Coding & Development / Documentation

Turn tribal knowledge (Slack threads, incident reviews) into structured runbooks: triggers, symptoms, diagnosis, remediation.
Difficulty: Advanced
Model: GPT-4 / Claude / Gemini
Use Case: On-Call Engineering, Incident Response, SRE
Updated: May 2026
Why This Prompt Exists
When the pager goes off at 3 AM, there's no time to search through Slack for "that one time this happened before."

You get:

- the same incident happening multiple times because no one documented the fix
- on-call engineers waking up senior devs for known issues
- incident post-mortems that no one reads
- tribal knowledge walking out the door when people leave
- MTTR (mean time to repair) staying high for repeat incidents

But great runbooks save lives at 3 AM:

- triggers: what alert or symptom starts this runbook
- symptoms: how to confirm this is the right runbook
- diagnosis: commands to run to understand severity
- remediation: step-by-step fix, with verification
- escalation: when to call for help

Without runbooks, every incident is a fire drill.

This prompt converts incident notes, Slack threads, and post-mortems into structured runbooks.

The Prompt

```
Assume the role of an SRE who writes runbooks for on-call engineers.

Your task is to convert incident knowledge into a structured runbook.

Generate:

1. RUNBOOK METADATA
   - Incident type (e.g., "Database connection pool exhaustion")
   - Severity (SEV-1 / SEV-2 / SEV-3)
   - Average time to fix (from historical data)

2. TRIGGERS
   - Alert name or condition
   - Example alert message

3. SYMPTOMS (how to confirm)
   - User-visible impact
   - Dashboard signals
   - Log patterns to grep

4. INITIAL DIAGNOSIS (ordered commands)
   - Command 1 to check health
   - Command 2 to check recent changes
   - Expected output vs. bad output
```

5. REMEDIATION STEPS (ordered, each with verification)
   - Step 1: immediate mitigation
   - Step 2: permanent fix
   - Step 3: verify fix worked

6. ESCALATION
   - Who to contact if steps fail
   - What information to bring them

7. POST-FIX ACTIONS
   - Data to capture for post-mortem
   - Commands to run to restore full service

INPUTS:

Incident notes / Slack threads / post-mortems:
[PASTE KNOWN INCIDENT DOCUMENTATION OR DESCRIBE THE PROBLEM]

Service/System affected:
[E.G., "Payment API", "Auth Service", "Main Database"]

Common fix (if known):
[E.G., "Restart the sidekiq workers"]

Tools available:
[DATADOG / GRAFANA / CLOUDWATCH / CUSTOM DASHBOARD / OTHER]

RULES:
- Every command must be copy-paste ready

```
- Assume the on-call engineer is tired and stressed
- Include expected output for every diagnostic command
- Flag any destructive commands with "⚠ DESTRUCTIVE"
- Include a "do not" section for common mistakes
```

How To Use It

- Write runbooks immediately after an incident — while memory is fresh.
- Run this prompt on your incident post-mortems to extract structured runbooks.
- Store runbooks where on-call engineers can find them quickly (not buried in Confluence).
- Review and update runbooks quarterly — things change.
- Use the “simulate an incident” game day to test runbooks before they’re needed.

Example Input

Incident notes / Slack threads / post-mortems:
“We had an incident where the payment API was timing out. Turned out the database connection pool was set to 10, but we had 20 pods. Each pod took 2 connections, so 20 pods tried to take 40 connections but only 10 available. Restarting didn’t help because they’d just fight again. Fix was increasing pool to 50. Symptoms: logs showed ‘timeout waiting for connection’ and payment failures. Alert was ‘Payment API 5xx > 5% for 2 minutes.'”

Service/System affected:
Payment API

Common fix (if known):
Increase DB connection pool size

Tools available:
DataDog, AWS RDS Console

Why It Works

Most incident documentation is narrative — great for learning, terrible for 3 AM emergencies.

This framework improves outcomes by forcing:

- triggers (so you pick the right runbook)
- symptoms (so you confirm before acting)
- diagnosis commands (so you don't guess)
- remediation steps with verification (so you know when you're done)
- escalation paths (so you don't waste time)

Great runbooks don't tell stories — they give commands.

Build Better AI Systems

Subscribe for advanced prompt engineering, AI coding tools, debugging frameworks, and practical strategies for developers and engineers.

Carefully engineered prompts for people doing real work.

Share this:

- Share on Facebook (Opens in new window) Facebook
- Share on X (Opens in new window) X

See also Architecture Decision Recorder