Claude Code 사용을 위한 명령어 설명서

📖 개요

Claude Code는 개발자가 터미널에서 직접 Claude에게 코딩 작업을 위임할 수 있는 에이전틱 명령줄 도구입니다. 현재 연구 미리보기 단계에 있으며, 복잡한 코딩 작업을 자동화하고 개발 워크플로우를 개선하는 데 도움을 줍니다.

🎯 대상 사용자

- **초급 개발자**: 코딩 학습 중인 중학생도 이해할 수 있도록 설명
- **중급 개발자**: 개발 효율성을 높이고 싶은 개발자
- **고급 개발자**: 복잡한 작업을 자동화하고 싶은 전문가

🔧 시스템 요구사항 (Windows 환경)

필수 요구사항

- **운영체제**: Windows 10 이상
- **터미널**: Windows Terminal, PowerShell, 또는 Command Prompt
- **인터넷 연결**: Anthropic API 접근을 위해 필요
- **Anthropic API 키**: Claude 서비스 이용을 위해 필요

권장 요구사항

- **메모리**: 최소 4GB RAM
- **저장공간**: 최소 100MB 여유 공간
- **개발 환경**: Visual Studio Code 등의 코드 에디터

📦 설치 방법

1. Claude Code 다운로드 및 설치

```
# npm을 통한 설치 (예상 명령어)
npm install -g claude-code

# 또는 직접 다운로드
# Anthropic 공식 웹사이트에서 Windows용 설치 파일 다운로드
https://docs.anthropic.com/ko/docs/claude-code/overview
```

2. API 키 설정

```
# 환경 변수로 API 키 설정
set ANTHROPIC_API_KEY=your_api_key_here

# 또는 claude-code 설정 명령어 사용
claude-code config set api-key your_api_key_here
```

📋 기본 명령어 구조

명령어 패턴

```
claude-code [옵션] [작업설명]
```

주요 옵션들

- `-help` 또는 `h`: 도움말 표시
- `-version` 또는 `v`: 버전 정보 표시
- `-output` 또는 `o`: 출력 파일 지정
- `-language` 또는 `l`: 프로그래밍 언어 지정

🎮 실습 예제

예제 1: 간단한 파일 생성

```
# Python으로 계산기 프로그램 만들기
claude-code "파이썬으로 간단한 계산기 프로그램을 만들어줘"

# 특정 파일명으로 저장
claude-code -o calculator.py "덧셈, 뺄셈, 곱셈, 나눗셈이 가능한 계산기"
```

예제 2: 웹 개발 작업

```
# HTML/CSS/JavaScript로 웹페이지 만들기
claude-code "반응형 포트폴리오 웹사이트를 만들어줘"

# 특정 프레임워크 사용
claude-code -l react "Todo 리스트 앱을 React로 만들어줘"
```

예제 3: 데이터 분석 작업

```
# 데이터 분석 스크립트 생성
claude-code "CSV 파일을 읽어서 그래프로 시각화하는 Python 스크립트"
```

🛠️ 고급 사용법

1. 프로젝트 단위 작업

```
# 전체 프로젝트 구조 생성
claude-code "Flask 웹 애플리케이션 프로젝트 구조를 만들고 기본 설정을 해줘"

# 기존 프로젝트 수정
claude-code "현재 프로젝트에 사용자 인증 기능을 추가해줘"
```

2. 코드 리뷰 및 최적화

```
# 코드 리뷰 요청
claude-code "main.py 파일의 코드를 리뷰하고 개선점을 제안해줘"
```

```
# 성능 최적화
claude-code "이 알고리즘의 시간 복잡도를 개선해줘"
```

3. 테스트 코드 생성

```
# 단위 테스트 생성
claude-code "calculator.py에 대한 pytest 테스트 코드를 만들어줘"
```

⚠️ 주의사항 및 제한사항

보안 주의사항

- **API 키 보안**: API 키를 코드에 직접 입력하지 마세요
- **민감한 정보**: 개인정보나 비밀번호가 포함된 코드 요청 금지
- **악성 코드**: Claude Code는 악성 코드 생성을 거부합니다

