Claude Code - Guía de Referencia Rápida

Cheat sheet conciso para usuarios avanzados de Claude Code.

🚀 Comandos CLI Esenciales

```
# Iniciar sesión interactiva
claude

# Ejecutar comando one-shot
claude "task description"

# Ejecutar query y salir
claude -p "query"

# Continuar conversación reciente
claude -c
claude --continue

# Resumir conversación previa
claude -r
claude --resume [session-name]

# Iniciar en Plan Mode
claude --permission-mode plan

# Configurar agents vía JSON
claude --agents '{"agent-name": {...}}'

# Output formats
claude -p "query" --output-format text
claude -p "query" --output-format json
claude -p "query" --output-format stream-json
```

⌨️ Keyboard Shortcuts

Shortcut	Acción
`Shift+Tab`	Cycle permission modes (default → acceptEdits → plan)
`Ctrl+O`	Toggle verbose mode (ver thinking)
`Ctrl+B`	Background current task
`Ctrl+R`	Reverse search command history
`Option+T` / `Alt+T`	Toggle extended thinking
`Tab`	Command autocomplete

Shortcut	Acción
↑ / ↓	Command history
?	Show keyboard shortcuts

💬 Slash Commands Críticos

```
/help                    # Ayuda general
/agents                  # Manage subagents
/mcp                     # Manage MCP servers
/skills                  # View available skills
/memory                  # Edit CLAUDE.md
/plan                    # Enter Plan Mode
/rename <name>           # Name current session
/resume                  # Resume previous session
/cost                    # Show token usage & cost
/context                 # Visualize context usage
/model                   # Change AI model
/permissions             # View/update permissions
/hooks                   # Manage hooks
/init                    # Initialize project (create CLAUDE.md)
/config                  # Update settings
/output-style            # Change output format
/release-notes           # View recent updates
```

🔧 MCP Server Management

Agregar MCP Server

HTTP Server (cloud services, recommended):

```
claude mcp add --transport http <name> <url>

# Con authentication header
claude mcp add --transport http <name> <url> \
  --header "Authorization: Bearer TOKEN"

# Ejemplos
claude mcp add --transport http github https://api.githubcopilot.com/mcp/
claude mcp add --transport http notion https://mcp.notion.com/mcp
```

Stdio Server (local tools):

```
claude mcp add --transport stdio <name> -- <command> [args...]

# Con environment variables
claude mcp add --transport stdio --env KEY=VALUE <name> -- <command>
```

```
# Ejemplo: PostgreSQL
claude mcp add --transport stdio db -- npx -y @bytebase/dbhub \
  --dsn "postgresql://user:pass@localhost:5432/database"
```

Scopes:

```
# Local (default) - solo este proyecto
claude mcp add <name> <url>

# Project - compartido con equipo (version control)
claude mcp add --scope project <name> <url>

# User - todos tus proyectos
claude mcp add --scope user <name> <url>
```

Gestionar MCP Servers

```
# Listar todos
claude mcp list

# Ver detalles
claude mcp get <name>

# Eliminar
claude mcp remove <name>

# Check status (dentro de Claude)
> /mcp
```

MCP Servers Útiles

Server	URL/Command	Uso
GitHub	`https://api.githubcopilot.com/mcp/`	Issues, PRs, repos
Notion	`https://mcp.notion.com/mcp`	Docs, databases
Sentry	`https://mcp.sentry.dev/mcp`	Error tracking
PostgreSQL	`npx -y @bytebase/dbhub --dsn "..."`	Database queries
Slack	`https://mcp.slack.com/`	Messaging
Google Drive	`https://mcp.google.com/drive`	Files, docs

🤖 Subagents

Built-in Agents

Agent	Model	Tools	Uso
Explore	Haiku	Read-only	Fast file discovery & code search
Plan	Inherit	Read-only	Research during plan mode
General-purpose	Inherit	All	Complex multi-step tasks

Custom Agent Configuration

Crear agent (`.claude/agents/agent-name.md`):

```
---
name: agent-name
description: When to use this agent (Claude reads this to decide)
tools: Read, Grep, Glob, Bash
model: sonnet | opus | haiku | inherit
permissionMode: default | acceptEdits | dontAsk | bypassPermissions | plan
skills: skill-name-1, skill-name-2
---

System prompt for the agent goes here.
Instructions on what the agent should do.
```

Tool Options: Read, Write, Edit, Bash, Grep, Glob, AskUserQuestion, Task, Skill, LSP

Permission Modes:

- `default`: Normal permission prompts
- `acceptEdits`: Auto-accept file edits
- `dontAsk`: Auto-deny prompts
- `bypassPermissions`: Skip all permissions (use carefully!)
- `plan`: Read-only mode

