

Article

Oct 2026

Automation best practices: 5 ways to build scripts that keep working

Automation should give you time back. You stop repeating the same steps, and your team can focus on the work that matters.



Megan Lansell
Software Consultant
Email megan.lansell@waterstons.com

When an automation goes wrong though, you're left with a backlog of unfinished tasks, no clear idea what failed and no quick way to put it right.

The good news is that most of that pain comes from a handful of avoidable habits. Here are the five foundations we'd put in place first.

1. Keep scripts small and safe to re-run

Take a new starter process. It's tempting to put every step into one script, but if it breaks halfway, you can't simply run it again because the user already exists. Everything after the failure is left undone, and someone has to untangle it by hand.

Scripts should be safe to re-run. Developers call this idempotent: run it twice and you get the same result. The easiest way to get there is to split the process into small, self-contained steps. For new starters, that could be separate scripts to create the user, set licensing and raise hardware requests. If one fails, you re-run that step and nothing else.

2. Handle dependencies before they bite

Some steps rely on earlier ones having finished. For example, you create a new starter and then try to add their permissions. Systems often take time to sync, so the second step can't always find the user straight away, and the script fails intermittently. Those are the failures that are hardest to track down.

Retries can help, but on their own they can mask the problem and lead to timeouts. A more reliable approach is to split the work into two scripts and schedule the second to run once your systems have had time to sync.

3. Plan for concurrent runs

Sooner or later, your automation will be triggered while it's already running. Without safeguards, one run succeeds and the other fails with errors such as "resource locked" or "file in use", and someone has to investigate.

Design for concurrency from the start. For example, if your new starter script updates a file that only one process can open at a time, two requests arriving together will clash. If the trigger is infrequent, check for a running job before starting. If it's frequent, use a queue so each job runs when the resource is free.

4. Test safely and track every change

Your business will change, and your automations will need to change with it. Treat those changes like any other code: put them under version control so you can see what changed and roll back quickly if something breaks.

Test environments are harder to set up for automations because they often touch several systems, so they're often skipped. That's a false economy, because fixing a problem across multiple live systems takes far longer than building a sandbox. Aim for one that mirrors your live setup as closely as you can. Even a partial one beats testing in production.

5. Log everything and monitor failures

Every script will fail at some point, so the aim is to make it easy to find and fix. Without monitoring, issues can go unnoticed until a significant backlog has built up.

Monitoring job failures means you catch problems early, before they escalate. Logging shows you exactly where a script failed and why, which makes debugging much faster. Without it, nobody can see what went wrong, and resolution drags on.

Get the foundations right

Get these five foundations right and your automations will stay reliable as they grow:

1. Keep scripts small and safe to re-run.
2. Handle dependencies before they bite.
3. Plan for concurrent runs.
4. Test safely and track every change.
5. Log everything and monitor failures.

Taking the time to get this right saves far more time than it costs.

Want a second pair of eyes on your automations? Whether you're starting out or untangling something that's got out of hand, we're happy to help. Get in touch for practical advice on automation in your business.

