Coding & Development / Documentation

Document breaking changes between versions with upgrade steps, deprecation notices, and codemod hints.
Difficulty: Advanced
Model: GPT-4 / Claude / Gemini
Use Case: Version Upgrades, Breaking Changes, Library Releases
Updated: May 2026
Why This Prompt Exists
Breaking changes without migration docs cause production outages and angry users.

You get:

- “my build broke after upgrading” tweets
- users stuck on old versions because migration is unclear
- support tickets asking “how do I migrate?”
- team members applying breaking changes incorrectly
- downtime from missed deprecation warnings

But great migrations are documented:

- what changed (before/after code examples)
- why it changed (rationale)
- upgrade steps (ordered, testable)
- deprecation timeline (when old behavior goes away)
- automated codemod if possible

Without migration guides, every upgrade is a crisis.

This prompt transforms changelogs and diffs into comprehensive migration guides.

The Prompt

```
Assume the role of a developer experience engineer who writes
migration guides.

Your task is to document breaking changes between two versions.

Generate:

1. VERSION INFO
   - From version: [X.X.X]
   - To version: [Y.Y.Y]

2. BREAKING CHANGES TABLE
   - Change type (API removed / behavior change / type change)
   - Old pattern (code example)
   - New pattern (code example)
   - Migration effort (Low / Medium / High)

3. UPGRADE STEPS (ordered)
   - Step 1: Update dependencies
   - Step 2: Run codemod (if available)
   - Step 3: Manual changes
   - Step 4: Test

4. DEPRECATION WARNINGS
   - What is deprecated but still works
   - Removal version
```

5. TESTING RECOMMENDATIONS
   - What to test after migration

6. ROLLBACK PLAN
   - How to revert if migration fails

INPUTS:

Changelog or release notes:
[PASTE CHANGELOG / RELEASE NOTES]

Old version:
[VERSION NUMBER]

New version:
[VERSION NUMBER]

Breaking changes list (if known):
[LIST OR "INFER FROM CHANGELOG"]

Language/Platform:
[JAVASCRIPT / PYTHON / RUBY / OTHER]

Codemod available?:
[YES / NO / LINK IF YES]

RULES:
- Every breaking change must have a before/after example
- Assume user has zero context about internal decisions

```
- Flag changes that require database migrations or data backfills
- Include search/replace patterns for common find operations
```

How To Use It

- Run this before releasing any major version — ideally during release planning, not after.
- Include a “tested on” section with example projects that successfully migrated.
- Link the migration guide in your upgrade error messages.
- Keep a running “deprecations” document between major releases.
- Ask a beta user to follow the guide before you publish it widely.

Example Input

Changelog or release notes:
“v3.0.0 breaking changes:
– Removed deprecated `setTimeout`-based API. Use `promiseDelay` instead.
– `fetchUser(id)` now throws `NotFoundError` instead of returning `null`.
– Renamed `map` to `transform` in the data pipeline.”

Old version:
2.5.0

New version:
3.0.0

Language/Platform:
TYPESCRIPT

Codemod available?:
NO

Why It Works

Most breaking change docs list what changed but not how to fix it.

This framework improves outcomes by forcing:

- before/after code examples (visual, actionable)
- ordered upgrade steps (reduces missing steps)
- effort estimation (sets expectations)
- rollback plan (builds confidence)
- testing recommendations (prevents silent failures)

Great migration guides don’t just announce breakage — they provide a path forward.

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 Runbook / Playbook Builder