Terminal Tools, Explained

tmux · mosh · OpenSSH · htop — and the WSL2 layer that runs them on Windows.

Prepared for: Eduardo Pergola Performance 2026-06-17 Reference / internal

---

This is a plain-English map of four classic command-line tools and the Windows layer (WSL2) they run on. They exist to do one thing well: **run work on a remote machine and not lose it when your connection, network, or laptop misbehaves.** Read top to bottom and the way they snap together at the end will make sense.

> **THE 10-SECOND MENTAL MODEL**
> **WSL2** is the Linux engine on your PC. **SSH / MOSH** are how you reach *another* machine. **TMUX** is the workspace that keeps running on that machine after you walk away. **HTOP** is the dashboard that shows you what the machine is doing.

0. Why these tools at all

A normal terminal has one fatal weakness for remote work: **if the connection drops, whatever was running dies with it.** Close the laptop, switch from Wi-Fi to your phone's hotspot, walk into an elevator — the session is gone, and so is the long build/deploy/script you had running.

These tools each remove one piece of that fragility:

- **SSH** — get to the other machine *securely* in the first place.
- **mosh** — keep that connection alive across network changes and sleep.
- **tmux** — keep the *work* alive even when the connection truly ends, and give you panes/tabs.
- **htop** — see what the machine is actually doing (CPU, memory, runaway processes).
- **WSL2** — the Linux environment that lets tmux and mosh exist on Windows at all.

1. WSL2 — the Linux engine on Windows

What it is: Windows Subsystem for Linux, version 2. It runs a **real Linux kernel** inside a fast, lightweight virtual machine that boots in about a second, so you can run a full Ubuntu (with its `apt` package manager) right alongside Windows. Your Windows files and Linux files can see each other.

Why you need it: `tmux` and `mosh` are Unix programs — there is **no native Windows build** of them. WSL2 gives them the Linux they expect. (SSH is the exception — Windows has a native port — but it's cleanest to keep the whole toolchain together in Ubuntu.)

Get started

```
wsl --install           # installs WSL2 + Ubuntu (one time, needs admin + a reboot)
wsl                     # drop into your Ubuntu shell
sudo apt update         # refresh the package lists inside Ubuntu
```

> **ON THIS MACHINE**
> We are installing WSL2 + Ubuntu and **RELOCATING IT TO D:** (your C: drive is tight), then installing the four tools inside it with one `apt` command. See the appendix.

2. OpenSSH — the secure way in

What it is: SSH = *Secure Shell*. It gives you an encrypted remote login to another computer — a terminal on a machine that may be across the room or across the world — plus secure file copy (`scp` / `sftp`). Everything over the wire is encrypted, so it's safe even on untrusted networks.

Two halves:

- **The client** (`ssh`) — what you run to connect *out*. Windows 11 ships this built in.
- **The server** (`sshd`) — runs on the machine you connect *to*, accepting connections. On Windows it's an optional feature; on Ubuntu it's the `openssh-server` package.

Password vs. keys (use keys)

You can log in with a password, but **public-key auth** is safer and lets you skip typing a password every time: you generate a keypair, keep the private half secret, and put the public half on the server.

```
ssh-keygen -t ed25519                # make a modern keypair (once)
ssh user@server.example.com          # connect (password or key)
ssh-copy-id user@server.example.com  # install your public key on the server
```

3. tmux — the workspace that never closes

What it is: a *terminal multiplexer*. It turns one terminal into many: **sessions** (whole workspaces) that each hold **windows** (like tabs) that each hold **panes** (splits). But its real superpower is **detach & reattach.**

The killer feature: a tmux session runs *on the server*, independent of your terminal. You can **detach** (Ctrl-b then d) and disconnect entirely — the session keeps running. Hours later, from a different computer, you `tmux attach` and everything is **exactly where you left it**: your long-running job, your open files, your layout. Dropped connections stop being a disaster.

The essentials

Almost everything is the **prefix** Ctrl-b, released, then one more key:

Keys	Does
Ctrl-b d	Detach (leave it running on the server)
Ctrl-b c	New window (tab)
Ctrl-b %	Split pane left/right
Ctrl-b "	Split pane top/bottom
Ctrl-b ←↑↓→	Move between panes
Ctrl-b [	Scroll-back mode (q to exit)

```
tmux new -s main          # start a named session "main"
tmux ls                   # list running sessions
tmux attach -t main       # re-attach to "main"
```

4. mosh — the connection that survives anything

What it is: *Mobile Shell* — a roaming, resilient replacement for the interactive part of SSH. It uses SSH to log in and authenticate, then switches to its own UDP-based protocol for the session itself.

Why it's better on flaky links:

- **Survives network changes** — Wi-Fi → cellular, new coffee-shop network, a new IP address: the session just continues.
- **Survives sleep** — close the laptop, reopen it hours later, keep typing.
- **Instant local echo** — your keystrokes appear immediately even on a high-latency link, because mosh predicts and shows them locally instead of waiting for the round trip. Typing feels local.

