

# Claude Code 101

한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

Jeremy.kim  
Software Engineer



# 전달하고 싶은 메시지



1. 못하면 가르치자
2. 해냈으면 회고하자
3. 복리의 마법을 믿자

# 첫 출근, 첫 업무 받은 날

그런데 팀에 나 혼자다

# 문서가 있는데 부실하다

# 코드가 너무 많다

# 어떤것부터 알아야 하지?

어떤 것부터 알아야 하지?

내가 뭘 알아야 하는지도 모른다

# 해결 방법?

# 물어보거나

# 스스로 고뇌한다



# 신규 입사자 온보딩

# 코딩 에이전트도 똑같다

# 물어보기 전에 알려주자

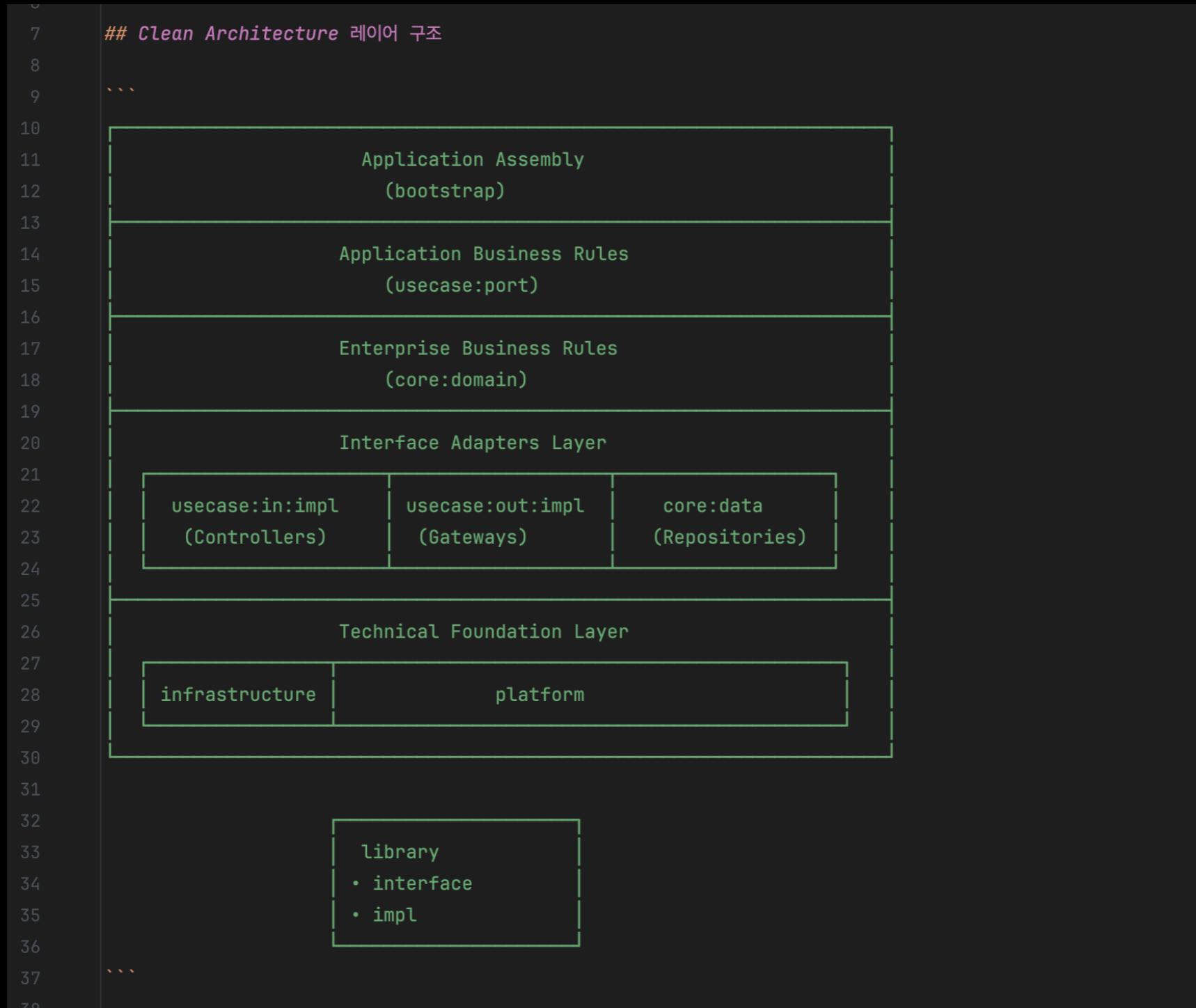
나와 수년을 함께 한 동료라고 생각하고  
**알고 있는 모든 것을 알려주자**

# Document

Claude Code 101:

한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- 프로젝트 모듈 구조가 있다면 **자세히** 알려주자



# Document

Claude Code 101:

한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- 프로젝트 모듈 구조가 있다면 **자세히** 알려주자

```
38
39     ### Clean Architecture 의존성 규칙
40
41     - **의존성 방향**: 위에서 아래로만 의존 (조립 → 기반)
42         - Bootstrap → Use Cases → Domain → Interface Adapters → Technical Foundation
43     - **기반 기술이 전체를 지탱**: infrastructure와 platform이 맨 아래에서 모든 시스템을 떠받침
44     - **Domain은 핵심**: core:domain은 비즈니스 핵심으로 외부 기술에 의존하지 않음
45     - **Bootstrap이 모든 것을 조립**: 최상위에서 모든 레이어의 구현체를 연결
46     - **의존성 역전**: 상위 레이어가 하위 기능을 사용할 때는 인터페이스를 정의하고, 하위에서 구현체를 제공
47
48     ## Gradle 모듈 구조 (Clean Architecture 관점)
49
50     ### 1. Application Assembly Layer (bootstrap)
51     모든 레이어의 의존성을 조립하고 애플리케이션을 실행하는 최상위 조립 레이어입니다.
52
53     ...
```

# Document

Claude Code 101:

한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- 코드 컨벤션이 있다면 **자세히** 알려주자
- 없으면 각 언어 공식 컨벤션이라도 알려주자

```
26
27     ### 파일 이름
28
29     Kotlin 파일이 단일 클래스나 인터페이스를 포함하는 경우(관련된 최상위 선언이 포함될 수 있음), 파일 이름은 클래스 이름과
30
31     파일에 여러 클래스가 포함되거나 최상위 선언만 포함하는 경우, 파일 내용을 설명하는 이름을 선택하고 대문자로 시작하는 카멜
32
33     ### 파일 구성
34
35     동일한 Kotlin 소스 파일에 여러 선언(클래스, 최상위 함수 또는 속성)을 배치하는 것은 이러한 선언이 의미적으로 서로 밀접
36
37     ### Class layout
38
39     일반적으로 클래스의 내용은 다음 순서로 정렬됩니다:
40
41     1. Property declarations and initializer blocks
42     2. Secondary constructors
43     3. Method declarations
44     4. Companion object
45
46     - Bad: 메서드 선언을 알파벳순이나 가시성으로 정렬하지 않는다
47     - Bad: 일반 메서드와 확장 메서드를 분리하지 마세요.
48     - Good: 관련된 것들을 함께 배치하여 클래스를 위에서 아래로 읽는 사람이 논리를 따라갈 수 있게 만든다
49     - Good: 중첩 클래스는 해당 클래스를 사용하는 코드 다음에 배치한다.
50     - Good: 중첩 클래스가 외부에서 사용되고 클래스 내부에서 참조되지 않는 경우 컴파니언 객체 뒤, 클래스 끝에 배치한다.
51
```

# Document

Claude Code 101:

