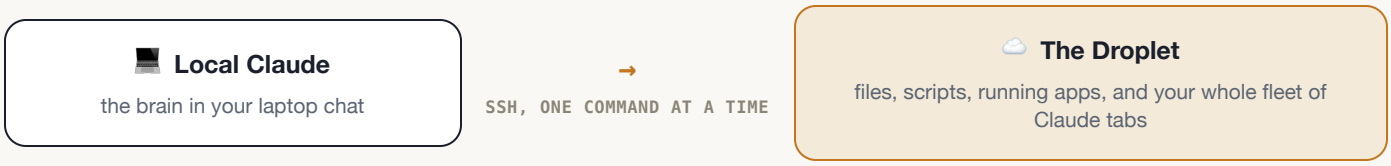
FIELD GUIDE, PART TWO

Local Claude, Remote Hands: Driving Your Server From One Chat

Part one got you the fleet: an always-on server running persistent Claude agents. This is the upgrade that ties it together: teaching the **Claude on your laptop** to reach into that server, run things, read things, and even **talk to every Claude in your fleet**, all by itself.



You talk to one Claude. That Claude operates everything else.

Hey, Mark again. You asked how my local Claude "speaks" to all the droplet chats. This is the whole trick, written down. It takes about ten minutes to set up and it changes how the whole system feels.

00 The idea in 30 seconds

why this is the unlock

Claude Code on your laptop can run terminal commands. SSH is a terminal command. Put those together:

- Your local Claude runs `ssh cs 'some command'`. The command executes **on the server**, the output comes back into the chat, and Claude reads it and decides what to do next.
- That means one conversation on your laptop can: read and edit server files, run and test scripts, check what's running, fix what's broken, deploy things, and pass work to the always-on agents.
- You never type server commands yourself. You say "check if the scraper on the server is still running and restart it if not," and your local Claude does the SSH round trips on its own.

Local brain, remote hands. The laptop Claude is the operator; the server is the workshop it operates.

01 One-time setup

~10 minutes, do once

Two things make this smooth: a short **alias** for your server, and a **key** so nothing asks for a password mid-task.

1 **Give your server a nickname.**
Add this to the file `~/.ssh/config` on your Mac (create it if missing), swapping in your server's IP and username:

~/.ssh/config on your Mac

```
Host cs
  HostName your-droplet-ip
  User your-username
```

1 **Set up a key so logins are automatic.**
Two commands in iTerm2 (the second asks for your server password one last time):

iTerm2 · your Mac

```
$ ssh-keygen -t ed25519   # press Enter through the prompts
$ ssh-copy-id cs          # type your server password once, ever
$ ssh cs 'echo it works'  # should print: it works, no password asked
```

WHY THE KEY MATTERS FOR CLAUDE SPECIFICALLY

Claude runs commands non-interactively. If SSH stops to ask for a password, the command hangs and the whole flow breaks. Key-based login means every `ssh cs '...'` just works, hundreds of times a session, silently.

02 The core move

one-shot commands

Everything is variations of one pattern: **a single, self-contained command wrapped in quotes.** Fire, read the output, decide, fire again.

what your local Claude actually runs

```
# look at something
$ ssh cs 'ls ~/projects/my-app/'
$ ssh cs 'tail -30 ~/projects/my-app/logs/app.log'

# do something
$ ssh cs 'cd ~/projects/my-app && node build.js'

# move files between machines
$ scp report.pdf cs:~/projects/my-app/
$ scp cs:~/projects/my-app/results.json ~/Downloads/

# run a whole script without saving it first (a "heredoc")
$ ssh cs 'bash -s' <<'EOF'
cd ~/projects/my-app
grep -c "error" logs/app.log
EOF
```

The habits that make it reliable

- **One shot, never interactive.** No plain `ssh cs` that opens a live session and waits. Every call is `ssh cs 'command'` that runs and returns.
- **Full paths, always.** Each command starts fresh on the server, so `cd` from a previous command is forgotten. Chain with `&&` or use absolute paths.
- **Add a timeout.** `ssh -o ConnectTimeout=25 cs '...'` so a sleepy server fails fast instead of hanging the chat.
- **Say which machine.** Ask your Claude to always label whether a command ran on your Mac or the server. When something goes wrong, this is the first thing you need to know.

03 Talking to the fleet

the part you asked about

Your fleet tabs are Claude sessions living inside tmux on the server. tmux has two magic commands that make them reachable from outside: **send-keys** (type into a tab) and **capture-pane** (read a tab's screen). Your local Claude can use both over SSH.

local Claude → a droplet Claude tab

```
# send a task to the Claude running in the "myapp" tab
$ ssh cs "tmux send-keys -t fleet:myapp 'run the test suite and summarize failures' Enter"

# wait a bit, then read that tab's screen to see the reply
$ ssh cs "tmux capture-pane -p -t fleet:myapp | tail -40"

# list all the tabs so Claude knows who it can talk to
$ ssh cs "tmux list-windows -t fleet"
```

Put that together and your laptop Claude becomes the **dispatcher**: it can hand project A's tab a task, check project B's tab's progress, and report back to you in one conversation. The fleet keeps grinding around the clock; the local Claude is

how you steer all of it from wherever you are.

TWO HONEST CAUTIONS

Typing into a live agent is powerful, so aim carefully. Have your local Claude read the tab first (`capture-pane`) before sending keys, so it does not interrupt something mid-task or answer the wrong prompt.

Watch the server load. Every live Claude tab uses memory. If the box gets slow or freezes, run fewer tabs, add swap, or upgrade the droplet (part one covers sizing).

04 Gear up your Claude

paste this once

You do not have to remember any of the above. Paste this block into your local Claude Code once, and better yet save it in your `CLAUDE.md` so every future session already knows it:

paste into Claude Code (then save to CLAUDE.md)

```
My cloud server is reachable as "cs" (ssh cs). Rules for working with it:
1. Use one-shot non-interactive commands only: ssh -o ConnectTimeout=25 cs 'command'.
   Never open an interactive session. Use full paths; chain with && when needed.
2. Use scp to move files between machines. Use heredocs (ssh cs 'bash -s' <<'EOF' ... EOF)
   for multi-line scripts.
3. My persistent Claude agents live in tmux session "fleet", one window per project.
   To message one: ssh cs "tmux send-keys -t fleet:NAME 'message' Enter"
   To read one:    ssh cs "tmux capture-pane -p -t fleet:NAME | tail -40"
   Always capture-pane BEFORE send-keys so you never interrupt mid-task.
4. Always tell me which machine each command ran on: my Mac, or the server.
5. Never run destructive commands (rm -rf, reboots) on the server without asking me first.
```

From then on you just talk outcomes: "check the server logs for errors since yesterday," "tell the myapp agent to ship the fix and confirm it deployed," "copy last night's results to my Downloads." Your Claude handles the machinery.

THE COMPOUNDING PART

This pattern is how the whole system levels up: the local Claude can maintain the server, and the server agents keep working when your laptop is closed. Each side covers the other's blind spot. Set it up once, and every future project inherits it.