

Codex 주요 기능 쉬운 가이드북

0. 이 가이드북의 목적

이 문서는 Codex를 사용할 때 자주 등장하는 핵심 기능들을 쉽게 이해하기 위한 안내서입니다.

Codex는 단순히 코드를 한 번 작성해주는 도구가 아니라, 프로젝트를 기억하고, 작업 목표를 추적하고, 필요한 도구를 연결하고, 코드 변경사항을 검토하며, 장기 작업을 이어갈 수 있게 도와주는 개발·업무 보조 시스템입니다.

이 가이드북에서는 다음 기능을 쉽게 설명합니다.

1. `/goal` — 큰 작업의 목표 설정과 추적
2. `AGENTS.md`와 `/init` — 프로젝트별 영구 지침 만들기
3. `Skills`와 `/skills` — 반복 작업용 기능 자동 불러오기
4. MCP와 `/mcp` — 외부 도구 연결하기
5. `/review`와 `/diff` — 코드 변경사항 검토하기
6. `/experimental`과 Subagents — 작업을 나눠 병렬 처리하기
7. `/resume`, `Ctrl+O`, `Ctrl+R`, Profiles — 세션 복귀와 작업 환경 전환

1. `/goal` — 큰 작업의 목표를 설정하고 추적하기

쉽게 말하면

`/goal`은 Codex에게 “이번 작업의 최종 목표가 무엇인지” 알려주고, 작업 중간중간 그 목표를 기준으로 진행 상황을 추적하게 만드는 기능입니다.

예를 들어 다음과 같은 작업에 유용합니다.

- 웹사이트 전체 리팩토링
- 신규 기능 개발
- 서비스 MVP 구현
- 버그 여러 개 수정
- 문서화 작업
- 테스트 코드 작성
- 대규모 코드베이스 분석

작업이 길어질수록 사람도 AI도 방향을 잃기 쉽습니다.

`/goal`은 그 방향을 고정해주는 역할을 합니다.

주요 명령어

`/goal set`

목표를 설정합니다.

예시:

```
/goal set 병원 마케팅 자동화 대시보드 MVP를 구현한다.
```

이렇게 입력하면 Codex는 이후 작업에서 이 목표를 기준으로 판단합니다.

```
/goal view
```

현재 설정된 목표를 확인합니다.

예시:

```
/goal view
```

사용 시점:

- 지금 어떤 목표로 작업 중인지 확인하고 싶을 때
 - 작업이 길어져서 방향을 다시 점검하고 싶을 때
-

```
/goal pause
```

목표 추적을 잠시 멈춥니다.

예시:

```
/goal pause
```

사용 시점:

- 지금은 목표와 상관없는 작은 질문을 하고 싶을 때
 - 사이드 작업을 잠깐 해야 할 때
-

```
/goal resume
```

멈춰둔 목표 추적을 다시 시작합니다.

예시:

```
/goal resume
```

`/goal clear`

현재 목표를 삭제합니다.

예시:

```
/goal clear
```

사용 시점:

- 프로젝트가 끝났을 때
- 완전히 다른 목표로 전환할 때
- 기존 목표가 더 이상 유효하지 않을 때

주의할 점

`/goal` 은 “며칠 동안 자동으로 실행되는 기능”이 아닙니다.

즉, Codex가 사용자가 자리를 비운 동안 며칠씩 알아서 작업을 계속하는 기능은 아닙니다.
블로그나 홍보 글에서 “장기 자동 실행”처럼 과장해서 설명하는 경우가 있다면 그대로 믿으면 안 됩니다.

정확히는 다음에 가깝습니다.

`/goal` 은 긴 작업의 목표를 기억하고, 현재 세션 안에서 작업 방향을 추적하는 기능이다.

2. `AGENTS.md` 와 `/init` — 프로젝트별 영구 지침 만들기

쉽게 말하면

`AGENTS.md` 는 Codex에게 “이 프로젝트에서는 이렇게 작업해라”라고 알려주는 지침 파일입니다.

사람으로 치면 프로젝트 인수인계 문서나 작업 규칙 문서에 가깝습니다.

예를 들어 다음 내용을 담을 수 있습니다.

- 이 프로젝트의 목적
- 코드 작성 규칙
- 폴더 구조
- 테스트 실행 방법
- 배포 방법
- 금지해야 할 작업
- 자주 쓰는 명령어
- 팀 내 컨벤션

`/init`은 무엇인가?

`/init`은 프로젝트에 필요한 기본 구조를 자동으로 잡아주는 명령어입니다.