한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- 코드 컨벤션이 있다면 **자세히** 알려주자
- 없으면 각 언어 공식 컨벤션이라도 알려주자

```
113    ## UseCase 네이밍
114
115    UseCase는 Alistair Cockburn의 원칙에 따라 "목표를 짧은 능동 동사구로 표현"합니다.
116
117    **패턴**: `<동사 + 역할 + UseCase 접미사>`
118
119    **Command와 Query 구분**:
120    - **CommandUseCase**: 상태를 변경하는 작업
121    - **QueryUseCase**: 데이터를 조회하는 작업
122
123    ```kotlin
124    // Good: 구체적인 동작과 역할이 명확
125    interface EnterInvitationCommandUseCase {
126        fun invoke(command: Command): Invitation
127
128        data class Command(
129            val inviteeUserId: Long,
130            val invitationCode: Long,
131        )
132    }
133
134    interface UpdateInvitationCommandUseCase {
135        fun invoke(command: Command): Invitation
136
137        data class Command(
138            val invitationId: Long,
139        )
140    }
141
```

- 테스트 작성법이 있다면 **자세히 알려주자**

```
5    ## 목차
6
7    1. [테스트 철학] (#테스트-철학)
8    2. [테스트 프레임워크] (#테스트-프레임워크)
9    3. [테스트 구조] (#테스트-구조)
10   4. [테스트 더블 사용] (#테스트-더블-사용)
11   5. [픽스처 관리] (#픽스처-관리)
12   6. [네이밍 컨벤션] (#네이밍-컨벤션)
13   7. [검증 방법] (#검증-방법)
14   8. [모듈 구조] (#모듈-구조)
15   9. [Outport Stub 관리] (...)
16  10. [테스트 작성 예시] (#테스트-작성-예시)
17
18  ## 테스트 철학
19
20  ### 기본 원칙
21
22  - ★★모킹 라이브러리 사용 금지★★: mockk와 같은 모킹 라이브러리를 사용하지 않습니다
23  - ★★직접 구현한 Stub 객체 사용★★: 테스트 더블은 직접 구현한 Stub 객체를 사용합니다
24  - ★★명확하고 읽기 쉬운 테스트★★: 테스트 의도가 명확하게 드러나도록 작성합니다
25  - ★★격리된 테스트★★: 각 테스트는 독립적으로 실행 가능해야 합니다
26
27  ### 장점
28
29  - ★★명시적 동작★★: Stub 객체의 동작이 코드로 명확하게 표현됩니다
30  - ★★컴파일 타임 안전성★★: 인터페이스 변경 시 컴파일 에러로 감지됩니다
31  - ★★디버깅 용이★★: 테스트 실패 시 디버깅이 쉽습니다
32  - ★★의존성 최소화★★: 외부 라이브러리 의존성을 줄입니다
--
```

# Document

Claude Code 101:

한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- 테스트 작성법이 있다면 **자세히 알려주자**

```
75
76     ### 주석을 통한 구분
77
78     각 단계를 주석으로 명확히 구분합니다:
79
80     - `// arrange`: 테스트 데이터 및 의존성 준비
81     - `// act`: 테스트 대상 메서드 실행
82     - `// assert`: 결과 검증
83
84     ## 테스트 더블 사용
85
86     ### Stub 객체 구현
87
88     모킹 라이브러리 대신 직접 구현한 Stub 객체를 사용합니다.
89
90     ````kotlin
91     class WalletRepositoryStub(
92         private val walletRegistry: List<Wallet> = emptyList()
93     ) : WalletRepository {
94         override fun findById(userId: Long): Wallet? {
95             return walletRegistry.find { it.userId == userId }
96         }
97
98         override fun save(wallet: Wallet): Wallet {
99             // 테스트에 필요한 동작만 구현
100            return wallet
101        }
102    }
103    ````
```

# Document

Claude Code 101:

한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

## • Git Commit Message 작성법 방법을 알려주자

```
5  ## 커밋 메시지 형식
6
7  ### 기본 형식
8
9  ``
10 GZ-{티켓번호} {타입}({모듈}): {설명}
11 ``
12
13 ### 커밋 타입
14
15 - `feat`: 새로운 기능 추가
16 - `fix`: 버그 수정
17 - `refactor`: 리팩토링 (기능 변경 없이 코드 구조 개선)
18 - `test`: 테스트 코드 추가 또는 수정
19 - `docs`: 문서 수정
20 - `chore`: 빌드 스크립트, 설정 파일 등 수정
21 - `style`: 코드 포맷팅, 세미콜론 누락 등 (로직 변경 없음)
22 - `perf`: 성능 개선
```

# Document

Claude Code 101:

한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

## • Git Commit Message 작성법 방법을 알려주자

```
23
24     ### 모듈 표기 규칙
25
26     모듈은 **도메인 단위**로만 표현합니다:
27
28     ✓ **올바른 예시**:
29     - `core:campaign`
30     - `core:user`
31     - `core:region`
32
33
34
35
36
37     ✗ **잘못된 예시**:
38     - `core:campaign:data` (하위 모듈까지 표현하지 않음)
39     - `core:user:domain` (하위 모듈까지 표현하지 않음)
40
41
42     ### 커밋 메시지 예시
43
44     ###### 기능 추가
45     ````bash
46     GZ-35 feat(core:campaign): implement accept for campaign invitation
47     GZ-42 feat(usecase:farm): add farm creation workflow
48     GZ-58 feat(platform:chat): implement message routing
49
50     ```
51
```

# Document

Claude Code 101:

한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- 복잡한 작업할 때는 CC가 생각 과정, 현재 작업 명세, 설계안, 요구사항 명세 등을 중간 문서(Markdown)를 작성하도록 만들자.
- 이 문서는 현재는 작업 메모로 활용되고, 나중에는 Context로도 쓸 수 있다. 이렇게 하면 코딩 과정을 검증하고 가이드하기가 훨씬 쉬워진다.
- 세션이 길어질수록 맥락이 사라지기 쉬운데, 문서를 다시 읽는 것만으로도 중요한 맥락을 되찾을 수 있다.

# 개발에 필요한 모든 가이드를 문서화하고

# 끊임 없이 개선하자



잘 하고 있는지 확인하는 방법?

가이드 문서가 얼마나 자주  
업데이트 되는지 보자

# 프롬프트가 완료되면 끝이 아니라 시작이다

> 잘 했어.

이 내용 `@docs/project_coding_conventions.md` 가이드 문서에 반영해 줘. ■

Claude Code 101:

한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

# 기본 사용법

# 기본 사용법

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- **CLAUDE.md** 파일은 Claude Code 필수 사항
- 특정 프로젝트에서 Claude를 어떻게 활용하는지에 대한 가이드
- /init 명령어를 실행하면 스스로 Codebase를 검토하고 작성
- **./CLAUDE.md**: 현재 프로젝트 공용 가이드
- **~/.claude/CLAUDE.md**: 모든 프로젝트를 대상으로 한 가이드

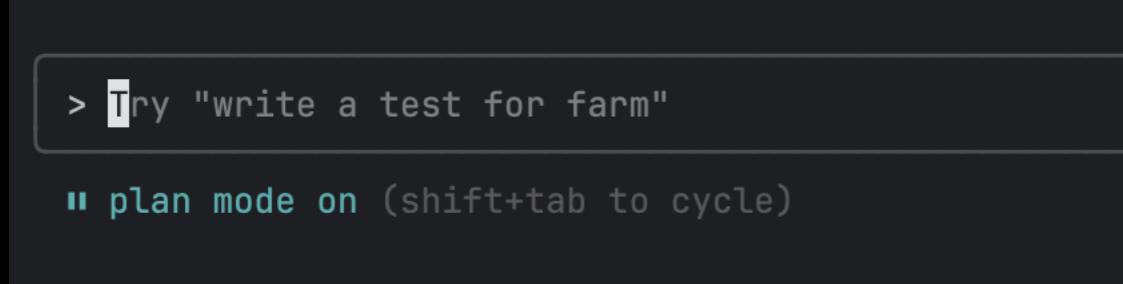
> /init█

/init	Initialize a new CLAUDE.md file with codebase documentation
/install-github-app	Set up Claude GitHub Actions for a repository
/migrate-installer	Migrate from global npm installation to local installation
/compact	Clear conversation history but keep a summary in context. Options
/doctor	Diagnose and verify your Claude Code installation and settings
/ide	Manage IDE integrations and show status
/upgrade	Upgrade to Max for higher rate limits and more Opus

# 기본 사용법

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- Shift + tab : Plan mode 전환
- CC의 신뢰성을 높임. 좀 더 똑똑해짐.



