V

바이브 메이커 · 무료 자료

실전 클로드코드 300% 활용 명령어

설치는 했는데 채팅만 하고 있다면 — 진짜 힘은 슬래시(/) 뒤에 있습니다.
실전에서 가장 강력한 **명령어 17개**를 한 장씩 정리했어요.

`/goal` `/loop` `/deep-research` `/effort ultracode` `플랜 모드`
`/agents` `/mcp` `/code-review`

바이브 메이커 Vibe Maker

docs.vibemake.kr
@vibemak2r

가장 강력한 기능은, 슬래시(/) 뒤에 숨어 있습니다

클로드코드를 설치하고 대화만 해봤다면, 절반도 못 쓴 겁니다. 채팅창에 **슬래시(/) 한 글자**를 치면, 작업을 통째로 자동화하는 특별한 명령어들이 뜹니다.

📖 카드 한 장 = 명령어 한 개, 이렇게 읽으세요

📌

이게 뭐예요?

코드 몰라도 이해되게, 한 줄로 개념부터.

🎯

언제 써요?

실전에서 이 명령어를 꺼내는 딱 그 순간.

⌨️

이렇게 써요

그대로 따라 칠 수 있는 실제 입력 예시.

⚡

파워 팁

한 끗 차이를 만드는 고급 활용·조합.

> 💡 **외울 필요 없어요**
>
> 명령어는 채팅창에 **/** 만 치면 목록과 설명이 자동으로 뜹니다. 이 자료는 "무엇이 있는지"를 보여주는 **지도**일 뿐 — 오늘은 마음에 드는 하나만 골라 써보세요.

/goal

목표를 이룰 때까지, 매번 “계속”이라 안 쳐도 **알아서 끝까지** 갑니다.

📌 이게 뭐예요?

보통 클로드코드는 한 번 시키면 한 번 하고 멈춥니다. **/goal 은 "끝 조건"을 걸어두면 그걸 만족할 때까지 스스로 반복**해서 작업해요. 중간중간 "계속해"라고 다시 칠 필요가 없습니다.

🎯 언제 써요?

여러 번 고쳐야 완성되는 일 — 기능 하나 끝까지 완성, 에러 0개 만들기, 테스트 전부 통과처럼 **"끝나는 지점"이 분명할 때.**

⌨️ 이렇게 써요

터미널 · 클로드코드

```
/goal 로그인 기능 다 만들고 테스트도 전부 통과할 때까지
```

⚡ 파워 팁

끝 조건을 **구체적으로** 쓸수록 정확합니다. "에러 없을 때까지"보다 "타입 에러 0개가 될 때까지"처럼 검증 가능한 조건을 주세요.

/loop

정해둔 간격마다(또는 알아서 판단해) **같은 일을 반복 확인**합니다.

📌 이게 뭐예요?

`/goal` 이 "될 때까지"라면, **/loop 은 "주기적으로 반복"**입니다. 5분마다·15분마다처럼 간격을 주거나, 간격 없이 두면 클로드코드가 알아서 확인 타이밍을 잡습니다.

🎯 언제 써요?

배포가 끝났는지, 자동 검사(CI)가 통과했는지처럼 **시간이 걸리는 걸 지켜보다가, 상황이 바뀌면 다음 행동을 이어갈 때.**

⌨️ 이렇게 써요

터미널 · 클로드코드

```
/loop 15m 깃허브 배포 확인하고, 실패하면 고쳐줘
```

⚡ 파워 팁

세션(창)을 **열어둬야** 돕니다. 창을 닫으면 멈추지만, 7일 안에 다시 열면 이어집니다. /goal 과 헷갈리지 마세요 — loop = 반복 감시, goal = 완성까지.

백그라운드 실행

뒤에서 돌리기 `run in background`

오래 걸리는 작업은 **뒤로 돌려두고**, 그 사이 다른 걸 시키세요.

📌 이게 뭐예요?

