Cursor AI

The AI Code Editor That Writes With You

The Complete Cliff Notes — 40 Pages

A Meshuga Guide — Crazy Simple Tech

Last Updated: February 19, 2026

Cursor AI Cliff Notes — Pages 1-10

Quick Start Guide

01 — It's VS Code, But Smarter

Cursor is a code editor built on top of VS Code. It looks identical. Your extensions work. Your settings import. The difference? AI lives inside it. Press Cmd+K and type what you want. Cursor writes the code. Press Cmd+L and ask it to explain something. It reads your entire codebase to answer.

02 — Who Should Use Cursor

You'll love it if you write code daily, waste time Googling syntax, copy-paste from Stack Overflow, work on unfamiliar codebases, or want AI help without leaving your editor.

03 — Download and Install

Visit cursor.sh, download for your OS, drag to Applications (Mac) or run installer (Windows). Open Cursor — it imports VS Code settings automatically.

04 — Sign In

Click Sign In top-right. Use Google, GitHub, or email. Free tier: 2,000 completions/month. Pro ($20/mo): unlimited.

05 — Cmd+K: AI Code Generation

Highlight code or place cursor. Press Cmd+K. Type instruction in plain English. Press Enter. Review generated code. Accept or reject.

06 — Cmd+L: AI Chat (Codebase-Aware)

Press Cmd+L. Chat panel opens. Ask about your code. Cursor searches your project and responds with file references.

07 — Write New Code from Scratch

Type a comment describing what you need. Highlight it. Press Cmd+K. Type "write this function." Cursor generates it with error handling.

08 — Refactor Existing Code

Highlight messy code. Press Cmd+K. Type "refactor this to be more readable." Cursor rewrites

with better names and modern syntax.

13 — Be Specific, Not Vague

Bad: "make this better." Good: "add input validation for email and phone number." Specificity = better results.

15 — Reject and Refine

Wrong output? Press Esc. Try Cmd+K again with more detail. Most code improves with one refinement.

16 — Check Generated Code Before Running

Always review. Check: correct libraries? Consistent names? Error handling? Security issues? Treat AI code like code from a smart but careless colleague.

Cursor AI Cliff Notes — Pages 11-20

Advanced Techniques & Workflows

Continuation of Cursor AI Quick Start Guide
Pages 11-20 of 40

Page 11: Advanced Prompting Techniques

21 — Layer Your Prompts

Technique: Don't try to do everything in one prompt. Build incrementally.

Example:

1. First prompt: "Add basic error handling"
2. Second prompt: "Add logging for the errors"
3. Third prompt: "Add retry logic with exponential backoff"

Each step is clearer than asking for all three at once.

---

22 — Use Comments as Specifications

Best practice: Write detailed comments first, then ask Cursor to implement them.

Example:

Authentication middleware that:

1. Checks for JWT token in Authorization header

2. Validates token signature and expiration

3. Extracts user_id and attaches to request context

4. Returns 401 if token is invalid or missing

5. Logs all authentication attempts (success and failure)

Highlight the comment, press `Cmd+K`, type "implement this middleware".

Cursor has a complete specification to work from.

Page 12: Multi-File Refactoring

23 — How Cursor Sees Multiple Files

By default: Cursor only sees the file you're currently editing.

To include other files:

1. Open multiple files in tabs
2. Cursor will use them as context automatically
3. OR: Use `Cmd+L` and mention specific files: "Check how authentication is handled in auth.js"

Pro tip: Keep relevant files open in tabs before prompting.

---

24 — Refactor Across Files

Scenario: You want to rename a function used in multiple files.

Steps:

1. Open all affected files in tabs
2. Highlight the function in the main file
3. Press `Cmd+K`
4. Type: "Rename this function to `authenticateUser` and update all references"

Cursor will suggest changes across all open files. Review each one before accepting.

![Screenshot: Multi-file refactoring with changes shown in multiple tabs]

Page 13: Custom AI Rules for Your Project

25 — Create a .cursorrules File

What it does: Tells Cursor about your project's conventions.