\* Thinking...

이제 구현 계획을 제시하겠습니다.

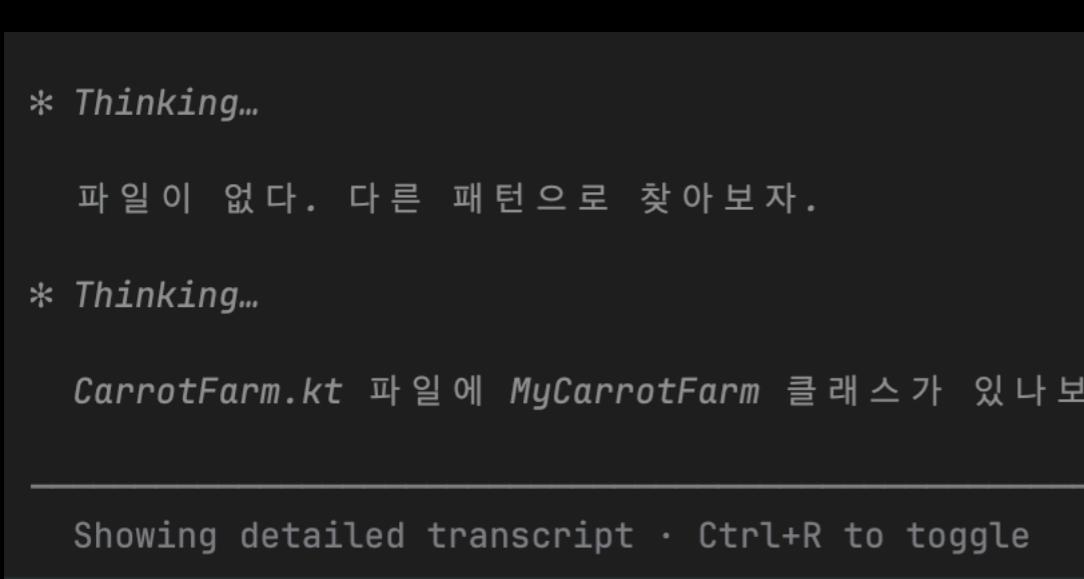
Ready to code?

Here is Claude's plan:

# 기본 사용법

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- CTRL + R : Claude가 보는 전체 맥락을 확인
- CC가 어떤 일을 하고 있는지 확인하기



# 기본 사용법

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- /clear : CC와의 현재 컨텍스트를 초기화

```
> /clear
└ (no content)

> █
▶▶ auto-accept edits on (shift+tab to cycle)
```

- 작업을 한 세션에서 처리할 수 있을 만큼 충분히 작은 단위로 나누기 추천
- (저는 clear 명령 거의 사용하지 않고 새로운 세션을 여는 편)

# 기본 사용법

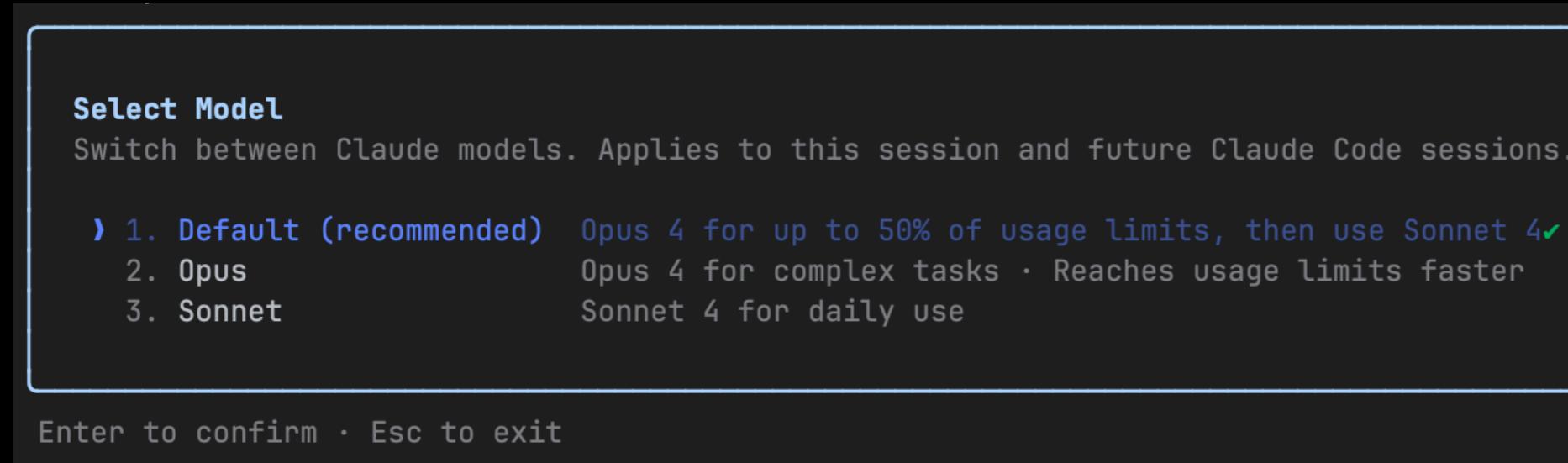
Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- /model : CC가 사용할 모델 선택하기

```
> /model [model]

/model      Set the AI model for Claude Code
/status     Show Claude Code status including version, mod
/vim        Toggle between Vim and Normal editing modes
```

- MAX 쓰니까 잘 모르겠으면 **Opus** 선택(Claude 4 Opus)



# 소소한 팁

# 소소한 팁 - Ultrathink

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- 복잡하거나 다단계의 문제 해결 시 **ultrathink** 쓰자

