Claude Code
Complete Usage Guide

The Definitive Reference for AI-Assisted Development

Extended Edition v2.0

Table of Contents

Part I: Fundamentals

1. Core Principles

1.1 Mindset for Effective Usage

Claude Code is an AI pair programmer. Quality output correlates directly with clarity of communication.

- **Clarity:** Be specific. 'Fix this' → 'Fix null reference on line 42'
- **Incremental:** Break complex tasks into smaller steps
- **Context-Aware:** Reference previous work in conversation
- **Feedback:** Tell Claude what worked and what didn't
- **Verify:** Always test generated code

2. Context Management

2.1 Understanding Context

Context = everything Claude remembers: messages, code, files. The window is finite (~200K tokens). Efficient management is crucial.

2.2 Optimization Strategies

- Share only relevant code, not entire files
- Use summaries for large codebases
- One topic per conversation
- Save outputs locally
- Use CLAUDE.md for persistent context

■ Run /cost periodically. Use /compact before hitting 80% context.

3. /compact and /clear

3.1 /compact — Summarize & Continue

Condenses conversation while preserving key info. Use when context > 70% and continuing same topic.

```
/compact
/compact keep current function and error messages
/compact preserve the API schema
```

3.2 /clear — Complete Reset

Erases all history. Use for new topics or when conversation went wrong.

■■ /clear is irreversible. Save important code first.

Part II: CLAUDE.md & Project Setup

4. Understanding CLAUDE.md

CLAUDE.md is automatically read at conversation start. It provides persistent project context.

4.1 CLAUDE.md Template

```
# Project: [Name]

## Tech Stack
- Runtime: Node.js 20.x
- Framework: React 18 + TypeScript
- State: Zustand
- Testing: Vitest

## Project Structure
src/
■■■ components/
■■■ hooks/
■■■ stores/
■■■ utils/

## Coding Conventions
- Functional components with hooks
- Named exports over default
- TypeScript strict mode

## Current Focus
[What you're working on now]

## Do NOT
- Do not use 'any' type
- Do not use class components
```

5. Project Initialization

5.1 Starting New Project

1. cd /project/path
2. /init
3. Edit CLAUDE.md
4. /doctor
5. Start working

Part III: Prompt Engineering

6. Writing Effective Prompts

6.1 The CTEFC Framework

- **C - Context:** Project info, tech stack
- **T - Task:** Specific, measurable request
- **E - Examples:** Input/output samples
- **F - Format:** Desired output structure
- **C - Constraints:** Limitations

6.2 Bad vs Good

■ Bad: "Make this faster"

```
■ Good:
## Context
React component rendering 1000+ items, currently 800ms.
## Task
Optimize to < 100ms
## Constraints
- Maintain current API
- No external libraries
```

7. Prompt Templates

■ Code Review Template

```
## Context
- Language: [TypeScript/Python]
- File: [path/to/file]

## Code
[paste code]

## Focus
- [ ] Logic  - [ ] Performance
- [ ] Security  - [ ] Readability
```

■ Bug Debugging Template

```
## Environment
Runtime: [Node.js 20], OS: [macOS]

## Expected vs Actual
[What should happen] vs [What happens]

## Error
```

```
[Full error message]

## Code
[Relevant code]

## Already Tried
- [Attempt 1]
```

■ Feature Implementation Template

```
## Project Context
Stack: [React 18 + TypeScript + Zustand]

## Feature: [Name]
### Requirements
- [Req 1]
- [Req 2]

### Constraints
- No new dependencies
- Must be accessible

### Output
1. Implementation
2. Types
3. Tests
```

■ Refactoring Template

```
## Current Code
[paste code]

## Goals
1. [e.g., Split into smaller functions]
2. [e.g., Improve testability]

## Must Preserve
- Public API
- Existing behavior
```

8. Real-World Examples

8.1 Memory Leak Fix

```
## Problem
React component - memory grows, components not unmounting

## Code
function useWebSocket(url) {
  const [messages, setMessages] = useState([]);
  useEffect(() => {
    const ws = new WebSocket(url);
    ws.onmessage = (e) => setMessages(p => [...p, e.data]);
  }, [url]);  // Missing cleanup!
}

## Request: Identify leak, provide fix, explain
```

Part IV: File & Git Operations

9. File Operations

■ Create File Template

```
## Task: Create Component
Path: src/components/UserCard.tsx

## Requirements
- Display avatar, name, email
- Loading skeleton state
- TailwindCSS styling

## Reference
[paste similar component]
```

■ Modify File Template

```
## Task: Modify File
File: src/utils/dateFormatter.ts

## Current Code
[paste current]

## Changes
1. Add timezone support
2. Add formatRelativeTime()

## Constraint: Maintain backward compatibility
```

■ Multi-File Rename Template

```
## Task: Rename Entity
"Product" → "Item" across project

## Files
- types/Product.ts → types/Item.ts
- api/productApi.ts → api/itemApi.ts

## Update: type names, functions, imports
```

10. Git Workflow

■ Commit Message Template

```
## Changes
[describe or paste diff]

## Format: Conventional Commits
feat/fix/refactor/docs/test/chore

## Output
1. Title (max 72 chars)
```

```
2. Body (if needed)
```

■ PR Description Template

```
## Summary
[What this PR does]

## Changes
- file1.ts - [change]
- file2.ts - [change]

## Checklist
- [ ] Tests  - [ ] Docs  - [ ] No console.logs
```

