AllowUsers / AllowGroups  | Only the named accounts may log in           |
| MaxAuthTries              | Limits guesses per connection                |
| X11Forwarding no          | Removes a rarely needed attack surface       |

| Setting             | Why it matters                                                 |
|---------------------|----------------------------------------------------------------|
| ClientAliveInterval | Drops dead sessions after a timeout                            |
| Port                | A non-default port reduces log noise, but is not real security |

## Test, then apply

```
$ sudo sshd -t           # check for errors
$ sudo systemctl reload sshd # RHEL, Fedora
$ sudo systemctl reload ssh  # Debian, Ubuntu
```

Then open a **new** terminal and confirm you can log in before closing the old one.

### TIP

On recent Ubuntu releases the SSH service may be started by a socket (ssh.socket). If a Port change does not take effect, check `systemctl status ssh.socket` and your distribution's documentation.

## Add layers of protection

- **Firewall.** Allow port 22 (or your chosen port) only from the addresses that need it, using `ufw`, `firewalld` or your cloud provider's security groups.
- **Fail2ban** (or similar) temporarily blocks addresses that fail to log in repeatedly.
- **Two-factor authentication.** Combine keys with a one-time code for sensitive systems.
- **Keep software updated.** Install OpenSSH security updates promptly.
- **Audit keys.** Review each `authorized_keys` file regularly and remove keys that are no longer needed.

## Restrict a key or user

Prefix a line in `authorized_keys` with options to limit what that key can do:

```
from="203.0.113.0/24",no-port-forwarding,no-pty \  
ssh-ed25519 AAAAC3Nza... backup-job
```

That key works only from one network and cannot open tunnels or a terminal, which suits an automated task.

## Useful server-side checks

```
$ sudo ss -tlnp | grep sshd      # is it listening?  
$ who                          # who is logged in  
$ last -n 20                   # recent logins  
$ sudo journalctl -u ssh -n 50  # service log  
$ sudo tail -f /var/log/auth.log # Debian, Ubuntu  
$ sudo tail -f /var/log/secure  # RHEL, Fedora
```

# Working in Long Sessions

A dropped Wi-Fi connection normally kills your SSH session and everything running in it. A few habits and tools prevent lost work.

## Escape sequences

If a session freezes, you cannot type `exit`. SSH has built-in escape sequences that work at the start of a line: press **Enter**, then the `~` (tilde) key, then the command key.

| Keys         | Action                                              |
|--------------|-----------------------------------------------------|
| Enter, ~, .  | <b>Disconnect</b> a frozen session                  |
| Enter, ~, ?  | Show the help list                                  |
| Enter, ~, C  | Open a command line to add port forwards on the fly |
| Enter, ~, ^Z | Suspend SSH into the background                     |
| Enter, ~, ~  | Send a literal tilde                                |

**TIP**

If you are using SSH inside SSH, each extra tilde sends the command one level deeper. Use `~~.` to close the inner session.

## Keep the connection alive

Firewalls and routers drop connections that stay silent. Send a small keep-alive message regularly, either on the command line or in your config:

```
$ ssh -o ServerAliveInterval=60 user@host

# or in ~/.ssh/config
Host *
    ServerAliveInterval 60
    ServerAliveCountMax 3
```

## Keep programs running with tmux

**tmux** (or the older **screen**) runs your shell on the server, so it keeps going even if you disconnect. Reconnect later and pick up exactly where you left off.

```
$ tmux new -s work          # start a named session
# ... do your work ...
# press Ctrl+B, then D    to detach
$ tmux ls                  # list sessions
$ tmux attach -t work       # reattach
$ tmux kill-session -t work # remove it
```

## Mosh for unreliable networks

**Mosh** (mobile shell) starts over SSH but then survives roaming between networks, sleep and high latency. It is excellent on mobile connections. It needs to be installed on both ends and uses UDP ports 60000 to 61000.

```
$ mosh user@host
```

## Run something and keep it going

```
$ nohup ./long_job.sh > job.log 2>&1 &
$ tail -f job.log
```

For anything important, tmux is usually the better choice, because you can look at the live output again later.

## Handy terminal shortcuts

| Shortcut     | Action                          |
|--------------|---------------------------------|
| Ctrl + C     | Stop the running command        |
| Ctrl + D     | End the session (like exit)     |
| Ctrl + R     | Search your command history     |
| Ctrl + L     | Clear the screen                |
| Ctrl + A / E | Jump to start / end of the line |

# Troubleshooting Errors

Most SSH problems fall into a few well-known categories. The first step is always the same: ask for more detail.