How to create:

1. Create a file called `.cursorrules` in your project root
2. Add rules in plain English

Example .cursorrules:

- Use ES6 arrow functions, not function keyword
- All API responses must include status, data, and error fields
- Error messages should be user-friendly, not technical
- Use async/await, not .then() chains
- All database queries must have timeout handling

Result: Cursor follows these rules when generating code.

---

26 — Project-Specific Patterns

Add common patterns to .cursorrules:

- React components: functional with hooks, no class components
- State management: use Context API, not Redux
- API calls: use our custom `apiClient.js` wrapper
- File naming: camelCase for files, PascalCase for components
- Testing: every new function needs a test in __tests__/

Cursor will match your team's style automatically.

![Screenshot: .cursorrules file in project root with sample rules]

Page 14: Debugging with AI Assistance

27 — Ask Cursor to Debug

Scenario: Code breaks, error message is confusing.

Steps:

1. Highlight the broken code
2. Press `Cmd+L`
3. Paste the error message
4. Ask: "Why is this failing?"

Example:

TypeError: Cannot read property 'name' of undefined
at processUser (app.js:42)

Cursor's response:

> "The error occurs because `user` is undefined at line 42. This happens when the API call on

line 38 fails or returns null. Add a check before accessing `user.name`."

Then it suggests the fix.

---

28 — Find Where Bugs Come From

Technique: Ask Cursor to trace the issue.

Prompt: "Where does the `user` variable come from and why might it be undefined?"

Cursor searches your codebase and explains:
> "The `user` variable is set on line 35 from `await fetchUser(id)`. The function is in `api/users.js` line 12. It returns null if the user isn't found. Add error handling after line 35."

You get the full path to the problem.

Page 15: Team Workflows & Best Practices

29 — Share .cursorrules with Your Team

Why: Everyone gets consistent AI-generated code.

How:

1. Create `.cursorrules` in your repo
2. Commit it to version control
3. Team members pull it down
4. Cursor reads it automatically

Result: Junior devs get code that matches senior dev standards.

---

30 — Code Review AI-Generated Code

Critical rule: Treat AI code like code from a junior developer.

Review checklist:

- ' Does it handle edge cases?
- ' Are there security issues? (SQL injection, XSS, etc.)
- ' Does it match our coding style?
- ' Are variable names clear?
- ' Is error handling sufficient?
- ' Does it work with our existing code?

Never merge AI code without review.

Page 16: CI/CD Integration

31 — Use Cursor in Automated Workflows

Scenario: You want AI to generate code during CI/CD.

Limitation: Cursor is a desktop app, not a CLI tool (yet).

Workaround: Use Cursor to generate code locally, commit, then CI/CD runs tests/builds as normal.

Future: Cursor API is coming (currently in beta). When available, you'll be able to call it from scripts.

---

32 — Pre-Commit Hooks with AI

Idea: Use Cursor to check code before committing.

Example workflow:

1. Write code
2. Before committing, ask Cursor: "Review this code for issues"
3. Fix issues it finds
4. Commit

How to make this a habit:

- Add a git pre-commit hook that reminds you to AI-review
- OR: Just make it part of your checklist

Page 17: Privacy & Security Settings

33 — What Data Does Cursor Send?

What gets sent to AI:

- Code you highlight or ask about
- Context from open files (if needed)
- Your prompts and questions

What doesn't get sent:

- Files you don't open in Cursor
- Your entire hard drive
- Private keys or secrets (unless you paste them)

Recommendation: Don't highlight sensitive data (API keys, passwords, etc.) when

prompting.

---

34 — Use .cursorignore to Block Files

What it does: Prevents Cursor from reading certain files.

How to create:

1. Create `.cursorignore` in your project root
2. List files/folders to ignore (same format as .gitignore)

Example .cursorignore:

```
.env
secrets/
config/production.yaml
*.key
*.pem
```

Cursor won't use these files as context, even if they're open.

