
Agentic Coding with Gemini CLI
Google’s AI Agent in Your Terminal
Press Space for next page →

Contact Info

Ken Kousen Kousen IT, Inc.

- ken.kousen@kousenit.com
- http://www.kousenit.com
- http://kousenit.org (blog)
- Social Media:
  - @kenkousen (twitter)
  - @kenkousen@foojay.social (mastodon)
  - @kousenit.com (bluesky)
- *Tales from the jar side* (free newsletter)
  - https://kenkousen.substack.com
  - https://youtube.com/@talesfromthejarside

Course Overview

- **Duration**: 5 hours of hands-on learning
- **Format**: Instructor-led with multiple labs
- **Hands-on Labs**: Real codebases in Python, JavaScript, Java
- **Prerequisites**: Command-line experience, development background

Topics Covered

- **Foundation**: Installation, CLI basics, authentication
- **Core Skills**: File operations, shell integration, context management
- **Customization**: GEMINI.md, custom commands, settings.json
- **Safety**: Sandbox mode, approval policies, checkpointing
- **Advanced**: MCP integration, extensions, session management

What is Gemini CLI?

- Open-source AI agent from Google
- **Gemini 3 Pro** (latest) or Gemini 2.5 Pro/Flash
- Built-in tools: Google Search, file ops, shell, web fetch
- Model Context Protocol (MCP) support
- Designed for developers who live in the terminal

Key Differentiators

- **Massive Context**: 1 million token context window
- **Google Search Grounding**: Real-time web access
- **Free Tier Available**: Generous API limits
- **Open Source**: Fully open, community-driven
- **MCP Native**: Built-in Model Context Protocol support

Models Available

- **Gemini 3 Pro**: Most intelligent, best for complex coding
- **Gemini 2.5 Pro**: Strong performance, 1M token context
- **Gemini 2.5 Flash**: Faster, lower cost option
- **Auto routing**: CLI picks best model for each task

API Access

- **Free Tier**: Generous daily limits
- **Google AI Ultra**: Full Gemini 3 Pro access
- **Paid API Key**: Pay-as-you-go pricing
- Enable via `/settings` → "Preview features"

📖 **Get API Key**: Google AI Studio

Installation

- **npm** (recommended): `npm install -g @google/gemini-cli`
- **npx** (no install): `npx @google/gemini-cli`
- Verify: `gemini --version`
- Current version: 0.19.x

```
# Install globally
npm install -g @google/gemini-cli

# Verify installation
gemini --version
```

Authentication

- **Environment Variable**: `export GEMINI_API_KEY="your-key"`
- **Global .env file**: `~/.gemini/.env`
- **Project .env file**: `./.gemini/.env`
- **Google Cloud**: `GOOGLE_CLOUD_PROJECT` for Vertex AI

```
# Option 1: Environment variable
export GEMINI_API_KEY="your-api-key"

# Option 2: Global .env file
echo 'GEMINI_API_KEY=your-api-key' >> ~/.gemini/.env

# Option 3: Project .env file
mkdir -p .gemini && echo 'GEMINI_API_KEY=your-api-key' >> .gemini/.env
```

Basic Usage Modes

- **Interactive REPL**: `gemini` - Start a conversation
- **One-shot**: `gemini "prompt"` - Single response
- **Piped input**: `echo "task" | gemini`
- **Interactive with context**: `gemini -i "initial context"`

```
# Interactive mode
gemini

# One-shot mode
gemini "Explain what this codebase does"

# With initial context
gemini -i "You are a Python expert"
```

Core Features

Essential Capabilities

Master the fundamentals

File References with @

- Reference files directly: `@./src/main.js`
- Reference directories: `@./src/` (recursive)
- Reference images: `@./screenshot.png`
- Multiple references in one prompt

```
# Reference a specific file
gemini "Explain @./src/app.py"

# Reference multiple files
gemini "Compare @./old.js and @./new.js"

# Reference a directory
gemini "Analyze the architecture in @./src/"
```

Shell Integration with !

