Building with AI at SectionAI

Your Systematic Guide to Claude Code + Superpowers

Section 1: Why This Matters

You Can Build. Right Now. Today.

There's a programmer in Beijing named Liu Xiaopai. Working mostly alone, he runs 1-2 dozen revenue-generating products simultaneously, clearing over $1M per year in profit. He's not a superhuman coder. He's using AI tools—specifically Claude Code—to automate "the entire product lifecycle."

A typical startup needs many people to build and maintain a single product. Liu does it solo by building **custom tools that build for him**.

This is what's called "hyperproductivity"—and it's not reserved for programmers.

This Could Be You

At SectionAI, we believe every team member should be able to build for their own use cases:

• Search and analyze your documents on your computer

• Build automations that save hours every week

• Create tools and products that solve your specific problems

• Deploy full-scale applications without waiting for engineering support

You don't need a CS degree. You don't need years of programming experience. **You just need a system.**

The Problem: Getting Stuck

Here's the truth: Most people try Claude Code once or twice, hit a bug they don't know how to fix, and give up.

Why? Because they don't know **the system**.

Without knowing about:

• `/superpowers:brainstorm` to refine unclear ideas

• `/superpowers:write-plan` to create executable plans

• `/superpowers:execute-plan` to build in controlled batches

• The `systematic-debugging` skill when things break

...you're just guessing. You're hoping Claude figures it out. Sometimes it does. Often it doesn't.

You Have Permission to Experiment

The biggest barrier isn't technical—it's psychological.

You have permission to:

• Try building things even if you're "not technical"

• Get stuck and need to debug

• Build something inefficient and improve it later

• Experiment with tools that might fail

• Ask for help when you're blocked

Building with AI is messy. You will hit bugs. Things will break. That's not failure—that's the process.

This guide gives you the **systematic approach** that turns those frustrating moments into solvable problems.

What You'll Learn

By the end of this guide, you'll know:

The workflow: When to brainstorm, plan, execute, and debug

The commands: Which slash commands and skills to use when

How to start: Your first project from zero to working

How to unstuck yourself: What to do when you hit issues

What to build: 7 starter project ideas to build this week

Let's begin.

Section 2: The System Overview

The Core Cycle

Building with Claude Code + Superpowers follows a clear cycle:

```
Brainstorm → Plan → Execute → Debug (when needed) → Verify → Ship
```

But the real power is knowing **when** to use each phase.

The Decision Tree

Here's how to think about every project:

```
Starting a new project?
■
■■ Do you have crystal-clear requirements?
■  ■■ YES → Jump to /superpowers:write-plan
■  ■■ NO → Start with /superpowers:brainstorm
■
After brainstorming...
■■ /superpowers:write-plan
   ■
   ■■ /superpowers:execute-plan [your-plan.md]
       ■
       ■■ Hit a bug or error?
       ■  ■■ Use: systematic-debugging skill
       ■
       ■■ Tests failing repeatedly?
       ■  ■■ Use: test-driven-development skill
       ■
       ■■ Not sure if it's working?
       ■  ■■ Use: verification-before-completion skill
       ■
       ■■ Major feature complete?
          ■■ Use: requesting-code-review skill
```

Why This Matters

Without Superpowers:

• Claude Code is powerful but unstructured

• It might brainstorm, then immediately try to execute (skipping planning)

• When bugs hit, it guesses at fixes instead of investigating

• You don't know if it's actually done or just claiming success

With Superpowers:

• Clear phases with review checkpoints

• Systematic debugging that finds root causes

• Test-driven development that ensures code actually works

• Verification before claiming completion

The Three Types of Work

Think of your work in three categories:

1. Exploration (Brainstorming Phase)

• You have a rough idea but unclear implementation

• You're not sure the best approach

• You need to explore alternatives

• **Use:** `/superpowers:brainstorm`

2. Planning (Planning Phase)

• You know what you want to build

• You need detailed step-by-step tasks

• You want to review before execution starts

• **Use:** `/superpowers:write-plan`

3. Execution (Building Phase)

• You have a plan file

• You want controlled, batch-by-batch execution

• You want review checkpoints to catch issues early

• **Use:** `/superpowers:execute-plan`

The Safety Net: Debugging Skills

When things go wrong (and they will), you have skills to invoke:

• **systematic-debugging**: When you hit bugs or unexpected behavior

• **test-driven-development**: Write tests first, then implement

• **verification-before-completion**: Actually verify it works before finishing

• **requesting-code-review**: Get structured feedback on major work

Key Insight

The system prevents common failure modes:

What's Next

Now that you understand the system, let's get you set up and building your first project.