Page 18: Troubleshooting Common Issues

35 — Cursor Won't Start

Symptoms: App crashes on launch or won't open.

Fixes:

1. **Restart your computer** (classic, but works)
2. **Clear Cursor cache:**
   - Mac: `~/Library/Application Support/Cursor`
   - Windows: `%APPDATA%\Cursor`
   - Delete the folder, restart Cursor
3. **Reinstall Cursor:**
   - Download latest version from cursor.sh
   - Uninstall old version first

---

36 — AI Responses Are Slow

Causes:

- High server load (lots of users)
- Large codebase (Cursor is indexing)
- Slow internet connection

Fixes:

1. **Upgrade to Pro:** Pro tier gets priority responses
2. **Reduce context:** Close extra files you don't need
3. **Check internet speed:** Cursor needs decent connection
4. **Wait for indexing to finish:** First-time indexing takes 1-2 minutes

---

Page 19: Power-User Shortcuts

37 — Keyboard Shortcuts You Should Memorize

Essential:

- `Cmd+K` — Generate/edit code
- `Cmd+L` — Open AI chat
- `Cmd+Shift+L` — Clear chat history
- `Esc` — Reject AI suggestion
- `Tab` — Accept AI suggestion (inline)

Pro:

- `Cmd+Option+K` — Generate in new tab
- `Cmd+Option+L` — Chat with new context
- `Cmd+/` — Comment/uncomment code (with AI context)

Learn these and you'll be 10x faster.

---

38 — Create Custom Snippets

Technique: Save common prompts as snippets.

Example:
Instead of typing "Add error handling with try-catch and log errors" every time, save it as a snippet.

How:

1. Use a snippet extension (like "Snippets" in VS Code extensions)
2. Add your common prompts
3. Trigger with short keyword

Example snippet:

- Trigger: `eh`
- Expands to: "Add comprehensive error handling with try-catch, log errors, and user-friendly messages"

Page 20: Cursor vs. Other AI Tools

39 — Cursor vs. GitHub Copilot

GitHub Copilot:

- ' Inline autocomplete (types code as you write)
- ' $10/month (cheaper)
- ' No codebase context (sees one file at a time)
- ' No chat interface (just autocomplete)

Cursor:

- ' Full codebase context (reads entire project)
- ' Chat interface (ask questions about your code)
- ' Inline generation AND autocomplete
- ' $20/month (more expensive)

Which to use:

- Small projects, tight budget !' Copilot
- Large codebases, need context !' Cursor
- Maximum power !' Use both (they work together)

---

40 — Cursor vs. ChatGPT/Claude

ChatGPT/Claude (separate):

- ' More powerful models (sometimes)
- ' Longer context windows
- ' No editor integration (copy-paste hell)
- ' No codebase context

Cursor:

- ' Lives in your editor (no copy-paste)
- ' Reads your entire project
- ' Generates code inline
- ' Slightly less powerful than GPT-4 Turbo or Claude Opus

Best approach:

- Use Cursor for 90% of coding tasks
- Use ChatGPT/Claude for complex architecture questions or design decisions

---

Continue to Pages 21-30 !'

This is part 2 of 4. Pages 21-40 coming next.

Cursor AI Cliff Notes — Pages 21-30

Team Collaboration & Enterprise Features

Continuation of Cursor AI Guide
Pages 21-30 of 40

Page 21: Working with Teams

41 — Team License ($40/user/month)

What you get:

- Everything in Pro ($20/month)
- Centralized billing
- Admin dashboard (manage team members)
- Usage analytics (who's using it, how much)
- Shared .cursorrules (enforce team standards)

When to upgrade:

- Team of 5+ developers
- Need usage tracking
- Want enforced coding standards

---

42 — Onboard New Team Members Fast

Problem: New hire doesn't know the codebase.

Solution: Cursor becomes their guide.

