# Implement three practical production processes.

By the end of this lesson, you will have three routines running in your own account, each performing actual work on your real code or data.

## This is a lesson you will learn when deploying a real product!

Seven lessons on theory and examples. Now it's time to get started. By the end of this lesson, you will have three routines running in your own account, each performing a real task on your code or real data.

This isn't a demo. This isn't a prompt example. These are actual production routines that you'll keep.

## A cautionary tale

This is a real post from April 14th, the day the Routines feature was launched. A programmer set up a routine to run overnight to analyze his codebase. They went to sleep. When they woke up, the routine had triggered, falling into an infinite loop of context expansion and exceeding the entire daily usage limit. The notification he received upon waking up was:

*Woke up. Checked email. Automation failed. Usage limit reached, reset at 7 PM.*

This happened to many people in the first 24 hours. Common points: No word limit, no initial state processing, and no testing run before activation.

Don't become that kind of person. Everything in this lesson is designed to prevent that outcome.

## Routine 1: Document change detection (scheduled)

**Objective** : Once a week, scan your code and documentation for areas where the code has changed but the documentation hasn't. Post a short report to Slack or your group chat.

### Configuration

1. **Trigger** : Schedule - every Monday at 9 AM
2. **Connectors** : GitHub (reads your repository), Slack (posts reports)
3. **Other connectors** : OFF

### Centuries

```
B?n là ng??i phát hi?n s? sai l?ch tài li?u cho d? án c?a chúng tôi. M?i th
? Hai, hãy ki?m tra xem tài li?u c?a chúng ta có b? l?ch so v?
i code hay không. Hãy làm nh? sau: 1. L?y danh sách các file ?ã thay ??
i trong 7 ngày qua trên nhánh chính. 2. ??i v?i m?i file ?ã thay ??
i, hãy ki?m tra xem có file tài li?u t??ng ?ng nào ?? c?p ??
n API public, hành vi ho?c c?u hình c?a nó hay không. 3. ?ánh d?u b?t k? tr
??ng h?p nào mà code ?ã thay ??i nh?ng tài li?u thì không. ??nh d?ng ??
u ra: ## Báo cáo sai l?ch tài li?u ({ngày}) ### Sai l?ch ???c tìm th?y: {s?
 l??ng} {??i v?i m?i tr??ng h?p sai l?ch:} - **File:** `{???ng d?
n}` - **Tài li?u b? ?nh h??ng:** `{???ng d?n tài li?u}` (ho?c "không tìm th
?y tài li?u phù h?p") - **Tóm t?t thay ??i:** {m?t câu} - **Hành ??ng c?
n thi?t:** {m?t câu} N?u không có sai l?ch, ch? xu?t ra: "Tài li?u ?ã ??
ng b?. Không c?n hành ??ng." Gi?i h?n c?ng: - T?i ?a 10 tr??ng h?p sai l?
ch ???c báo cáo - Không bao gi? v??t quá 400 t? - Không có bình lu?
n, không có l?i m? ??u - b?t ??u tr?c ti?p v?i tiêu ?? báo cáo ??ng toàn b?
 báo cáo lên kênh Slack #engineering.
```

### Test run checklist

Before activating the schedule:

1. Click on Run now
2. Verify that the Slack message has been received.
3. Verify that the report format matches the template.
4. Verify the word count is under 400.
5. Check the run history to see the number of tokens and duration.

If any of those steps fail, fix the prompt and run the test again. Do not allow the scheduler to be triggered until the test run is successful.

# Routine 2: Bot evaluates PR (GitHub events)

**Objective** : When a PR is opened to the main branch with the label needing review, Claude will post a structured review comment. One session per PR, so subsequent commits will build on the previous response.

### Configuration

1. **Trigger** : GitHub event — Pull request ? opened + synchronized
2. **Repository** : your storage
3. **Base branch filter:** `main`
4. **Label steel:** `needs-review`
5. **Author exclude:** `dependabot[bot]`,`renovate[bot]`
6. **Paths** : only the paths to your code (ignore documentation and lock files).
7. **Session** : the same session for each PR
8. **Connectors** : GitHub only

### Centuries

Use the PR evaluation prompt from Lesson 3. Copy it precisely into the routine form. The filtering principle + the clear "do not push to main" rule + detection limits are what make this routine safe.

### Test

1. Launch a small test PR campaign with `main`labels .`needs-review`
2. Observe the routine being executed - confirm within 1-2 minutes.
3. Verify that the review/rating matches the format.
4. Add a new commit to the PR; verify that the subsequent review is contextually appropriate.
5. Remove the label `needs-review`and add a new commit; verify the routine is NOT executed.

If it's executed without a label, your filter isn't working. Don't proceed until it's working correctly.

## Routine 3: Customizable

Choose a routine that suits the actual work you're doing. Here are five strong starting points, all based on the "Use Examples" section in the launch guide:

### Option A: Sentry / Error Classification

1. **Trigger** : API (Your Sentry or alert system sends POST events to a URL regularly)
2. **Connectors** : GitHub (to check for patches), Linear (to create request tickets)
3. **Prompt** : Classify the error, determine its severity, check for related issues, create a new request ticket if none, and link it back to the error event.

### Option C: Security Scanner

1. **Trigger** : Schedule weekly OR push to GitHub for `package.json`/ `requirements.txt`/ `go.mod`/ `Cargo.toml`
2. **Connectors** : GitHub
3. **Prompt** : Check dependencies for known CVEs, flag any findings as high/critical, and open a PR with upgrade suggestions.