- Execute shell commands: `!git status`
- Toggle persistent shell mode: `!`
- Gemini can observe and analyze output
- Combine with AI analysis

```
# In interactive mode:
> !npm test
# Gemini sees the test output

> !git diff
# Ask Gemini to analyze the changes

# Toggle persistent shell mode
> !
```

Slash Commands: Navigation

- `/help` - Show available commands
- `/clear` - Clear conversation history
- `/memory show` - View loaded context
- `/memory refresh` - Reload GEMINI.md files

Slash Commands: Sessions

- `/init` - Generate project GEMINI.md
- `/chat save <tag>` - Save conversation
- `/compress` - Summarize conversation
- `/restore` - Recover from checkpoint

Slash Commands in Action

```
# Show what context is loaded
/memory show

# Reload all GEMINI.md files
/memory refresh

# Generate a GEMINI.md for current project
/init

# Save current conversation
/chat save feature-implementation

# Compress long conversation
/compress
```

Keyboard Shortcuts: Editing

- `Ctrl+L` - Clear screen
- `Ctrl+V` - Paste text/images
- `Ctrl+X` - Open external editor

Keyboard Shortcuts: Control

- `Ctrl+Y` - Toggle auto-approval (YOLO mode)
- `Ctrl+C` - Cancel current operation
- `Ctrl+D` - Exit Gemini CLI

Built-in Tools

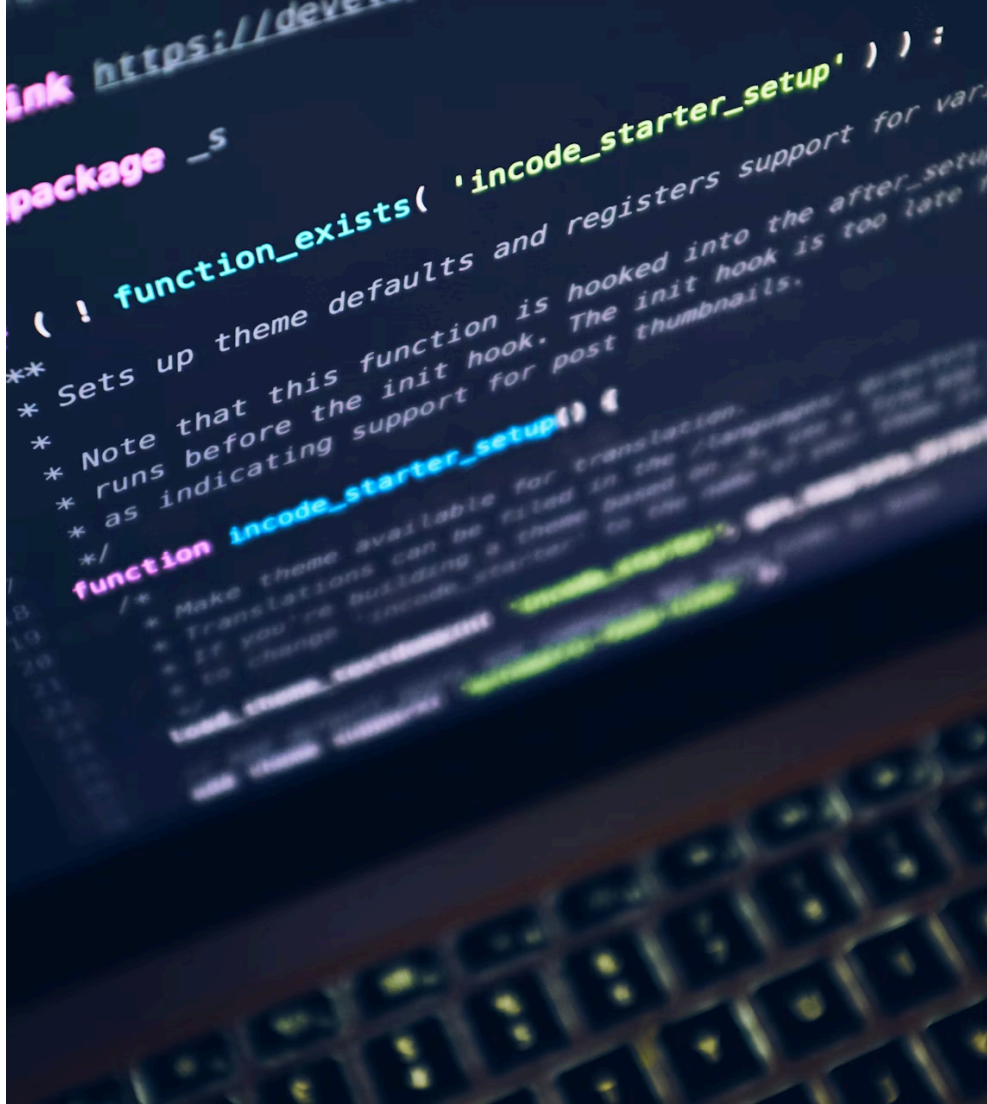
- **File System**: `read_file()`, `write_file()`, `replace()`, `glob()`
- **Shell**: Execute terminal commands
- **Web**: `google_web_search()`, `web_fetch()`
- **Memory**: `save_memory()` for cross-session recall

```
# Gemini automatically uses appropriate tools
"Search the web for React 19 new features"
# Uses google_web_search()

"Read all Python files in src/"
# Uses glob() and read_file()

"Update the README with the changes we made"
# Uses write_file()
```

