Coding & Development / Documentation

Audit existing READMEs for missing sections, then rewrite — onboarding friction killer.
Difficulty: Beginner
Model: GPT-4 / Claude / Gemini
Use Case: Project Onboarding, Open Source Docs, Internal Tools
Updated: May 2026
Why This Prompt Exists
A bad README wastes hours of every new contributor’s time.

You get:

- “how do I set this up?” questions in Slack every week
- missing environment variables that cause cryptic errors
- outdated setup commands that no longer work
- no troubleshooting section for common problems
- developers giving up and using something else

But great READMEs follow a checklist:

- what this project does (one sentence)
- prerequisites (Node version, Docker, API keys)
- installation (step by step, copy-paste ready)
- configuration (all env vars documented)
- running (dev, test, prod commands)
- troubleshooting (top 3 problems)

Without a checklist, READMEs rot.

This prompt audits and rewrites READMEs against a complete standards checklist.

The Prompt

```
Assume the role of a developer onboarding specialist who fixes broken
README s.

Your task is to audit and rewrite a README file.

Generate:

1. AUDIT REPORT
   - Missing sections (list)
   - Outdated information (list with evidence)
   - Unclear instructions (list)

2. REWRITTEN README
   - Project title and one-line description
   - Prerequisites (with version constraints)
   - Installation (copy-paste commands)
   - Configuration (all environment variables)
   - Running locally (dev server command)
   - Running tests
   - Troubleshooting (top 3 common issues)
   - Contributing (link or brief guidelines)

INPUTS:

Current README:
[PASTE EXISTING README CONTENT OR WRITE "NONE"]
```

```
Project Type:
[WEB_APP / CLI_TOOL / LIBRARY / API / OTHER]

Primary Language/Runtime:
[NODEJS / PYTHON / GO / RUST / OTHER]

Known pain points from team (optional):
[E.G., "Everyone struggles with setting up the database"]

RULES:
- Preserve any unique or valuable content from original
- Flag any assumptions that may not be true
- Commands must be runnable as written
- Use code fences for all commands
- Add a "first time setup" section if the project is complex
```

How To Use It

- Run this on every new project before sharing it externally.
- Update the README quarterly — run the prompt again with the current README as input.
- Ask a new team member to test the generated README before merging.
- Keep the troubleshooting section alive by adding real issues your team encounters.
- Link to the README from your issue templates (“Did you check the README troubleshooting?”)

Example Input

Current README:
“My cool project. Run npm start. You need a database probably.”

Project Type:
WEB_APP

Primary Language/Runtime:
NODEJS

Known pain points from team (optional):
“Postgres setup is never documented. New people take 2 days to get running.”

Why It Works
Most READMEs are written by the person who knows the project best — which makes them blind to what newcomers need.

This framework improves outcomes by forcing:

- audit before rewrite (you see what’s missing)
- prerequisites section (no more “it doesn’t work” surprises)
- configuration table (env vars are never optional)
- troubleshooting (acknowledges things will break)
- runnable commands (not pseudocode)

Great READMEs don’t assume — they guide.

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