Section 3: Getting Started

Prerequisites

You're ready to start if you:

• Have used Claude, ChatGPT, or Gemini before

• Can open a terminal/command line (don't worry, we'll walk through it)

• Have a computer running macOS, Linux, or Windows

That's it. No programming experience required.

Installation

Step 1: Install Claude Code

Open your terminal and run:

```
npm install -g @anthropic-ai/claude-code
```

Don't have npm? Install Node.js first from nodejs.org (choose the LTS version), then run the command above.

Step 2: Install Superpowers Plugin

Once Claude Code is installed, run:

```
claude plugin install superpowers
```

This adds the systematic workflow commands and skills we've been talking about.

Step 3: Verify Installation

Test that everything works:

```
claude --version
```

You should see a version number. You're ready!

Your First Project: File Renamer

Let's build something simple but useful: a script that renames files in a folder based on a pattern.

Why this project?

• Takes 30-60 minutes

• You'll use the full workflow: brainstorm → plan → execute

• Solves a real problem (cleaning up messy filenames)

• You'll hit at least one bug (good practice for debugging)

Step-by-Step Walkthrough

1. Create a project folder

```
mkdir ~/my-first-ai-project
cd ~/my-first-ai-project
```

2. Start Claude Code

```
claude
```

You'll see a chat interface. Claude is now ready to help you build.

3. Brainstorm the idea

Type this (exactly):

```
/superpowers:brainstorm I want to build a script that renames all files in a folder. It should let me specify a
```

What happens:

• Claude will ask clarifying questions (answer them!)

• Questions might include: "Should it work recursively on subfolders?" "What if files already exist with the new names?" "Do you want to preview changes before applying?"

• Answer based on what you want (there's no wrong answer)

Example interaction:

```
Claude: Should the script work recursively on subfolders, or just the target folder?
You: Just the target folder for now.

Claude: What should happen if a file with the new name already exists?
You: Skip it and warn me.

Claude: Would you like a preview mode to see what would change before actually renaming?
You: Yes, that sounds safe.
```

After you've answered the questions, Claude will summarize the approach and ask if you're ready to plan.

4. Create the plan

Type:

```
/superpowers:write-plan
```

What happens:

• Claude generates a detailed implementation plan

• Saves it as a markdown file (e.g., `file-renamer-implementation-plan.md`)

• Shows you the plan with exact file paths, code examples, and verification steps

Review the plan! Read through it. Ask questions if anything is unclear:

• "Why are you using this approach?"

• "What does this code do?"

• "Can you explain the testing strategy?"

5. Execute the plan

When you're ready:

```
/superpowers:execute-plan file-renamer-implementation-plan.md
```

What happens:

• Claude executes in **batches** (usually 2-4 tasks at a time)

• After each batch, it pauses and reports progress

• You can review the work and give feedback

• If everything looks good, tell it to continue

Important: Don't just say "continue" automatically. Look at what it built. Does it make sense?

6. Hit a bug (probably)

You'll likely encounter an error. This is normal and expected. Here's what to do:

```
@superpowers:systematic-debugging
```

Or just describe the error and ask Claude to use the systematic debugging skill.

What happens:

• Claude investigates the root cause (not just symptoms)

• Analyzes patterns to understand what's wrong

• Tests hypotheses before implementing fixes

• Actually fixes the underlying issue

7. Verify it works

Before finishing, test it:

```
# Create some test files
touch test1.txt test2.txt test3.txt

# Run your script
python file_renamer.py --pattern "test*.txt" --prefix "document" --preview
```

Does it show the expected renames? If yes:

```
# Actually rename them
python file_renamer.py --pattern "test*.txt" --prefix "document"
```

8. Celebrate!

You just built your first tool with AI.

What You Just Learned

In this first project, you:

■ Used `/superpowers:brainstorm` to refine a rough idea

■ Used `/superpowers:write-plan` to get a detailed implementation plan

■ Used `/superpowers:execute-plan` to build in controlled batches

■ Debugged an issue systematically (instead of guessing)

■ Verified the final product actually works

This is the complete cycle. Every project follows this pattern.

Common First-Project Issues

"Claude says it's done but the script doesn't work"

• Use: `@superpowers:verification-before-completion`

• This forces Claude to actually run the code and verify it works

"I'm getting an error I don't understand"

• Copy the full error message and paste it to Claude

• Ask: "Can you use the systematic debugging skill to investigate this?"