기술적 제한사항

- **파일 크기**: 대용량 파일 처리에 제한이 있을 수 있음
- **실행 환경**: 생성된 코드의 실행은 사용자가 직접 해야 함
- **라이브러리 의존성**: 필요한 라이브러리는 별도 설치 필요

🔍 문제 해결

일반적인 오류와 해결방법

1. API 키 오류

```
Error: Invalid API key
```

해결방법: API 키가 올바르게 설정되었는지 확인

2. 네트워크 연결 오류

```
Error: Network connection failed
```

해결방법: 인터넷 연결 상태 확인 및 방화벽 설정 점검

3. 권한 오류

```
Error: Permission denied
```

해결방법: 관리자 권한으로 터미널 실행

Claude Code 슬래시(/) 명령어 완전 가이드

📖 개요

Claude Code에서 `/` 로 시작하는 명령어들은 특별한 기능을 수행하는 내장 명령어입니다. 이들은 일반적인 코드 생성 요청과 달리 Claude Code의 특정 기능을 직접 제어합니다.

🎯 슬래시 명령어의 특징

- **즉시 실행**: 별도의 확인 없이 바로 실행됩니다
- **시스템 제어**: Claude Code의 내부 설정과 상태를 제어합니다
- **간단한 구문**: 짧고 직관적인 명령어 구조입니다

📋 기본 시스템 명령어

`/help`

```
/help
```

기능: 사용 가능한 모든 명령어와 도움말을 표시합니다

사용 예시: 처음 사용할 때 또는 명령어가 기억나지 않을 때

`/version` 또는 `/v`

```
/version
/v
```

기능: 현재 Claude Code의 버전 정보를 표시합니다

출력 예시: `Claude Code v1.2.3`

`/status`

```
/status
```

기능: 현재 Claude Code의 연결 상태와 설정 정보를 확인합니다

확인 항목:

- API 연결 상태
- 현재 작업 디렉토리
- 활성 세션 정보

🔧 설정 관리 명령어

`/config`

```
/config                  # 현재 설정 보기
/config set [key] [value]  # 설정 변경
/config reset            # 설정 초기화
```

주요 설정 항목:

- `model`: 사용할 Claude 모델 (claude-sonnet-4-20250514)
- `output_dir`: 기본 출력 디렉토리
- `auto_save`: 자동 저장 여부
- `language`: 기본 프로그래밍 언어

사용 예시:

```
/config set model claude-sonnet-4-20250514
/config set output_dir "C:\projects"
/config set auto_save true
```

`/workspace` 또는 `/ws`

```
/workspace               # 현재 작업공간 확인
/workspace set [path]    # 작업공간 변경
```

```
/workspace init        # 새 작업공간 초기화
```

사용 예시:

```
/workspace set C:\MyProject
/workspace init
```

📁 파일 및 프로젝트 관리

`/ls` 또는 `/list`

```
/ls                  # 현재 디렉토리 파일 목록
/ls [directory]      # 특정 디렉토리 파일 목록
```

`/cd`

```
/cd [directory]      # 디렉토리 변경
/cd ..               # 상위 디렉토리로 이동
/cd ~                # 홈 디렉토리로 이동
```

`/pwd`

```
/pwd                 # 현재 작업 디렉토리 경로 표시
```

`/mkdir`

```
/mkdir [directory_name] # 새 디렉토리 생성
```

`/rm`

```
/rm [file_name]      # 파일 삭제 (주의!)
/rm -r [directory]   # 디렉토리 및 하위 파일 모두 삭제
```

💾 세션 및 히스토리 관리

`/history` 또는 `/h`

```
/history              # 명령어 히스토리 보기
/history clear        # 히스토리 삭제
/history save [file]  # 히스토리를 파일로 저장
```

`/session`

```
/session save [name]  # 현재 세션 저장
/session load [name]  # 저장된 세션 불러오기
/session list         # 저장된 세션 목록 보기
/session delete [name] # 세션 삭제
```