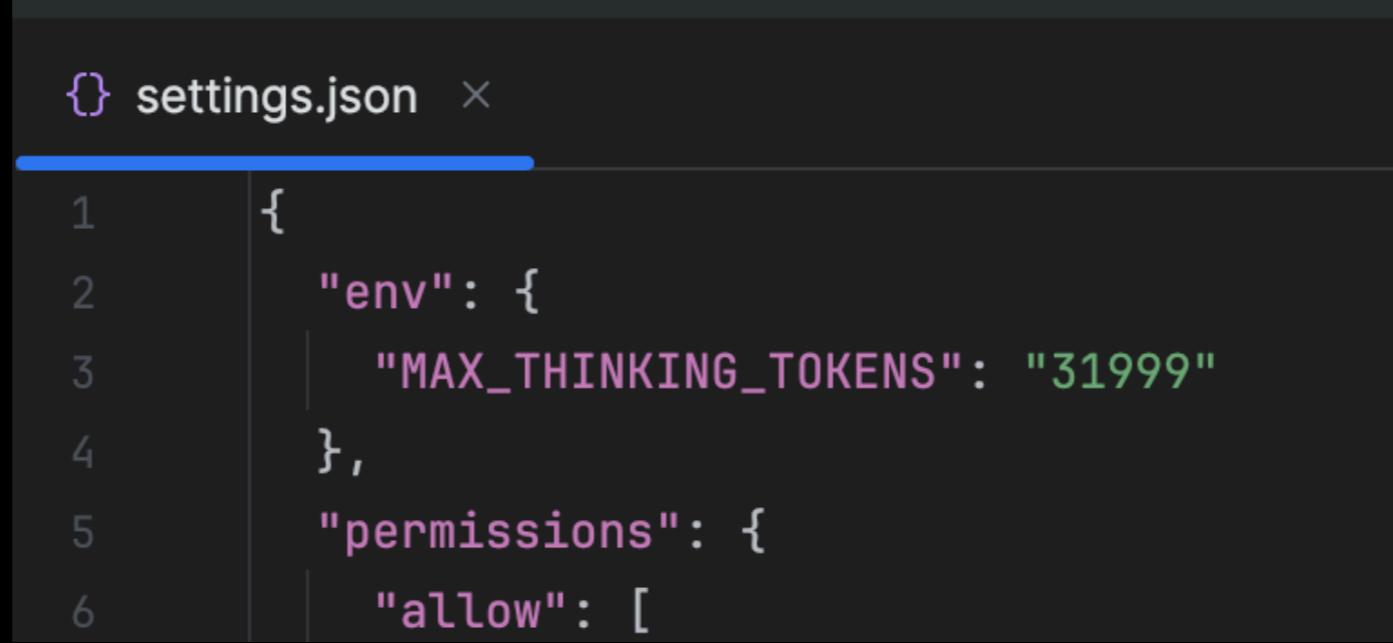
The screenshot shows a terminal window with a dark background. In the top-left corner, there is a red rectangular box highlighting the command `> ultrathink`. Below this, the terminal displays several sections of code, each starting with a label like `@bo`, `@us`, or `@us` followed by a list of items. On the right side of the terminal, there are some numbers and labels such as `48`, `armCo`, and `daang`. At the bottom of the terminal window, there is a status bar with the text `!! plan mode on (shift+tab to cycle)`.

- think: 4000 토큰
- think hard, think more, megathink: 10000 토큰
- ultrathink, think very hard, think super hard: 31999 토큰

# 소소한 팁 - Ultrathink

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- **MAX\_THINKING\_TOKENS = 31999** 명시하기
- 추론에 사용할 최대 토큰 수를 설정해놓고 쓰자



The image shows a dark-themed code editor window with a tab labeled "settings.json". The file contains the following JSON configuration:

```
{  
  "env": {  
    "MAX_THINKING_TOKENS": "31999"  
  },  
  "permissions": {  
    "allow": [  
    ]  
  }  
}
```

# 소소한 팁 - Hooks

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- **Self-sufficient loops:** CC가 스스로 Test, Lint, Build 등을 자동으로 실행하며 스스로 작업을 검증할 수 있는 환경을 만드는게 중요
- ~ 끝나면 Lint 실행하고 검증까지 하도록 요구하기
- ~ 했을때 Build 실행하고 검증까지 하도록 요구하기
- ...

# 소소한 팁 - Hooks

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

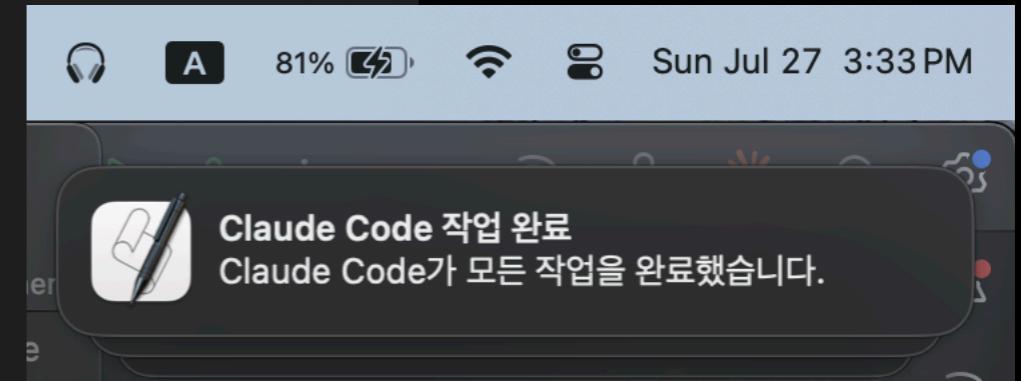
- **UserPromptSubmit**: 사용자가 프롬프트를 제출할 때 즉시 실행
- **PreToolUse**: 도구 호출 전에 실행 (도구 호출을 차단할 수 있음)
- **PostToolUse**: 도구 실행 완료 후 실행
- **Notification**: CC가 알림을 보낼 때 실행
- **Stop**: CC가 응답을 완료할 때 실행
- **SubagentStop**: 하위 에이전트가 응답을 완료할 때 실행

# 소소한 팁 - Hooks

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

```
"hooks": {  
  "PreToolUse": [  
    {  
      "matcher": "Edit|Write|MultiEdit",  
      "hooks": [  
        {  
          "type": "command",  
          "command": "$CLAUDE_PROJECT_DIR/.claude/commands/check-kotlin-files.sh"  
        }  
      ]  
    }  
  ],  
  "Stop": [  
    {  
      "matcher": "",  
      "hooks": [  
        {  
          "type": "command",  
          "command": "$CLAUDE_PROJECT_DIR/.claude/commands/notify-intellij.sh"  
        }  
      ]  
    }  
  ]  
}
```

## Lint & Build



# 소소한 팁 - Commands

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- 자주 사용하는 프롬프트를 **명령어**를 만들어 놓자.
- **반복**적으로 하는 행동이 있다? -> Custom Command 구성
  - Project Command: .claude/commands/
  - Global Command: ~/.claude/commands/
  - Ex) .claude/commands/commit.md

# 소소한 팁 - Commands

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- 커맨드가 많아지면 관련 명령을 **네임스페이스**를 묶자
- 네임스페이스 내에 명령은 하나의 명확한 목적만 다루자
- CLAUDE.md 에 명령 인덱스를 문서화하자

```
.claude/commands/
  └── infra/
    └── monitor.md
  └── frontend/
    ├── component.md
    └── test.md
  └── backend/
    ├── db/
    └── migration.md
  └── api/
    └── spec.md
```

# 소소한 팁 - Commands

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- 예시 1. Git Commit

# 소소한 팁 - Commands

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- 예시 1. Git Commit
- Commit 가이드 만들고, Commit을 CC에게 맡기자

The screenshot shows a terminal window with two panes. The left pane contains a template for a git commit message, and the right pane shows an example of how it's used.

**Left Pane (Template):**

```
> [ ] --- r 작업 내용만 GZ-[번호] 티켓으로 커밋해 줘  
* Thinking...  
  
사용자가 [ ] --- r 작업 내용만 GZ-[번호] 티켓으로 커밋해 줘  
docs/git_commit_guide.md를 참조하여 티켓을 생성합니다.  
- 형식: GZ-{번호} {타입}({모듈}): {설명}  
- 모듈은 도메인 단위로 표현  
  
순서:  
1. git status로 변경된 파일 확인  
2. git diff로 변경 내용 확인  
3. git log로 최근 커밋 메시지 스타일  
4. [ ] --- r 관련 파일만 선택  
5. 커밋 생성
```

**Right Pane (Implementation):**

```
> GZ-56 티켓으로 코드 커밋해 줘  
* Thinking...  
  
사용자가 GZ-56 티켓으로 코드를 커밋해 달라고 요청했습니다.  
  
현재 git status를 보면:  
- 수정된 파일들이 있습니다 (M으로 표시)  
- 새로 추가된 파일들이 있습니다 (AD로 표시)  
- 추적되지 않는 디렉토리가 있습니다 (?? growthz-oas/)  
  
커밋을 만들기 전에:  
1. git status, git diff, git log를 병렬로 실행해서 현재 상태를 파악해야 합니다  
2. 변경사항을 분석해서 적절한 커밋 메시지를 작성해야 합니다  
3. 필요한 파일들을 staging하고 커밋을 생성해야 합니다  
  
Git 커밋 가이드라인에 따르면:  
- 커밋 메시지 형식: GZ-{번호} {타입}({모듈}): {설명}  
- 모듈은 도메인 단위로만 표현 (예: core:campaign, usecase:farm)  
- 커밋 타입: feat, fix, refactor, test, docs, chore, style, perf  
  
먼저 TodoWrite 도구를 사용해서 작업을 계획해야 합니다.
```

● 먼저 작업을 계획하겠습니다.