Safety & Control

Work Safely

Approval modes and sandboxing



Approval Modes

- **default**: Prompt for approval on tool calls
- **auto_edit**: Auto-approve file edit tools only
- **yolo**: Auto-approve ALL tool calls

```
# Default - asks for approval
gemini

# Auto-approve edits only
gemini --approval-mode auto_edit

# Auto-approve everything (YOLO mode)
gemini --yolo
# Or use Ctrl+Y in interactive mode
```

Sandbox Mode

- Isolate file operations in a container
- Requires Docker or Podman
- Prevents accidental system changes
- Perfect for exploring unfamiliar code

```
# Run in sandbox mode
gemini --sandbox

# Or with -s flag
gemini -s "Refactor this entire codebase"
```

Checkpointing

- **Automatic**: Snapshots created before each file modification
- **Shadow Git**: Stored in `~/.gemini/history/` (not your repo)
- **Includes**: Files + conversation + tool call
- **Disabled by default**: Must enable in settings

```
// ~/.gemini/settings.json
{ "checkpointing": { "enabled": true } }
```

Restoring Checkpoints

```
# List and select a checkpoint to restore
/restore

# Shows timestamps + filename + tool name
# e.g., 2025-06-22T10-00-00_000Z-app.py-write_file
```

Restores files AND resets conversation to that point

Context Management

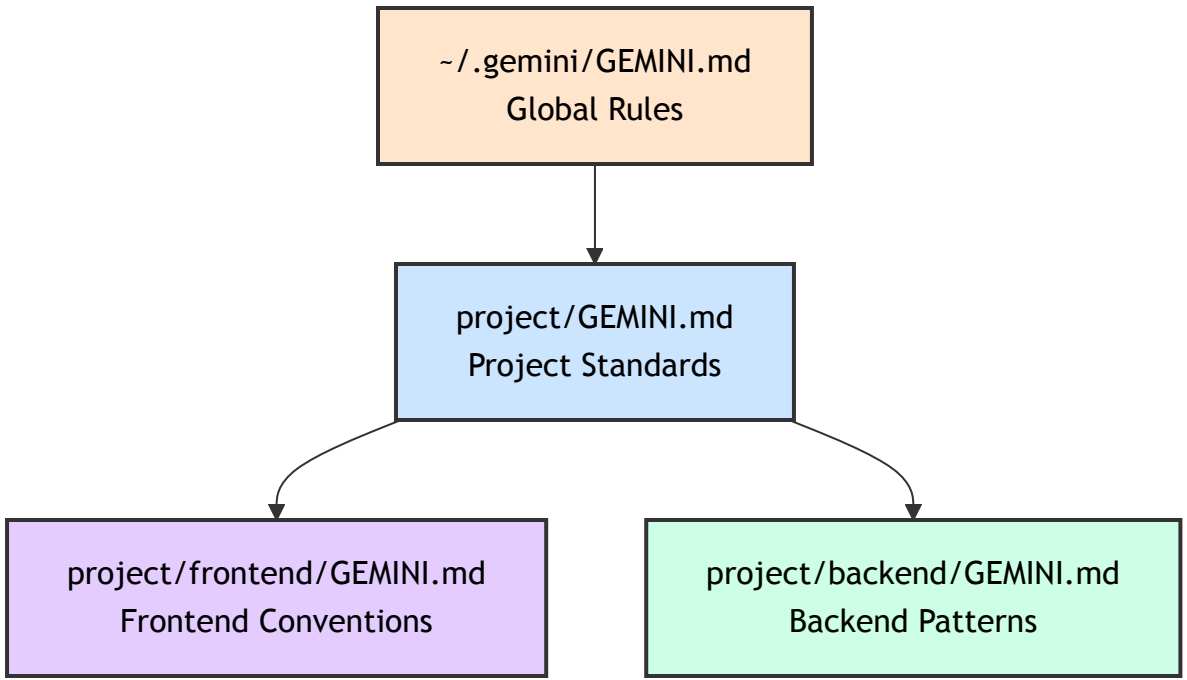
GEMINI.md Files

Project memory and instructions