쉽게 말하면, Codex에게 이렇게 말하는 것입니다.

“이 프로젝트를 이해할 수 있도록 기본 설정 파일과 지침 구조를 만들어줘.”

그러면 Codex가 프로젝트를 분석하고, 필요한 경우 `AGENTS.md` 같은 지침 파일을 생성하거나 스캐폴드 구조를 잡아줍니다.

전역 지침과 프로젝트 지침

`AGENTS.md`는 계층적으로 적용될 수 있습니다.

전역 지침

모든 프로젝트에 공통으로 적용되는 규칙입니다.

예시:

항상 변경 전후를 설명한다.
테스트가 있으면 실행 결과를 확인한다.
보안 정보는 코드에 직접 넣지 않는다.

프로젝트 지침

특정 프로젝트에만 적용되는 규칙입니다.

예시:

이 프로젝트는 Next.js 기반이다.
UI 컴포넌트는 `components/ui` 안에 둔다.
API 호출 로직은 `services` 폴더에 작성한다.

언제 사용하면 좋은가?

`AGENTS.md`와 `/init`은 다음 상황에서 특히 유용합니다.

- 새 프로젝트를 Codex와 함께 시작할 때
- 기존 프로젝트를 Codex에게 이해시키고 싶을 때
- 반복적으로 같은 규칙을 설명하기 귀찮을 때

- 팀 규칙을 AI 작업에도 일관되게 반영하고 싶을 때

3. Skills와 `/skills` — 반복 작업용 능력 자동 불러오기

쉽게 말하면

Skills는 Codex가 특정 작업을 더 잘 수행하도록 미리 준비된 작업 설명서입니다.

예를 들어 매번 이렇게 설명하지 않아도 됩니다.

CSV 파일을 분석할 때는 결측치를 확인하고, 이상값을 찾고, 주요 통계를 요약하고, 그래프를 만들어줘.

대신 `SKILL.md` 라는 파일에 이 절차를 적어두면, Codex가 해당 작업에 맞는 Skill을 자동으로 불러와 사용할 수 있습니다.

`SKILL.md` 란?

`SKILL.md` 는 특정 작업을 처리하는 방법을 적어둔 파일입니다.

예시로 이런 Skill을 만들 수 있습니다.

- CSV 분석 Skill
- 블로그 글 작성 Skill
- 코드 리뷰 Skill
- 병원 마케팅 리포트 Skill
- 상세페이지 기획 Skill
- 광고 성과 분석 Skill
- 고객 인터뷰 분석 Skill

`/skills` 는 무엇인가?

`/skills` 는 현재 사용할 수 있는 Skill을 확인하거나 선택적으로 적용하는 명령어입니다.

예시:

```
/skills
```

사용 시점:

- 어떤 Skill이 있는지 보고 싶을 때
- 특정 Skill을 수동으로 적용하고 싶을 때

- 자동 매칭이 아니라 내가 직접 작업 방식을 지정하고 싶을 때

쉽게 이해하는 예시

Skill이 없을 때

사용자가 매번 이렇게 길게 설명해야 합니다.

이 CSV를 보고 매출, 전환율, 객단가, 광고비 대비 효율을 분석하고, 이상치와 개선 포인트를 알려줘.

Skill이 있을 때

`marketing-report/SKILL.md`에 분석 방식이 저장되어 있다면, 사용자는 짧게 말할 수 있습니다.

이 파일 분석해줘.

Codex가 작업 유형을 보고 관련 Skill을 자동으로 불러옵니다.

4. MCP와 `/mcp` — 외부 도구 연결하기

쉽게 말하면

MCP는 Codex가 외부 도구와 연결될 수 있게 해주는 방식입니다.

예를 들어 Codex가 단순히 코드만 보는 것이 아니라, 필요한 경우 외부 시스템과 연결될 수 있습니다.

연결 가능한 예시는 다음과 같습니다.

- 데이터베이스
- 사내 문서
- GitHub
- Slack
- Notion
- 파일 시스템
- 브라우저 자동화 도구
- 내부 API
- 분석 도구

`config.toml`은 무엇인가?

MCP 연결 설정은 보통 `config.toml` 파일에 적습니다.

쉽게 말하면 “어떤 외부 도구를 어떻게 연결할지”를 적는 설정 파일입니다.

예시 개념:

```
[mcp_servers.github]  
command = "github-mcp-server"
```

이런 식으로 설정해두면 세션 시작 시 Codex가 해당 도구를 인식할 수 있습니다.