Usar Agents

```
# Explícito
> Use the security-auditor agent to analyze this code

# Implícito (Claude decide based on description)
> Perform a security audit

# Background execution
> Run security audit in background
[Ctrl+B durante execution]

# Resume agent
> Continue the previous security audit
```

📚 Skills

Crear Skill (`.claude/skills/skill-name/SKILL.md`)

```
---
name: skill-name
description: What this skill does and when to use it
allowed-tools: Read, Grep, Glob
model: sonnet | opus | haiku | inherit
context: fork | inherit
---

Instructions for using this skill.
Can be as detailed as needed.
```

Skill Locations

Location	Scope	Priority
`~/.claude/skills/`	Personal (all projects)	Low
`.claude/skills/`	Project (team shared)	High
Plugin	Plugin scope	Medium

Multi-file Skills (Progressive Disclosure)

```
skill-name/
├── SKILL.md           # Overview (what Claude reads first)
├── DETAILS.md         # Deep dive documentation
├── EXAMPLES.md        # Code examples
└── scripts/           # Helper scripts
    └── helper.sh
```

🎯 Custom Slash Commands

Project Command (team shared)

```
mkdir -p .claude/commands
echo "Prompt text here. Use \$ARGUMENTS for params." >
.claude/commands/command-name.md
```

Con frontmatter:

```
---
argument-hint: [arg1] [arg2]
description: What this command does
allowed-tools: Bash, Read
---

Command prompt here.
First arg: $1
Second arg: $2
All args: $ARGUMENTS
```

Ejecutar command:

```
> /command-name arg1 arg2
```

Personal Command (solo tú)

```
mkdir -p ~/.claude/commands
echo "Personal prompt" > ~/.claude/commands/my-command.md
```

Bash Execution en Commands

```
---
allowed-tools: Bash(git status:*), Bash(git diff:*)
---

## Context
Git status: !`git status`
Git diff: !`git diff`

Based on the above, create a commit.
```

⚙️ Configuration Files

`.claude/settings.json`

```
{
  "permissions": {
    "defaultMode": "default",
    "rules": [
      {
        "working_directory": "src/sensitive/*",
        "tools": ["Read", "Grep"],
```

```
        "deny": ["Write", "Edit", "Bash"]
      }
    ]
  },
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "./scripts/lint.sh $FILE_PATH"
          }
        ]
      }
    ]
  },
  "alwaysThinkingEnabled": true,
  "env": {
    "MAX_THINKING_TOKENS": "10000"
  },
  "enabledPlugins": {
    "plugin-name@marketplace": true
  }
}
```

`.mcp.json` (Project MCP Servers)

```
{
  "mcpServers": {
    "github": {
      "type": "http",
      "url": "https://api.githubcopilot.com/mcp/"
    },
    "database": {
      "type": "stdio",
      "command": "npx",
      "args": ["-y", "@bytebase/dbhub", "--dsn", "${DATABASE_URL}"],
      "env": {
        "DATABASE_URL": "${DATABASE_URL}"
      }
    }
  }
}
```

`CLAUDE.md` (Project Knowledge)

```
# Project Overview
[What this project does]
```

```
## Architecture
[System design, key components]

## Conventions
- Coding style
- Naming conventions
- Testing approach

## Important Files
- src/core/ - Core business logic
- src/api/ - REST API endpoints

## Common Tasks
- How to add a feature
- How to debug issues
- How to deploy
```

🔐 Permission Management

Permission Modes

Mode	Behavior	Use When
`default`	Ask for permission	Normal development, safety first
`acceptEdits`	Auto-accept file edits	Rapid iteration, prototyping
`plan`	Read-only exploration	Planning, research, reviewing

Switch modes: `Shift+Tab` cycles through modes

Path-Specific Rules

```
{
  "permissions": {
    "rules": [
      {
        "working_directory": "tests/*",
        "defaultMode": "acceptEdits"
      },
      {
        "working_directory": "production/*",
        "defaultMode": "plan",
        "deny": ["Write", "Edit", "Bash"]
      }
    ]
  }
}
```

🧠 Extended Thinking

Toggle: `Option+T` (Mac) / `Alt+T` (Windows/Linux)

Configure default:

```
> /config
# Toggle alwaysThinkingEnabled
```

Limit budget:

```
export MAX_THINKING_TOKENS=10000
claude
```

View reasoning: `Ctrl+O` para verbose mode

📊 Context & Cost Management

```
# Ver context usage
> /context

# Ver token costs
> /cost

# Limpiar context con subagent
> Use a subagent to search for X and return only the result
```