"Claude keeps trying the same fix that doesn't work"

• Say: "This approach isn't working. Let's use systematic-debugging to find the root cause instead of guessing."

"I want to change the approach mid-execution"

• That's fine! Just explain what you want to change

• Claude can adjust the plan or you can abort and start fresh

Next Steps

Now that you've completed your first project, you're ready to dive deeper into the Superpowers workflow.

Section 4: The Superpowers Workflow (Deep Dive)

This section explains **when and how** to use each command and skill. Bookmark this section—you'll reference it often.

4.1 Brainstorming Phase

When to Use

Use `/superpowers:brainstorm` when:

• You have a rough idea but unclear implementation

• You're not sure about the best approach

• You need to explore alternatives before committing

• You're solving a problem but don't know the solution shape yet

Don't use it when:

• You have crystal-clear requirements (jump straight to planning)

• You're fixing a small, well-defined bug (just describe it)

How to Use

Command:

```
/superpowers:brainstorm [describe your idea in 1-3 sentences]
```

Example:

```
/superpowers:brainstorm I want to build a tool that searches all my markdown files for specific keywords and ge
```

What Happens

Claude will:

Ask clarifying questions to understand your needs

Explore different implementation approaches with you

Discuss tradeoffs of each approach

Help you validate assumptions incrementally

Summarize the refined idea and ask if you're ready to plan

Pro tip: Answer the questions honestly. If you don't know, say "I don't know—what do you recommend?" Claude will guide you.

Example Interaction

```
You: /superpowers:brainstorm I want to automate sending weekly team updates via email

Claude: Let me ask some clarifying questions:

1. Where does the team update content come from?
   - Do you write it manually?
   - Should it pull data from tools (Jira, GitHub, etc.)?
   - Should it be generated from templates?

You: I write it manually in a markdown file each week. I just want to automate the sending part.

Claude: Got it. A few more questions:

2. Which email service are you using?
   - Gmail?
   - Outlook?
   - Something else?

You: Gmail

Claude: 3. Should this run automatically on a schedule, or should you trigger it manually?

You: Manually is fine. I want to review before sending.

Claude: Perfect. Here's what I'm thinking:

APPROACH:
- Python script that reads a markdown file
- Converts markdown to HTML for nice formatting
- Uses Gmail API to send the email
- You run it manually after writing your update
- It can store recipient list in a config file

Does this match what you're looking for?

You: Yes! That's exactly it.

Claude: Great. Ready to create a detailed implementation plan?
```

At this point, you'd move to planning.

Red Flags During Brainstorming

Claude isn't asking clarifying questions

• Say: "Can you ask me some clarifying questions about this idea? I want to make sure we're aligned before planning."

Claude jumps straight to implementation

• Say: "Hold on—let's brainstorm the approach first using the brainstorming skill. I'm not sure about implementation details yet."

You're getting overwhelmed with questions

• Say: "Let's focus on the core functionality first. We can add nice-to-haves later."

4.2 Planning Phase

When to Use

Use `/superpowers:write-plan` when:

• You've finished brainstorming and know what to build

• You have clear requirements (can skip brainstorming entirely)

• You want a reviewable plan before execution starts

• You're about to tackle a multi-step project

Always use planning unless it's a trivial 1-2 line change.

How to Use

Command:

```
/superpowers:write-plan
```

No arguments needed—Claude has context from your conversation or brainstorming session.

What Happens

Claude will:

Create a detailed implementation plan

Break it into bite-sized tasks (each task is 15-30 minutes of work)

Include exact file paths where code will go

Show complete code examples for complex parts

Add verification steps to confirm each task works

Save the plan as a markdown file

Example Plan Structure

```
# Email Automation Tool - Implementation Plan

## Overview
Build a Python CLI tool that sends markdown-formatted emails via Gmail API.

## Prerequisites
- Python 3.8+
- Gmail account with API access enabled
- pip for package installation

## Tasks

### Task 1: Project Setup
**Goal:** Create project structure and install dependencies

**Steps:**
1. Create project directory structure:
   ```
   email-automation/
   ■■■ email_sender.py
   ■■■ config.json
   ■■■ requirements.txt
   ■■■ README.md
   ```

2. Create requirements.txt with:
   ```
   google-auth-oauthlib==1.0.0
   google-auth-httplib2==0.1.0
   google-api-python-client==2.88.0
   markdown2==2.4.8
   ```

3. Install dependencies: `pip install -r requirements.txt`

**Verification:** Run `pip list` and confirm all packages installed

---

### Task 2: Gmail API Setup
**Goal:** Set up authentication for Gmail API

[...continues with detailed steps...]

---

### Task 3: Markdown to HTML Conversion
[...continues...]
```

Review the Plan Carefully

Before executing, ask yourself:

• Do I understand what each task does?