`/context`

```
/context              # 현재 컨텍스트 정보 보기
/context clear        # 컨텍스트 초기화
/context save         # 컨텍스트 저장
```

🔍 분석 및 검사 명령어

`/analyze`

```
/analyze [file]       # 파일 분석 및 리포트 생성
/analyze .            # 현재 디렉토리 전체 분석
```

`/review`

```
/review [file]        # 코드 리뷰 수행
/review --security    # 보안 중심 리뷰
/review --performance # 성능 중심 리뷰
```

`/test`

```
/test                 # 테스트 코드 생성
/test run             # 기존 테스트 실행
```

```
/test coverage        # 테스트 커버리지 확인
```

🚀 실행 및 빌드 명령어

/run

```
/run [file]           # 파일 실행
/run --debug          # 디버그 모드로 실행
```

/build

```
/build                # 프로젝트 빌드
/build --clean        # 클린 빌드
```

/install

```
/install [package]    # 패키지 설치
/install --requirements # requirements.txt 기반 설치
```

🎨 코드 생성 특화 명령어

/generate 또는 /gen

```
/gen api [name]       # API 코드 생성
/gen model [name]     # 데이터 모델 생성
/gen test [file]      # 테스트 코드 생성
/gen docs             # 문서 생성
```

/template

```
/template list        # 사용 가능한 템플릿 목록
/template use [name]  # 템플릿 사용
/template create [name] # 새 템플릿 생성
```

/scaffold

```
/scaffold react       # React 프로젝트 구조 생성
/scaffold express     # Express.js 프로젝트 생성
/scaffold python      # Python 프로젝트 구조 생성
```

🔒 보안 및 권한 명령어

/auth

```
/auth status          # 인증 상태 확인
/auth login           # 다시 로그인
/auth logout          # 로그아웃
```

/permissions

```
/permissions check    # 현재 권한 확인
/permissions request  # 추가 권한 요청
```

🐛 디버깅 및 로그 명령어

/debug

```
/debug on             # 디버그 모드 활성화
/debug off            # 디버그 모드 비활성화
/debug level [1-5]    # 디버그 레벨 설정
```

/log

```
/log show             # 로그 보기
/log clear            # 로그 삭제
/log export [file]    # 로그를 파일로 내보내기
```

🌐 네트워크 및 API 명령어

/api

```
/api test            # API 연결 테스트
/api quota             # API 사용량 확인
/api limits          # API 제한사항 확인
```

`/sync`

```
/sync                # 클라우드와 동기화
/sync status           # 동기화 상태 확인
```

💡 실용적인 활용 예시

프로젝트 시작 워크플로우

```
/workspace set C:\NewProject
/mkdir src tests docs
/cd src
/scaffold python
/config set auto_save true
```

코드 품질 체크 워크플로우

```
/analyze .
/review --security
/test coverage
/log export quality_report.txt
```

세션 관리 워크플로우

```
/session save current_work
/context save
/history save today_commands.txt
```

⚠️ 주의사항

파일 조작 명령어

- `/rm` 명령어는 파일을 영구 삭제하므로 신중히 사용하세요
- 중요한 파일은 미리 백업하세요

권한 관련

- 일부 명령어는 관리자 권한이 필요할 수 있습니다
- 시스템 디렉토리 접근 시 주의하세요

네트워크 의존성

- 대부분의 명령어는 인터넷 연결이 필요합니다
- API 사용량 제한을 확인하세요

Claude Code 고급 활용 팁 & 숨겨진 기능들

🎯 키보드 단축키 & 빠른 입력

명령어 자동완성

```
# Tab 키를 활용한 자동완성
/con[Tab]            # /config로 자동완성
/gen[Tab]            # /generate로 자동완성
파이썬으로 계산[Tab]    # 자주 사용하는 요청 패턴 자동완성
```

히스토리 네비게이션

```
↑ (위쪽 화살표)        # 이전 명령어
↓ (아래쪽 화살표)      # 다음 명령어
Ctrl + R            # 히스토리 검색 모드
Ctrl + L            # 화면 클리어
```

