

Seven Automation Patterns

There are fifteen automations running on my laptop right now. I didn't build fifteen different things. They are all the same seven shapes, and here is the prompt for each one.

By Mabon ap Gwyn Built with Claude Code maponos.co.uk

15	7	1	<10¢
JOB	PAT	LAP	CHEAPEST ONE, PER
S RUNNING	TTERNS	TOP	MONTH

BEFORE YOU START

Every automation on my machine is one of seven shapes, or two of them stuck together. Once you can name the shape, the building part takes an evening.

This guide is written for Claude Code on a Mac. If you have not got it yet, step zero below takes about two minutes. The prompts then work as they are written: open Claude Code in an empty folder, paste one in, fill in the angle brackets, and answer the questions it asks you. I have left the prompts deliberately conversational, because the useful part of building this way is the arguing, not the typing.

Three rules hold for all seven, and they are the difference between an automation that runs and an automation you *believe* is running.

- **It has to run when you are not there.** Working in your terminal proves nothing. The scheduler starts your job with almost no environment, so a tool that lives in /opt/homebrew/bin is simply missing.
- **It has to write to a log file.** A job with no log cannot be debugged and cannot be trusted.
- **Something has to watch it.** That is pattern seven, and it is the one everybody skips.

STEP ZERO: GETTING CLAUDE CODE

Skip this if you already have it.

Claude Code is Anthropic's coding tool, and it runs in your terminal, which on a Mac is the Terminal app you already have. It reads and writes files in whichever folder you start it in, and nowhere else. That is why every prompt in this guide begins by naming a folder.

Install it with either of these. The first needs nothing in place beforehand. The second is for you if you already have Node 18 or newer.

```
IN YOUR TERMINAL

curl -fsSL https://claude.ai/install.sh | bash

# or, if you already have Node installed
npm install -g @anthropic-ai/claude-code
```

Now make a folder for your first automation and start it in there. It will ask you to sign in the first time, using the same Claude account you already have. Anthropic charge for this either through a Claude subscription or by usage, so check which you are on before you leave anything running overnight.

```
IN YOUR TERMINAL

mkdir -p ~/automations/my-first-one
cd ~/automations/my-first-one
claude
```

One folder per automation is the habit worth forming on day one. It keeps each job's script, its schedule file and its logs in the same place, which is exactly what you will want at two in the morning when one of them is misbehaving.

One more thing to know before the patterns, because two of them lean on it. Running `claude -p "your question"` gives you the same tool with no conversation around it: it answers once, prints the answer, and exits. That is how a script running at three in the morning, with nobody watching, gets an opinion out of an AI in the middle of doing something else.

THE SEVEN, AT A GLANCE

#	PATTERN	RUNS	THE ONE I RUN
1	Run X on a schedule	daily 08:00	Cloud spend dashboard, rebuilt every morning
2	Read my inbox and pull out the useful stuff	every 30 min	Flat hunt: scores property alerts out of 100
3	Leave a message for your laptop to collect	hourly + at login	Ideas typed on my phone, landing on my laptop
4	Watch a file and react when it changes	always on	A call starts, the note taker starts itself

#	PATTERN	RUNS	THE ONE I RUN
5	Check for work, do the work	every 2 min	My CRM queues a job, my laptop runs it
6	Take a weekly snapshot for your own record	Mondays 06:10	Who a company is hiring, kept week by week
7	Put the answer where I'll actually see it	always on	A menu bar icon showing all fifteen at once

1 Run X on a schedule

The plainest one. A clock goes off, a script runs, you read the result later.

REACH FOR IT

When you are doing the same thing at the same time every week, and the inputs are somewhere a script can reach. If you have ever said "I really should check that more often", this is the pattern.

THE ONE I RUN

`com.mab.aws-cost-monitor / daily 08:00`

Every morning at eight, one script asks every cloud account of mine what it spent yesterday and rebuilds a single page I can look at over coffee. Before it existed I found out what I was spending when the bill arrived, which is the worst possible moment to find out.

THE PROMPT

PASTE INTO CLAUDE CODE

I want a job on this Mac that runs every morning at 08:00.

What it should do: <one sentence, plainly>

Build it as a single script in ~/automations/<name>/, then:

- write the launchd plist for it, with an explicit PATH in EnvironmentVariables (/opt/homebrew/bin:/usr/local/bin:/usr/bin:/bin)
- send stdout and stderr to logs/ in the same folder
- install it with launchctl bootstrap
- start it once with
launchctl kickstart -k gui/\$(id -u)/<label>
and show me the log

That last step matters. I don't want to know it works in my shell, I want to know it works the way the scheduler will actually run it.