Part V: Testing & Documentation

11. Test Templates

■ Unit Test Template

```
## Code to Test
function calculateDiscount(price, percent, max?) {
  const discount = price * (percent / 100);
  return max && discount > max ? price - max : price - discount;
}

## Test Requirements
- Normal cases
- Edge cases (0%, 100%, negative)
- maxDiscount cap

## Framework: Vitest
```

■ Component Test Template

```
## Component: LoginForm.tsx

## Scenarios
1. Renders default state
2. Validation errors on invalid input
3. Calls onSubmit with data
4. Loading state during submit
5. Error message on failure

## Stack: Vitest + React Testing Library
```

12. Documentation

■ README Template

```
## Include Sections
1. Title + Description
2. Features
3. Prerequisites
4. Installation
5. Usage
6. API docs
7. Contributing
8. License
```

■ JSDoc Template

```
## Code
function processOrder(order, options?) { }

## Generate JSDoc with:
- @param descriptions
- @returns
```

- @throws
- @example

Part VI: Advanced Topics

13. Multi-File Refactoring

```
## Task: Extract Shared Module

## Current: Duplicated across
- features/orders/utils.ts
- features/products/utils.ts

## Goal: Extract to shared/utils/

## Phases
1. Create shared module
2. Update orders imports
3. Update products imports
4. Remove duplicates
```

14. Security

14.1 Never Share in Prompts

- ■ API keys, tokens, passwords
- ■ Database connection strings
- ■ Private keys
- ■ Real user data
- ■ Internal URLs

14.2 Safe Pattern

```
## DON'T
const apiKey = "sk-1234567890abcdef";

## DO
const apiKey = process.env.API_KEY;
// or: const apiKey = "[REDACTED]";
```

15. Troubleshooting

- **Connection errors:** /doctor, check internet
- **Slow responses:** /compact, simplify prompts
- **Wrong file edits:** Be specific about paths
- **Forgotten context:** Update CLAUDE.md

Part VII: Game Development

16. Game Dev Templates

■ Game Architecture Template

```
## Game Type
[e.g., Vampire Survivors-style roguelike]

## Tech
- Platform: Web (Canvas)
- Engine: Vanilla JS
- Target: 60 FPS, 1000+ entities

## Systems Needed
1. Game loop
2. Entity management
3. Collision detection
4. Rendering
5. Input handling
```

■ Game System Template

```
## System: Weapon Upgrades

## Existing
class EntityManager { entities = []; }

## Requirements
1. XP from enemies
2. Level-up at 100, 300, 600 XP
3. Show 3 upgrade options
4. Pause during selection

## Output
1. UpgradeSystem class
2. UI component
3. Integration points
```

■ Performance Optimization Template

```
## Current
- 200 enemies: 60 FPS ✓
- 500 enemies: 35 FPS ✗

## Code
update(dt) {
  for (const e of entities) {
    for (const o of entities) {  // O(n²)!
      if (checkCollision(e, o)) { }
    }
  }
}

## Target: 60 FPS with 1000 entities
## Request: Spatial partitioning, object pooling
```

17. Canvas & Animation

■ Sprite Animation Template

```
## Spritesheet
- Tile: 32x32, Layout: 4x4
- Row 0: Idle, Row 1: Walk
- Row 2: Attack, Row 3: Death

## Requirements
- 10 FPS animation
- State transitions
- Callback on complete

## Output: AnimationController class
```

■ Particle System Template

```
## Use Cases
- Enemy death explosion
- XP collection effect
- Hit sparks

## Requirements
- Pool: 500 particles
- Properties: pos, vel, color, life
- Minimal GC

## Output: ParticleSystem with pooling
```

■ Collision Detection Template

```
## Current: O(n²)
## Entities: 500 enemies, 200 projectiles

## Pairs Needed
- Player ↔ Enemies
- Projectiles ↔ Enemies

## Request
1. Spatial hash/quadtree
2. Layer filtering
3. < 2ms collision phase
```

18. Multiplayer & WebRTC

■ P2P Setup Template

```
## Game: Turn-based (Chess/Janggi)

## Architecture
- P2P via WebRTC
- WebSocket signaling
- LAN priority

## Requirements
1. Connection setup
2. State sync
3. Reconnection
```

```
## Output: SignalingClient, PeerConnection
```

■ State Sync Template

```
## Strategy: Host-authoritative

## State
{ tick, players, enemies, projectiles }

## Requirements
1. Delta compression
2. Input prediction
3. Reconciliation
4. Interpolation

## Conditions: 50-200ms latency
```

Part VIII: Reference

19. Command Reference

Command	Description
`/help`	Show all commands
`/compact`	Summarize, reduce context
`/clear`	Reset completely
`/cost`	Token usage & cost
`/init`	Initialize, create CLAUDE.md
`/memory`	Edit CLAUDE.md
`/config`	Settings
`/doctor`	Diagnose
`/bug`	Bug report
`/review`	Review staged changes
`@file`	Reference file
`@folder/`	Reference folder

20. Quick Checklists

■ Before Prompting

- ■ Context provided?
- ■ Request specific?
- ■ Only relevant code?
- ■ Format specified?

■ Context Management

- ■ Check /cost
- ■ /compact at 70%
- ■ /clear for new topics
- ■ Save outputs
- ■ Update CLAUDE.md

■ After Receiving Code

- ■ Test code
- ■ Ask follow-ups
- ■ Save/commit
- ■ Provide feedback

■ Security

- ■ No API keys in prompts
- ■ No real user data
- ■ Use placeholders
- ■ Manual security review

■ Game Dev

- ■ Profile with many entities
- ■ Test target devices
- ■ Object pooling
- ■ Save system planned

■ Claude Code works best as a collaborative partner. Clear communication = better results.