> **MOSH DOES NOT REPLACE TMUX**
> mosh keeps your **CONNECTION** alive across network blips. tmux keeps your **SESSION/WORK** alive even when the connection truly ends, and gives you panes and windows. They solve different halves of the problem — **USE BOTH TOGETHER.**

```
mosh user@server.example.com     # connect (needs mosh installed on BOTH ends)
                                 # uses UDP ports 60000-61000 (open them in the firewall)
```

5. htop — the live dashboard

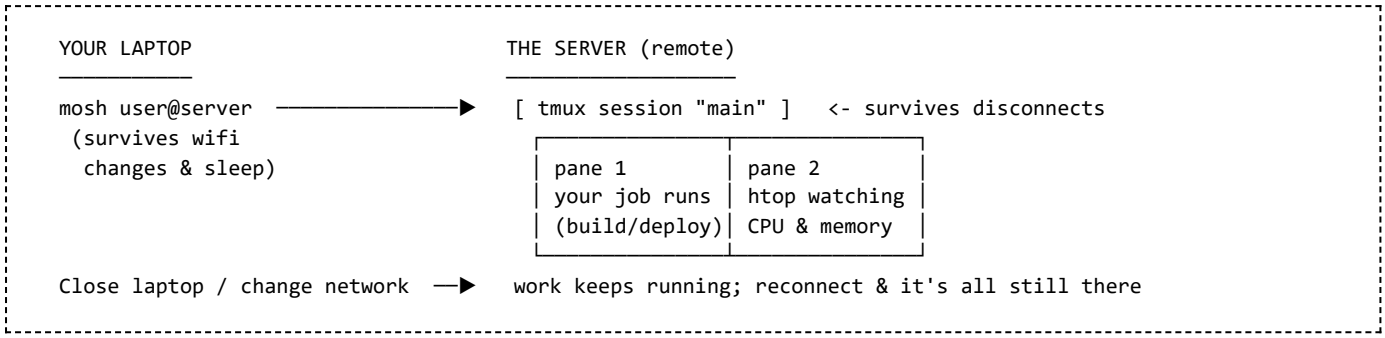
What it is: an interactive, color, scrollable process and resource monitor — a much friendlier `top`. Run `htop` and you instantly see the health of the machine.

- **Top bars:** one bar **per CPU core**, plus memory and swap usage, plus the load average and uptime.
- **Process list:** every running process, sortable and scrollable, with a tree view (F5) to see what spawned what.
- **Act on it:** search (F3), filter (F4), sort (F6), kill a runaway process (F9), renice priority (F7/F8), quit (F10 or q).

When a server "feels slow," `htop` is the first place you look: it shows the one process eating all the CPU or RAM.

6. How they fit together

The classic remote-work flow chains all four. You connect resiliently, drop into a persistent workspace, and keep an eye on the box while your real job runs:



In commands:

```
mosh user@server                      # 1. resilient connection in
tmux attach -t main || tmux new -s main   # 2. join (or start) the persistent workspace
# Ctrl-b %  to split, then in the new pane:
htop                                  # 3. watch the machine while your job runs
# Ctrl-b d  to detach and walk away — everything keeps running
```

7. One-page cheat sheet

Tool	In one line	Start it
WSL2	Real Linux on Windows; the engine the others run on	`wsl`
ssh	Secure encrypted login to another machine	`ssh user@host`
mosh	Like ssh, but survives network changes & sleep	`mosh user@host`
tmux	Persistent workspace + panes; survives disconnects	`tmux new -s main`
htop	Live CPU / memory / process dashboard	`htop`

8. Windows notes

Windows Terminal is the modern host app (usually already on Windows 11). It gives you tabs and a profile per shell — PowerShell, Command Prompt, and your **Ubuntu (WSL)** profile. Open the Ubuntu profile and `tmux`, `mosh`, and `htop` all run inside it.

> **DON'T CONFUSE WINDOWS TERMINAL PANES WITH TMUX PANES**
> Windows Terminal's own split panes are **LOCAL ONLY** — they vanish if you close the window, and they don't exist on the remote server. **TMUX** panes live *on the server* and persist through disconnects. That's exactly why you still want tmux even though Windows Terminal can split — they operate at different layers.

9. Appendix — how it's being set up here

On this PC we install WSL2 + Ubuntu, move it off the cramped C: drive onto D:, then install all four tools in one shot:

```
# 1. Windows side (admin PowerShell, one time):
wsl --install

# 2. Move the distro off C: -> D: (so it doesn't eat the small C: drive)
#    (export the fresh install, re-import it under D:\wsl\ubuntu)

# 3. Inside Ubuntu, install the toolchain:
sudo apt update
sudo apt install -y tmux mosh openssh-client openssh-server htop

# 4. Verify:
tmux -V ;  mosh --version ;  ssh -V ;  htop --version
```

After that, the daily entry point is: open **Windows Terminal → Ubuntu**, then `mosh` or `ssh` out to wherever you're working, attach `tmux`, and you're set.