# 소소한 팁 - Commands

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- 예시 1. Git Commit
- ~~Commit 가이드 만들고, Commit을 CC에게 맡기자~~
- 이것도 귀찮다!!!



# 소소한 팁 - Commands

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- 예시 1. Git Commit
- 현재 작업한 내용들을 바탕으로 커밋을 작성하는 Command

The screenshot shows a terminal window with a dark background and light-colored text. At the top, there's a command line input field containing the text '> /commit'. Below this, a large text area displays the output of the command:

```
/commit      Smart Git Commit (AI-Powered & Ticket-Aware) (project)

* Thinking...

커밋이 성공적으로 완료되었습니다.

●✓ 커밋 완료

GZ-1 refactor(          직을 리팩토링하여 활성화 상태 검증 개선

변경 내용

- 5개 파일 수정 (48줄 추가, 17줄 삭제)
- C
- E
- A
- Value class들에 toString() 메서드 추가
- 테스트 예외 메시지 형식 수정

개선 효과

- 코드의 재사용성이 향상
- 로직 중앙화
- 로그 가독성 개선
```

The text area includes several sections: 'Thinking...', '커밋이 성공적으로 완료되었습니다.', '커밋 완료' (with a checked green circle icon), 'GZ-1 refactor()', '직을 리팩토링하여 활성화 상태 검증 개선', '변경 내용', '개선 효과', and a bulleted list of changes.

# 소소한 팁 - Commands

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- 예시 2. Pull Request 생성
- Pull Request Template에 따라 PR 본문을 작성하고 PR 생성

The screenshot shows a comparison between a command-line interface and a GitHub pull request template.

**Command-line interface:**

```
> /pull-request
▶▶ auto-accept edits on (shift+tab to ...)
```

**Github Pull Request Template:**

jeremy-kim\_karrot commented 2 days ago

Review

- 리뷰 긴급도: MEDIUM

개요

Pull Request 자동 생성을 위한 slash command 문서를 추가합니다.

변경사항

- .claude/commands/pull-request.md 파일 추가
- API 에러 해결을 위해 문서 간소화

커밋 내역

- [725c3a1](#) GZ-1 docs: simplify pull-request command to fix API error
- [bba5073](#) GZ-1 docs: Pull Request 생성 커맨드 문서 추가

테스트

로컬 환경에서 동작 확인

Generated with [Claude Code](#)

...

# 소소한 팁 - Commands

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- 예시 3. Code Review
- 현재 수정된 코드 기준으로 보안·버그·성능·품질 스캔하고 수정 제안

```
* Thinking...
Now let me create the review report:
● Code Review Report

TODO 목록
- security
- security
- security
- bug | gra
- perf | gl
- quality
 추가

상세 리포트
● 보안 이슈

추가를 권장합니다.

g FROM node:24-alpine :1
기본적으로 root 사용자로 실행됩니다. 다음과 같이 non-root 사용자를 추가하세요:
RUN addgroup -g 1001 -S nodejs
RUN adduser -S nodejs -u 1001
USER nodejs

| IE not set()

● 버그/오류 처리
g serve-handler 실행 중 오류 처리가 없습니다. try-catch 블록이나 에러 핸들러를 추가하세요.

● 성능 개선
현재 전체 node_modules를 포함한 큰 이미지를 생성합니다. Multi-stage 빌드로 최종 이미지 크기를 줄일 수 있습니다.

● 코드 품질
g 개인 이메일이 하드코딩되어 있습니다. 환경 변수나 설정 파일로 관리하는 것이 좋습니다.

분석 완료
- 스캔 파일: 5개 (index.js, Dockerfile, package.json, build.gradle.kts, openapi.yaml)
- 발견 이슈: 6개 (보안 3, 버그 1, 성능 1, 품질 1)
- 분석 실패: 0개
```

# 소소한 팁 - Commands

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- 예시 4. OpenAI o3 활용(MCP)
- 다단계 추론과 문제 해결이 필요할 때 o3 모델에 프롬프팅 활용

```
> /start is running... @.claude/commands 에 선언된 명령어들이 Claude code 사용 환경에 적합한 것들인지 분석해줘.  
[REDACTED]
```

\* Thinking...

사용자가 `.claude/commands`에 선언된 명령어들이 `Claude code` 사용 환경에 적합한지 분석해달라고 요청했습니다.

`/start` 명령이 실행되고 있으며, 이는 다음과 같은 단계를 요구합니다:

1. 개발 가이드라인 출력
2. `TODO` 도구를 사용한 작업 계획 수립
3. 코드베이스 탐색 및 논의

먼저 `.claude/commands` 디렉토리의 파일들을 확인해보겠습니다. 현재 다음 파일들이 있습니다:

- `commit.md`
- `pull-request.md`
- `review.md`
- `start.md`
- `test.md`

이 파일들을 읽어서 내용을 분석해야 합니다. 그전에 개발 가이드라인을 먼저 확인해야 합니다.

- 먼저 개발 가이드라인을 확인하고, 작업 계획을 수립한 후 명령어들을 분석하겠습니다.

● **Update Todos**

- └  `rules/` 디렉토리의 개발 가이드라인 문서들 읽기
- `.claude/commands` 디렉토리의 모든 명령어 파일 읽기
- 각 명령어가 `Claude code` 사용 환경에 적합한지 분석
- o3와 함께 분석 결과 논의 및 개선점 도출

# 소소한 팁 - Commands

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- 예시 4. OpenAI o3 활용(MCP)
- 다단계 추론과 문제 해결이 필요할 때 o3 모델에 프롬프팅 활용

- 이제 각 명령어가 Claude code 사용 환경에 적합한지 분석해보겠습니다.