Onboarding workflow:

1. New hire installs Cursor
2. Clone the repo
3. Ask Cursor: "Give me an overview of this codebase"
4. Ask: "Where is user authentication handled?"
5. Ask: "How do I add a new API endpoint?"

They're productive in hours, not weeks.

Page 22: Code Style Enforcement

43 — Use .cursorrules for Style Consistency

Problem: Every dev codes differently.

Solution: Define style in .cursorrules.

Example rules:

- Functions must have JSDoc comments
- All variables use camelCase
- No magic numbers — use named constants
- Database queries must have error handling
- API responses follow {status, data, error} format

Cursor enforces these automatically.

---

44 — Lint Before Accepting AI Code

Best practice: Run linter on AI-generated code before merging.

Workflow:

1. Cursor generates code
2. Save file
3. Linter runs (ESLint, Prettier, etc.)
4. Fix lint errors
5. Commit

Pro tip: Configure Cursor to auto-format on save.

Page 23: Handling Large Codebases

45 — Cursor Gets Slow on Big Projects

Why: Indexing millions of lines takes time.

Solutions:

1. **Use .cursorignore:** Exclude node_modules, build folders, test data
2. **Close unused files:** Only keep relevant tabs open
3. **Split into smaller contexts:** Work on one module at a time
4. **Upgrade to Pro:** Faster indexing and responses

---

46 — Ask About Specific Modules

Instead of: "How does authentication work?"

Try: "How does authentication work in src/auth/login.js?"

Being specific = faster, more accurate answers.

Page 24: Version Control Best Practices

47 — Commit AI Code in Separate Commits

Why: Makes code review easier.

Workflow:

1. Human-written code !' commit
2. AI-generated code !' separate commit with clear message

Example commit messages:

- "Add user authentication (human-written)"
- "Add error handling to auth flow (AI-assisted)"

Reviewers know what to scrutinize.

---

48 — Use Branches for AI Experiments

Best practice: Don't let AI commit directly to main.

Workflow:

1. Create feature branch: `git checkout -b feature/ai-refactor`
2. Use Cursor to refactor
3. Test thoroughly
4. Code review
5. Merge to main

Protects main branch from AI mistakes.

Page 25: Testing AI-Generated Code

49 — AI Code Needs Tests (Duh)

Rule: Every AI-generated function gets a test.

Why: AI doesn't always consider edge cases.

Example:
AI generates a function that divides two numbers. Did it handle division by zero? Write a test to

find out.

---

50 — Ask Cursor to Write Tests

Technique: After generating code, immediately ask for tests.

Workflow:

1. Generate function with `Cmd+K`
2. Highlight the function
3. Press `Cmd+L`
4. Ask: "Write unit tests for this function"

Cursor generates test cases based on the code.

Review the tests. AI might miss edge cases.

Page 26: Security Considerations

51 — Don't Paste Secrets into Cursor

Bad:

```
const API_KEY = "sk-proj-abc123def456..."
// Don't ask Cursor to refactor this
```

Good:

```
const API_KEY = process.env.API_KEY
// Safe to refactor — no actual secret exposed
```

Cursor sends highlighted code to the cloud. Keep secrets in environment variables.

---

52 — Review AI Code for Security Issues

Common AI security mistakes:

- SQL injection vulnerabilities
- XSS vulnerabilities
- Missing authentication checks
- Exposing sensitive data in logs

Always review AI code for security before deploying.

Page 27: Performance Optimization

53 — Ask Cursor to Optimize Slow Code

Scenario: Function is slow, you don't know why.

Steps:

1. Highlight the slow function
2. Press `Cmd+L`
3. Ask: "Why is this function slow? How can I optimize it?"

Example response:
> "This function makes N+1 database queries. Instead of querying inside the loop, fetch all records at once and use a lookup table."

Cursor suggests the optimization.

---

54 — Benchmark Before and After

Best practice: Measure performance before trusting AI optimizations.

