Code-Cave: Gemini CLI Tips & Tricks

Free AI Coding Assistant in Your Terminal

Out The Mud Work Shops

🎯 Your Coding Mud with Gemini CLI

Your mud isn't paying for AI tools - it's the friction between your terminal and AI assistance. Gemini CLI is FREE (60 requests/min with Google account) and lives right in your terminal. Let's rise above subscription fatigue and context switching.

🚀 Why Gemini CLI for Developers

The Free Tier Advantage

- **60 requests/minute** with personal Google account
- **1,000 requests/day** limit
- **Gemini 2.5 Pro** access with 1M token context
- **No credit card** required
- **Built-in tools** included (file ops, shell, web search)

Terminal-First Benefits

```
# Instead of copy-paste between browser and terminal:
$ gemini
> Fix the failing test in src/auth/login.test.js

# Gemini reads the file, analyzes the error, and fixes it
```

💡 Core Prompting Principles

1. Use Context, Not Copy-Paste

```
❌ Bad: [Pasting entire file contents]

✅ Better: "Review the error handling in src/services/PaymentService.js
and improve it based on best practices"
```

2. Reference Multiple Files

```
"Compare the patterns in src/controllers/UserController.js
with src/controllers/OrderController.js and make them consistent"
```

3. Let Gemini See Your Project

```
"Analyze the project structure and suggest improvements
for better organization"
```

🔧 Built-in Tools Mastery

File Operations

```
# Read and analyze
> "What does the config/database.js file do?"

# Search across files
> "Find all TODO comments in the src directory"

# Batch updates
> "Update all import statements from '../utils' to '@/utils' in src/"
```

Shell Commands

```
# Run tests and analyze results
> "Run npm test and fix any failing tests"

# Check dependencies
> "Check for outdated packages and suggest updates"

# Git operations
> "Show me what changed in the last commit and suggest improvements"
```

Web Search Integration

```
# Research best practices
> "Search for the latest React 18 performance best practices
and apply them to src/components/"

# Debug errors
> "Search for this error: 'Cannot read property of undefined'
in context of our Redux setup"
```

📊 Real-World Test Cases

Test Case 1: API Endpoint Creation

```
> "Create a new REST endpoint for user profile updates:
- POST /api/users/:id/profile
- Validate input with Joi
- Follow the pattern in src/routes/userRoutes.js
- Include proper error handling
- Add tests to match our existing test structure"
```

Test Case 2: Refactoring Legacy Code

```
> "Refactor src/legacy/processPayment.js:
- Convert callbacks to async/await
- Add proper error handling
- Extract magic numbers to constants
- Add TypeScript types
- Maintain the same public API"
```

Test Case 3: Performance Optimization

```
> "Analyze src/components/Dashboard.jsx for performance issues:
- Check for unnecessary re-renders
- Suggest React.memo usage
- Find N+1 query problems
- Add performance logging
- Create before/after metrics"
```

🎨 Project-Wide Operations

Codebase Analysis

```
> "Analyze the entire codebase and create a report:
- Code duplication instances
- Inconsistent naming patterns
- Missing test coverage
- Overly complex functions (cyclomatic complexity > 10)
- Security vulnerabilities
Output as markdown"
```

Dependency Management

```
> "Review package.json:
- Find unused dependencies
- Suggest newer alternatives for deprecated packages
- Check for security vulnerabilities
- Recommend performance improvements"
```

Documentation Generation

```
> "Generate comprehensive documentation:
- API endpoints with examples
- Component props documentation
- Setup instructions
- Architecture overview
Based on the current codebase"
```

🧪 Test-Driven Development

Smart Test Generation

```
> "For src/services/EmailService.js:
- Generate unit tests covering all methods
- Include edge cases and error scenarios
- Mock external dependencies
- Aim for 100% coverage
- Follow the pattern in src/services/__tests__/"
```