What is GEMINI.md?

- **Project memory** loaded automatically
- **Coding standards** and conventions
- **Architecture context** for the AI
- **Persistent instructions** across sessions
- Similar to CLAUDE.md or AGENTS.md

Hierarchical Loading



More specific files override general ones

Example GEMINI.md

```
# Project: Weather API

## Tech Stack
- Backend: Python Flask
- Database: PostgreSQL
- Testing: pytest

## Coding Standards
- Use type hints for all functions
- Follow PEP 8 style guide
- Write docstrings for public APIs

## Current Focus
Implementing caching layer for API responses
```

Memory Commands

- `/memory show` - View combined context
- `/memory refresh` - Reload all GEMINI.md files
- `/memory add <text>` - Append to global GEMINI.md
- `/init` - Generate starter GEMINI.md

```
# See what context is loaded
/memory show

# Add a quick note to global memory
/memory add "Always use async/await for database calls"

# Generate a project-specific GEMINI.md
/init
```

Modular Imports

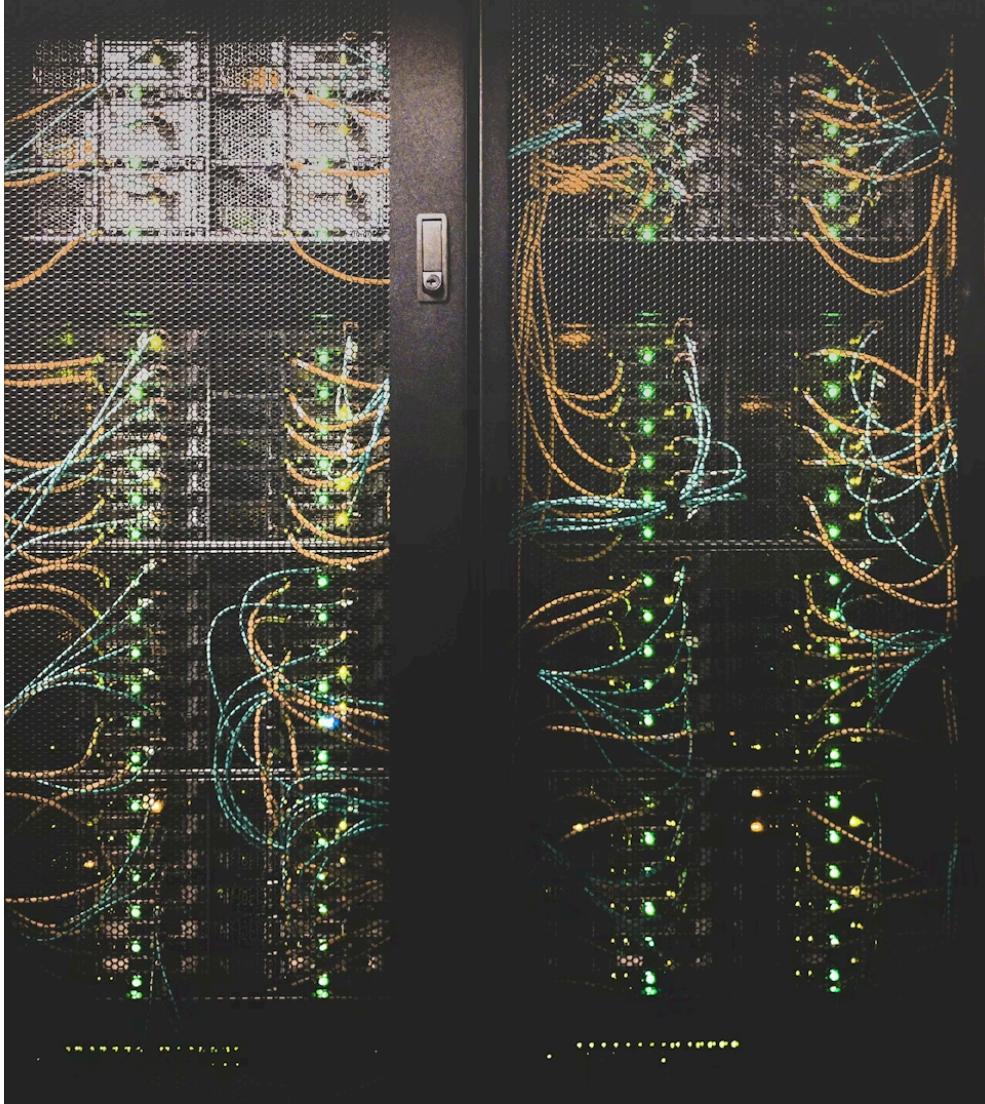
- Import other files with `@file.md` syntax
- Break large context into components
- Supports relative and absolute paths

```
# GEMINI.md

## Project Overview
@./docs/architecture.md

## Coding Standards
@./docs/style-guide.md

## API Documentation
@./docs/api-reference.md
```

