AI Development Workflow

Based on Zevi Arnowitz's Workflow

A systematic approach to building software with AI

The Pipeline:

/create-issue --> /explore --> /create-plan --> /execute
|
v
/review --> /peer-review --> /document

/learning-opportunity (use anytime)

The Development Pipeline

This workflow separates thinking from doing. Each phase has a specific purpose:

1. /create-issue	Capture bugs/features quickly while mid-development
2. /explore	Understand the problem before writing any code
3. /create-plan	Design the implementation with status tracking
4. /execute	Build according to the plan
5. /review	Have Claude review its own code
6. /peer-review	Cross-check with other AI models
7. /document	Update documentation after changes
8. /learning-opportunity	Shift into teaching mode when confused

"This is the big difference between vibe coding and building serious apps. I spend a lot of time going back and forth and understanding." - Zevi

Command Details

1. /create-issue - Capture Ideas Fast

When to use:

You're mid-development and think of a bug, feature, or improvement you don't want to work on right now.

How it works:

```
/create-issue
> I want to add dark mode toggle to the settings page
```

Claude asks 2-3 quick questions, then creates a complete issue with title, TL;DR, current vs expected state, relevant files, and priority labels. Takes ~2 minutes max.

2. /explore - Understand Before Building

When to use:

You're ready to work on an issue and need to deeply understand the problem.

How it works:

```
/explore
> [describe the feature or paste the issue]
```

Claude reads your codebase thoroughly, identifies affected files, dependencies, and edge cases. It asks clarifying questions until it fully understands. Go back and forth until Claude has no more questions.

Key principle:

Your job is NOT to implement yet. Just explore, plan, and clarify ambiguities.

3. /create-plan - Design the Implementation

When to use:

After exploration is complete and all questions are answered.

How it works:

```
/create-plan
```

Claude generates a markdown plan file with:

- TL;DR summary of what you're building
- Critical decisions made during exploration
- Step-by-step tasks with status emojis
- Overall progress percentage tracking

Status emojis:

Red = To Do | Yellow = In Progress | Green = Done

Pro tip:

This markdown file can be shared with different models (Gemini for UI, Claude for backend) for execution.

4. /execute - Build It

When to use:

After the plan is reviewed and approved.

How it works:

```
/execute
> [reference the plan file]
```

Claude implements step by step, following existing code patterns and conventions. It updates the plan's status emojis as it progresses through each task.

5. /review - Self-Check

When to use:

After implementation is complete and you've done manual QA.

How it works:

```
/review
```

Claude reviews its own code checking for:

- Security issues and vulnerabilities
- TypeScript errors (no 'any' types)
- Production readiness (no console.log, no TODOs)
- React/Hooks issues
- Performance problems
- Architecture violations

Output is organized by severity: CRITICAL, HIGH, MEDIUM, LOW

6. /peer-review - Cross-Model Review

When to use:

After self-review. This is Zevi's secret weapon for catching issues.

The process:

1. Run /review in Claude
2. Run review in another tool (Cursor, Codex, Gemini)
3. Copy findings from other models
4. Run /peer-review with their feedback:

```
/peer-review
Dev Lead 1 (Codex):
[paste Codex findings]

Dev Lead 2 (Gemini):
[paste Gemini findings]
```

Why this works:

Claude acts as the 'team lead' who has more context than the reviewers. It will either:

- Confirm issues and add them to the fix plan
- Dismiss invalid findings with clear explanation

"Claude sometimes gets sassy: This has been raised for the third time, and for the third time I'm telling you this is not an issue."

7. /document - Update Documentation

When to use:

After all fixes are complete and code is ready to ship.

How it works:

```
/document
```

Claude checks git diff, reads the actual implementation (not existing docs), and updates CHANGELOG and relevant documentation. It verifies everything against real code, not assumptions.

8. /learning-opportunity - Teaching Mode

When to use:

Anytime you encounter something you don't understand.

How it works:

```
/learning-opportunity
> What is dependency injection and why did you use it here?
```

Claude explains at three levels of complexity:

- Level 1: Core concept - what it is and why it exists
- Level 2: How it works - mechanics and tradeoffs
- Level 3: Deep dive - production implications

Uses examples from your actual codebase, not generic tutorials.

Pro Tips from Zevi

Don't skip exploration

Rushing to code creates gnarly bugs later. Planning is critical for serious apps.

Use different models for different tasks

Gemini excels at UI/design, Claude for architecture and communication, GPT/Codex for solving hard bugs.

Do post-mortems on failures

Ask Claude: "What in your system prompt made you make this mistake?" Then update your documentation.

Manual QA before AI review

You catch obvious issues while testing, AI catches subtle code problems.

Keep slash commands updated

Every time AI fails, improve the prompts so it doesn't happen again.

Models have personalities

Claude = communicative CTO. GPT = genius coder who doesn't explain. Gemini = brilliant but chaotic artist.

Example Session

Here's what a typical development session looks like:

```
# Morning: Pick up a task

/explore
> I want to add user authentication with Google OAuth

[Answer Claude's clarifying questions back and forth]

/create-plan

[Review the plan, make adjustments, approve it]

/execute

[Test manually in the browser]

/review

[Run review in Cursor/Codex too, then:]

/peer-review
[paste findings from other models]

[When confused about something Claude did:]

/learning-opportunity
> Why did you use JWT instead of sessions?

[After everything works:]

/document
```

The key insight: Planning and review matter more than execution speed. Don't rush to code.

Key Takeaways

1. Separate thinking from doing - explore and plan before you execute.

2. Use multiple AI models as a team - they each have different strengths.

3. AI is your thought partner, not just a code generator.

4. Every failure is an opportunity to improve your prompts and documentation.

5. You own your outputs - if something is wrong, it's your responsibility to catch it.

"It's not that you will be replaced by AI. You'll be replaced by someone who's better at using AI than you." - Zevi

Based on the Lenny's Podcast episode with Zevi Arnowitz, PM at Meta.
Slash commands and workflow created by Zevi for his AI development process.