빌드·테스트·긴 리서치처럼 시간이 걸리는 일을 '백그라운드'로 돌리면, 끝날 때까지 멍하니 기다릴 필요 없이 **대화를 계속 이어갈 수 있습니다.** 작업이 끝나면 알아서 알려줘요.

🎯 언제 써요?

오래 걸리는 명령을 돌려야 하는데, 그동안 **다른 화면·다른 작업도 함께 진행**하고 싶을 때.

⌨️ 이렇게 써요

터미널 · 클로드코드

```
이 테스트 백그라운드로 돌려줘. 그동안 다음 화면 같이 만들자
```

> ⚡ **파워 팁**
>
> /loop·/goal 과 함께 쓰면 **"뒤에서, 될 때까지, 알아서"** 조합이 완성됩니다 — 기다리는 시간이 사라져요.

/deep-research

웹을 여러 각도로 뒤져 교차검증하고, **출처까지 붙은 리포트**로 정리.

📌 이게 뭐예요?

질문을 던지면 여러 검색을 **동시에 펼치고**, 자료를 읽고, 서로 사실을 대조한 뒤, **출처가 달린 정리 리포트**를 만들어줍니다. 그냥 검색 한 번보다 훨씬 깊고 믿을 만합니다.

🎯 언제 써요?

시장 조사, 경쟁사 비교, 기술 조사처럼 **"대충 말고 근거 있게"** 알아야 할 때.

⌨️ 이렇게 써요

터미널 · 클로드코드

```
/deep-research 국내 노코드 결제 서비스 3곳 수수료·정산주기 비교
```

⚡ 파워 팁

질문에 **대상·기간·비교기준**을 넣을수록 리포트가 날카로워집니다. (최신 버전·일부 유료 플랜에서 동작해요.)

플랜 모드

`/plan` `Shift + Tab`

코드를 건드리기 전에, **계획부터 세우고 확인받게** 만듭니다.

📌 이게 뭐예요?

플랜 모드에선 클로드코드가 파일을 고치지 않고 **'무엇을, 어떤 순서로 할지' 계획만** 세웁니다. 계획을 읽고 승인하면 그때 실행 — 큰 변경 전에 방향을 먼저 맞추는 안전장치예요.

🎯 언제 써요?

규모가 크거나, **잘못되면 되돌리기 번거로운 작업** 앞에서. "일단 바로 고쳐"가 무서울 때.

⌨️ 이렇게 써요

터미널 · 클로드코드

```
/plan 회원가입을 이메일 인증 방식으로 바꾸는 계획 세워줘
```

> ⚡ **파워 팁**
>
> 계획을 읽고 **"여기만 바꿔"**라고 하면 실행 전에 방향을 교정할 수 있어, 엉뚱하게 다 만들어놓고 되돌리는 사고가 확 줄어듭니다. Shift+Tab 으로도 모드 전환.

확장 사고

`think` `think hard` `ultrathink`

"더 깊이 생각해줘" 한마디로 **추론의 깊이를 끌어올립니다.**

📌 이게 뭐예요?

요청에 **think**(깊이) 또는 **ultrathink**(아주 깊이) 같은 말을 붙이면, 클로드코드가 **더 오래·꼼꼼히 따져보고** 답합니다. 어려운 설계나 까다로운 버그에서 정답률이 눈에 띄게 올라가요.

🎯 언제 써요?

복잡한 구조 설계, 원인이 안 잡히는 버그, **여러 선택지를 비교**해야 하는 판단 앞에서.

⌨️ 이렇게 써요

터미널 · 클로드코드

```
이 결제 구조 ultrathink 로 설계해줘 — 놓친 예외 상황까지
```

> ⚡ **파워 팁**
>
> 쉬운 일엔 안 써도 됩니다(느려져요). **어려운 판단이 필요한 순간에만** 꺼내 쓰는 게 요령입니다.

/effort ultracode

한 줄로 **일하는 강도를 최대**로 — 에이전트를 병렬로 굴리는 파워 모드.

📌 이게 뭐예요?

