# \*\*Comprehensive SuperClaude Configuration Guide\*\*

Based on analysis of Claude configuration files, here's a complete guide on what to use with Claude, when, and where.

## \*\*🎯 Overview\*\*

SuperClaude is a sophisticated AI assistant framework with 18 commands, 4 MCP servers, 9 personas, and extensive optimization patterns. It's designed for evidence-based development with security, performance, and quality as core principles.

---

## \*\*🔧 Core System Components\*\*

### \*\*1. Main Configuration Files\*\*

- \*\*`.claude/settings.local.json`\*\* - Basic Claude permissions and settings

- \*\*`.claude/shared/superclaude-core.yml`\*\* - Core philosophy, standards, and workflows

- \*\*`.claude/shared/superclaude-mcp.yml`\*\* - MCP server integration details

- \*\*`.claude/shared/superclaude-rules.yml`\*\* - Development practices and rules

- \*\*`.claude/shared/superclaude-personas.yml`\*\* - 9 specialized personas

### \*\*2. Command Architecture\*\*

- \*\*18 Core Commands\*\* with intelligent workflows

- \*\*Universal Flag System\*\* with inheritance patterns

- \*\*Task Management\*\* with two-tier architecture

- \*\*Performance Optimization\*\* including UltraCompressed mode

---

## \*\*🎭 Personas: When & Where to Use\*\*

### \*\*Development Personas\*\*

```yaml

--persona-frontend: "UI/UX focus, accessibility, React/Vue components"

When: Building user interfaces, design systems, accessibility work

Use with: Magic MCP, Puppeteer testing, --magic flag

--persona-backend: "API design, scalability, reliability engineering"

When: Building APIs, databases, server architecture

Use with: Context7 for patterns, --seq for complex analysis

--persona-architect: "System design, scalability, long-term thinking"

When: Designing architecture, making technology decisions

Use with: Sequential MCP, --ultrathink for complex systems

```

### \*\*Quality Personas\*\*

```yaml

--persona-analyzer: "Root cause analysis, evidence-based investigation"

When: Debugging complex issues, investigating problems

Use with: All MCPs for comprehensive analysis

--persona-security: "Threat modeling, vulnerability assessment"

When: Security audits, compliance, penetration testing

Use with: --scan --security, Sequential for threat analysis

--persona-qa: "Testing, quality assurance, edge cases"

When: Writing tests, quality validation, coverage analysis

Use with: Puppeteer for E2E testing, --coverage flag

--persona-performance: "Optimization, profiling, bottlenecks"

When: Performance issues, optimization opportunities

Use with: Puppeteer metrics, --profile flag

```

### \*\*Improvement Personas\*\*

```yaml

--persona-refactorer: "Code quality, technical debt, maintainability"

When: Cleaning up code, reducing technical debt

Use with: --improve --quality, Sequential analysis

--persona-mentor: "Teaching, documentation, knowledge transfer"

When: Creating tutorials, explaining concepts, onboarding

Use with: Context7 for official docs, --depth flag

```

---

## \*\*🔌 MCP Servers: Capabilities & Usage\*\*

### \*\*Context7 (Library Documentation)\*\*

```yaml

Purpose: "Official library documentation & examples"

When\_to\_Use:

- External library integration

- API documentation lookup

- Framework pattern research

- Version compatibility checking

Command\_Examples:

- "/analyze --c7" (research library patterns)

- "/build --react --c7" (React with official docs)

- "/explain --c7" (official documentation explanations)

Best\_For: "Research-first methodology, evidence-based implementation"

Token\_Cost: "Low-Medium"

```

### \*\*Sequential (Complex Analysis)\*\*

```yaml

Purpose: "Multi-step problem solving & architectural thinking"

When\_to\_Use:

- Complex system design

- Root cause analysis

- Performance investigation

- Architecture review

Command\_Examples:

- "/analyze --seq" (deep system analysis)

- "/troubleshoot --seq" (systematic investigation)

- "/design --seq --ultrathink" (architectural planning)

Best\_For: "Complex technical analysis, systematic reasoning"

Token\_Cost: "Medium-High"