• Are there any tasks that seem wrong or unnecessary?

• Are there missing steps?

Ask Claude questions:

• "Why are you using this library instead of X?"

• "What does Task 4 accomplish?"

• "Can we simplify Task 6?"

Modifying the Plan

You can request changes:

• "Can you add error handling for missing files?"

• "Let's skip the web interface for now and focus on CLI"

• "Can you break Task 5 into smaller subtasks?"

Claude will update the plan file.

4.3 Execution Phase

When to Use

Use `/superpowers:execute-plan` when:

• You have a plan file you've reviewed

• You're ready to build

• You want controlled, batch-by-batch execution with checkpoints

How to Use

Command:

```
/superpowers:execute-plan path/to/your-plan.md
```

Example:

```
/superpowers:execute-plan email-automation-plan.md
```

What Happens

Claude will:

Load the plan

Execute tasks in **batches** (typically 2-4 tasks per batch)

After each batch, pause and report what was completed

Wait for your review and approval before continuing

Continue through all batches until the plan is complete

During Execution

After each batch, Claude will report:

```
■ Batch 1 Complete

Completed tasks:
- Task 1: Project setup
- Task 2: Gmail API setup
- Task 3: Markdown conversion

Next batch will cover:
- Task 4: Email sending logic
- Task 5: Configuration management

Review the code above. Ready to continue? (yes/no/modify)
```

Your options:

• **"yes" or "continue"**: Proceed to next batch

• **"wait, let me test this first"**: Pause while you verify

• **"Task 2 didn't work correctly—here's the error: [paste error]"**: Claude will debug before continuing

• **"Can you explain what you did in Task 3?"**: Claude will explain

Pro tip: Actually review the code between batches. Catch issues early!

Batch Execution Benefits

Why batches instead of all-at-once?

Early error detection: Catch problems after 3 tasks, not after 15

Course correction: Adjust approach mid-execution if needed

Understanding: You see progress and can learn as you go

Less overwhelming: Review 2-4 tasks at a time, not 20

Example Execution Flow

```
You: /superpowers:execute-plan my-plan.md
Claude: Starting execution. Plan has 8 tasks, I'll execute in batches.
[Executes Tasks 1-3]
Claude: ■ Batch 1 complete. Created project structure, set up dependencies, and implemented core logic. Review
You: [Reviews code] Looks good! Continue.
[Executes Tasks 4-6]
Claude: ■ Batch 2 complete. Added error handling and configuration management. However, I'm getting a test fail
[Claude debugs the issue]
Claude: Found the issue - incorrect file path. Fixed. Tests now passing. Ready to continue to final batch?
You: Yes
[Executes Tasks 7-8]
Claude: ■ All tasks complete! Running final verification...
[Runs verification]
Claude: ■ All tests passing. Project is ready to use.
```

4.4 Debugging Phase

When to Use

Use the `systematic-debugging` skill when:

• You hit a bug or unexpected behavior

• Tests are failing

• Code runs but produces wrong output

• You're getting errors you don't understand

Critical: Use this INSTEAD of letting Claude guess at fixes.

How to Use

Method 1: Explicit invocation

```
@superpowers:systematic-debugging
```

Method 2: Ask Claude to use it

```
I'm getting this error: [paste error]
Can you use the systematic-debugging skill to find the root cause?
```

What Happens

The skill follows a 4-phase framework:

Phase 1: Root Cause Investigation

• Reproduces the error

• Examines relevant code

• Checks assumptions

• Identifies where the problem originates

Phase 2: Pattern Analysis

• Looks for similar issues in the codebase

• Identifies whether it's an isolated bug or systemic issue

• Checks for related problems

Phase 3: Hypothesis Testing

• Forms hypotheses about what's wrong

• Tests each hypothesis

• Gathers evidence

• Confirms the actual root cause

Phase 4: Implementation

• Implements the fix

• Tests that the fix works

• Verifies no side effects

• Confirms the issue is resolved

Why This Matters

Without systematic-debugging:

```
You: I'm getting "FileNotFoundError: config.json"

Claude: Let me add a check for that:
[writes code that catches the error]

You: Still not working...

Claude: Let me try this instead:
[writes different error handling]

You: Still broken...

[10 iterations later, still guessing]
```