Workflow:

1. Benchmark current code (e.g., `console.time()`)
2. Ask Cursor to optimize
3. Benchmark optimized code
4. Compare results

Sometimes AI "optimizations" are actually slower.

Page 28: Debugging Cursor Itself

55 — Check Cursor's Status

If something feels off:

1. Click your profile icon (top-right)
2. Select "Check for Updates"
3. Look at server status

Common issues:

- Outdated version !' update
- Server downtime !' wait
- Account issue !' check billing

---

56 — Clear Cursor Cache

When to do this: Cursor behaves strangely (wrong suggestions, crashes, etc.)

How:

1. Close Cursor
2. Delete cache folder:
- Mac: `~/Library/Application Support/Cursor`
- Windows: `%APPDATA%\Cursor`
- Linux: `~/.config/Cursor`
3. Restart Cursor

Note: You'll need to re-index your projects.

Page 29: Advanced Configuration

57 — Customize AI Model Settings

Location: Settings !' Cursor !' Model

Options:

- **Model temperature:** Higher = more creative, Lower = more deterministic
- **Max tokens:** How long responses can be
- **Context window:** How much code Cursor reads

Recommendation: Leave defaults unless you know what you're doing.

---

58 — Set Up Custom Keybindings

Why: Default shortcuts might conflict with your habits.

How:

1. Open Settings !' Keyboard Shortcuts
2. Search for "cursor"
3. Rebind `Cmd+K` or `Cmd+L` if needed

Example custom bindings:

- `Cmd+Shift+A` !' AI code generation
- `Cmd+Shift+Q` !' AI chat

Page 30: Language-Specific Tips (Part 1)

59 — Python: Use Type Hints

Why: Cursor generates better code when it knows types.

Bad:

```
def process_data(data):
```

Cursor doesn't know what 'data' is

pass

Good:
def process_data(data: list[dict]) -> dict:

Cursor knows it's a list of dicts

pass

Ask Cursor to add type hints to existing code.

---

60 — JavaScript: Specify ES Version

In .cursorrules:

- Use ES6 syntax (arrow functions, const/let, template literals)
- Avoid var keyword
- Use async/await instead of promises

Cursor will match your project's JavaScript style.

Continue to Pages 31-40 !’

This is part 3 of 4. Final pages 31-40 coming next.

Cursor AI Cliff Notes — Pages 31-40

Power User Techniques & Real-World Applications

Final Section of Cursor AI Guide
Pages 31-40 of 40

Page 31: Language-Specific Tips (Part 2)

61 — TypeScript: Leverage Strong Typing

Cursor excels with TypeScript because types provide context.

Example:

```
interface User {
id: number;
email: string;
role: 'admin' | 'user';
}

function processUser(user: User) {
// Cursor knows exactly what fields exist
// and what types they are
}
```

Ask Cursor: "Add a function to validate User objects" — it will use the interface.

---

62 — React: Component Best Practices

In .cursorrules for React projects:

- Use functional components with hooks
- No class components
- Props must be typed with TypeScript interfaces
- Use descriptive prop names
- Each component in its own file
- Export default at bottom of file

Cursor will generate React components that match your team's style.

Page 32: Go-Specific Workflows

63 — Go: Error Handling Patterns

Tell Cursor your error handling style:

.cursorrules:

- Always check errors immediately after function calls
- Use fmt.Errorf for error wrapping
- Log errors before returning them
- No panic() in production code

Example prompt: "Add error handling to this database query"

Cursor generates Go-style error checks automatically.

---

64 — Go: Struct and Interface Generation

Scenario: You need a new struct and interface.

Prompt:
Create a User struct with fields: ID (int), Email (string), CreatedAt (time.Time)
Also create a UserRepository interface with methods: Create, GetByID, Update, Delete

Cursor generates both, properly formatted.

Page 33: Rust-Specific Workflows

65 — Rust: Ownership and Borrowing

Cursor understands Rust's ownership rules (mostly).