● **o3-search - o3-search (MCP)**(input: "Context": \n- .claude/commands 디렉토리에 5개의 명령어가 정의되어 있음\n- 각 명령어는 개발 워크플로우를 자동화하는 역할을 함\n- Claude code는 Anthropic의 공식 CLI 도구로 AI 어시스턴트가 개발 작업을 지원함\n\*\*명령어 목록\*\*: \n1. `/commit` - Git 커밋 메시지를 AI가 자동 생성 (티켓 번호 + C\n2. `/pull-request` - 현재 변경사항을 기반으로 PR 자동 생성 (gh CLI 사용)\n3. `/review` - 변경된

● **o3-search - o3-search (MCP)**(input: "Context": \n- .claude/commands 디렉토리에 5개의 명령어가 정의되어 있음\n- 각 명령어는 개발 워크플로우를 자동화하는 역할을 함\n- Claude code는 Anthropic의 공식 CLI 도구로 AI 어시스턴트가 개발 작업을 지원함\n\*\*명령어 목록\*\*: \n1. `/commit` - Git 커밋 메시지를 AI가 자동 생성 (티켓 번호 + Conventional Commits 형식)\n2. `/pull-request` - 현재 변경사항을 기반으로 PR 자동 생성 (gh CLI 사용)\n3. `/review` - 변경된 파일의 보안/버그/성능/품질 문제를 자동 분석\n4. `/start` - 개발 가이드라인 출력 → TODO 계획 → o3와 논의 프로세스\n5. `/test` - 테스트 자동 실행 및 실패 진단/해결 가이드 제시\n\*\*Objective\*\*: 각 명령어가 Claude code 환경에서 효과적으로 작동할 수 있는지 분석하고, 개선이 필요한 부분이 있는지 평가해주세요.\n\*\*Constraints\*\*: \n- Claude code는 파일 시스템 접근, 명령어 실행, 코드 분석 등이 가능함\n- 사용자가 명시적으로 요청하지 않는 한 자동으로 코드를 수정하거나 커밋하지 않아야 함\n- 보안과 안전성을 최우선으로 고려해야 함\n\*\*Output Format\*\*: \n각 명령어 별로 다음을 포함한 분석:\n- Claude code 환경에서의 적합성 (적합/부분적합/부적합)\n- 장점과 우려사항\n- 개선 제안 사항")

# 소소한 팁 - Commands

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- 예시 4. OpenAI o3 활용(MCP)
- 다단계 추론과 문제 해결이 필요할 때 o3 모델에 프롬프팅 활용

\* *Thinking...*

이제 o3의 구체적인 개선안을 얻었습니다. *TODO*를 완료로 표시하고 분석 결과를 정리해보겠습니다.

- **Update Todos**

- └ ☐ rules/ 디렉토리의 개발 가이드라인 문서들 읽기
- ☐ .claude/commands 디렉토리의 모든 명령어 파일 읽기
- ☐ 각 명령어가 Claude code 사용 환경에 적합한지 분석
- ☐ o3와 함께 분석 결과 논의 및 개선점 도출

# 소소한 팁 - Commands

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- 예시 4. OpenAIo3 활용(MCP)
- 다단계 추론과 문제 해결이 필요할 때 o3 모델에 프롬프팅 활용

\* Thinking...

이제 o3의 구체적인 개선안을 얻었습니다. *TODO*를

## ● Update Todos

- rules/ 디렉토리의 개발 가이드라인 문서들
- .claude/commands 디렉토리의 모든 명령어 API
- 각 명령어가 Claude code 사용 환경에 적합한지
- o3와 함께 분석 결과 논의 및 개선점 도출

### • .claude/commands 명령어 분석 결과

#### 1. 전반적 평가

- 5개 명령어 모두 Claude code 환경에서 적합하거나 부분적합으로 평가됨
- 주요 강점: 개발 워크플로우 자동화로 생산성 향상
- 주요 개선점: 안전성 강화 및 사용자 확인 프로세스 필요

#### 2. 명령어별 적합성 평가

명령어	적합성	주요 개선 필요사항
/commit	✓ 적합	--dry-run 옵션, 사용자 확인 절차
/pull-request	⚠️ 부분적합	gh 토큰 안전성, draft PR 우선
/review	✓ 적합	분석 레벨 선택, 패치 제안 기능
/start	✓ 적합	가이드라인 경로 설정 가능
/test	⚠️ 부분적합	스마트 테스트 스코프, 자동 패치 확인

#### 3. 핵심 개선 제안

##### 💡 공통 안전성 강화

- 모든 명령에 --dry-run 옵션 추가
- 실행 전 변경사항 요약 및 사용자 확인 프롬프트
- 작업 로그를 ~/.claude-logs/에 자동 저장

##### 👉 명령어별 주요 개선점

###### /commit

- 멀티스테이지 메시지 생성 (diff → 요약 → 제목/본문)
- .claude-config.yml에서 커밋 규칙 커스터마이징

###### /pull-request

- draft PR로 먼저 생성 후 --ready 옵션으로 최종화
- 대용량 diff 자동 요약 처리

###### /review

- 분석 모드 선택: quick(2초)/standard(30초)/deep(2분)
- 제안사항별 패치 파일 생성 옵션

###### /start

- README/Makefile 자동 분석으로 빠른 시작
- TODO를 로컬(.todo.md) 또는 GitHub Issues 중 선택

###### /test

- 변경 파일 연관 테스트만 우선 실행 (--smart)
- 실패 원인 자동 분석 및 수정 제안

#### 4. 구현 권장 사항

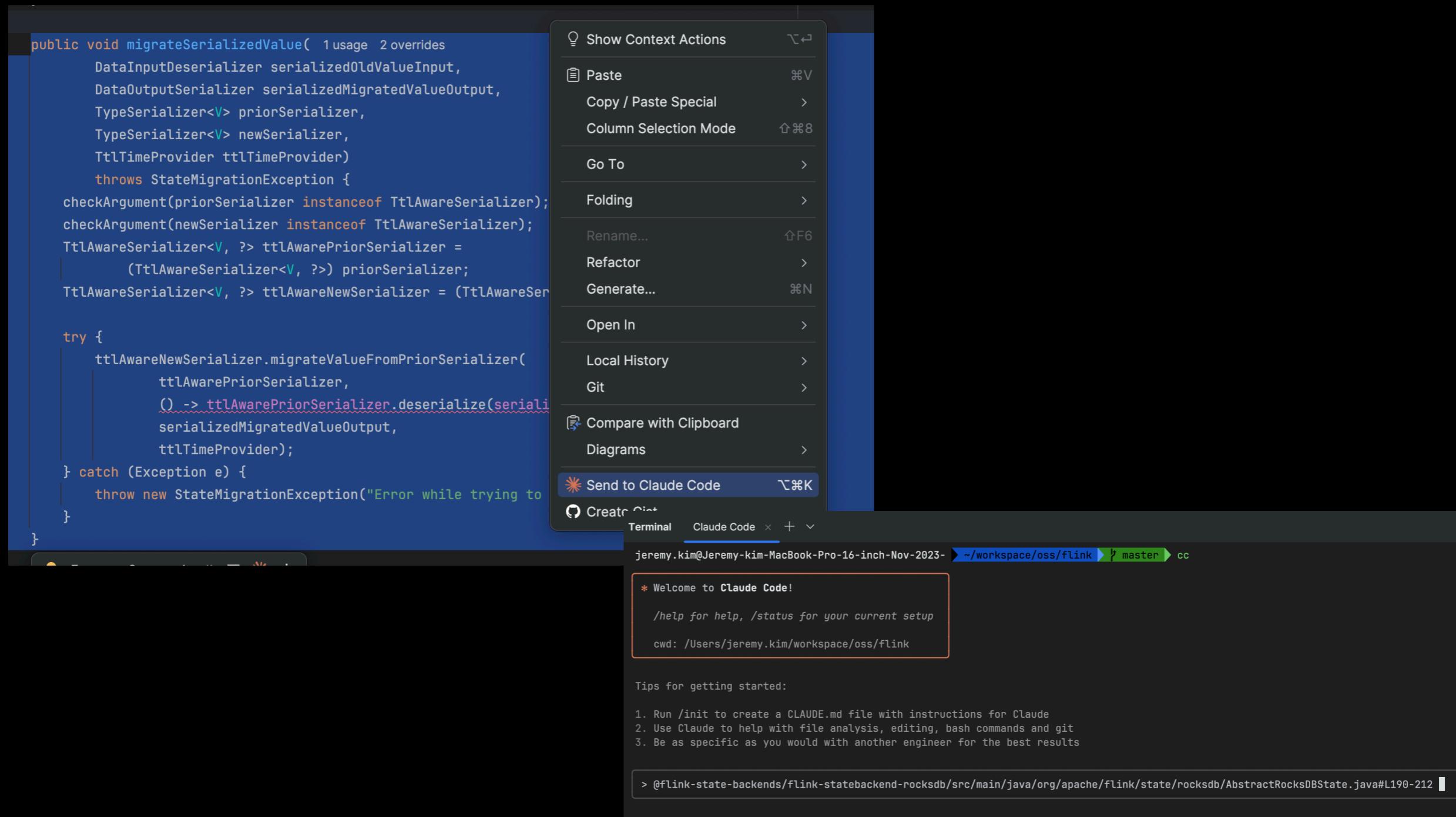
1. 투명성 확보: 모든 bash 명령 실행 내용을 사용자에게 표시
2. 설정 파일 도입: .claude-config.yml로 프로젝트별 규칙 관리
3. 비대화형 구현: 대화형 입력 대신 플래그/옵션 활용
4. 백업 메커니즘: 위험한 작업 전 자동 백업 생성

이러한 개선사항들을 적용하면 Claude code 환경에서 더욱 안전하고 효율적인 개발 워크플로우를 구축할 수 있습니다.