WHERE IT BITES

My off machine backup ran for a month writing "nothing to do" and I never looked. The first time it had something to commit, it failed, because the scheduler handed it a bare PATH and the signing tool it needed was not on it. A month of no backups, on a machine with sixteen automations. Ask for that kickstart line every single time.

2 Read my inbox and pull out the useful stuff

Something already emails you. Let the machine read it first and only pass on what matters.

REACH FOR IT

When the information you want is already arriving and the problem is volume, not access. This is also the honest way to automate a website that does not want to be scraped: turn on its own email alerts and read those instead.

THE ONE I RUN

```
com.mab.flatfinder / every 30 minutes
```

My tenancy ends in October, so the property portals email me every new listing. A job reads that inbox, hands each listing to a small AI model with my criteria written in plain English, and scores it out of 100. I get an email when three have scored over 70, or immediately if one scores over 90.

The criteria live in a separate file in ordinary sentences, so I can change my mind about what I want without touching any code. That is the part worth copying.

THE PROMPT

PASTE INTO CLAUDE CODE

```
Read my email over IMAP, read only, and look at anything new
from <sender> since the last run.
```

```
For each message, pull out <the fields I care about>, then send that
text to 'claude -p' asking it to score the item 0 to 100 and return
JSON with a score and a one line reason.
```

```
Keep my criteria in criteria.json as plain English sentences, so I can
edit them without touching code. Mark some of them as hard rules that
cap the score at 20 when they fail.
```

```
Keep seen.json of message ids so nothing is handled twice.
Only tell me when three items have scored over 70, or one scores
over 90. Show me what a scored item looks like before you build
the alerting.
```

WHERE IT BITES

The first version scored every listing as unreadable and I nearly binned the whole thing. The portal wraps every link in a click tracker about 700 characters long, so the text I was sending the model was almost entirely tracking gibberish with a flat buried in it. Print the exact text you are about to send the model. Do that before you touch anything else.

3 Leave a message for your laptop to collect

Something drops a note in a tray. Your laptop empties the tray whenever it is next awake.

REACH FOR IT

Any time the thing that starts the work and the thing that does the work are not in the same place, or not awake at the same time. A laptop that sleeps all night is not a problem here, it just catches up in the morning.

THE ONE I RUN

`com.mab.wiki-idea-queue / hourly and at login`

I have a password protected page bookmarked on my phone with three boxes: an idea, a todo, a thing to buy. Whatever I type sits on a queue. Whenever my laptop is awake, it empties that queue, files each item in the right place, and for ideas it asks AI to flesh the one liner out into something I can actually react to later.

It costs pennies a month, because a queue that is empty most of the time is nearly free.

THE PROMPT

PASTE INTO CLAUDE CODE

I want to type something on my phone and have it land on my laptop.

Build three pieces:

- a password protected web page with a text box
- a small endpoint that puts whatever I type on a queue
- a script on my laptop that empties the queue hourly and at login, writes each item to a file, then deletes it from the queue

Rules:

- the password lives in a secret store, never in the page source
- the laptop's credentials can read and delete on this one queue and nothing else
- tell me the monthly cost before you build anything

Start by telling me the cheapest way to do this, and why you would not do it the other ways.

WHERE IT BITES

Delete the message only after the file is safely written, never before. And if your script gets compiled before it runs, remember it runs from a different folder than the one you wrote it in. Mine worked perfectly and filed everything one directory above where I meant.

4 Watch a file and react when it changes

Most apps say what they are doing in a log file. You can listen to that without asking anyone's permission.

REACH FOR IT

When an app you use has no integration, no API and no plugin, but obviously knows when the thing happened. It almost always writes it down somewhere.

THE ONE I RUN

```
com.mabon.discord-notetaker / always on
```

I kept forgetting to start the note taker before client calls. Discord writes a line to its own log the moment a call connects, so now a job sits watching that file, and starting the call starts the recording. I have not thought about it since.

THE PROMPT

PASTE INTO CLAUDE CODE

```
<app> writes a log file somewhere on this Mac. When <thing> happens, I think it writes a line about it.
```

```
First: find the log file and show me the exact line that marks <thing> starting. Don't write any code until I have confirmed you found the right one.
```

```
Then build a job that follows that file and runs <command> when the line appears. It has to:
```

- follow the end of the file, not re-read it
- survive the app restarting and the file being rotated
- refuse to fire twice for the same event
- log what it saw and what it did

WHERE IT BITES

Log formats change when the app updates, quietly, and your automation simply stops. Make it shout when it has seen nothing for longer than usual, rather than assuming silence means nothing happened.

5 Check for work, do the work

Something out in the world writes down a job. Your machine asks "anything for me?" and gets on with it.

REACH FOR IT

When a button somewhere needs to trigger something that can only run on your machine, because that is where the tool, the licence or the login lives.