Configuration

Customize Your Setup

settings.json and environment



Configuration Layers

1. **Default values** - Built-in defaults
2. **User settings** - `~/.gemini/settings.json`
3. **Project settings** - `.gemini/settings.json`
4. **Environment variables** - Including `.env` files
5. **Command-line arguments** - Highest priority

settings.json Options

```
{
  "theme": "Default",
  "vimMode": false,
  "hideTips": false,
  "hideBanner": false,
  "autoAccept": false,
  "sandbox": false,
  "checkpointing": true,
  "preferredEditor": "vscode"
}
```

Tool Configuration

- **coreTools**: Whitelist available tools
- **excludeTools**: Blacklist specific tools
- Restrict dangerous operations for safety

```
{
  "coreTools": ["read_file", "write_file", "glob"],
  "excludeTools": ["shell"]
}
```

File Filtering

- **respectGitIgnore**: Honor .gitignore patterns
- **enableRecursiveFileSearch**: Recursive completion
- **.geminiignore**: Custom ignore patterns

```
{
  "fileFiltering": {
    "respectGitIgnore": true,
    "enableRecursiveFileSearch": true
  }
}
```

Environment Variables

Variable	Purpose
`GEMINI_API_KEY`	API authentication (required)
`GEMINI_MODEL`	Override default model
`GOOGLE_CLOUD_PROJECT`	GCP project for Vertex AI
`GOOGLE_CLOUD_LOCATION`	GCP region
`HTTP_PROXY`	Network proxy

Custom Context Filename

- Change from `GEMINI.md` to custom name
- Support multiple filenames
- Include additional directories

```
{
  "context": {
    "fileName": ["CONTEXT.md", "GEMINI.md"],
    "includeDirectories": ["~/shared-context"],
    "loadFromIncludeDirectories": true
  }
}
```

Advanced Features

Power User Tools

MCP, Extensions, Sessions

Model Context Protocol (MCP)

- Standard protocol for AI-to-system connections
- Connect to external tools and services
- Supports local commands, HTTP, and SSE
- OAuth 2.0 for remote authentication

MCP Configuration: Firecrawl

```
{
  "mcpServers": {
    "firecrawl": {
      "command": "npx",
      "args": ["@modelcontextprotocol/server-firecrawl"],
      "env": {
        "FIRECRAWL_API_KEY": "${FIRECRAWL_API_KEY}"
      }
    }
  }
}
```