### Option D: Incident Log Analyzer

1. **Trigger** : API (your monitoring system triggers when the alert threshold is exceeded)
2. **Connectors** : GitHub (to link to recent commits), Slack (to post summaries)
3. **Prompt** : Read the log excerpt, identify the possible cause, and list the 3 most suspicious recent commits. Post it to the Slack incident channel.

### Option E: Automated customer support responses (Draft only)

1. **Trigger** : API (your platform's supported API sends a new request)
2. **Connectors** : Support system (read-only + create drafts - NEVER send)
3. **Prompt** : Categorize the request, find relevant KB articles, draft the response, and save it as a draft for real review.

### Option F: Check database migration

1. **Trigger** : Push to GitHub on migrations/**
2. **Connectors** : GitHub
3. **Prompt** : Review the migration process to ensure safety - missing downshifts, non-immutable operations, locking mechanisms on large tables. Comment on the findings in the PR.

### Choose an option and build it.

Following the same principles as routines 1 and 2:

1. Filter aggressively
2. A prompt with range queries, fixed formatting, hard limits, and empty state handling.
3. Only the connectors you actually need.
4. Run a test before activating the schedule or trigger.

## First week's notebook

Now you have 3 routines running. Here are the things to do in week 1:

**Day 1 (today)**

1. Test each routine.
2. Activate all three
3. Check the run history from the evening - how many runs were performed, and were there any errors?

**Days 2-3**

1. Consider the output quality of each routine.
2. Adjust the prompts that are producing abnormal output.
3. Tighten the filter if unexpected runs occur.

**Days 4-5**

1. Check your quota usage – are you adhering to your daily limit?
2. Check token costs - are any routines significantly more expensive than expected?
3. Remove unused connectors if you accidentally left them activated.

**Days 6-7**

1. Reflect: Does each routine actually save you time?
2. Eliminate any ineffective routines.
3. Focus on the routine that brings the most success.

Most routines require 3-5 repetitions in the first week before they "complete." That's not a failure—it's the natural format of the process. Plan for that.

## Things you can do right now

You can:

1. Configure any of the three types of triggers (schedules, GitHub events, APIs) in a disciplined manner.
2. Write production-level prompts with queries that have fixed scope, formatting, hard limits, and handle empty state.
3. Limit MCP connectors to the minimum OAuth privilege level.
4. Avoid top cost leakage errors (long-winded prompts, unfiltered triggers, bloated connector lists).
5. Decide when Routines is the right tool compared to GitHub Actions, Zapier, or cron.
6. Implement production routines with trial and iterative workflows.

You have built three actual running routines in your account. You have learned the discipline of distinguishing between scalable routines and those that exceed your limits.

Your certification is attached to your account. Pin it to LinkedIn or keep it private – either way, you now know something most engineers don't: How to run your own cloud-based AI agents to get real work done.

## Next step

Now that Routines are a part of your system, let's delve deeper:

1. Mastering Claude Code - the interactive aspect of Claude Code, where you still spend most of your programming time.
2. Gain a deeper understanding of AI Agent - multi-agent orchestration, a model that goes beyond single routines.
3. Automated workflows - connect Routines with the rest of your tools to achieve cumulative results.

The automation system of 2026 isn't "Routines OR GitHub Actions OR Zapier." It's all of them, chosen based on the format of each task. Now you have the judgment to make that choice.

## Key points to remember

1. Implement three practical routines, not three perfect routines.
2. Run a test before activating - always do so.
3. Monitor the run history on the first day; repeat the prompts on days 2-3; make edits on days 4-5.
4. Most routines require 3-5 iterations before they stabilize – plan accordingly.
5. You have learned the core principles: Filtering, formatting, limiting, initial state, and range.

1. Question 1:

One of your new routines is producing strange results on its first real-world run. What should you do?

1. A. Disable the routine immediately.
2. B. Blame Claude and continue
3. C. Switching to a different automation tool – a common misconception that overlooks how this actually works.
4. D. Open the run history, view the input and output, adjust the prompt, and run a test to verify the fix.

EXPLAIN:

Each routine prompt will need at least one iteration after its initial exposure to real data. That's not an error—it's the expected workflow. The run history shows you the exact inputs and outputs; use that to iterate.

2. Question 2:

After implementing a new routine, what is the FIRST thing you should check in the first 24 hours?

1. A. Running history and number of runs exceeding the limit used.
2. B. Will the output win a design award?
3. C. Will your colleagues like it?
4. D. Trend in the number of tokens over a month

EXPLAIN:

During the first 24 hours, monitor the run history. Is it running as expected? Are the filters working? Is each run within budget? Detecting quota leaks from day one will save you a whole week of wasted trial runs.

3. Question 3:

What is the right attitude to have when implementing initial production processes?

1. A. Build them perfectly before activating the schedule.
2. B. Let Claude write the entire process himself without supervision.
3. C. Copying someone else's prompt exactly without editing – a common mistake leading to suboptimal results.
4. D. Deploy drafts, run them for a week, repeat - perfection is impossible before the actual run.

EXPLAIN:

Routines will improve significantly in the first week of actual running. Don't overcomplicate the initial design. Implement, observe, and iterate. The run history is your feedback loop.

Submit your work

## Training results

You have completed **0** questions.

-- / --

Review the lesson

You finished reading the article "**Implement three practical production processes.**" edited by the **TipsMake** team. We hope this article has provided you with many useful tech tips and tricks. You can search for similar articles on tips and guides. Thank you for reading and for following us regularly.

© 2019 TipsMake.com