With systematic-debugging:

```
You: I'm getting "FileNotFoundError: config.json"
Use systematic-debugging please.

Claude: Using systematic-debugging skill.

Phase 1: Investigating root cause
- The error occurs in config.py line 23
- Code assumes config.json is in current working directory
- But script is being run from parent directory
- So it's looking in wrong place

Phase 2: Pattern analysis
- This affects any file operations using relative paths
- Other files (email templates, logs) might have same issue

Phase 3: Hypothesis
- Root cause: Hard-coded relative paths break when script run from different directory
- Solution: Use absolute paths or path relative to script location

Phase 4: Implementation
- Updated all file paths to use os.path.dirname(__file__)
- Tested from multiple directories - works correctly now
- ■ Fixed

You: [one iteration, problem solved]
```

Example Debugging Session

```
You: My script runs but produces the wrong output. Expected "Hello World" but getting "Hello None"

@superpowers:systematic-debugging

Claude: Starting systematic debugging.

PHASE 1: Root Cause Investigation

Let me trace the execution:
1. Function get_name() returns None
2. This happens when config file exists but 'name' key is missing
3. The code assumes 'name' key will always be present

Found it: Line 45 doesn't handle missing keys.

PHASE 2: Pattern Analysis
```

```
Checking for similar issues:
- Line 52: Also assumes 'email' key exists
- Line 67: Also assumes 'recipients' key exists
- This is a systemic issue with config handling

PHASE 3: Hypothesis Testing

Hypothesis: Config parsing doesn't validate required keys.

Test 1: Remove 'name' from config → Confirms the issue
Test 2: Remove 'email' from config → Same problem
Test 3: Add validation → Fixes the issue

Root cause confirmed: Missing input validation.

PHASE 4: Implementation

Adding validation:
1. Create required_keys list
2. Check all required keys exist on load
3. Raise clear error if missing
4. Provide default values where appropriate

Testing fix:
■ With valid config: Works correctly
■ With missing 'name': Clear error message
■ With missing 'email': Clear error message

Issue resolved!

You: Perfect! That makes sense now.
```

When NOT to Use Systematic Debugging

• Typos or obvious syntax errors (just fix them)

• You intentionally want to try a quick experiment

• You're still in brainstorming/exploratory mode

4.5 Other Essential Skills

Test-Driven Development (TDD)

When to use:

• Before implementing any feature

• When you want high confidence code works correctly

• When building something with clear success criteria

How to invoke:

```
@superpowers:test-driven-development
```

What it does:

Write tests FIRST (before implementation)

Run tests and watch them FAIL (confirms tests are valid)

Write minimal code to make tests pass

Verify tests now pass

Why this matters: Ensures your tests actually verify behavior (not just passing because they're wrong).

Verification Before Completion

When to use:

• Before claiming a task is "done"

• When Claude says something works but you're not sure

• Before committing code or creating a PR

How to invoke:

```
@superpowers:verification-before-completion
```

What it does:

• Runs actual verification commands

• Checks that tests pass

• Confirms expected output

• Requires proof before marking complete

Why this matters: Prevents "it's done!" when it's actually broken.

Requesting Code Review

When to use:

• After completing a major feature

• Before merging work

• When you want structured feedback on your implementation

How to invoke:

```
@superpowers:requesting-code-review
```

What it does:

• Reviews implementation against requirements/plan

• Checks for potential issues

• Suggests improvements

• Validates the approach

Quick Reference: When to Use What

Section 5: Starter Projects

Now that you understand the system, here are 7 projects you can build this week. Each includes what you'll learn, the workflow to use, and estimated time.

Project 1: Document Search Tool

What it does:

Build a CLI tool to search through all your markdown notes/documents for specific keywords, showing matches with context.

