remote-run: long jobs on another machine

Key takeaways

1. A long remote job needs a home that outlasts your SSH connection.
2. Keep failures longer than successes: failures are debugging material.
3. Write the policy where the agent reads it: tmux default, ssh exception.

Two standard tools, 255 lines of glue

ssh and tmux do the work; remote-run makes them one command

- remote-run invents no new mechanism; it makes the safe one the default.
- One verb to launch a remote job; one verb to check on it; one verb to see what it printed.
- Successful jobs get tidied up automatically. Failed jobs stick around so you can debug them.

Vocabulary

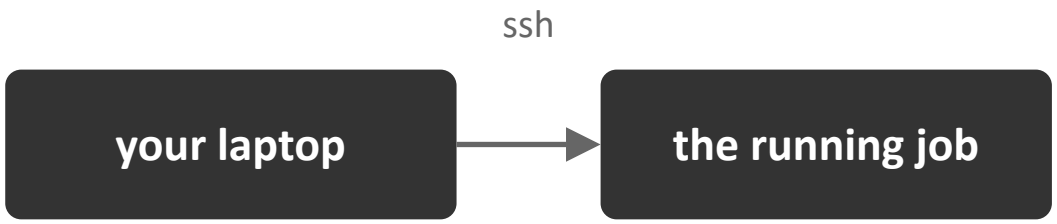
Four terms used throughout — if you've never run a job on a remote computer, start here

- SSH (Secure Shell)
  - A way to run a command on another computer over the network.
  - You type a command locally; the remote machine runs it and sends its output back to you.
- tmux
  - Software that lets ONE terminal hold MANY separate command-line sessions side-by-side.
  - Key property: those sessions keep running even after you disconnect, so a long build doesn't die when your laptop sleeps.
- Exit code (a.k.a. exit status)
  - A number a program returns when it finishes. 0 means success; any non-zero number means failure.
  - Humans rarely care about the exact non-zero value — only the success-vs-failure distinction.
- Reaping a process
  - Cleaning up the bookkeeping after a program ends. Unix leaves a small 'zombie' entry behind until something acknowledges the exit — reaping is that acknowledgement.

The problem

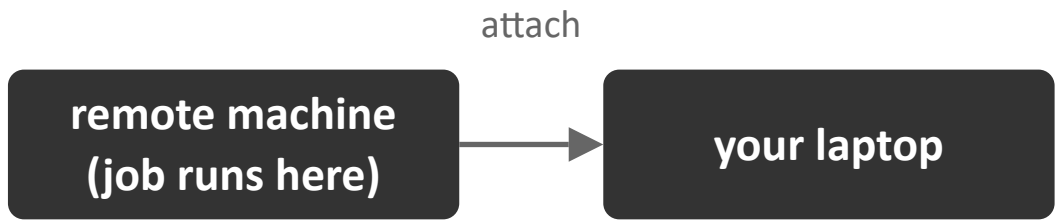
Why plain ssh isn’t enough, and what tmux gives us

Plain ssh — job tied to your connection



laptop sleeps → the job dies

tmux — job tied to the remote machine



laptop sleeps → the job keeps running

Using tmux by hand is fiddly — ssh in, start tmux, name the window, run the command, detach, remember the name — every single time. remote-run automates the whole dance into one command.

A new problem appears

Every job you launch leaves a window behind, even after it ends

- What ‘leaving a window behind’ means
  - When a command finishes inside tmux, the window can be set to stay open so you can read the final output.
  - remote-run uses that setting — otherwise you’d lose the output the instant the job ended.
  - Trade-off: the window persists forever unless someone removes it.
- After a few weeks of daily use
  - Dozens of dead windows from yesterday’s tests, last week’s builds, a month-old benchmark.
  - Finding today’s job in the list becomes its own chore.
  - Something has to clean up. The question is: what, and when, and which windows are safe to delete?

Two ways to clean up

The choice rests on a single observation about success vs failure

Naive: a timer (the first attempt)
Every night, delete all windows older than 36 hours. Simple. Reliable. Throws away evidence from yesterday's failed build — the one you wanted to investigate.

Better: ask the exit code
When you launch a new job, delete only the windows whose previous job succeeded (exit 0). Keep the failed ones (non-zero) indefinitely — they're the ones worth reading.

- Plain-English principle: keep failures longer than successes. Failures are debugging material; successes are just history.
- tmux tells you the exit code of each finished window via a field called pane_dead_status — the cleanup script just reads it.