`/mcp`는 무엇인가?

`/mcp`는 현재 연결된 MCP 도구 목록을 확인하는 명령어입니다.

예시:

```
/mcp
```

더 자세히 보고 싶을 때는 verbose 옵션을 사용할 수 있습니다.

```
/mcp verbose
```

언제 유용한가?

MCP는 다음 상황에서 특히 강력합니다.

- Codex가 실제 데이터베이스 구조를 보고 작업해야 할 때
- GitHub 이슈나 PR 정보를 참고해야 할 때
- 사내 문서 기반으로 답변해야 할 때
- Notion, Slack, Drive 등 외부 업무 도구와 연결해야 할 때
- 코드와 실제 운영 환경 정보를 함께 봐야 할 때

5. `/review`와 `/diff` — 코드 변경사항 검토하기

쉽게 말하면

`/review`와 `/diff`는 Codex가 현재 코드 변경사항을 확인하고, 문제가 없는지 검토하게 만드는 기능입니다.

개발자가 직접 Git diff를 보는 것처럼 Codex가 변경된 내용을 분석합니다.

/diff

`/diff` 는 현재 작업 중인 변경사항을 보여줍니다.

일반적인 Git diff와 비슷하지만, untracked 파일까지 포함해서 확인할 수 있습니다.

즉, 아직 Git에 추가하지 않은 새 파일도 검토 대상에 포함될 수 있습니다.

사용 예시:

```
/diff
```

/review

`/review` 는 현재 워킹트리를 기준으로 코드 리뷰를 수행합니다.

예시:

```
/review
```

Codex가 확인할 수 있는 항목은 다음과 같습니다.

- 버그 가능성
- 타입 오류
- 보안 문제
- 예외 처리 누락
- 성능 문제
- 불필요한 복잡성
- 테스트 부족
- 컨벤션 위반

/diff 와 /review 의 차이

기능	목적
<code>/diff</code>	무엇이 변경됐는지 확인
<code>/review</code>	변경사항에 문제가 없는지 검토

쉽게 말하면:

```
/diff = 바뀐 내용을 보여줘  
/review = 바뀐 내용에 문제 없는지 봐줘
```

언제 사용하면 좋은가?

- 커밋하기 전
- PR 만들기 전
- 큰 기능을 구현한 후
- 리팩토링 후
- 버그 수정 후
- AI가 수정한 코드를 검증하고 싶을 때

6. `/experimental` 과 Subagents — 작업을 나눠 병렬 처리하기

쉽게 말하면

`/experimental` 은 실험 기능을 켜고 끄는 명령어입니다.

그중 하나가 Subagents입니다.

Subagents는 큰 작업을 여러 개의 하위 작업으로 나눠 처리하는 개념입니다.

예를 들어 하나의 AI가 모든 것을 순서대로 하는 것이 아니라, 여러 보조 에이전트가 역할을 나눠 작업하는 방식입니다.

예시

사용자가 이렇게 요청했다고 가정합니다.

이 프로젝트 전체를 분석해서 성능 문제, 보안 문제, 리팩토링 포인트를 찾아줘.

Subagents가 있다면 작업을 이렇게 나눌 수 있습니다.

- 에이전트 1: 성능 문제 분석
- 에이전트 2: 보안 문제 분석
- 에이전트 3: 코드 구조 분석
- 에이전트 4: 테스트 부족 영역 분석

그 후 결과를 합쳐서 최종 보고서를 만들 수 있습니다.

언제 유용한가?

Subagents는 다음과 같은 대형 작업에 적합합니다.

- 전체 코드베이스 리뷰
- 대규모 리팩토링 계획 수립
- 여러 파일 동시 분석
- 보안·성능·테스트를 동시에 점검
- 복잡한 기능 구현 전 사전 조사

주의할 점

Subagents는 실험 기능입니다.

따라서 항상 완벽하게 안정적이라고 가정하면 안 됩니다.
중요한 작업에서는 최종 결과를 사람이 검토해야 합니다.

7. `/resume`, `Ctrl+O`, `Ctrl+R`, Profiles — 작업 흐름 이어가기

`/resume`

`/resume`은 이전 세션으로 돌아가는 기능입니다.

작업을 하다가 터미널을 닫았거나, 이전에 하던 작업을 다시 이어가고 싶을 때 사용합니다.

예시:

```
/resume
```

사용 시점:

- 어제 하던 작업을 이어서 하고 싶을 때
- 긴 프로젝트를 여러 번에 나눠 진행할 때
- 이전 대화 흐름을 다시 불러오고 싶을 때

`Ctrl+O`

`Ctrl+O`는 출력 내용을 복사하는 기능입니다.