What you'll learn:

• File I/O (reading multiple files)

• Text processing and pattern matching

• Basic CLI argument parsing

• Working with file paths

Workflow:

```
1. /superpowers:brainstorm → Clarify search features (case-sensitive? regex? show context?)
2. /superpowers:write-plan → Get detailed implementation
3. /superpowers:execute-plan → Build in batches
4. Test and refine
```

Estimated time: 2-3 hours

Starting prompt:

```
/superpowers:brainstorm I want to build a tool that searches all markdown files in a directory for specific key
```

Expected challenges:

• Handling different file encodings → systematic-debugging will help identify encoding issues

• Performance with large directories → Brainstorming will help explore efficient approaches

Project 2: Email Automation Script

What it does:

Automate sending recurring emails (weekly updates, reminders, reports) from markdown templates.

What you'll learn:

• API integration (Gmail API)

• Markdown to HTML conversion

• Configuration file management

• Authentication flows

Workflow:

```
1. /superpowers:brainstorm → Clarify email service, triggers, template format
2. /superpowers:write-plan → Plan includes OAuth setup, template rendering
3. /superpowers:execute-plan → Build with review checkpoints
4. Test with sample emails before going live
```

Estimated time: 3-4 hours

Starting prompt:

```
/superpowers:brainstorm I want to automate sending weekly team updates via email. I write the content in markdo
```

Expected challenges:

• Gmail API authentication → Plan will include detailed OAuth setup steps

• HTML email formatting → May need iteration during execution phase

Project 3: Meeting Notes Processor

What it does:

Process meeting transcripts (from Otter.ai, Zoom, etc.) to extract action items, decisions, and key points.

What you'll learn:

• Text parsing and extraction

• Working with LLM APIs (Claude/GPT for extraction)

• Structured output generation

• File format conversion

Workflow:

```
1. /superpowers:brainstorm → Discuss transcript format, extraction criteria
2. /superpowers:write-plan → Plan prompt engineering for extraction
3. /superpowers:execute-plan → Build extraction + formatting
4. Test with real meeting transcripts
```

Estimated time: 4-5 hours

Starting prompt:

```
/superpowers:brainstorm I get meeting transcripts from Otter.ai. I want a tool that processes them and extracts
```

Expected challenges:

• Prompt engineering for reliable extraction → Brainstorming helps refine extraction criteria

• Handling different transcript formats → May need systematic-debugging for edge cases

Project 4: Data Analysis Dashboard

What it does:

Build a simple web dashboard that loads CSV data and creates interactive visualizations.

What you'll learn:

• Web basics (HTML/CSS/JavaScript)

• Data visualization libraries (Chart.js, Plotly)

• CSV parsing and data manipulation

• Local web server setup

Workflow:

```
1. /superpowers:brainstorm → Choose visualization library, decide on chart types
2. /superpowers:write-plan → Plan frontend + data processing
3. /superpowers:execute-plan → Build in batches (backend first, then frontend)
4. Test with your data, refine visualizations
```

Estimated time: 5-6 hours

Starting prompt:

```
/superpowers:brainstorm I have CSV files with sales data (date, amount, category). I want to build a simple web
```

Expected challenges:

• JavaScript/frontend work if new to web dev → Plan will include complete code examples

• Chart customization → May iterate during execution based on how it looks

Project 5: Slack Bot for Team Updates

What it does:

Bot that posts automated reports or notifications to Slack channels on a schedule or trigger.

What you'll learn:

• Slack API integration

• Webhook handling

• Scheduling/cron jobs

• Message formatting (Slack blocks)

Workflow:

```
1. /superpowers:brainstorm → Discuss bot triggers, message content, channels
2. /superpowers:write-plan → Plan includes Slack app setup, webhook config
3. /superpowers:execute-plan → Build core bot, then add scheduling
4. Test in private channel first
```

Estimated time: 4-6 hours

Starting prompt:

```
/superpowers:brainstorm I want to build a Slack bot that posts a daily summary to our #team-updates channel. Th
```

Expected challenges:

• Slack app configuration → Plan will walk through app setup in Slack admin

• Message formatting → May need iteration to get formatting right

Project 6: Personal CRM (Contact Manager)

What it does:

Track contacts, interactions, and follow-ups. Simple database-backed app for managing professional relationships.

What you'll learn:

• Database basics (SQLite)

• CRUD operations (Create, Read, Update, Delete)

• Simple CLI or web interface

• Data modeling

Workflow:

```
1. /superpowers:brainstorm → Design data model (what to track about contacts)
2. /superpowers:write-plan → Plan database schema, interface
3. /superpowers:execute-plan → Build database layer first, then interface
4. Test with sample contacts
```

Estimated time: 6-8 hours

Starting prompt:

```
/superpowers:brainstorm I want to build a simple CRM to track professional contacts. For each contact, I want t
```

Expected challenges:

• Database design if new to SQL → Brainstorming will help design schema

• Query complexity for search → May need systematic-debugging for complex queries

Project 7: Custom Workflow Automation (Your Own Idea!)