## Step 1: Turn on verbose output

```
$ ssh -v user@host      # some detail
$ ssh -vvv user@host    # maximum detail
```

Read the output from the top. It shows which config file was read, which keys were offered and why each step succeeded or failed. If you control the server, also check its log (see Chapter 7).

## Common errors and fixes

| Message                                | Likely cause and fix                                                                                                                                                                      |
|----------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Permission denied (publickey)          | The server did not accept your key. Check the key is in <code>authorized_keys</code> , that you use the right user and <code>-i</code> key, and that permissions are correct (Chapter 3). |
| Permission denied (publickey,password) | Wrong password, or password login is disabled on the server. Confirm the username.                                                                                                        |
| Connection refused                     | Nothing is listening on that port. Check the SSH service is running, the port number, and the server's local firewall.                                                                    |
| Connection timed out                   | The packets never arrive. Check the IP address, the network, a firewall or cloud security group, and that the server is on.                                                               |
| Could not resolve hostname             | The name is mistyped or DNS is failing. Try the IP address.                                                                                                                               |

| Message                                  | Likely cause and fix                                                                             |
|------------------------------------------|--------------------------------------------------------------------------------------------------|
| Host key verification failed             | The server's key differs from the saved one. See the next section.                               |
| Too many authentication failures         | The agent offered many keys. Use <code>-o IdentitiesOnly=yes -i KEY</code> .                     |
| UNPROTECTED PRIVATE KEY FILE             | Key permissions are too open. Run <code>chmod 600</code> on the key.                             |
| no matching key exchange / host key type | Client and server share no common algorithm. Update the older side; avoid re-enabling weak ones. |
| Connection reset / closed by remote host | The server dropped you. Look at its log; may be a ban from fail2ban or a policy limit.           |
| Broken pipe                              | An idle session was cut. Use keep-alives (Chapter 8).                                            |

## "REMOTE HOST IDENTIFICATION HAS CHANGED"

This scary-looking warning means the server's key no longer matches the one in your `known_hosts` file. Often it is harmless, for example after a server was rebuilt or an IP address was reassigned. But it can also indicate a **man-in-the-middle attack**.

1. Do not simply ignore the warning. Ask: did this server change recently?
2. Verify the new fingerprint with the server's owner or through your provider's console.
3. If it is legitimate, remove the old entry: `ssh-keygen -R host`.
4. Connect again and accept the new key.

### WARNING

Never fix this by disabling host key checking (`StrictHostKeyChecking no`) on a habit. That removes the protection that tells you when someone is impersonating your server.

## Quick network checks

```
$ ping host           # is it reachable?
$ nc -vz host 22      # is the port open?
$ ssh -v -p 2222 user@host # right port?
$ ssh -G host         # what settings apply?
```

## Debug the server (if you have access)

```
$ sudo sshd -t          # config errors
$ sudo /usr/sbin/sshd -d -p 2200 # debug on a spare port
```

The second command runs a one-off debug server on port 2200. Connect with `ssh -p 2200 user@host` and watch the output as you log in.

### TIP

Work from the outside in: can you reach the machine, then the port, then authenticate, then get a shell? The step where it fails tells you where to look.

# Security Best Practices

SSH is secure by design, but only when it is used carefully. These practices protect both your own keys and the servers you manage.

### For your own keys

- Use **Ed25519** keys (or RSA with 3072 bits or more).
- Always protect private keys with a **strong passphrase**.
- Keep private keys only on devices you control, and keep them out of backups, cloud folders and version control.
- Use a **separate key** for each role or service.
- Rotate keys periodically, and immediately if a device is lost or compromised.
- Consider **hardware security keys** (FIDO2) for the strongest protection: `ssh-keygen -t ed25519 -sk`.

### For connections

- **Verify host fingerprints** the first time you connect.
- Be careful with **agent forwarding** (-A). A compromised server can use your forwarded agent. Prefer ProxyJump instead.
- Avoid logging in from untrusted computers, such as shared or public machines.
- Do not run ssh with options that disable checking, except as a deliberate, temporary test.

### For servers

- Disable password login and root login (Chapter 7).
- Allow only specific users or groups.
- Restrict access by firewall to known addresses where possible.