Codex가 만든 답변, 코드, 명령어, 설명 등을 빠르게 복사할 때 유용합니다.

사용 시점:

- 생성된 코드를 다른 곳에 붙여넣고 싶을 때
- 답변 내용을 문서화하고 싶을 때
- 명령어를 복사해서 실행하고 싶을 때

Ctrl+R

Ctrl+R 은 히스토리 검색 기능입니다.

이전에 입력했던 명령어나 요청을 다시 찾을 수 있습니다.

사용 시점:

- 이전에 썼던 프롬프트를 다시 쓰고 싶을 때
- 과거 명령어를 빠르게 재실행하고 싶을 때
- 반복 작업을 효율화하고 싶을 때

Profiles

Profiles는 권한이나 작업 환경을 전환하는 기능입니다.

예를 들어 상황에 따라 Codex에게 허용하는 작업 범위를 다르게 설정할 수 있습니다.

예시

Profile	용도
읽기 전용	코드 분석만 허용
개발 모드	파일 수정 허용
제한 모드	위험한 명령어 차단
관리자 모드	더 넓은 권한 허용

왜 중요한가?

AI에게 모든 권한을 항상 열어두는 것은 위험할 수 있습니다.

Profiles를 사용하면 작업 상황에 맞게 권한을 조절할 수 있습니다.

예를 들어:

- 분석만 할 때는 읽기 전용
- 코드 수정이 필요할 때는 개발 모드

• 배포나 삭제 작업은 더 엄격하게 관리

이런 식으로 안전한 작업 흐름을 만들 수 있습니다.

전체 기능 요약표

기능	쉽게 말하면	주 사용 상황
<code>/goal</code>	큰 작업의 목표를 설정하고 추적	장기 작업, 대형 리팩토링, MVP 개발
<code>/init</code>	프로젝트 기본 지침과 구조 생성	새 프로젝트 시작, 기존 프로젝트 정리
<code>AGENTS.md</code>	Codex용 프로젝트 작업 규칙	코드 컨벤션, 테스트 규칙, 금지사항 저장
<code>/skills</code>	반복 작업용 Skill 확인 및 적용	분석, 문서화, 코드 리뷰, 마케팅 리포트
<code>SKILL.md</code>	특정 작업 수행법을 적은 파일	자주 반복하는 작업 자동화
MCP	외부 도구 연결 방식	GitHub, DB, Notion, Slack, Drive 연결
<code>/mcp</code>	연결된 MCP 목록 확인	도구 연결 상태 점검
<code>/diff</code>	변경된 코드 확인	커밋 전, PR 전
<code>/review</code>	변경사항 코드 리뷰	버그, 보안, 성능, 테스트 점검
<code>/experimental</code>	실험 기능 관리	Subagents 등 고급 기능 사용
Subagents	작업을 나눠 병렬 처리	대규모 분석, 복합 작업
<code>/resume</code>	이전 세션 이어가기	장기 프로젝트 복귀
<code>Ctrl+O</code>	출력 복사	코드/답변 복사
<code>Ctrl+R</code>	히스토리 검색	이전 명령어 재사용
Profiles	권한과 작업 환경 전환	안전한 작업 제어

입문자를 위한 추천 사용 순서

처음부터 모든 기능을 쓰려고 하면 복잡합니다.
아래 순서대로 익히는 것이 좋습니다.

1단계: 기본 작업 흐름 익히기

먼저 아래 기능만 익혀도 충분합니다.

```
/init
/diff
```

/review
/resume

이 4개만 알아도 프로젝트 시작, 변경사항 확인, 코드 리뷰, 세션 복귀가 가능합니다.

2단계: 프로젝트 지침 만들기

그다음 `AGENTS.md` 를 정리합니다.

추천 내용:

프로젝트 목적
폴더 구조
코드 작성 규칙
테스트 방법
배포 방법
주의사항

이렇게 해주면 Codex가 매번 프로젝트 규칙을 다시 묻지 않아도 됩니다.

3단계: 큰 작업에는 `/goal` 사용하기

작업이 30분 이상 걸리거나 여러 단계로 나뉜다면 `/goal` 을 설정하는 것이 좋습니다.

예시:

`/goal set 관리자 대시보드의 결제 내역 조회 기능을 안정적으로 구현한다.`

이렇게 하면 Codex가 작업 중간에 목표를 기준으로 판단하기 쉬워집니다.