The interface — one tool, a small set of verbs

HOST is the remote machine; NAME is the window you give the job

```
remote-run HOST CMD             # launch CMD on HOST in a fresh tmux window
remote-run --check  HOST NAME   # running / finished / failed?
remote-run --log    HOST NAME   # dump everything the job printed
remote-run --attach HOST NAME   # live view: watch it in real time
remote-run --list   HOST        # every job on HOST and its status
remote-run --kill   HOST NAME   # stop the job and drop its window
remote-run --clean  HOST        # reap all finished windows now
# options, before HOST:  --name NAME   --cd DIR   --session NAME
```

All jobs share one tmux session named worker — one place to find them all.

A worked session — dispatch, poll, read

Launch a build, poll until it finishes, then read its output

```
$ remote-run --name build chambers 'make -j 2>&1 | tee build.log'  # launch
Started on chambers in worker:build
  Check:  remote-run --check chambers build
  Log:    remote-run --log chambers build
  Attach: remote-run --attach chambers build

$ remote-run --check chambers build     # poll
Running

$ remote-run --check chambers build     # ...eight minutes later
Finished (exit 0)

$ remote-run --list chambers            # survey every job
build      (done exit 0)
sweep      running
```

Mental model

tmux by default, plain ssh by exception — written in the agent's config

Default: remote-run via tmux
If a command could outlast the SSH connection — a test, a build, a benchmark, a server, a training run, a data job — launch it through remote-run.

Exception: raw ssh
Only for trivial one-shots whose output is short and deterministic — ssh HOST hostname, ssh HOST git pull. Nothing longer.

- The agent (Claude Code) reads this rule from a global config file at the start of every session — no per-session reminder needed.
- After launching a job, the agent reports the window name and the three follow-up verbs — check, log, and attach — so anyone can pick up where it left off.

Developer story — cleanup by exit status

What we tried first
A timer: every 36 hours, delete every dead window. Simple, but blind to the difference between succeeding and failing.

What we learned
tmux already records the exit code of each window in a field called pane_dead_status. The exit code is a richer ‘done’ signal than the clock.

What we shipped
When a new job is launched, sweep up only the exit-0 windows. Leave non-zero windows alone — possibly forever.

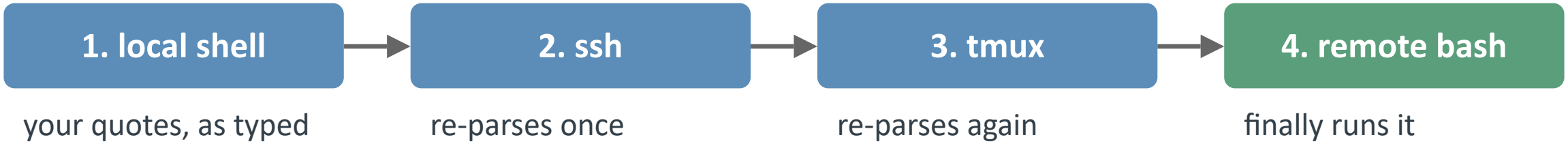
- The age-based version worked, but threatened to silently delete the evidence from yesterday’s failed build — the evidence a human would still want to read.
- Exit 0 means ‘safe to forget’. Non-zero means ‘a human probably wants to look at this’. Two outcomes, two retention policies.
- This policy arrived five weeks into daily use, not at design time — the tool had to accumulate dead windows before the right rule was obvious.
- Also discarded: a scheme where each job printed a ‘done’ marker — crashes don’t print markers, so it can’t distinguish a crash from a still-running job.

source: dotfiles/HISTORY.md @ 2026-04-20; commit 41158a9

What took longer than expected

Three surprises in a 255-line script

- macOS silently broke the duplicate-name check
  - BSD grep rejected dash-leading names, so duplicates slipped through — the fix was a six-character guard, two weeks to find.
- Removing a feature was harder than adding it
  - A sibling tool took over secrets, so the flag became dead weight: when responsibility moves, shrink the wrapper.
- Quoting had to survive four layers of shell
  - Each hop below re-parses the quotes; remote-run ships a temp script and runs that instead.



Key Takeaways

1. Long remote jobs need somewhere to live that outlasts your connection. tmux provides that; remote-run makes it usable.
2. Asymmetric retention by outcome — keep failures longer than successes, because failures are debugging material and successes are just history.
3. Write the policy down where the agent will read it (‘tmux by default, ssh by exception’), so the rule survives across sessions without per-prompt reminders.
4. When a sibling tool takes over a responsibility, shrink the wrapper instead of growing it — the dispatcher should own dispatching, not everything adjacent to it.