MCP Configuration: Database

```
{
  "mcpServers": {
    "postgres": {
      "command": "npx",
      "args": ["@modelcontextprotocol/server-postgres"],
      "env": {
        "CONNECTION_STRING": "${DATABASE_URL}"
      }
    }
  }
}
```

Managing MCP Servers

```
# List configured MCP servers
gemini mcp list

# Add a new MCP server
gemini mcp add github

# Remove an MCP server
gemini mcp remove github

# Test MCP server connection
gemini mcp test github
```

MCP Server Options

- **command**: Shell command to start server
- **args**: Command arguments
- **env**: Environment variables
- **cwd**: Working directory
- **timeout**: Startup timeout in ms
- **includeTools/excludeTools**: Filter available tools

Popular MCP Servers

- **Firecrawl** - Web scraping, search, content extraction
- **PostgreSQL** - Database queries and schema
- **Filesystem** - Extended file operations
- **Slack** - Team communication
- **Playwright** - Browser automation

📖 **Registry**: modelcontextprotocol.io/registry

Extensions System

- Extend Gemini CLI capabilities
- Place in `~/.gemini/extensions/`
- Configure with `gemini-extension.json`
- List with `gemini --list-extensions`

```
# List available extensions
gemini --list-extensions

# Use specific extensions only
gemini -e extension1 -e extension2

# Manage extensions
gemini extensions list
gemini extensions enable my-extension
```

Custom Commands

- Create reusable prompt templates
- Place TOML files in `~/.gemini/commands/`
- Access via slash command syntax

```toml
# ~/.gemini/commands/review.toml
[command]
name = "review"
description = "Review code for issues"

[prompt]
template = """
Review the following code for:
- Security vulnerabilities
- Performance issues
- Best practices violations

{{content}}
"""
```

Session Management

- **Resume sessions**: Continue previous conversations
- **List sessions**: View available sessions
- **Delete sessions**: Clean up old conversations

```
# Resume the latest session
gemini --resume latest

# Resume by index
gemini --resume 3

# List available sessions
gemini --list-sessions

# Delete a session
gemini --delete-session 5
```

Output Formats

- **text**: Default human-readable output
- **json**: Structured JSON for scripting
- **stream-json**: Streaming JSON for real-time

```
# JSON output for automation
gemini -o json "List all TODO comments"

# Stream JSON for real-time processing
gemini -o stream-json "Analyze this codebase"
```

IDE Integration

- **VS Code Integration**: Connect to workspace
- **Native diff viewing**: Review changes in editor
- **Context sharing**: IDE context available to Gemini

```
{
  "preferredEditor": "vscode"
}
```

Practical Applications

Real-World Workflows

Common use cases

Code Exploration

- Understand unfamiliar codebases
- Trace dependencies and data flow
- Find patterns and conventions
- Generate architecture documentation

```
gemini "Analyze the architecture of @./src/ and explain
how the components interact"

gemini "Trace the flow from the API endpoint to the database
in @./src/controllers/ and @./src/services/"
```

Test Generation

- Generate unit tests for existing code
- Identify edge cases automatically
- Create integration test scaffolding
- Mock setup and fixtures

```
gemini "Create comprehensive unit tests for @./src/utils.py
with pytest, including edge cases"

gemini "Generate integration tests for @./src/api/users.py
with proper mocking"
```

Documentation Generation

- README files for projects
- API documentation
- Architecture diagrams (Mermaid)
- Code comments and docstrings

```
gemini "Generate a comprehensive README.md for this project"

gemini "Add detailed docstrings to all public functions
in @./src/services/"

gemini "Create a Mermaid diagram showing the system architecture"
```

Refactoring & Modernization

- Upgrade legacy code patterns
- Apply modern language features
- Improve code organization
- Fix anti-patterns

```
gemini "Refactor @./src/legacy.py to use modern Python 3.12
features like type hints and match statements"

gemini "Convert this callback-based code to async/await
@./src/api.js"
```

Debugging Workflows

- Analyze error messages and stack traces
- Identify root causes
- Suggest fixes with context
- Test and verify solutions

```
gemini "This test is failing with @./tests/output.log.
Analyze the error and fix the issue in @./src/app.py"

gemini "Debug why the API returns 500 errors.
Check @./src/routes.py and @./src/middleware.py"
```