When Cursor makes mistakes:

- It might use `.clone()` too much (inefficient)
- It might not use lifetimes correctly

Fix: Review for unnecessary clones, adjust lifetimes manually.

Prompt tip: "Optimize this code to avoid unnecessary clones"

---

66 — Rust: Error Handling with Result<T, E>

Tell Cursor your error handling style:

.cursorrules:

- Use Result<T, E> for functions that can fail
- Use ? operator for error propagation
- Custom error types for domain errors
- No unwrap() or expect() in production code

Cursor will generate idiomatic Rust error handling.

Page 34: Database Operations

67 — Generate SQL Queries Safely

Always ask for parameterized queries to avoid SQL injection.

Bad prompt: "Write a query to get user by email"

Good prompt: "Write a parameterized SQL query to get user by email (prevent SQL injection)"

Result:

```
-- Bad (vulnerable)
SELECT * FROM users WHERE email = '$email';

-- Good (parameterized)
SELECT * FROM users WHERE email = $1;
```

---

68 — ORM Code Generation

Scenario: You use an ORM (Sequelize, SQLAlchemy, Prisma, etc.)

In .cursorrules:

- Use Prisma for database operations
- Always include error handling
- Use transactions for multi-step operations

Prompt: "Create a Prisma query to update user email"

Cursor generates ORM-specific code matching your library.

Page 35: API Development

69 — Generate REST Endpoints

Prompt format:

Create a REST endpoint:

- Route: POST /api/users
- Body: { email, password, name }
- Response: 201 with user object, or 400 with error
- Include validation and error handling

Cursor generates the full endpoint with proper status codes.

---

70 — API Documentation Comments

After generating an endpoint, ask Cursor to document it:

Prompt: "Add OpenAPI/Swagger comments to this endpoint"

Result:

/**

- @swagger
- /api/users:
- post:
- summary: Create a new user
- requestBody:
- required: true
- content:
- application/json:
- schema:
- type: object
- properties:
- email:
- type: string

*/

Documentation stays in sync with code.

Page 36: Frontend-Specific Tips

71 — CSS and Styling

Cursor can generate CSS, but specify your framework.

In .cursorrules:

- Use Tailwind CSS for styling
- No inline styles

- Responsive by default (mobile-first)

Prompt: "Style this button component with Tailwind"

Cursor generates Tailwind classes instead of raw CSS.

---

72 — State Management

Tell Cursor your state management library:

.cursorrules:

- Use React Context API for global state
- No Redux
- Keep state close to where it's used

Prompt: "Create a Context provider for user authentication"

Cursor generates code matching your stack.

Page 37: Testing Strategies

73 — Generate Unit Tests Automatically

Workflow:

1. Write a function
2. Highlight it
3. Press `Cmd+L`
4. Ask: "Write unit tests for this function (include edge cases)"

Review the tests. AI might miss:

- Null/undefined inputs
- Empty arrays/objects
- Race conditions
- Async timing issues

Add tests for cases AI missed.

---

74 — Integration Test Scaffolding

Prompt:
Create integration tests for this API endpoint:

- Test successful user creation
- Test duplicate email error
- Test invalid email format

- Test missing required fields

Cursor generates the test structure. You fill in assertions.

Page 38: Real-World Case Studies

75 — Case Study: Refactoring Legacy Code

Scenario: 500-line function, nobody understands it.

Workflow:

1. Open the file
2. Highlight the function
3. Ask: "Explain what this function does"
4. Ask: "Break this into smaller functions"
5. Review and test each piece

Time saved: Hours of reverse-engineering.

---

76 — Case Study: Learning a New Framework

Scenario: You need to learn Next.js for a new project.

Workflow:

1. Clone a Next.js starter
2. Ask Cursor: "How do I add a new page in Next.js?"
3. Ask: "How do I fetch data server-side in Next.js?"
4. Ask: "How do I handle authentication in Next.js?"

Cursor teaches you the framework as you build.

