CLAUDE COMMAND REFERENCE

공식 DOCS + COMMAND/CONFIG/SHORTCUT 중심 재구성 PDF 받기 PNG 받기

INSTALLATION · COMMANDS · FLAGS · SLASH COMMANDS · SHORTCUTS · CONFIG · PERMISSION MODES · HOOKS · MCP

PREMIUM DARK COMMAND SHEET

Claude Code

설명형 사용법이 아니라 바로 찾아보는 커맨드·플래그·설정·단축키 중심 레퍼런스

기본 실행	`claude` / `claude "질문"` / `claude -p "질문"`
핵심 구조	명령 → 플래그 → slash → shortcut → config → permission
좋은 ASK	파일 + 목표 + 제약 + 검증을 같이 준다
주의	설명 요청과 실제 수정 요청을 한 문장에 섞지 않는다

01 설치 / 인증 — installation

- `curl -fsSL https://claude.ai/install.sh | bash`
- `brew install --cask claude-code`
- `claude doctor`
- `claude auth login`
- `claude auth status`
- `claude auth logout`

- 처음엔 프로젝트 루트에서 시작.
- 설치 확인은 `claude doctor` 가 제일 빠름.

02 핵심 실행 명령 — cli commands

`claude`	대화형 세션 시작
`claude "질문"`	질문과 함께 바로 시작
`claude -p "질문"`	출력만 받고 종료. 파이프/스크립트용
`cat logs.txt | claude -p "explain"`	파이프 입력 분석
`claude -c`	이전 세션 이어가기
`claude -r "세션" "질문"`	특정 세션 재개
`claude update`	업데이트
`claude agents`	에이전트/병렬 작업 관리

03 자주 쓰는 플래그 — cli flags

`-p`	single query 후 종료
`-c`	마지막 세션 계속
`--add-dir`	추가 작업 디렉토리 허용
`--agent`	특정 에이전트 실행
`--agents`	에이전트 정의 주입
`--append-system-prompt`	세션 규칙 추가
`--append-system-prompt-file`	파일로 규칙 추가
`--bare`	CLAUDE.md / hooks / skills 없이 최소 시작
`--debug`	디버그 채널 출력
`--effort high`	더 깊게 생각하도록 유도

04 Slash commands — quick control

`/help`	현재 세션에서 가능한 기능 보기
`/clear`	대화 맥락 비우고 새로 시작
`/compact`	긴 컨텍스트 압축
`/config`	설정/동작 확인 계열
`/permissions`	권한 상태 확인/조정
`/status`	현재 세션 상태 빠르게 확인

- slash는 작업 전환·정리·상태 확인에 유용.
- `/compact` 는 긴 세션에서 특히 자주 씀.

05 키보드 단축키 — shortcuts

`Shift+Tab`	권한 모드 순환
`Ctrl+G`	계획을 에디터에서 직접 열어 수정
`↑ / history`	이전 입력 재사용
`Tab`	자동완성/탐색 계열
`⌘K / Ctrl+K`	검색/명령 팔레트 계열 접근

06 설정 파일 / 위치 — config files

`CLAUDE.md`	프로젝트 지속 규칙
`.claude/`	rules, settings, 에이전트 관련 구조
`settings.json`	세션/도구/동작 관련 설정
`.claude/rules/`	범위별 규칙 분리
`AGENTS.md`	에이전트 동작/역할 문서화

07 CLAUDE.md 빠른 규칙 — memory

넣기 좋은 것
코드 스타일, 테스트 순서, 배포 규칙, 금지사항, 디렉토리 규약

넣지 말 것
이번 세션 임시 TODO, 일회성 진행 상황, 금방 바뀌는 잡정보

좋은 규칙 예
항상 최소 diff, public API 변경 금지, 테스트 먼저

운영 팁
팀 규칙은 파일에, 개인 선호는 메모리에 분리

08 Permission modes — safe autonomy

Plan mode
읽기 전용 분석. 위험하거나 큰 작업에 적합

기본 모드
행동 전 확인을 받는 안전한 기본값

Auto mode
덜 끊긴다. 대신 검증 기준을 강하게 줘야 함

원칙
자동화 수준이 올라갈수록 테스트/린트/빌드를 더 명시

09 Hooks — automation

`PreToolUse`	도구 실행 전 정책 검사/가드
`PostToolUse`	포맷팅, 테스트, 후처리 자동화
`PostToolUseFailure`	실패 시 로깅/알림/정리
`UserPromptSubmit`	프롬프트 입력 시 규칙 주입/검사
`Stop`	세션 종료 직전 마무리 작업

- 가장 흔한 패턴은 Edit/Write 뒤 formatter 자동 실행.
- Bash 실행 전 금지 명령 검사도 hooks로 자주 건다.

10 MCP — tool connectivity

`claude mcp add`	MCP 서버 추가
`--transport http`	원격 HTTP 서버 연결
`--transport stdio`	로컬 stdio 서버 연결
`--scope project`	프로젝트 공유 설정
`--scope user`	개인 전역 설정
`.mcp.json`	프로젝트 MCP 정의 파일

- GitHub, DB, 내부 API 같은 외부 도구 연결에 핵심.
- 팀 공유는 project scope, 개인 실험은 user scope가 편함.

11 Subagents — parallel delegation

`.claude/agents/`	프로젝트별 에이전트 정의 위치
`~/.claude/agents/`	개인 전역 에이전트 정의 위치
`explore`	코드 탐색/관련 파일 찾기
`review`	변경 검토/리스크 점검
`implement`	구현 작업 병렬 분리
`research`	배경조사/문서 확인/비교 작업

언제 쓰나
독립 태스크를 나눌 수 있을 때. 탐색/구현/리뷰 분업에 특히 유리.

주의
같은 파일을 두 녀석에 수정하게 만들면 충돌과 검수 비용이 커진다.

12 Skills — reusable workflows

`/skills`	사용 가능한 스킬 목록 확인
`skill view`	특정 스킬 내용 읽기
`skill create`	반복 작업을 재사용 절차로 저장
`skill patch`	낡은 스킬 내용을 즉시 보정
`~/.claude/skills/`	개인 스킬 저장 위치

- 복잡한 성공 절차를 다시 쓰게 되는 순간 스킬 후보.
- 한번 쓴 스킬이 틀렸으면 바로 patch하는 게 핵심.

13 Plugins — extensions

`claude plugin`	플러그인 관리 진입점
`claude plugin install ...`	플러그인 설치
`claude plugins`	plugin 명령의 alias
`--channels`	플러그인 채널 알림 수신
`plugin:<name>@marketplace`	채널 대상 지정 형식

- plugin은 도구 확장, channels는 이벤트/알림 수신에 가깝다.
- MCP가 외부 도구 연결이라면 plugin은 생태계 확장 느낌으로 보면 편하다.