# 소소한 팁 - IntelliJ IDEA

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

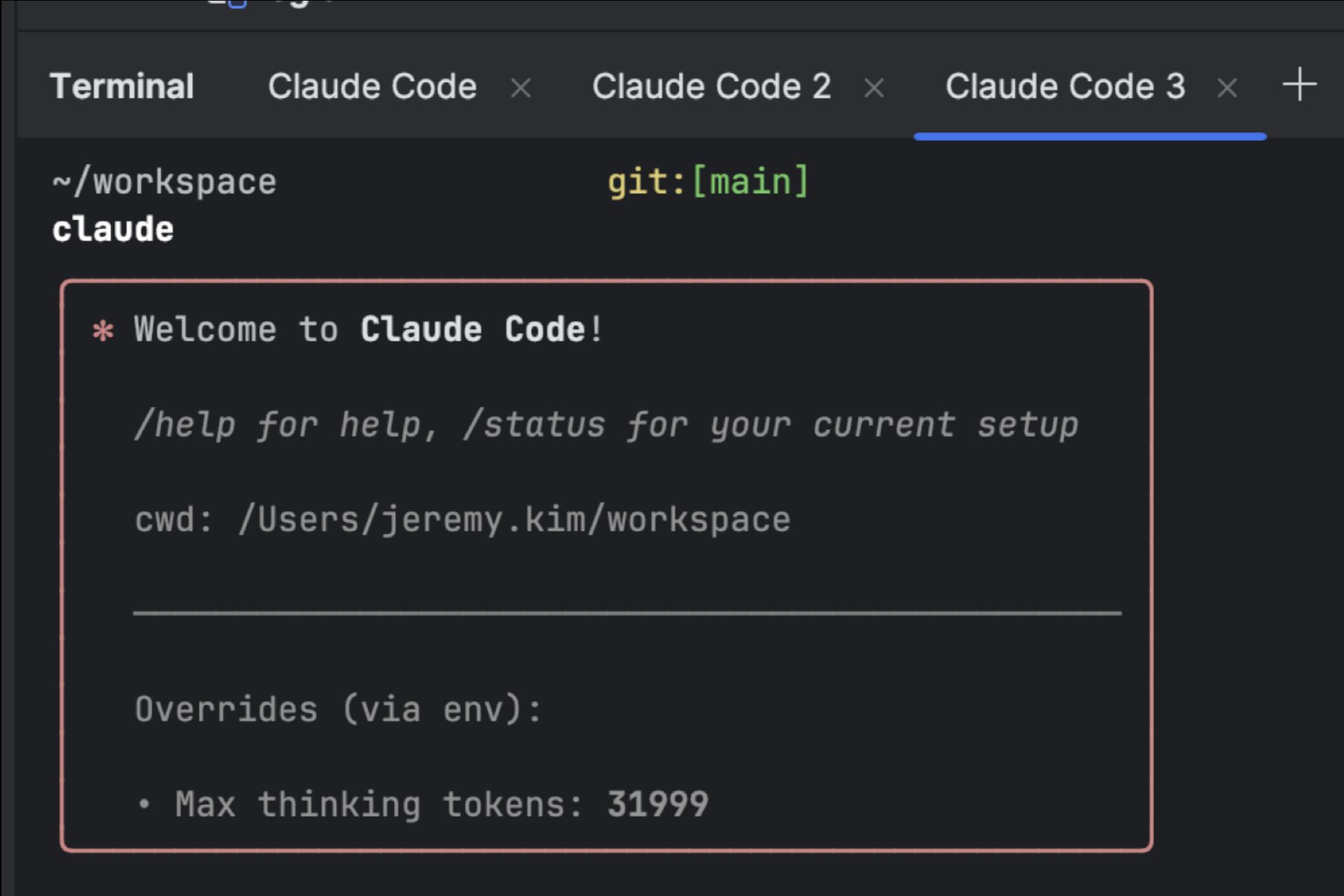
- 프롬프트에 코드 위치 공유하기



# 소소한 팁 - IntelliJ IDEA

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- N개의 터미널에 CC Instance 실행하기



The screenshot shows the IntelliJ IDEA interface with three terminal tabs open at the top: "Terminal", "Claude Code", and "Claude Code 2". The "Claude Code" tab is active, displaying a terminal session. The session shows the user has navigated to their workspace directory (~/.workspace) and run the command "claude". The output from the "claude" command is highlighted with a red box and contains the following text:

```
* Welcome to Claude Code!
/help for help, /status for your current setup
cwd: /Users/jeremy.kim/workspace
_____
Overrides (via env):
• Max thinking tokens: 31999
```

# 소소한 팁 - ccusage

Claude Code 101:

한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- ccusage로 사용량 추적하자

2025-07-18	- opus-4	142	888	88,173	553,824	643,027	\$2.55	
2025-07-19	- opus-4 - sonnet-4	1,804	8,168	899,270	19,987,931	20,897,173	\$26.63	
2025-07-20	- opus-4 - sonnet-4	659	327	198,239	4,470,378	4,669,603	\$6.89	
2025-07-21	- opus-4 - sonnet-4	1,542	9,385	483,242	5,338,958	5,833,127	\$9.95	
2025-07-22	- opus-4 - sonnet-4	1,653	15,277	1,027,726	7,588,931	8,633,587	\$18.23	
2025-07-24	- opus-4 - sonnet-4	8,293	34,435	3,961,542	48,242,416	52,246,686	\$51.74	
2025-07-25	- opus-4 - sonnet-4	1,705	23,930	602,866	8,853,751	9,482,252	\$5.84	
2025-07-26	- opus-4 - sonnet-4	4,395	10,730	1,644,074	26,067,831	27,727,030	\$67.23	
2025-07-27	- opus-4	25,696	3,097	1,185,206	16,921,728	18,135,727	\$48.22	
Total		89,512	374,242	18,827,238	299,637,720	318,928,712	\$350.03	

# 소소한 팁 - ccusage

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- 실시간 대시보드(**ccusage blocks --live**)로 에이전트가 속도 제한에 가까워지고 있는지 파악하자

```
npx ccusage@latest blocks --live

CLAUDE CODE - LIVE TOKEN USAGE MONITOR

⌚ SESSION [██████████] 93.8%
Started: 01:00:00 PM Elapsed: 4h Remaining: 18m (06:00:00 PM)

🔥 USAGE [██████████] 22.0% (7983.4k/36340.1k)
Tokens: 7,983,430 (Burn Rate: 32,384 token/min ✓ NORMAL) Limit: 36,340,072 tokens Cost: $23.25

📈 PROJECTION [██████████] 23.6% (8588.3k/36340.1k)
Status: ✓ WITHIN LIMIT Tokens: 8,588,333 Cost: $25.01

⚙️ Models: opus-4

↻ Refreshing every 1s • Press Ctrl+C to stop
```