What it does:

Automate your own repetitive task. The best projects solve your actual problems.

What you'll learn:

• Problem decomposition

• Tool design for real-world use

• Everything from Projects 1-6!

Workflow:

```
1. /superpowers:brainstorm → Deep exploration of your problem and solutions
2. /superpowers:write-plan → Detailed plan tailored to your needs
3. /superpowers:execute-plan → Build with review checkpoints
4. Iterate based on actual use
```

Estimated time: Varies (4-10 hours)

Starting prompt:

```
/superpowers:brainstorm [Describe your repetitive task in detail. What do you do manually? What's frustrating?
```

Examples:

• Automatically organize downloads folder based on file types and dates

• Generate weekly reports from multiple data sources

• Sync data between different tools (Notion ↔ Airtable)

• Process and archive old emails based on rules

• Generate social media content from blog posts

• Backup and organize photos automatically

Expected challenges:

• Unique to your problem → This is where the full workflow shines

• Likely to hit unexpected issues → systematic-debugging is essential

Tips for All Projects

Start small, then expand:

• Get core functionality working first

• Add nice-to-haves later

• Don't over-plan the first version

Test frequently:

• Test after each batch during execution

• Don't wait until the end to discover issues

• Use verification-before-completion before finishing

Document as you go:

• Ask Claude to add comments explaining complex parts

• Create a README with usage instructions

• Future you will thank present you

Share your work:

• Post in #ai-building Slack channel at SectionAI

• Show teammates what you built

• Help others who want to build similar things

Section 6: Getting Unstuck

You WILL get stuck. It's not a matter of if, but when. Here's your guide to getting unstuck quickly.

The Problem-Solution Matrix

The Universal Unstuck Process

When you're stuck and the matrix above doesn't help, follow these steps:

Step 1: Stop and describe the problem

```
I'm stuck. Here's what I'm trying to do: [goal]
Here's what's happening: [actual behavior]
Here's what I expected: [expected behavior]
```

Step 2: Ask Claude which approach to use

```
What skill or command should we use to address this?
```

Claude will recommend systematic-debugging, brainstorming, etc.

Step 3: Follow the recommended approach

Trust the system. If Claude says "let's use systematic-debugging," actually use it.

Step 4: If still stuck, take a break

Sometimes the best debugging happens when you step away for 15 minutes.

Common Anti-Patterns (Don't Do These!)

■ Letting Claude guess at fixes

```
Bad:
You: Got an error
Claude: Let me try this [random fix]
You: Still broken
Claude: Let me try this instead [another guess]
[repeat 10 times]
```

■ Using systematic debugging

```
Good:
You: Got an error - let's use systematic-debugging to find the root cause
Claude: [investigates properly]
[fixed in 1-2 iterations]
```

■ Skipping planning for complex tasks

```
Bad:
You: Build me a dashboard with data viz
Claude: [starts writing code immediately]
[3 hours later, everything needs to be rewritten]
```

■ Planning first

```
Good:
You: Build me a dashboard with data viz. Let's use /superpowers:write-plan first.
Claude: [creates reviewable plan]
You: [reviews, asks questions, refines]
/superpowers:execute-plan [plan-file]
[builds correctly in 2 hours]
```

■ Accepting "it's done" without verification

```
Bad:
Claude: Done! The script should work now.
You: Great! [doesn't test it]
[Later: script is broken]
```

■ Verifying before finishing

```
Good:
Claude: Done! The script should work now.
You: Use verification-before-completion to confirm
Claude: [actually runs it, finds issue, fixes it]
Claude: ■ Verified working
```

■ Not reading error messages

```
Bad:
You: There's an error
Claude: [guesses at random solutions]
```

■ Sharing full error details

```
Good:
You: Getting this error:
[pastes complete error message with stack trace]

Use systematic-debugging to investigate

Claude: [investigates based on actual error details]
```

When to Ask for Human Help

Sometimes you need another human. Ask for help in #ai-building Slack channel when:

You've used systematic-debugging but still can't solve it (after 2-3 attempts)

You need domain expertise (e.g., specific API knowledge, security practices)

You're not sure if your approach is right (architectural decisions)

You've been stuck for more than an hour (fresh eyes help)

How to ask for help effectively:

```
Hey team! Working on [project].

Goal: [what you're trying to build]
Approach: [your current approach]
Problem: [specific issue]
What I've tried: [list debugging attempts]

Looking for: [specific question or guidance needed]
```

Emergency Restart

If everything is completely broken and you don't know how to fix it:

Save your current work (even if broken)

Start a new Claude session

Describe what you were trying to build

Ask Claude to review the broken code and create a fresh plan

Starting fresh is OK! Sometimes it's faster than debugging a mess.