THE ONE I RUN

```
com.mab.maponos-jobs / every 2 minutes
```

I built my own CRM rather than pay \$297 a month for one. It runs in the cloud, but some of the work it triggers can only happen on my laptop. So the dashboard writes a job, and every two minutes my laptop asks whether there is one, claims it, runs it, and writes the answer back.

I press a button in a browser. Two minutes later something happens on a machine in my flat. Nothing clever, and it has never once needed fixing.

THE PROMPT

PASTE INTO CLAUDE CODE

```
My <app> needs to trigger work that can only run on my laptop.
```

```
Add a jobs list to it: type, arguments, status, result.
```

```
Then build a runner on this machine that every 2 minutes:
```

- asks for the oldest queued job
- claims it, so a second copy of the runner can't grab it too
- runs the matching local script
- writes back the result, or the error, and moves on

```
One job at a time. If it fails, mark it failed with the error text.  
Do not retry forever. Log every job it picks up, including the ones  
it decides to ignore.
```

WHERE IT BITES

Every two minutes is plenty. Every ten seconds is how you discover what somebody's rate limit is. And claim the job before you start it, or the day your laptop is slow you will run the same job twice.

6 Take a weekly snapshot for your own record

Fetch the same page every week and never overwrite the old one. The archive is the product.

REACH FOR IT

When the thing you want to know is not in today's page, it is in the difference between today's page and last month's. Nobody can catch up with you on this later, which is what makes it worth starting on a Monday you do not feel like it.

THE ONE I RUN

`com.maponos.quantum-snapshot / Mondays 06:10`

For one client I take a picture every Monday of who is on a set of company team pages and what jobs they have advertised. Any one week of that is worthless. Six months of it tells you which companies are quietly growing, which ones lost half a team, and who is hiring for a problem they are about to need help with.

THE PROMPT

PASTE INTO CLAUDE CODE

Every Monday at 06:10, fetch <page or list> and save the result to snapshots/YYYY-MM-DD.json.

Never overwrite or tidy up an old snapshot. They are the point.

Then compare this week against the most recent previous snapshot and write a short summary of what is new, what has gone, and what has changed. Email it to me.

Save the first snapshot now, so next Monday has something to compare itself to. Check the site's terms first and tell me if what I have asked for is not allowed.

WHERE IT BITES

The temptation is to store only the latest version and a tidy summary. Resist it. You will want to ask a question in March that you had not thought of in January, and only the raw snapshots can answer it. Check what you are allowed to collect before you start collecting it.

7 Put the answer where I'll actually see it

An automation nobody can see is a rumour. This is the one that turns fourteen working jobs into a system you trust.

REACH FOR IT

The moment you have more than about three of these. Not later. The cost of skipping it is not that things break, it is that you stop knowing whether they are working, which slowly turns the whole thing back into something you have to check by hand.

THE ONE I RUN

`com.mab.automation-hub / always on`

A small icon in my menu bar and a page on my own laptop. Every job as a card: what it is, when it runs, whether it is loaded, when it last wrote to its log, and buttons to turn it off, run it now, or read the log. There is also a button that opens Claude Code already loaded with everything about that one job, for when it misbehaves.

The important decision: it works out what exists by asking the machine, every time you look. It does not keep its own list. A list would have been easier and it would have started lying the first time I added something without telling it.

THE PROMPT

PASTE INTO CLAUDE CODE

I have several scheduled jobs on this Mac. Build me a menu bar app and a local web page showing all of them at once.

For each one: name, schedule, whether it is currently loaded, when it last wrote to its log, and whether that is longer ago than it should be for its schedule. Turn it off, run it now, tail the log.

Work out what exists by reading the machine on every refresh. Do not keep your own list of what should exist and trust that. A dashboard that can be wrong is worse than no dashboard, because it turns "I should check" into "I already checked".

Anything you find that you don't recognise, show it anyway, under "unregistered".

WHERE IT BITES

I built this one last, and it immediately told me three of my automations had been failing for weeks, including the one that was supposed to be backing everything up. Nothing was watching the watchers. Build this when you have three jobs, not fifteen.

WHAT TO BUILD FIRST

Pick the thing you do every Monday morning and resent doing by Monday lunchtime. It is almost always pattern one or pattern two.

Build that one this week. Build pattern seven the week you have three of them. Everything else on my machine is just those seven shapes wearing different clothes, and I have never once needed an eighth.

One last thing, since it is the most common way this goes wrong: the AI part is usually the cheapest and easiest bit. The hard parts are messy input, knowing what happens when the thing is wrong, and noticing when it has quietly stopped. Spend your evening there.

Mabon ap Gwyn

I build these for businesses at maponos.co.uk

Found this useful, or got one running? Tell me what it does, I collect them.