# 소소한 팁 - Env Variables

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- CLAUDE\_CODE\_DISABLE\_NONESSENTIAL\_TRAFFIC=true
  - : 텔레메트리, 오류 보고, 자동 업데이트를 비활성화
- ENABLE\_BACKGROUND\_TASKS=true
  - : 백그라운드 작업 기능을 활성화 (장시간 실행되는 명령에 유용)
- FORCE\_AUTO\_BACKGROUND\_TASKS=true
  - : 장시간 작업을 자동으로 백그라운드로 전환 (사용자 확인 불필요)
- CLAUDE\_CODE\_ENABLE\_UNIFIED\_READ\_TOOL=true
  - : 파일 읽기 기능을 통합 (예: Jupyter Notebook 포함)

# 소소한 팁 - Env Variables

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- 장시간 작업(예: 개발 서버) 자동 백그라운드 처리
- 텔레메트리 및 불필요한 네트워크 트래픽 차단
- 통합된 파일 읽기 기능
- 편리한 옵션 스위치( -y: 자동 승인 -r: 세션 이어가기)

```
function cc() {
    local env_vars=(
        "ENABLE_BACKGROUND_TASKS=true"
        "FORCE_AUTO_BACKGROUND_TASKS=true"
        "CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC=true"
        "CLAUDE_CODE_ENABLE_UNIFIED_READ_TOOL=true"
    )

    local claude_args=()

    if [[ "$1" = "-y" ]]; then
        claude_args+=("--dangerously-skip-permissions")
    elif [[ "$1" = "-r" ]]; then
        claude_args+=("--resume")
    elif [[ "$1" = "-ry" || "$1" = "-yr" ]]; then
        claude_args+=("--resume" "--dangerously-skip-permissions")
    fi

    env "${env_vars[@]}" claude "${claude_args[@]}"
}
```

# 소소한 팁 - Env Variables

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법



```
function cc() {
    local env_vars=(
        "ENABLE_BACKGROUND_TASKS=true"
        "FORCE_AUTO_BACKGROUND_TASKS=true"
        "CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC=true"
        "CLAUDE_CODE_ENABLE_UNIFIED_READ_TOOL=true"
    )

    local claude_args=()

    if [[ "$1" = "-y" ]]; then
        claude_args+=("--dangerously-skip-permissions")
    elif [[ "$1" = "-r" ]]; then
        claude_args+=("--resume")
    elif [[ "$1" = "-ry" || "$1" = "-yr" ]]; then
        claude_args+=("--resume" "--dangerously-skip-permissions")
    fi

    env "${env_vars[@]}" claude "${claude_args[@]}"
}
```

kspace/daangn/growthz ➔ ↗ main ➔ cc

> Try "refactor farm"

? for shortcuts

# 소소한 팁 - parallel

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- `$ claude config set --global parallelTasksCount 5`
- 큰 작업들을 병렬로 나누어서 실행하고 싶을 때 사용
- <https://github.com/anthropics/clause-code/issues/2581>

# 소소한 팁 - Git Worktree

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- 하나의 Git 저장소에서 여러 작업 디렉토리를 동시에 관리하는 기능
- 하나의 .git 데이터를 공유 → 중복 저장소 없이 공간 절약
- 빠른 브랜치 전환 없이 여러 작업 병행 가능

# 소소한 팁 - Git Worktree

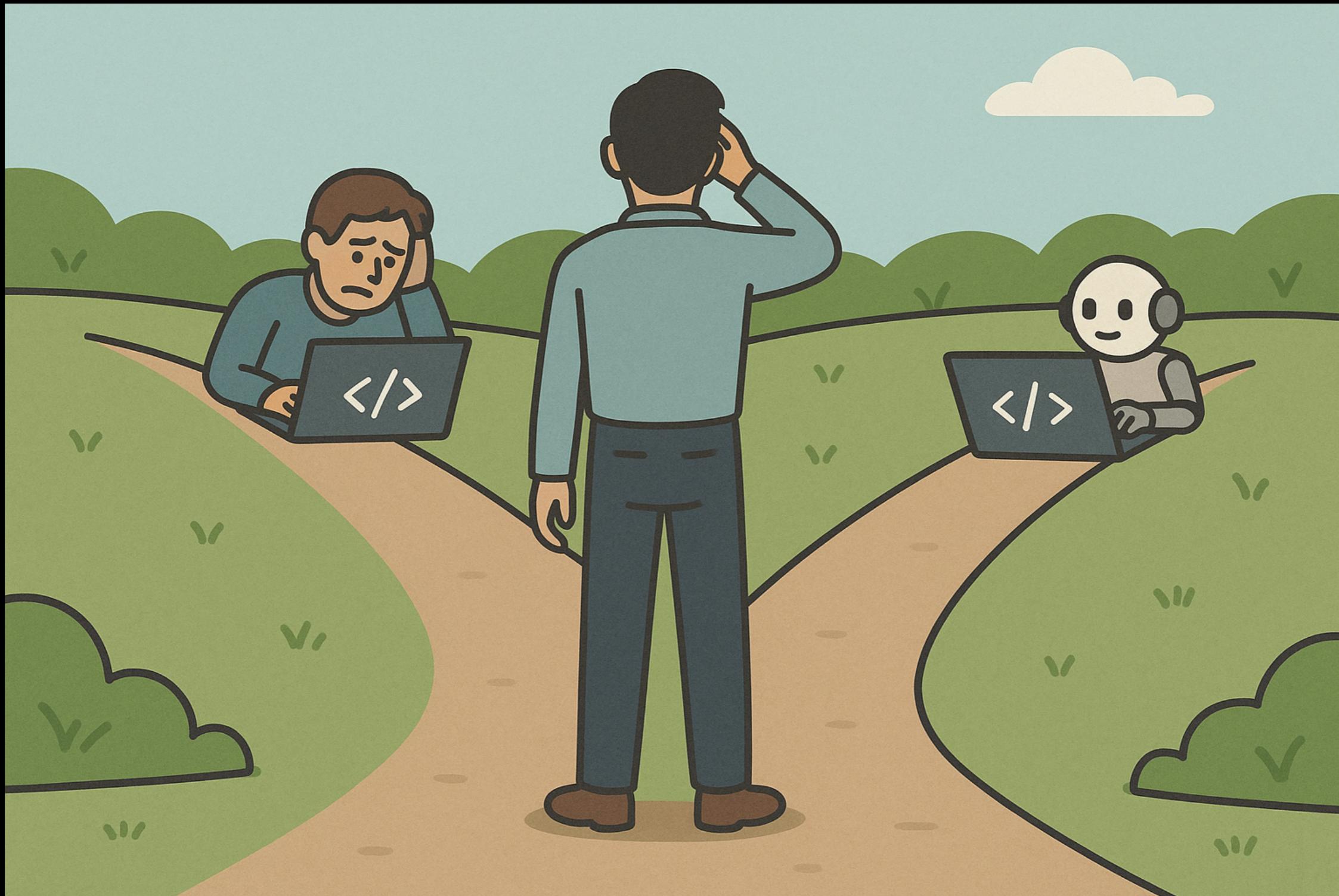
Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법

- <https://github.com/smtg-ai/clause-squad>
- <https://conductor.build/>
- <https://www.npmjs.com/package/wm2>

끝  
≡

# 가장 중요한 것 - 중꺾마

Claude Code 101:  
한 번 쓰고 끝이 아니다: 끊임없이 진화하는 코딩 에이전트 활용법



못하면 가르치자  
해냈으면 회고하자  
복리의 마법을 믿자