클로드코드의 **일하는 강도(effort)를 최고 단계**로 올립니다. 켜두면 큰 일을 시켰을 때 **여러 에이전트로 나눠 동시에** 처리하고, 더 깊이 따져보며 끝까지 밀어붙여요. 어려운 작업일수록 결과 품질이 확 올라갑니다.

🎯 언제 써요?

경쟁사 전수조사, 코드 전체 점검, 큰 기능 설계처럼 **한 번에 크고 무거운 일**을 통째로 맡길 때. (가벼운 일엔 안 켜도 됩니다 — 느려져요.)

⌨️ 이렇게 써요

터미널 · 클로드코드

```
/effort ultracode → 켠 다음, 큰 일을 통째로 시키기
```

⚡ 파워 팁

/goal 과 함께 쓰면 **"팀 단위로, 끝날 때까지, 알아서"** — 직원 한 명이 아니라 팀 하나가 일합니다. 다 끝나면 /effort 로 다시 낮춰 아껴 쓰세요.

/compact

길어진 대화를 요약해, **계속 이어갈 여유(기억 공간)**를 되찾습니다.

📌 이게 뭐예요?

대화가 길어지면 클로드코드가 기억할 공간이 꽉 찹니다. **/compact 는 지금까지의 핵심(결정·변경사항)만 요약으로 압축**해 공간을 되찾아 줘요. 하던 작업을 잃지 않고 계속할 수 있습니다.

🎯 언제 써요?

긴 세션에서 응답이 느려지거나, **"맥락이 가득 찼다"**는 신호가 뜰 때.

⌨️ 이렇게 써요

터미널 · 클로드코드

```
/compact 결제 관련 결정은 꼭 남겨줘
```

> ⚡ **파워 팁**
>
> 요약할 때 **"무엇을 꼭 남길지"**를 함께 지시하면, 중요한 맥락을 잃지 않고 깔끔하게 이어갈 수 있습니다.

/rewind

잘못된 지점 이전으로 **통째로 되감기** — 강력한 '실행취소'.

📌 이게 뭐예요?

클로드코드는 작업 도중 **되돌릴 지점(체크포인트)을 자동으로 저장**합니다. /rewind 로 원하는 시점을 골라, 그때의 **코드와 대화 상태로 되돌릴** 수 있어요.

🎯 언제 써요?

방향이 틀어졌거나, **몇 단계 전이 더 나았을 때.** 이것저것 고치다 꼬였을 때 깔끔하게 되돌리기.

⌨️ 이렇게 써요

터미널 · 클로드코드

```
/rewind  → 되돌릴 지점 선택
```

⚡ 파워 팁

되돌릴 수 있으니 **과감하게 시도하세요.** 아니다 싶으면 되감으면 그만 — 실험이 자유로워지는 게 진짜 이득입니다.

/clear · /resume

새 주제는 **깔끔하게 초기화**, 지난 작업은 **그대로 복구**.

📌 이게 뭐예요?

/clear 는 지금 대화를 비우고 새 출발 — 이전 내용에 끌려다니지 않습니다. **/resume** 은 예전에 하던 세션 목록에서 골라 **이어서** 작업해요. 주제가 바뀔 때 대화가 섞이지 않게 관리하는 습관입니다.

🎯 언제 써요?

완전히 다른 일로 넘어갈 땐 **/clear**, 어제 하던 걸 이어서 할 땐 **/resume**.

⌨️ 이렇게 써요

터미널 · 클로드코드

```
/clear  (새 주제 시작)  ·  /resume  (지난 세션 선택)
```

> ⚡ **파워 팁**
>
> 주제마다 세션을 나누면 **각 대화가 짧고 정확**해집니다 — 한 창에 이것저것 다 몰아넣지 마세요.

@파일 참조

원하는 파일·폴더를 **콕 집어** "이거 보고 해줘".

📌 이게 뭐예요?

메시지에 **@** 를 치고 파일 이름을 넣으면, 그 파일을 정확히 지목해 읽게 합니다. **"그 파일 어디였더라"** 설명할 필요 없이 바로 맥락을 넘겨줄 수 있어요.