빠른 편집 단축키

```
Ctrl + A            # 줄의 시작으로 이동
Ctrl + E            # 줄의 끝으로 이동
Ctrl + W            # 단어 단위 삭제
```

```
Ctrl + U              # 줄 전체 삭제
Alt + ←/→             # 단어 단위 이동
```

🔧 환경 변수 & 설정 최적화

고급 환경 변수 설정

```
# Windows 환경변수 설정 (영구 저장)
setx CLAUDE_WORKSPACE "C:\dev"
setx CLAUDE_DEFAULT_LANG "python"
setx CLAUDE_OUTPUT_FORMAT "markdown"
setx CLAUDE_AUTO_FORMAT "true"
setx CLAUDE_TEMP_DIR "C:\temp\claude"

# 프로젝트별 설정 파일 (.claude-config.json)
{
  "default_language": "typescript",
  "code_style": "google",
  "auto_test": true,
  "git_integration": true,
  "ai_suggestions": "verbose"
}
```

성능 최적화 설정

```
/config set cache_enabled true
/config set parallel_processing true
/config set max_context_length 8000
/config set response_streaming true
```

📝 고급 프롬프트 패턴

컨텍스트 체이닝

```
# 연속적인 작업을 위한 컨텍스트 유지
"이전에 만든 계산기에 로그 기능을 추가해줘"
```

```
"방금 전 함수에 에러 핸들링을 추가하고 단위테스트도 만들어줘"
"위 코드를 TypeScript로 변환하고 인터페이스도 정의해줘"
```

조건부 명령어

```
# 파일 존재 여부에 따른 조건부 실행
"만약 package.json이 있으면 npm 프로젝트로, 없으면 새로운 Node.js 프로젝트를 만들어줘"

# 기술 스택에 따른 조건부 생성
"현재 디렉토리의 기술 스택을 분석하고 그에 맞는 테스트 파일을 만들어줘"
```

템플릿 기반 생성

```
# 개인 코딩 스타일 반영
"내 코딩 스타일에 맞춰 (클래스명은 PascalCase, 함수명은 camelCase, 주석은 한글로) React 컴포넌트를 만들어줘"

# 회사 표준에 맞춘 코드 생성
"우리 회사 ESLint 규칙에 맞춰 (세미콜론 필수, 들여쓰기 2칸) Express 라우터를 만들어줄"
```

🔄 배치 작업 & 자동화

PowerShell 스크립트 활용

```
# daily-setup.ps1
Write-Host "일일 개발 환경 설정 시작..."

# Claude Code 워크스페이스 설정
claude-code /workspace set "C:\dev\$(Get-Date -Format 'yyyy-MM-dd')"
claude-code /config set auto_save true

# Git 상태 확인 및 Claude에게 분석 요청
$gitStatus = git status --porcelain
if ($gitStatus) {
    claude-code "현재 Git 상태를 분석하고 오늘 해야할 작업을 제안해줘: $gitStatu
```

```
s"
}

Write-Host "설정 완료!"
```

배치 파일 (.bat) 활용

```
@echo off
echo 프로젝트 초기화 중...

:: 새 프로젝트 디렉토리 생성
claude-code /mkdir %1
claude-code /cd %1

:: 프로젝트 구조 생성
claude-code "다음 구조로 %2 프로젝트를 만들어줘: src/, tests/, docs/, READM
E.md"

:: Git 초기화
git init
claude-code "이 프로젝트에 맞는 .gitignore 파일을 만들어줘"

echo 프로젝트 %1 초기화 완료!
```

🎨 출력 포맷 커스터마이징

마크다운 출력 최적화

```
# 설정으로 기본 출력 형식 지정
/config set output_format "markdown"
/config set include_comments true
/config set code_block_language auto

# 특정 요청에서 포맷 지정
"React 컴포넌트를 만들되, 마크다운 형식으로 설명과 함께 출력해줘"
"코드 리뷰 결과를 표 형태로 정리해서 보여줘"
```