🔄 Git Integration

Commits

```
# Claude analiza cambios y crea commit
> create a commit with all my changes

# Mensaje específico
> commit with message "feat: add authentication"

# Ver estado
> show git status
> what have I changed?
```

Pull Requests

```
# Crear PR
> create a pr

# Con detalles
> create a pr with title "Add authentication" to main branch

# Review PR
> review pr #123
```

Git Worktrees (Parallel Sessions)

```
# Crear worktree
git worktree add ../project-feature-a -b feature-a

# Usar Claude en cada worktree
cd ../project-feature-a
claude "Implement feature A"

# En otro terminal
cd original-project
claude "Fix bug B"

# Listar worktrees
git worktree list

# Remover worktree
git worktree remove ../project-feature-a
```

🔍 Reference Syntax

```
# Files
> Explain @src/auth.js

# Directories
> Analyze @src/controllers/

# MCP Resources
> Review @github:issue://123
> Query @db:table://users
> Read @docs:file://api/auth
```

🎨 Output Styles

Crear Custom Output Style

`~/.claude/output-styles/concise.md`:

```
---
name: concise
description: Brief, focused responses
---

Keep responses concise:
- Max 3 paragraphs
- Bullet points preferred
- Focus on actionable info
```

Use:

```
> /output-style concise
```

🪝 Hooks (Automation)

Available Hooks

- `PreToolUse`: Before tool execution
- `PostToolUse`: After tool execution
- `PermissionRequest`: When permission needed
- `UserPromptSubmit`: After user submits prompt
- `Stop`: When conversation ends

Example Hook

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "npm run lint $FILE_PATH",
            "continueOnError": true
          }
        ]
      }
    ],
    "UserPromptSubmit": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "echo 'Analyzing prompt...'",
            "blocking": false
```

```
            }
          ]
        }
      ]
    }
  }
```

🚀 CI/CD Integration

Package.json Scripts

```
{
  "scripts": {
    "review": "claude -p 'Review changes for security and style issues'",
    "pre-commit": "claude -p 'Check staged files for critical issues'",
    "translate": "claude -p 'Translate new strings in locales/en.json to
ES, FR, DE'"
  }
}
```

GitHub Actions

```
- name: Claude Code Review
  run: |
    claude -p "Review PR changes. Output markdown." > review.md
    gh pr comment ${{ github.event.number }} -F review.md
```

Unix Pipes

```
# Analyze logs
tail -f app.log | claude -p "Alert on errors or anomalies"

# Process data
cat data.csv | claude -p "Summarize key insights"

# Generate docs
git diff | claude -p "Generate changelog from this diff"
```

🔧 Troubleshooting

Common Issues

PATH not set:

```
export PATH="$HOME/.claude/bin:$PATH"
source ~/.zshrc
```

Permission denied:

```
chmod +x ~/.claude/bin/claude
```

API rate limits:

```
export MAX_THINKING_TOKENS=5000
claude config set apiKey YOUR_KEY
```

Context window full:

```
> /context  # Check usage
> Use a subagent for this high-volume task
```

MCP server not connecting:

```
> /mcp  # Check status
claude mcp remove <name>
claude mcp add <name> <url>  # Re-add
```

📱 Model Selection

```
# Ver modelo actual
> /model

# Cambiar modelo
> /model opus    # Claude Opus 4.5 (most capable)
> /model sonnet  # Claude Sonnet 4.5 (balanced)
> /model haiku   # Claude Haiku 3.5 (fast & cheap)
```

En agents: Set `model: opus|sonnet|haiku|inherit` en frontmatter

💡 Pro Tips

1. **Plan first, execute later**: Use Plan Mode para complejidad
2. **Subagents for isolation**: High-volume operations en subagents
3. **MCP for integration**: External tools via MCP, not bash scripts

4. **Skills for knowledge**: Team expertise en skills reutilizables
5. **Context management**: Monitor con `/context`, cleanup con subagents
6. **Git worktrees**: Parallel development sin cambiar branches
7. **Unix philosophy**: Pipe data through Claude como cualquier tool
8. **Background tasks**: `Ctrl+B` para long-running operations
9. **Explicit > Implicit**: Specific prompts dan mejores resultados
10. **Iterate**: First attempt rara vez es perfecto, refina

🔗 Links Útiles

- **Docs**: https://code.claude.com/docs/
- **MCP Spec**: https://modelcontextprotocol.io/
- **Examples**: https://github.com/anthropics/claude-code-examples
- **MCP Servers**: https://github.com/modelcontextprotocol/servers
- **Discord**: https://discord.gg/anthropic

Mantén este cheat sheet a mano durante desarrollo con Claude Code!