Section 7: Quick Reference Cheat Sheet

Keep this page handy. Print it out. Bookmark it. You'll reference it constantly.

The Workflow (One-Page Version)

```
Starting a project?
■
■■ Clear requirements?
■  ■■ YES → /superpowers:write-plan
■  ■■ NO  → /superpowers:brainstorm
■           ■■ Then /superpowers:write-plan
■
Have a plan?
■■ /superpowers:execute-plan [plan.md]
    ■
    ■■ Bug? → @superpowers:systematic-debugging
    ■■ Tests failing? → @superpowers:test-driven-development
    ■■ Design unclear? → @superpowers:brainstorming
    ■■ Claude says "done"? → @superpowers:verification-before-completion
    ■■ Major work complete? → @superpowers:requesting-code-review
```

Commands Quick Reference

Skills Quick Reference

Installation Commands

```
# Install Claude Code
npm install -g @anthropic-ai/claude-code

# Install Superpowers plugin
```

```
claude plugin install superpowers

# Start Claude in your project
cd your-project
claude
```

First Session Template

Copy/paste this for your first project:

```
I'm new to Claude Code + Superpowers. I want to build [describe your idea in 1-2 sentences].

Let's start by brainstorming the approach:
/superpowers:brainstorm [your idea]
```

Then follow the prompts!

Debugging Decision Tree

```
Hit a bug?
■
■■ Is it an obvious typo?
■  ■■ YES → Just fix it
■
■■ Is it a clear error message?
■  ■■ YES → Share full error with Claude
■           → Use @superpowers:systematic-debugging
■
■■ Code runs but wrong output?
■  ■■ YES → Use @superpowers:systematic-debugging
■           → It will trace execution and find issue
■
■■ Tests failing?
   ■■ Use @superpowers:systematic-debugging
      → It will investigate why tests fail
```

Common Commands You'll Use

During Development:

```
# Start Claude
claude

# Check version
claude --version

# List installed plugins
claude plugin list
```

```
# Update Superpowers
claude plugin update superpowers
```

In Claude Chat:

```
# Brainstorm
/superpowers:brainstorm [idea]

# Create plan
/superpowers:write-plan

# Execute plan
/superpowers:execute-plan plan-file.md

# Invoke debugging
@superpowers:systematic-debugging

# Verify completion
@superpowers:verification-before-completion

# Request review
@superpowers:requesting-code-review
```

Pro Tips

Always brainstorm when unsure - It's better to spend 10 minutes clarifying than 2 hours building the wrong thing

Review plans before executing - Catch issues in the plan, not in the code

Don't skip batch reviews - Look at code between batches during execution

Use systematic-debugging immediately - Don't let Claude guess for 3+ iterations

Verify before finishing - "It's done!" means nothing without verification

Start with starter projects - Build muscle memory with Projects 1-3 before tackling Project 7

Document as you build - Ask Claude to add comments and README

Share your wins - Post in #ai-building when you complete projects

Resources

SectionAI:

• #ai-building Slack channel - Ask questions, share projects

• Team members building with AI - Learn from their experiences

External:

• Claude Code docs: https://docs.claude.ai/code

• Superpowers GitHub: https://github.com/colefortier/superpowers

• This guide: Keep it handy!

You're Ready

You now have:

• ■ The system (Brainstorm → Plan → Execute → Debug)

• ■ The commands (brainstorm, write-plan, execute-plan)

• ■ The skills (systematic-debugging, TDD, verification)

• ■ Starter projects (7 concrete ideas)

• ■ Troubleshooting guide (how to get unstuck)

• ■ Quick reference (cheat sheet for daily use)

Your next step: Pick a starter project (I recommend #1 or #2) and build it this week.

Remember:

• You have permission to experiment

• Getting stuck is part of the process

• The system will help you get unstuck

• Ask for help when you need it (#ai-building)

Welcome to hyperproductivity. Now go build something.

Last updated: November 2025

Questions? Feedback? Share in #ai-building