```

### \*\*Magic (UI Components)\*\*

```yaml

Purpose: "UI component generation & design system integration"

When\_to\_Use:

- React/Vue component building

- Design system implementation

- UI pattern consistency

- Rapid prototyping

Command\_Examples:

- "/build --react --magic" (component generation)

- "/design --magic" (UI design systems)

- "/improve --accessibility --magic" (accessible components)

Best\_For: "Consistent design implementation, quality components"

Token\_Cost: "Medium"

```

### \*\*Puppeteer (Browser Automation)\*\*

```yaml

Purpose: "E2E testing, performance validation, browser automation"

When\_to\_Use:

- End-to-end testing

- Performance monitoring

- Visual validation

- User interaction testing

Command\_Examples:

- "/test --e2e --pup" (E2E testing)

- "/analyze --performance --pup" (performance metrics)

- "/scan --validate --pup" (visual validation)

Best\_For: "Quality assurance, performance validation, UX testing"

Token\_Cost: "Low (action-based)"

```

---

## \*\*⚡ Key Commands & When to Use\*\*

### \*\*Analysis Commands\*\*

```yaml

/analyze: "Comprehensive codebase analysis"

Flags: --code --arch --security --performance --c7 --seq

When: Understanding codebase, identifying issues, research

/troubleshoot: "Systematic problem investigation"

Flags: --investigate --seq --evidence

When: Debugging complex issues, root cause analysis

/scan: "Security, quality, and compliance scanning"

Flags: --security --owasp --deps --validate

When: Security audits, vulnerability assessment

```

### \*\*Development Commands\*\*

```yaml

/build: "Feature implementation & project creation"

Flags: --init --feature --react --api --magic --tdd

When: Building features, creating projects, implementing

/design: "Architectural design & system planning"

Flags: --api --ddd --microservices --seq --ultrathink

When: System architecture, API design, planning

/test: "Comprehensive testing & validation"

Flags: --coverage --e2e --pup --validate

When: Quality assurance, test coverage, validation

```

### \*\*Quality Commands\*\*

```yaml

/improve: "Code quality & performance optimization"

Flags: --quality --performance --security --iterate

When: Refactoring, optimization, quality improvements

/cleanup: "Technical debt & maintenance"

Flags: --code --all --dry-run

When: Removing unused code, cleaning up technical debt

```

### \*\*Operations Commands\*\*

```yaml

/deploy: "Production deployment & operations"

Flags: --env --validate --monitor --checkpoint

When: Deploying to production, operational tasks

/migrate: "Data & schema migrations"

Flags: --database --validate --dry-run --rollback

When: Database changes, data migrations

```

---

## \*\*🎛 Universal Flags: Always Available\*\*

### \*\*Planning & Execution\*\*

```yaml

--plan: "Show execution plan before running"

--dry-run: "Preview changes without execution"

--force: "Override safety checks"

--interactive: "Step-by-step guided process"

```

### \*\*Thinking Modes\*\*

```yaml

--think: "Multi-file analysis (4K tokens)"

--think-hard: "Deep architectural analysis (10K tokens)"

--ultrathink: "Critical system redesign (32K tokens)"

```

### \*\*Compression & Performance\*\*

```yaml

--uc: "UltraCompressed mode (~70% token reduction)"

--profile: "Detailed performance profiling"

--watch: "Continuous monitoring"

```

### \*\*MCP Control\*\*

```yaml

--c7: "Enable Context7 documentation lookup"

--seq: "Enable Sequential complex analysis"

--magic: "Enable Magic UI component generation"

--pup: "Enable Puppeteer browser automation"

--all-mcp: "Enable all MCP servers"

--no-mcp: "Disable all MCP servers"

```

---

## \*\*📋 Task Management System\*\*

### \*\*Two-Tier Architecture\*\*

```yaml

Level\_1\_Tasks: "High-level features (./claudedocs/tasks/)"

Purpose: "Session persistence, git branching, requirement tracking"

Scope: "Features spanning multiple sessions"

Level\_2\_Todos: "Immediate actionable steps (TodoWrite/TodoRead)"

Purpose: "Real-time execution tracking within session"

Scope: "Current session specific actions"