Test Improvement

```
> "Review all tests in src/__tests__ and:
- Identify missing test cases
- Find tests that always pass
- Suggest better assertions
- Add performance benchmarks"
```

💎 Power User Workflows

The "Morning Review" Workflow

```
> "Morning review:
1. Check git status and summarize changes
2. Run all tests and report failures
3. Search for new TODO comments
4. Check for console.log statements
5. List files changed in last 24 hours"
```

The "Pre-Commit" Workflow

```
> "Pre-commit check:
1. Lint all staged files
2. Run related tests
3. Check for security issues
4. Verify no sensitive data in code
5. Generate commit message based on changes"
```

The "Code Review" Assistant

```
> "Review the diff between main and feature/user-auth:
- Security vulnerabilities
- Performance concerns
- Code style violations
- Missing tests
- Suggest improvements"
```

🔄 Iterative Development

Progressive Enhancement

```
# Start simple
> "Create a basic user model"

# Add validation
```

```
> "Add validation to the user model"

# Add relationships
> "Add relationship methods to connect with posts"

# Optimize
> "Optimize the model for better query performance"
```

Debugging Sessions

```
> "Debug mode for src/api/auth.js:
1. Add console.log statements at key points
2. Run the auth flow
3. Analyze the output
4. Fix identified issues
5. Clean up debug code"
```

⚡ Performance Tips

Batch Operations

```
# Process multiple files efficiently
> "For all React components in src/components:
- Add prop-types validation
- Convert class components to functional
- Add display names for debugging"
```

Memory-Efficient Requests

```
# Break large tasks into chunks
> "Analyze src/data/ directory in chunks of 10 files
to find optimization opportunities"
```

🚫 Common Pitfalls to Avoid

1. **Don't Ignore Context**: Gemini needs to understand your project structure
2. **Don't Skip Verification**: Always test generated code
3. **Don't Overload Requests**: Break complex tasks into steps
4. **Don't Forget Project Rules**: Set up GEMINI.md first
5. **Don't Waste Requests**: Plan before prompting

📈 Measuring Success

You're mastering Gemini CLI when:

- ✅ Development speed increases by 40%+
- ✅ Less context switching to browser
- ✅ Consistent code style across team
- ✅ Tests written alongside features
- ✅ Documentation stays current

🎯 Daily Challenge Ideas

Monday: Refactoring Day

```
> "Find the most complex function in src/ and refactor it
for better readability while maintaining functionality"
```

Tuesday: Test Tuesday

```
> "Find a file with < 80% test coverage and bring it to 100%"
```

Wednesday: Performance Wednesday

```
> "Profile the slowest endpoint and optimize it by 50%"
```

Thursday: Documentation Day

```
> "Document one undocumented module completely"
```

Friday: Cleanup Friday

```
> "Find and fix all ESLint warnings in the codebase"
```

🔥 Advanced Techniques

Multi-Step Workflows

```
> "Create a complete feature:
1. Design the database schema
2. Create migration files
3. Build the API endpoints
4. Add validation
5. Write tests
6. Generate documentation
For a 'user favorites' feature"
```

AI-Powered Code Reviews

```
> "Act as a senior developer and review src/new-feature/:
- Architecture decisions
- Security concerns
- Performance implications
- Maintainability
- Test coverage
Provide actionable feedback"
```

Learning Mode

```
> "Explain the architecture of this project as if
to a new team member, including:
- Key design patterns used
- Important conventions
- Common pitfalls
- Best practices specific to this codebase"
```

🆓 Maximizing the Free Tier

Request Optimization

- Bundle related questions
- Use specific file references
- Cache responses for common queries
- Plan your 60 requests/minute wisely

Team Collaboration

```
# Share discoveries via GEMINI.md
> "Document the patterns you just helped me implement
in .gemini/GEMINI.md for the team"
```

Remember: Gemini CLI is your free AI pair programmer living in your terminal. No subscriptions, no browser tabs, just intelligent assistance where you work.

Rise above the paywall. Code with intelligence, not invoices.