- Keep OpenSSH and the operating system patched.
- Monitor logs for repeated failures and unexpected logins.
- Remove keys for people and systems that no longer need access.
- Use certificates or a central identity system when managing many servers.

## Good habits

- Use `~/.ssh/config` for consistency and fewer typing mistakes.
- Name your keys clearly, for example `id_ed25519_work`.
- Add a comment with `-C` when generating keys, so you can tell them apart later.
- Back up your **public** keys and your config, and store private keys in a password manager or encrypted vault.
- Test any change to a server's SSH settings from a second session before closing the first.

### TIP

If you remember only one thing: **keys with passphrases, no passwords, no root login, and a second terminal open when you change settings.**

# Appendix A: Quick Reference

| Task                  | Command                                           |
|-----------------------|---------------------------------------------------|
| Connect               | <code>ssh user@host</code>                        |
| Custom port           | <code>ssh -p 2222 user@host</code>                |
| Use a key             | <code>ssh -i ~/.ssh/key user@host</code>          |
| Run a command         | <code>ssh user@host 'uptime'</code>               |
| Debug                 | <code>ssh -vvv user@host</code>                   |
| New key (Ed25519)     | <code>ssh-keygen -t ed25519 -C "comment"</code>   |
| Install key on server | <code>ssh-copy-id user@host</code>                |
| Load key into agent   | <code>ssh-add ~/.ssh/id_ed25519</code>            |
| List agent keys       | <code>ssh-add -l</code>                           |
| Forget a host key     | <code>ssh-keygen -R host</code>                   |
| Show key fingerprint  | <code>ssh-keygen -lf key.pub</code>               |
| Jump host             | <code>ssh -J bastion user@host</code>             |
| Show effective config | <code>ssh -G alias</code>                         |
| Upload a file         | <code>scp file user@host:~/</code>                |
| Download a file       | <code>scp user@host:~/file .</code>               |
| Copy a folder         | <code>scp -r dir user@host:~/</code>              |
| Sync a folder         | <code>rsync -avz dir/ user@host:~/dir/</code>     |
| Interactive transfer  | <code>sftp user@host</code>                       |
| Local forward         | <code>ssh -L 8080:localhost:80 user@host</code>   |
| Remote forward        | <code>ssh -R 9000:localhost:3000 user@host</code> |

| Task                      | Command                                          |
|---------------------------|--------------------------------------------------|
| SOCKS proxy               | <code>ssh -D 1080 -N user@host</code>            |
| Background tunnel         | <code>ssh -N -f -L 5433:db:5432 user@host</code> |
| Disconnect frozen session | Enter, then ~ then .                             |
| Check server config       | <code>sudo sshd -t</code>                        |

# Appendix B: Glossary

---

**Agent (ssh-agent).** A background program that holds your decrypted private keys so you do not retype the passphrase.

**authorized\_keys.** The server-side file listing the public keys allowed to log in to an account.

**Bastion / jump host.** A hardened server used as the single entry point to a private network.

**Client.** The program, such as ssh, that you run to connect to a server.

**Ed25519.** A modern, fast and secure type of key, recommended for new keys.

**Fingerprint.** A short hash that identifies a key, used to verify a server or key is the one you expect.

**Host key.** The server's own key, which proves its identity to clients.

**known\_hosts.** The client-side file recording the host keys of servers you have connected to.

**Man-in-the-middle.** An attack in which someone secretly intercepts and relays a connection.

**OpenSSH.** The most widely used implementation of the SSH tools.

**Passphrase.** A password that protects a private key stored on disk.

**Port.** A numbered channel on a computer; SSH normally uses 22.

**Port forwarding.** Carrying other network traffic through an SSH connection; also called tunnelling.

**Private key.** The secret half of a key pair. It must never be shared.

**Public key.** The shareable half of a key pair, placed on servers.

**rsync.** A tool that copies and synchronises files efficiently, often over SSH.

**SCP.** Secure copy: a simple file transfer command that uses SSH.

**Server (sshd).** The program on the remote machine that accepts SSH connections.

**SFTP.** SSH File Transfer Protocol: interactive file transfer over SSH.

**SOCKS proxy.** A general-purpose proxy; SSH can create one with -D.

**tmux.** A terminal multiplexer that keeps sessions running after you disconnect.

**Tunnel.** An encrypted path through which other traffic is sent.