Coding & Development / Debugging

Convert cryptic errors into precise fixes — without guessing or rabbit holes.
Difficulty: Intermediate
Model: GPT-4 / Claude / Gemini
Use Case: Error Resolution, Stack Trace Analysis, Root Cause Diagnosis
Updated: May 2026
Why This Prompt Exists
Most developers (and AI users) ask “how do I fix this error?” and get generic advice: check indentation, reinstall packages, restart the server.

That wastes hours. Real debugging requires hypothesis-driven investigation.

You get:

- vague, non-actionable suggestions
- the same “check your syntax” advice repeated
- rabbit holes that don’t address the root cause
- fixes that introduce new bugs
- hours lost on trial and error

But debugging is not magic.

It follows predictable logic:

- reproduce the context → isolate variables → test hypotheses
- contradiction between expected and actual behavior points to root cause
- confirmation test before fix prevents guesswork
- minimal changes reduce regression risk

Without structure, debugging becomes random.

This prompt forces AI to act like a senior debugger — structured, testable, precise.

The Prompt

```
Assume the role of a senior debugging engineer who finds root causes,
not symptoms.

Your task is to analyze an error and provide a structured diagnosis.

Generate:

1. CONTRADICTION
   - One sentence restating the gap between expected and actual
behavior

2. ROOT CAUSES (ranked 1-5)
   - List each with reasoning based on error signature
   - #1 most likely must have direct evidence from inputs

3. CONFIRMATION TEST
   - Minimal, isolated test (code or command) to confirm #1 root cause
   - Do not propose a full fix yet

4. MINIMAL FIX
   - Exact code change (diff format or before/after)
   - No extra refactoring

5. RISKS
   - Data loss, security, race conditions, performance
```

```
INPUTS:

LANGUAGE/Framework:
[PASTE LANGUAGE AND FRAMEWORK]

Full Error Message:
[PASTE COMPLETE STACK TRACE OR ERROR TEXT]

Relevant Code Snippet:
[PASTE CODE WHERE ERROR OCCURS]

Expected Behavior:
[WHAT SHOULD HAPPEN]

Actual Behavior:
[WHAT ACTUALLY HAPPENS]

What I've Tried:
[LIST ATTEMPTED FIXES]

RULES:
- Never skip the confirmation test
- Rank root causes by likelihood, not convenience
- Flag dangerous side effects explicitly
- If error message is unfamiliar, explain which keywords matter for searching
- Suggest search queries when appropriate
```

How To Use It

- Copy the full stack trace — not just the last line. The prompt works best with exception type, line numbers, and context.
- Be honest about “what I’ve tried” — this prevents the AI from suggesting the same dead ends.
- Run the confirmation test first — don’t skip to the fix. That single step separates guesswork from diagnosis.
- Use for any language — Python, JavaScript, Go, Rust, PHP, even SQL errors. Fill the inputs precisely.
- Iterate — paste the confirmation test output back into a new run if the root cause changes.

Example Input

LANGUAGE/Framework: Python / FastAPI

Full Error Message:
sqlalchemy.exc.IntegrityError: (psycopg2.errors.NotNullViolation) null value in column “user_id” violates not-null constraint

Relevant Code Snippet:
@router.post(“/review”)
def add_review(review: ReviewCreate, db: Session = Depends(get_db)):
new_review = ReviewModel(rating=review.rating, comment=review.comment)
db.add(new_review)
db.commit()
return {“status”: “ok”}

Expected Behavior:
Review saved with user_id automatically from logged-in user session.

Actual Behavior:

IntegrityError: user_id cannot be null.

What I've Tried:
Checked that user is authenticated (prints user email in logs). Added print(type(review.user_id)) → None. Tried setting default=0 in DB schema.

Why It Works
Most debugging prompts produce generic suggestions and surface-level fixes.

This framework improves outcomes by forcing:

- contradiction statement (clarity)
- ranked root causes (prioritization)
- confirmation test (verification before action)
- minimal fix (precision, not scope creep)
- risk flagging (safety in production)

Great debugging doesn't guess — it forms hypotheses, tests them, and fixes exactly what's broken.

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 The Null/Undefined Error Fixer