

way of the hax0r
n00b to console punk

Chapter 12

Work Across Machines

The shell you have been using runs on the computer in front of you. Much of the interesting work happens elsewhere: a server in a data center, a Raspberry Pi on your desk, a lab machine, a virtual machine in the cloud. The console reaches all of them the same way, and a shell on a remote machine is the same shell you already know.

The concepts in this chapter are:

- **SSH**, the secure shell: a client, a server, and an encrypted connection between them;
- **host keys** and **user keys**, and why the first connection asks a question;
- running a **remote command** without an interactive session;
- **transferring files** with `scp`, `sftp`, and `rsync`;
- **downloading** with `curl`, `wget`, and `Invoke-WebRequest`;
- **troubleshooting** a connection in order: name, network, port, authentication;
- **persistent sessions** with `tmux`, so that work survives a dropped connection.

You need a machine to connect to. A second computer on your network, a virtual machine, a cloud instance, or a Raspberry Pi all work; on Linux and macOS you can also enable the SSH server on your own machine and connect to `localhost`. The examples use `student@lab.example.com`; substitute your own user and host.

SSH

SSH is a protocol and a pair of programs. The **server**, `sshd`, runs on the remote machine and listens on port 22. The **client**, `ssh`, runs on yours, opens an encrypted connection, proves who you are, and gives you a shell on the other end. Everything you type travels encrypted, and so does everything that comes back. [1]

Every operating system in this book ships the client. Windows 10 and later include OpenSSH, so `ssh` works from PowerShell and Command Prompt; the server is an optional feature there. [2]

Connect

```
$ ssh student@lab.example.com
The authenticity of host 'lab.example.com (203.0.113.10)' can't be established.
ED25519 key fingerprint is SHA256:Q7...
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

The first connection to a machine shows its **host key** fingerprint and asks you to confirm it. The host key identifies the server; once you answer yes, ssh stores it in `~/.ssh/known_hosts` and checks it silently on every later connection. If someone puts a different machine at that name, the check fails loudly, which is the point. Confirm the fingerprint against one the administrator gave you when it matters. [1]

Then comes the password prompt, and then a prompt on the remote machine. Your terminal is still local; the shell is remote. `exit` ends the session and returns you to your own shell.

Omit `user@` when your username is the same on both machines. `-p 2222` selects a different port.

Keys Instead of Passwords

A **key pair** replaces the password: a private key that stays on your machine and a public key that you place on each server. It is more secure than a password and, once set up, faster.

```
$ ssh-keygen -t ed25519 -C 'student laptop'
$ ssh-copy-id student@lab.example.com
```

`ssh-keygen` creates `~/.ssh/id_ed25519` and `~/.ssh/id_ed25519.pub`. Accept the default location and give the key a passphrase; the passphrase protects the file if your laptop is lost. `ssh-copy-id` appends the public key to `~/.ssh/authorized_keys` on the server, after one last password login. The private key file must be readable only by you, mode `600`, and ssh refuses to use it otherwise. [3]

Windows has no `ssh-copy-id`. Copy the public key by hand:

```
$ type $env:USERPROFILE\.ssh\id_ed25519.pub | ssh student@lab.example.com "mkdir -p ~/.ssh && cat -> authorized_keys"
```

An **agent** holds the unlocked key so that you type the passphrase once per session. `ssh-agent` with `ssh-add` does this on Linux and macOS; on Windows the OpenSSH Authentication Agent service does, once started. [1], [4]

A Configuration File

`~/.ssh/config` gives a host a short name and remembers its settings:

```
Host lab
    HostName lab.example.com
    User student
    Port 22
    IdentityFile ~/.ssh/id_ed25519
```

After that, `ssh lab` does everything the long form did, and `scp` and `rsync` read the same file. [5]

Run a Command Remotely

A command after the host runs there and returns, with its output arriving in your terminal:

```
$ ssh lab uptime
$ ssh lab 'ls -l /var/log | head'
$ ssh lab 'df -h' > lab-disk.txt
```

Quote the remote command so that your local shell does not expand its pipes and variables first. The last line shows the two shells working together: the remote `df` runs on the server, and the local `>` writes the result on your machine. Chapter 7's exit status carries across too, so `ssh lab test -f /tmp/done && echo finished` works. [1]

Transfer Files

scp

`scp` copies files over SSH with `cp`'s syntax, where a path with `host:` in front is remote:

```
$ scp notes.txt lab:~/
$ scp lab:~/results.csv .
$ scp -r docs lab:~/docs
```

`-r` copies a directory. `scp` overwrites without asking, like `cp`. [6]

sftp

`sftp lab` opens an interactive session with `ls`, `cd`, `get`, and `put` commands, plus `lls` and `lcd` for the local side. It suits a look around before deciding what to copy. [7]

rsync

`rsync` copies only what changed, which makes it the tool for repeated transfers and backups:

```
$ rsync -av docs/ lab:~/docs/
$ rsync -av --delete lab:~/results/ ./results/
```

`-a` preserves permissions and times and recurses; `-v` lists what it copied. The trailing slash on a source directory means “the contents of”, and leaving it off copies the directory itself into the destination. `--delete` removes files from the destination that are gone from the source, so preview it first with `-n`, the dry run. `rsync` is standard on Linux and macOS; on Windows it is available through WSL. [8]



Trap: `rsync -av docs lab:~/docs/` and `rsync -av docs/ lab:~/docs/` put the files in different places. Run with `-n` and read the file list before any transfer with `--delete`.

Download from the Web

Most tools and data arrive over HTTP, and a command-line download can be scripted, resumed, and checked.

Bash/zsh and Command Prompt:

```
$ curl -O https://example.com/index.html
$ curl -o page.html https://example.com/
$ curl -sSL https://example.com/api | head
```

-O saves with the remote filename, -o with a name you choose. -L follows redirects, -s silences the progress bar, and -S still shows errors. Without -o or -O, curl prints to standard output, so it fits into a pipeline. Windows 10 and later ship curl.exe. [9]

wget https://example.com/file.zip saves the file and resumes with -c; it is common on Linux and absent by default on macOS and Windows. [10]

PowerShell:

```
$ Invoke-WebRequest -Uri https://example.com/file.zip -OutFile file.zip
$ Invoke-RestMethod -Uri https://api.github.com/repos/git/git
```

Invoke-WebRequest downloads; Invoke-RestMethod fetches and parses JSON or XML into objects that the pipeline can use. iwr and irm are their aliases. [11], [12]



Wut: In Windows PowerShell 5.1, curl and wget are aliases for Invoke-WebRequest, so a curl command from a tutorial fails with parameter errors. PowerShell 7 removed those aliases. Type curl.exe to be sure you get the real program.

Whatever you download, check it before running it. A checksum, which Chapter 13 covers, confirms that the file is the one the publisher meant; reading a script before running it confirms what it does.

When the Connection Fails

Work through the layers in order, and stop at the first one that fails.

1. **Name.** Does the hostname resolve? ping lab.example.com, nslookup lab.example.com, or Test-Connection lab.example.com in PowerShell. “Could not resolve hostname” is a name problem, not an SSH problem. [13], [14]
2. **Network.** Does the machine answer at all? A ping reply says the machine is up, though some networks block ping.
3. **Port.** Is the SSH server reachable? ssh -v lab prints each step of the connection, and “Connection refused” or “Connection timed out” points at the server or a firewall. On Windows, Test-NetConnection lab.example.com -Port 22 tests one port. [15]
4. **Authentication.** “Permission denied (publickey)” means the server accepts only keys and did not accept yours. Check that the public key is in authorized_keys, that the private key’s permissions are 600, and that ssh -v shows it being offered.

5. **Host key.** A warning that the host identification has changed means the server's key differs from the one in `known_hosts`. If the machine was reinstalled, remove the old line with `ssh-keygen -R lab.example.com`; if it was not, stop and ask.

`ssh -v` answers most questions; `-vv` and `-vvv` add detail.

Sessions That Survive

A connection can drop: your laptop sleeps, the Wi-Fi changes, the train enters a tunnel. When it does, the remote shell and everything it was running end with it. A **terminal multiplexer** keeps a session alive on the server so that you can reconnect to it.

tmux is the usual choice, installed with a package manager on the server:

```
$ tmux new -s work
```

That opens a session named `work` with a status bar at the bottom. Work in it as usual. To leave the session running and return to the plain shell, press `Ctrl+B` and then `d`, which **detaches**. Later, from a new connection: [16]