Git Workflows

- Generate commit messages
- Create pull request descriptions
- Analyze diffs and changes
- Resolve merge conflicts

```
# Analyze staged changes
!git diff --staged
"Generate a conventional commit message for these changes"

# Create PR description
"Create a pull request description summarizing the changes
from the last 5 commits"
```

CI/CD Integration

- Non-interactive mode for pipelines
- JSON output for parsing
- Exit codes for success/failure
- Automated code reviews

```
# In CI/CD pipeline
gemini "Review @./src/ for security issues" -o json > review.json

# Check exit code
if gemini "Verify all tests pass" -o json; then
  echo "All checks passed"
fi
```

Best Practices

Professional Workflows

Tips for success

Effective Prompting

- **Be specific** about what you want to achieve
- **Provide context** about goals and constraints
- **Use file references** to ground the conversation
- **Iterate** for complex tasks
- **Include examples** when possible

Safety Guidelines

- **Start in default mode** - Get comfortable first
- **Enable checkpointing** - Safety net for mistakes
- **Review generated code** - Don’t blindly accept
- **Use sandbox** for exploratory work
- **Commit often** - Git is your safety net

Project Setup

1. Create comprehensive GEMINI.md
2. Set up project-specific settings.json
3. Configure relevant MCP servers
4. Create custom commands for common tasks
5. Establish team conventions

GEMINI.md Best Practices

- **Keep it focused** - Relevant project info only
- **Update regularly** - Reflect current state
- **Use imports** - Modularize large contexts
- **Include examples** - Show expected patterns
- **Document conventions** - Style guides, patterns

Team Collaboration

- Share GEMINI.md in version control
- Standardize settings.json across team
- Create shared custom commands
- Document AI-assisted workflows
- Review AI-generated code together

Common Pitfalls

- **Overly broad prompts** → Be specific
- **Missing context** → Use GEMINI.md
- **Skipping review** → Always verify output
- **YOLO mode too early** → Build trust first
- **Ignoring checkpoints** → Enable early

Troubleshooting

- **Authentication issues**: Check `GEMINI_API_KEY`
- **Rate limits**: Use appropriate tier
- **Tool failures**: Check MCP server status
- **Context not loading**: Run `/memory refresh`
- **Debug mode**: Use `gemini -d` for details

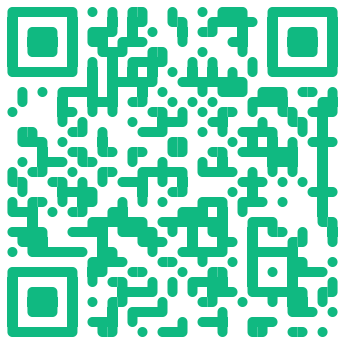
```
# Debug mode for troubleshooting
gemini -d

# Check memory/context
/memory show

# Refresh context files
/memory refresh
```

Quick Access

Gemini CLI Docs

github.com/google-gemini/gemini-cli

Course Repository



github.com/kousen/gemini-training

Important Links

📚 Official Documentation

`https://github.com/google-gemini/gemini-cli`

🔑 Get API Key

`https://aistudio.google.com/apikey`

📦 MCP Server Registry

`https://modelcontextprotocol.io/registry`

💻 Course Materials

`https://github.com/kousen/gemini-training`

Command Reference: Basic Usage

```
# Interactive mode
gemini

# One-shot mode
gemini "prompt"

# Interactive with initial context
gemini -i "context"
```

Command Reference: Safety

```
# Run in sandbox mode
gemini --sandbox

# Auto-approve all tool calls
gemini --yolo

# Auto-approve file edits only
gemini --approval-mode auto_edit
```

Command Reference: Sessions & Output

```
# Resume last session
gemini --resume latest

# List available sessions
gemini --list-sessions

# JSON output for scripting
gemini -o json "prompt"

# Debug mode
gemini -d
```

Thank You!

Questions?

Kenneth Kousen*Author, Speaker, Java & AI Expert*

kousenit.com | @kenkousen