Page 39: Common Pitfalls & How to Avoid Them

77 — Pitfall: Over-Relying on AI

Problem: You stop thinking, just accept AI code blindly.

Solution:

- Always review AI code
- Understand what it does before merging
- If you don't understand it, ask Cursor to explain
- Treat AI like a junior dev (helpful but needs oversight)

---

78 — Pitfall: Vague Prompts

Problem: "Make this better" !’ AI guesses, gets it wrong.

Solution: Be specific.

- "Add input validation for email format"
- "Optimize this loop to reduce time complexity"
- "Refactor to use async/await instead of callbacks"

Specificity = better results.

---

79 — Pitfall: Ignoring Edge Cases

Problem: AI code works for happy path, breaks on edge cases.

Solution:

- Ask explicitly: "What edge cases should I handle?"
- Write tests for edge cases
- Review AI code with a critical eye

---

80 — Pitfall: Not Updating .cursorrules

Problem: Team conventions change, Cursor still uses old patterns.

Solution:

- Update .cursorrules when you change conventions
- Commit it to version control
- Review periodically (quarterly)

Page 40: What's Next & Future Features

81 — Cursor API (Coming Soon)

What it is: Call Cursor AI from command line or scripts.

Use cases:

- Automate code generation in CI/CD
- Batch refactoring across repos
- Generate boilerplate for new projects

Status: Currently in private beta. Public release TBD.

---

82 — Multimodal Support (Future)

What it means: Show Cursor a UI screenshot, ask it to generate the code.

Example:

- Take screenshot of a design mockup
- Ask: "Generate React components for this UI"
- Cursor outputs the code

Status: Experimental, not yet released.

---

83 — Voice Input (Possible Future)

Imagine: Describe code changes out loud, Cursor writes it.

Example: "Add error handling to the login function" (spoken, not typed)

Status: Not officially announced, but AI voice integration is trending.

---

84 — Join the Community

Where to learn more:

- **Cursor Discord:** discord.gg/cursor — Ask questions, share tips
- **Cursor Twitter:** @cursor_ai — Updates and announcements
- **Cursor GitHub:** github.com/getcursor/cursor — Report bugs, request features
- **Cursor Docs:** cursor.sh/docs — Official documentation

Pro tip: Follow @cursor_ai on Twitter for early feature announcements.

Final Thoughts

85 — You're Now a Cursor Power User

You've learned:

- ' Core commands (Cmd+K, Cmd+L)
- ' Advanced prompting techniques
- ' Multi-file refactoring
- ' Custom project rules (.cursorrules)
- ' Team workflows
- ' Security best practices
- ' Debugging strategies

- ' Language-specific tips
- ' Real-world applications

What to do next:

1. Use Cursor daily on your actual projects
2. Experiment with .cursorrules for your team
3. Share what you learn with your team
4. Join the Cursor community
5. Keep this guide handy as a reference

Quick Reference Card

Essential Shortcuts:

- `Cmd+K` — Generate/edit code
- `Cmd+L` — Open AI chat
- `Esc` — Reject suggestion
- `Tab` — Accept suggestion

Key Files:

- `.cursorrules` — Project-specific AI rules
- `.cursorignore` — Files to exclude from AI

Best Practices:

- Be specific in prompts
- Review all AI code before merging
- Use .cursorrules for consistency
- Write tests for AI-generated code
- Keep secrets out of prompts

Thank You

Thank you for reading the complete Cursor AI Cliff Notes guide. If this saved you time, share it with your team.

Questions? Feedback? Find a bug in the guide?

Contact: support@meshuga.com

Want more guides like this?

Visit techcliffnotes.com for guides on:

- Claude AI
- Perplexity
- Wispr Flow
- And more trending tools

Cursor AI Cliff Notes — Complete 40-Page Guide

Published: February 19, 2026
Last Updated: February 19, 2026
Version: 1.0

Thank You

You now have everything you need to master Cursor AI.
Practice on a real project — that's the fastest way to learn.

Browse more guides at meshuga.com