```
$ tmux ls
work: 1 windows (created Sun Sep 20 10:00:00 2026)
$ tmux attach -t work
```

Everything is as you left it, including a program that kept running while you were gone. `tmux kill-session -t work` ends it when you are done.

tmux also splits the terminal: `Ctrl+B %` splits into two panes side by side, `Ctrl+B "` one above the other, and `Ctrl+B` with an arrow key moves between them. `Ctrl+B c` creates a new window and `Ctrl+B n` moves to the next one. The `Ctrl+B` **prefix** is what makes **tmux** keys different from terminal and shell keys, which Chapter 4 sorted out. [17]

GNU **screen** is the older tool with the same idea, and some servers have only that; its prefix is `Ctrl+A`. On Windows, run **tmux** inside WSL, or rely on Windows Terminal's tabs for local work and **tmux** on the remote side where it matters.

Key Points

- `ssh user@host` opens an encrypted shell on another machine; the first connection records the server's host key in `known_hosts`.
- `ssh-keygen -t ed25519` and `ssh-copy-id` replace passwords with a key pair; keep the private key at mode 600 and give it a passphrase.
- `~/.ssh/config` names hosts and remembers their settings for `ssh`, `scp`, and `rsync`.
- `ssh host command` runs one command remotely; quote it so the local shell leaves it alone.
- `scp` copies like `cp`, `sftp` browses, and `rsync -av` copies only what changed; a trailing slash on a source directory changes what is copied.
- `curl -O`, `wget`, and `Invoke-WebRequest -OutFile` download; PowerShell 5.1's `curl` is an alias for something else.
- Troubleshoot name, network, port, authentication, and host key in that order, with `ssh -v`.

- `tmux new -s name, Ctrl+B d`, and `tmux attach -t name` keep a remote session alive across disconnections.

Level Up

n00bs finish the chapter here. You can connect with a key, run a command remotely, move files both ways, and reattach to a session. Hackers use SSH as a building block.

Learn **port forwarding**: `ssh -L 8080:localhost:80 lab` brings a remote web server to your browser, and `-R` does the reverse. Explain what a **jump host** is and set one up with ProxyJump in your config.

Read about `ssh-agent` forwarding, then read why many administrators disable it. Compare with hardware keys and `sk-ssh-ed25519`.

Explore `mosh`, which survives roaming and sleep better than plain SSH, and explain what it changes about the connection.

Configure `tmux`: a `~/.tmux.conf` with a different prefix, mouse support, and a status line you like. Then look at **Zellij** as an alternative and compare its defaults.

Write a script that uses `rsync` to back up a directory to another machine, with a dry run first and a log. Chapter 11 has everything you need.

Exercises

1. **What does this do?** Predict what `ssh lab 'echo $HOME'` and `ssh lab "echo $HOME"` print when your local and remote home directories differ, then explain the difference.
2. **What does this do?** Predict the difference between `rsync -av src lab:~/dest/` and `rsync -av src/ lab:~/dest/`, then test with `-n`.
3. **What is the problem?** `ssh lab` says “Permission denied (publickey)” right after you copied your public key with `scp` to the server’s home directory. Explain what is missing.
4. **What is the problem?** A download with `curl` in Windows PowerShell 5.1 fails with “A parameter cannot be found that matches parameter name ‘O’”. Explain and fix.
5. **Why is it like this?** SSH asks you to confirm a host key on the first connection but not afterward. Explain what it is protecting against and what a later warning means.
6. **Why is it like this?** A program started over SSH dies when the connection drops, but the same program inside `tmux` survives. Explain what each one’s parent process is.
7. **How do you** copy a directory tree to a remote machine so that a second run copies only the files that changed?
8. **How do you** find out which step of an SSH connection is failing, without guessing?

References

- [1] OpenBSD Project, “Ssh(1): OpenSSH remote login client.” Available: <https://man.openbsd.org/ssh>
- [2] Microsoft, “OpenSSH for windows overview.” Available: https://learn.microsoft.com/en-us/windows-server/administration/openssh/openssh_overview
- [3] OpenBSD Project, “Ssh-keygen(1): OpenSSH authentication key utility.” Available: <https://man.openbsd.org/ssh-keygen>
- [4] Microsoft, “Key-based authentication in OpenSSH for windows.” Available: https://learn.microsoft.com/en-us/windows-server/administration/openssh/openssh_keymanagement
- [5] OpenBSD Project, “Ssh_config(5): OpenSSH client configuration file.” Available: https://man.openbsd.org/ssh_config
- [6] OpenBSD Project, “Scp(1): OpenSSH secure file copy.” Available: <https://man.openbsd.org/scp>
- [7] OpenBSD Project, “Sftp(1): OpenSSH secure file transfer.” Available: <https://man.openbsd.org/sftp>
- [8] Samba Team, “Rsync(1) manual page.” Available: <https://download.samba.org/pub/rsync/rsync.1>
- [9] Daniel Stenberg, “Curl(1) manual page.” Available: <https://curl.se/docs/manpage.html>
- [10] GNU Project, “GNU wget manual.” Available: <https://www.gnu.org/software/wget/manual/wget.html>
- [11] Microsoft, “Invoke-WebRequest.” Available: <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/invoke-webrequest>
- [12] Microsoft, “Invoke-RestMethod.” Available: <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/invoke-restmethod>
- [13] Microsoft, “Windows commands: nslookup.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/nslookup>
- [14] Microsoft, “Test-connection.” Available: <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/test-connection>
- [15] Microsoft, “Test-NetConnection.” Available: <https://learn.microsoft.com/en-us/powershell/module/nettcpip/test-netconnection>
- [16] tmux Project, “Tmux wiki: Getting started.” Available: <https://github.com/tmux/tmux/wiki/Getting-Started>
- [17] tmux Project, “Tmux(1) manual page.” Available: <https://man.openbsd.org/tmux>