```

### \*\*Auto-Trigger Rules\*\*

```yaml

Complex\_Operations: "3+ steps → Auto-trigger TodoList"

High\_Risk: "Database changes, deployments → REQUIRE todos"

Long\_Tasks: "Over 30 minutes → AUTO-TRIGGER todos"

Multi\_File: "6+ files → AUTO-TRIGGER for coordination"

```

---

## \*\*🔒 Security Configuration\*\*

### \*\*OWASP Top 10 Integration\*\*

- \*\*A01-A10 Coverage\*\* with automated detection patterns

- \*\*CVE Scanning\*\* for known vulnerabilities

- \*\*Dependency Security\*\* with license compliance

- \*\*Configuration Security\*\* including hardcoded secrets detection

### \*\*Security Command Usage\*\*

```yaml

/scan --security --owasp: "Full OWASP Top 10 scan"

/analyze --security --seq: "Deep security analysis"

/improve --security --harden: "Security hardening"

```

---

## \*\*⚡ Performance Optimization\*\*

### \*\*UltraCompressed Mode\*\*

```yaml

Activation: "--uc flag | 'compress' keywords | Auto at >75% context"

Benefits: "~70% token reduction | Faster responses | Cost efficiency"

Use\_When: "Large codebases | Long sessions | Token budget constraints"

```

### \*\*MCP Caching\*\*

```yaml

Context7: "1 hour TTL | Library documentation"

Sequential: "Session duration | Analysis results"

Magic: "2 hours TTL | Component templates"

Parallel\_Execution: "Independent MCP calls run simultaneously"

```

---

## \*\*🚀 Quick Start Workflows\*\*

### \*\*New Project Setup\*\*

```bash

/build --init --feature --react --magic --c7

# Creates React project with Magic components and Context7 documentation

```

### \*\*Security Audit\*\*

```bash

/scan --security --owasp --deps --strict

/analyze --security --seq

/improve --security --harden

```

### \*\*Performance Investigation\*\*

```bash

/analyze --performance --pup --profile

/troubleshoot --seq --evidence

/improve --performance --iterate

```

### \*\*Feature Development\*\*

```bash

/analyze --code --c7

/design --api --seq

/build --feature --tdd --magic

/test --coverage --e2e --pup

```

---

## \*\*📊 Best Practices\*\*

### \*\*Evidence-Based Development\*\*

- \*\*Required Language\*\*: "may|could|potentially|typically|measured|documented"

- \*\*Prohibited Language\*\*: "best|optimal|faster|secure|better|always|never"

- \*\*Research Standards\*\*: Context7 for external libraries, official sources required

### \*\*Quality Standards\*\*

- \*\*Git Safety\*\*: Status→branch→fetch→pull workflow

- \*\*Testing\*\*: TDD patterns, comprehensive coverage

- \*\*Security\*\*: Zero tolerance for vulnerabilities

### \*\*Performance Guidelines\*\*

- \*\*Simple→Sonnet | Complex→Sonnet-4 | Critical→Opus-4\*\*

- \*\*Native tools > MCP for simple tasks\*\*

- \*\*Parallel execution for independent operations\*\*

---

## \*\*🎯 When to Use What: Decision Matrix\*\*

| \*\*Scenario\*\* | \*\*Persona\*\* | \*\*MCP\*\* | \*\*Command\*\* | \*\*Flags\*\* |

|--------------|-------------|---------|-------------|-----------|

| \*\*New React Feature\*\* | `--persona-frontend` | `--magic --c7` | `/build --feature` | `--react --tdd` |

| \*\*API Design\*\* | `--persona-architect` | `--seq --c7` | `/design --api` | `--ddd --ultrathink` |

| \*\*Security Audit\*\* | `--persona-security` | `--seq` | `/scan --security` | `--owasp --strict` |

| \*\*Performance Issue\*\* | `--persona-performance` | `--pup --seq` | `/analyze --performance` | `--profile --iterate` |

| \*\*Bug Investigation\*\* | `--persona-analyzer` | `--all-mcp` | `/troubleshoot` | `--investigate --seq` |

| \*\*Code Cleanup\*\* | `--persona-refactorer` | `--seq` | `/improve --quality` | `--iterate --threshold` |

| \*\*E2E Testing\*\* | `--persona-qa` | `--pup` | `/test --e2e` | `--coverage --validate` |

| \*\*Documentation\*\* | `--persona-mentor` | `--c7` | `/document --user` | `--examples --visual` |

| \*\*Production Deploy\*\* | `--persona-security` | `--seq` | `/deploy --env prod` | `--validate --monitor` |