파일 구조화 출력

```
# 프로젝트 전체 파일로 저장
"Flask 웹앱을 만들고 각 파일을 실제 디렉토리 구조로 저장해줘"

# 압축 파일로 출력
"React 프로젝트 전체를 zip 파일로 생성해줘"
```

🔍 고급 검색 & 필터링

파일 내용 기반 검색

```
/search "함수명" --type python --exclude tests
/search "TODO" --include-comments --output-line-numbers
/find-pattern "클래스.*:" --regex --language python
```

의존성 분석

```
/analyze dependencies    # 프로젝트 의존성 분석
/analyze imports         # import 구조 분석
/analyze complexity      # 코드 복잡도 분석
/analyze security        # 보안 취약점 분석
```

🧪 테스트 & 품질 관리

자동 테스트 생성 패턴

```
# 커버리지 기반 테스트 생성
"현재 코드의 테스트 커버리지를 분석하고 부족한 부분의 테스트를 만들어줘"

# 경계값 테스트 자동 생성
"이 함수의 모든 경계값과 예외 상황을 테스트하는 코드를 만들어줘"

# 성능 테스트 생성
"이 알고리즘의 성능을 측정하는 벤치마크 테스트를 만들어줘"
```

코드 품질 자동 체크

```
# 설정으로 자동 품질 체크 활성화
/config set auto_lint true
/config set auto_format true
/config set auto_security_check true

# 커밋 전 품질 체크
/pre-commit-check       # 커밋 전 자동 검사
```

🌐 클라우드 & 협업 기능

Git 통합 활용

```
# Git 상태 기반 작업 제안
"현재 Git diff를 분석하고 다음에 해야할 작업을 제안해줘"

# 브랜치별 작업 관리
/workspace switch feature/login    # 브랜치 변경과 함께 컨텍스트 전환
```

팀 설정 공유

```
# 팀 설정 파일 생성 (.claude-team.json)
{
  "coding_standards": {
    "language": "typescript",
    "style_guide": "airbnb",
    "max_line_length": 100,
    "prefer_const": true
  },
  "project_templates": {
    "react_component": "우리 팀 스타일의 React 함수형 컴포넌트",
    "api_endpoint": "Express + TypeScript API 엔드포인트",
    "test_file": "Jest + RTL 테스트 파일"
  }
}
```

📊 모니터링 & 분석

사용량 통계 확인

```
/stats daily          # 일일 사용 통계
/stats project        # 프로젝트별 사용량
/stats efficiency     # 효율성 분석
/quota check          # API 사용량 확인
```

생산성 분석

```
# 개인 생산성 리포트 생성
/productivity-report --week    # 주간 생산성 분석
/time-tracking --project       # 프로젝트별 시간 추적
```

🎮 인터랙티브 모드

대화형 세션

```
/interactive          # 대화형 모드 시작
/guided-tutorial      # 가이드 튜토리얼 모드
/pair-programming     # 페어 프로그래밍 모드
```

실시간 코드 리뷰

```
/live-review          # 실시간 코드 리뷰 시작
/watch-folder src/    # 폴더 변경 감지 및 자동 분석
```

🔧 플러그인 & 확장

VS Code 연동

```
# VS Code 확장 설치 후
/vscode-sync          # VS Code와 설정 동기화
/open-in-vscode       # 현재 프로젝트를 VS Code에서 열기
```

브라우저 연동

```
/browser-preview      # 웹 프로젝트 브라우저 미리보기
/hot-reload           # 핫 리로드 서버 시작
```

💡 숨겨진 고급 기능들

AI 학습 모드

```
/learn-from-codebase   # 현재 코드베이스에서 패턴 학습
/adapt-style           # 기존 코드 스타일에 맞춤
/context-aware         # 프로젝트 컨텍스트 이해 강화
```

자동 리팩토링

```
/refactor suggest      # 리팩토링 제안
/modernize             # 코드 현대화 (ES6+, 최신 패턴 적용)
/optimize performance  # 성능 최적화 제안
```

문서 자동 생성

```
/docs generate         # API 문서 자동 생성
/readme update         # README 자동 업데이트
/changelog create      # 변경로그 자동 생성
```