🎯 언제 써요?

특정 파일을 고치거나, **두 파일을 비교·참고**해서 작업할 때.

⌨️ 이렇게 써요

터미널 · 클로드코드

```
@page.tsx 를 @styles.css 기준으로 스타일 맞춰줘
```

⚡ 파워 팁

파일뿐 아니라 **폴더째 @ 로 지목**하면 그 안 전체를 한 번에 맥락으로 줄 수 있습니다.

/agents

일을 나눠 **여러 'AI 일꾼'에게 동시에** 시키기.

📌 이게 뭐예요?

큰 일을 쪼개 **여러 서브에이전트(보조 AI)에게 병렬로** 맡길 수 있습니다. /agents 로 역할별 전문 에이전트를 만들고 관리해요. 혼자 순서대로 하는 것보다 빠르고, 각자 전문화됩니다.

🎯 언제 써요?

조사+작성+검토처럼 **성격이 다른 일을 나눠** 맡기거나, 많은 파일을 동시에 훑을 때.

⌨️ 이렇게 써요

터미널 · 클로드코드

```
/agents  → 역할별 에이전트 생성 · 관리
```

> ⚡ **파워 팁**
>
> **"리서치용", "코드 리뷰용"**처럼 목적별 에이전트를 한 번 만들어두면, 다음부터 매번 재사용됩니다.

/mcp

노션·피그마·깃허브 같은 **외부 도구를 클로드코드에 연결**.

📌 이게 뭐예요?

MCP(외부 도구 연결 규격)를 통해 클로드코드가 다른 서비스와 직접 대화하게 합니다. 연결해두면 **"노션에 정리해줘", "피그마에서 가져와"** 같은 일이 됩니다. /mcp 로 연결 상태를 관리해요.

🎯 언제 써요?

쓰던 툴(노션·구글·깃허브 등)의 데이터를 **클로드코드가 직접 읽고 쓰게** 하고 싶을 때.

⌨️ 이렇게 써요

터미널 · 클로드코드

```
/mcp  → 연결된 도구 확인 · 재연결
```

⚡ 파워 팁

자주 쓰는 툴 **1~2개만 연결해도** 복사·붙여넣기 왕복이 통째로 사라집니다.

스킬 & 나만의 명령어

`Skills` `.claude/commands`

반복하는 작업을 **'스킬'과 '/내명령어'로 박제**해 재사용.

📌 이게 뭐예요?

매번 똑같이 설명하던 작업 방식을, **스킬(작업 설명서)이나 커스텀 슬래시 명령어**로 저장해두면 다음엔 한 단어로 불러 씁니다. `.claude/commands` 폴더에 넣으면 **/내명령어** 로 바로 실행돼요.

🎯 언제 써요?

"또 그 작업이네" 싶은 반복 절차가 생겼을 때 — 배포 점검, 특정 형식 글쓰기 등.

⌨️ 이렇게 써요

파일 · `.claude/commands/배포점검.md`

저장 후 → **/배포점검**

> ⚡ **파워 팁**
>
> 잘 만든 명령어 하나가 **매번 10줄 설명을 한 단어로** 줄여줍니다 — 팀원과 파일로 공유하면 모두가 같은 방식으로 일하게 돼요.

/code-review

저장 전에, 바뀐 코드를 자동으로 훑어 **문제·개선점**을 찾습니다.

📌 이게 뭐예요?

방금 바꾼 코드를 **여러 관점(버그·중복·비효율)에서 점검**해 리포트합니다. 옵션을 주면 발견한 걸 바로 고치거나, 깃허브 PR에 코멘트로 달 수도 있어요.

🎯 언제 써요?

저장·배포 전 마지막 점검, 큰 수정을 끝낸 뒤 품질 확인.

⌨️ 이렇게 써요

터미널 · 클로드코드

```
/code-review (원하면 자동수정 옵션과 함께)
```

⚡ 파워 팁

"내가 놓친 것 없나?"를 대신 봐주는 두 번째 눈입니다. 배포 전 습관으로 삼으면 실수가 확 줄어요.