---

## \*\*🔍 Advanced Configuration Details\*\*

### \*\*Core Philosophy\*\*

```yaml

Philosophy: "Code>docs | Simple→complex | Security→evidence→quality"

Communication: "Format | Symbols: →|&|:|» | Structured>prose"

Workflow: "TodoRead()→TodoWrite(3+)→Execute | Real-time tracking"

Stack: "React|TS|Vite + Node|Express|PostgreSQL + Git|ESLint|Jest"

```

### \*\*Evidence-Based Standards\*\*

```yaml

Prohibited\_Language: "best|optimal|faster|secure|better|improved|enhanced|always|never|guaranteed"

Required\_Language: "may|could|potentially|typically|often|sometimes|measured|documented"

Evidence\_Requirements: "testing confirms|metrics show|benchmarks prove|data indicates|documentation states"

Citations: "Official documentation required | Version compatibility verified | Sources documented"

```

### \*\*Token Economy & Optimization\*\*

```yaml

Model\_Selection: "Simple→sonnet | Complex→sonnet-4 | Critical→opus-4"

Optimization\_Targets: "Efficiency | Evidence-based responses | Structured deliverables"

Template\_System: "@include shared/\*.yml | 70% reduction achieved"

Symbols: "→(leads to) |(separator) &(combine) :(define) »(sequence) @(location)"

```

### \*\*Intelligent Auto-Activation\*\*

```yaml

File\_Type\_Detection:

tsx\_jsx: "→frontend persona"

py\_js: "→appropriate stack"

sql: "→data operations"

Docker: "→devops workflows"

test: "→qa persona"

api: "→backend focus"

Keyword\_Triggers:

bug\_error\_issue: "→analyzer persona"

optimize\_performance: "→performance persona"

secure\_auth\_vulnerability: "→security persona"

refactor\_clean: "→refactorer persona"

explain\_document\_tutorial: "→mentor persona"

design\_architecture: "→architect persona"

```

---

## \*\*📁 Directory Structure & File Organization\*\*

### \*\*Documentation Paths\*\*

```yaml

Claude\_Docs: ".claudedocs/"

Reports: ".claudedocs/reports/"

Metrics: ".claudedocs/metrics/"

Summaries: ".claudedocs/summaries/"

Checkpoints: ".claudedocs/checkpoints/"

Tasks: ".claudedocs/tasks/"

Project\_Documentation: "docs/"

API\_Docs: "docs/api/"

User\_Docs: "docs/user/"

Developer\_Docs: "docs/dev/"

```

### \*\*Configuration Files Structure\*\*

```yaml

Main\_Config: ".claude/settings.local.json"

Shared\_Configs: ".claude/shared/"

Command\_Patterns: ".claude/commands/shared/"

Personas: ".claude/shared/superclaude-personas.yml"

MCP\_Integration: ".claude/shared/superclaude-mcp.yml"

```

---

This configuration system provides unprecedented power and flexibility for AI-assisted development. Use the personas to match expertise to your task, leverage MCP servers for specialized capabilities, and apply the appropriate flags for optimal results.

## \*\*🚀 Getting Started\*\*

1. \*\*Choose your persona\*\* based on the type of work you're doing

2. \*\*Select appropriate MCP servers\*\* for your specific needs

3. \*\*Use the right command\*\* with relevant flags

4. \*\*Apply evidence-based practices\*\* throughout development

5. \*\*Leverage UltraCompressed mode\*\* for efficiency when needed

The system is designed to be intelligent, adaptive, and focused on delivering high-quality, evidence-based solutions while maintaining security and performance standards.