4단계: 반복 작업은 Skills로 만들기

자주 반복하는 작업이 있다면 Skill로 분리합니다.

예시:

- 코드 리뷰 Skill
- 광고 성과 분석 Skill
- 병원 블로그 SEO 점검 Skill
- 상세페이지 기획 Skill
- 유튜브 스크립트 분석 Skill

반복 작업을 Skill로 만들면 작업 품질이 일정해지고, 매번 긴 설명을 하지 않아도 됩니다.

5단계: 외부 도구 연결은 MCP로 확장하기

프로젝트가 커지면 Codex가 코드만 보는 것으로는 부족합니다.

이때 MCP를 활용하면 다음과 같은 연결이 가능해집니다.

- GitHub 이슈 확인
- DB 스키마 확인
- Notion 문서 참고
- Slack 대화 참고
- 내부 API 호출

단, 외부 도구 연결은 권한 관리가 중요합니다.

처음에는 읽기 전용 연결부터 시작하는 것이 안전합니다.

실무 추천 사용 시나리오

시나리오 1. 새 프로젝트 시작

```
/init
```

이후 Codex가 프로젝트 구조를 잡고, 필요한 경우 `AGENTS.md`를 생성합니다.

그다음 목표를 설정합니다.

```
/goal set AI 교육 플랫폼의 수강 신청 MVP를 구현한다.
```

시나리오 2. 기능 개발 후 검토

기능을 구현한 뒤 변경사항을 확인합니다.

```
/diff
```

그다음 리뷰를 요청합니다.

```
/review
```

이 과정을 거치면 커밋 전에 버그 가능성을 줄일 수 있습니다.

시나리오 3. 장기 작업 이어가기

이전 작업으로 돌아갑니다.

```
/resume
```

현재 목표를 확인합니다.

```
/goal view
```

필요하면 다시 목표 추적을 시작합니다.

```
/goal resume
```

시나리오 4. 대규모 코드베이스 분석

실험 기능에서 Subagents를 활성화합니다.

```
/experimental
```

그다음 전체 분석을 요청합니다.

```
이 프로젝트를 성능, 보안, 테스트, 코드 구조 관점에서 나눠 분석해줘.
```

실무자가 기억해야 할 핵심 원칙

1. 긴 작업은 반드시 목표를 먼저 설정한다

작업이 길어질수록 AI도 방향을 잃을 수 있습니다.

```
/goal set 최종적으로 무엇을 완성할지
```

이렇게 시작하면 결과물이 더 안정적입니다.

2. 프로젝트 규칙은 말로 반복하지 말고 `AGENTS.md`에 저장한다

매번 같은 설명을 반복하는 것은 비효율적입니다.

반복되는 규칙은 문서화해야 합니다.

3. 코드 변경 후에는 `/diff` 와 `/review` 를 습관화한다

AI가 만든 코드도 검토가 필요합니다.

```
/diff
/review
```

이 두 개는 커밋 전 기본 루틴으로 두는 것이 좋습니다.

4. 외부 도구 연결은 편리하지만 권한 관리가 중요하다

MCP는 강력하지만, 강력한 만큼 신중하게 써야 합니다.

처음에는 읽기 전용으로 시작하고, 필요한 경우에만 쓰기 권한을 열어야 합니다.

5. 실험 기능은 보조 수단으로 사용한다

Subagents 같은 실험 기능은 대규모 작업에 유용하지만, 최종 판단은 사람이 해야 합니다.

특히 보안, 결제, 배포, 데이터 삭제와 관련된 작업은 반드시 사람이 확인해야 합니다.

최종 요약

Codex의 핵심 기능은 다음 한 문장으로 정리할 수 있습니다.

Codex를 단발성 코드 생성기가 아니라, 목표를 기억하고, 프로젝트 규칙을 따르고, 외부 도구와 연결되며, 코드 변경사항을 검토하고, 장기 작업을 이어가는 개발 파트너로 만드는 기능들이다.

가장 먼저 익혀야 할 명령어는 다음 5개입니다.

```
/init
/goal set
/diff
/review
/resume
```

그다음 프로젝트가 커지면 다음 기능을 추가로 익히면 됩니다.

```
AGENTS.md  
/skills  
/mcp  
/experimental  
profiles
```

초보자는 모든 기능을 한 번에 쓰기보다, 프로젝트 시작 → 목표 설정 → 작업 → 변경 확인 → 리뷰 → 세션 복귀 순서로 익히는 것이 가장 효율적입니다.