권한 모드

Shift + Tab

기본 · 자동승인 · 플랜

매번 "해도 돼요?" 확인을 **줄이거나**, 반대로 **더 조심스럽게**.

📌 이게 뭐예요?

클로드코드가 파일을 고치기 전 **물어보는 정도를 조절**합니다. Shift+Tab 으로 모드를 바꿔요 — **기본**(매번 확인) / **자동 승인**(편집 알아서) / **플랜**(계획만). 상황에 맞게 고르면 됩니다.

🎯 언제 써요?

믿고 맡길 땐 **자동 승인**으로 속도를, 위험한 작업엔 **기본**으로 안전을.

⌨️ 이렇게 써요

터미널 · 클로드코드

```
Shift + Tab  → 모드 순환 (화면 하단에 현재 모드 표시)
```

> ⚡ **파워 팁**
>
> 익숙한 반복 작업은 **'자동 승인'**으로 두면 확인 클릭 수십 번이 사라집니다. 낯선 코드·중요한 폴더에선 **'기본'**으로 되돌려 안전하게.

알아두면 좋은
미니 명령어

한 페이지 분량은 아니지만, 매일 쓰면 편한 짧은 명령어들.

`/model`	쓸 모델 고르기 — **opus**(깊게)·**haiku**(빠르게) 등 상황에 맞게 전환
`/init`	프로젝트에 **CLAUDE.md**(작업 규칙서)를 자동으로 만들어 초기 세팅
`/memory`	클로드코드가 **기억하는 규칙·메모** 확인 및 관리
`/cost`	지금까지 쓴 **사용량·비용**을 한눈에 확인
`/doctor`	설치·설정에 문제가 있는지 **자동 진단** (안 될 때 1순위)
`/help`	전체 **명령어 목록**과 설명 보기

💡 이 외에도 많습니다. 채팅창에 **/** 만 쳐보세요 — 지금 쓸 수 있는 명령어가 **설명과 함께** 쭉 뜹니다.

한 장 요약

CHEAT SHEET

17개 명령어, 한 장 요약

① 반복 · 자동화	
`/goal`	끝 조건을 만족할 때까지 무중단 반복
`/loop`	정해둔 간격마다 반복 감시·실행
백그라운드 실행	긴 작업은 뒤로 돌리고 대화 계속
② 리서치 · 계획	
`/deep-research`	웹 교차검증 + 출처 붙은 리포트
`/plan` · Shift+Tab	고치기 전에 계획부터 세우고 확인
think · ultrathink	추론 깊이를 끌어올려 정답률 ↑
`/effort ultracode`	일하는 강도 최대 — 에이전트 병렬 가동
③ 컨텍스트 · 세션	
`/compact`	대화 요약으로 기억 공간 확보
`/rewind`	체크포인트로 되감기 (강력한 실행취소)
`/clear` · `/resume`	새 주제 초기화 · 지난 세션 복구
@파일	파일·폴더를 콕 집어 맥락 전달
④ 확장 — 나만의 클로드코드	
`/agents`	여러 보조 AI에게 병렬로 분담
`/mcp`	노션·피그마 등 외부 도구 연결
스킬 · /내명령어	반복 절차를 저장해 한 단어로 재사용
`/code-review`	저장 전 코드 자동 점검·수정
⑤ 운영 · 설정	
권한 모드 (Shift+Tab)	확인 빈도를 상황에 맞게 조절
`/model /init /cost /doctor` …	매일 쓰는 짧은 운영 명령어

여기까지 읽으셨다면

이제 '대화'가 아니라, 자동화입니다

오늘은 딱 **하나만** 골라 써보세요. 손에 익으면 다음 걸 얹으면 됩니다.
명령어는 외우지 마세요 — **/ 만 치면** 다 뜨니까요.

- `/goal` 끝날 때까지 알아서 — 오늘 첫 자동화
- `/deep-research` 근거 있는 리서치 리포트 한 번
- `/plan` 큰 작업 전, 계획부터 확인하는 습관