Claude Code

전체 문서

By Anthropic

제작일 : 2026-03-17

터미널에서 작동하는 에이전틱 코딩 도구로,
코드베이스를 이해하고 더 빠르게 코딩할 수 있도록 도와줍니다.

code.claude.com/docs/ko 에서 컴파일
최신 버전은 웹사이트를 방문하세요.

Generated by: github.com/bepsvpt/cc-docs

Table of Contents

Part 1: Getting Started

Claude Code 개요

> Claude Code는 코드베이스를 읽고, 파일을 편집하고, 명령을 실행하고, 개발 도구와 통합하는 에이전트 코딩 도구입니다. 터미널, IDE, 데스크톱 앱, 브라우저에서 사용할 수 있습니다.

Claude Code는 기능을 구축하고, 버그를 수정하고, 개발 작업을 자동화하는 데 도움이 되는 AI 기반 코딩 어시스턴트입니다. 전체 코드베이스를 이해하고 여러 파일과 도구에 걸쳐 작업할 수 있습니다.

시작하기

환경을 선택하여 시작하세요. 대부분의 환경에는 Claude 구독 또는 Anthropic Console 계정이 필요합니다. Terminal CLI와 VS Code는 타사 제공자도 지원합니다.

Terminal

터미널에서 Claude Code를 직접 사용하기 위한 모든 기능을 갖춘 CLI입니다. 파일을 편집하고, 명령을 실행하고, 명령줄에서 전체 프로젝트를 관리할 수 있습니다.

To install Claude Code, use one of the following methods:

Native Install (Recommended)

macOS, Linux, WSL:

```
curl -fsSL https://claude.ai/install.sh | bash
```

Windows PowerShell:

```
irm https://claude.ai/install.ps1 | iex
```

Windows CMD:

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd && del
install.cmd
```

Windows requires Git for Windows. Install it first if you don’t have it.

> **Info:**
>
> Native installations automatically update in the background to keep you on the latest version.

Homebrew

```
brew install --cask claude-code
```

> **Info:**
>
> Homebrew installations do not auto-update. Run `brew upgrade claude-code` periodically to get the latest features and security fixes.

WinGet

```
winget install Anthropic.ClaudeCode
```

> **Info:**
>
> WinGet installations do not auto-update. Run `winget upgrade Anthropic.ClaudeCode` periodically to get the latest features and security fixes.

그런 다음 모든 프로젝트에서 Claude Code를 시작합니다:

```
cd your-project
claude
```

처음 사용할 때 로그인하라는 메시지가 표시됩니다. 완료되었습니다! 빠른 시작으로 계속하기 →

> **Tip:**

> 설치 옵션, 수동 업데이트 또는 제거 지침은 고급 설정을 참조하세요. 문제가 발생하면 문제 해결을 방문하세요.

VS Code

VS Code 확장 프로그램은 편집기에서 직접 인라인 diff, @-mentions, 계획 검토 및 대화 기록을 제공합니다.

- VS Code용 설치
- Cursor용 설치

또는 확장 프로그램 보기(`Mac에서 Cmd+Shift+X`, `Windows/Linux에서 Ctrl+Shift+X`)에서 "Claude Code"를 검색합니다. 설치 후 명령 팔레트(`Cmd+Shift+P` / `Ctrl+Shift+P`)를 열고 "Claude Code"를 입력한 다음 **새 탭에서 열기**를 선택합니다.

VS Code 시작하기 →

Desktop app

IDE 또는 터미널 외부에서 Claude Code를 실행하기 위한 독립 실행형 앱입니다. diff를 시각적으로 검토하고, 여러 세션을 나란히 실행하고, 반복되는 작업을 예약하고, 클라우드 세션을 시작할 수 있습니다.

다운로드 및 설치:

- macOS (Intel 및 Apple Silicon)
- Windows (x64)
- Windows ARM64 (원격 세션만 해당)

설치 후 Claude를 실행하고, 로그인한 다음, **Code** 탭을 클릭하여 코딩을 시작합니다. 유료 구독이 필요합니다.

데스크톱 앱에 대해 자세히 알아보기 →

Web

로컬 설정 없이 브라우저에서 Claude Code를 실행합니다. 오래 실행되는 작업을 시작하고 완료되면 다시 확인하고, 로컬에 없는 리포지토리에서 작업하거나, 여러 작업을 병렬로 실행할 수 있습니다. 데스크톱 브라우저 및 Claude iOS 앱에서 사용할 수 있습니다.

claude.ai/code에서 코딩을 시작합니다.

웹에서 시작하기 →

JetBrains

IntelliJ IDEA, PyCharm, WebStorm 및 기타 JetBrains IDE용 플러그인으로 대화형 diff 보기 및 선택 컨텍스트 공유 기능이 있습니다.

JetBrains Marketplace에서 Claude Code 플러그인을 설치하고 IDE를 다시 시작합니다.

JetBrains 시작하기 →

할 수 있는 작업

Claude Code를 사용할 수 있는 몇 가지 방법은 다음과 같습니다:

Claude Code는 하루를 낭비하는 지루한 작업을 처리합니다: 테스트되지 않은 코드에 대한 테스트 작성, 프로젝트 전체의 lint 오류 수정, 병합 충돌 해결, 종속성 업데이트, 릴리스 노트 작성.

```
claude "write tests for the auth module, run them, and fix any failures"
```

원하는 내용을 일반 언어로 설명합니다. Claude Code는 접근 방식을 계획하고, 여러 파일에 걸쳐 코드를 작성하고, 작동하는지 확인합니다.

버그의 경우 오류 메시지를 붙여넣거나 증상을 설명합니다. Claude Code는 코드베이스를 통해 문제를 추적하고, 근본 원인을 파악하고, 수정 사항을 구현합니다. 더 많은 예제는 일반적인 워크플로우를 참조하세요.

Claude Code는 git과 직접 작동합니다. 변경 사항을 스테이징하고, 커밋 메시지를 작성하고, 브랜치를 생성하고, 풀 요청을 엽니다.

```
claude "commit my changes with a descriptive message"
```

CI에서는 GitHub Actions 또는 GitLab CI/CD를 사용하여 코드 검토 및 이슈 분류를 자동화할 수 있습니다.

Model Context Protocol (MCP)는 AI 도구를 외부 데이터 소스에 연결하기 위한 개방형 표준입니다. MCP를 사용하면 Claude Code는 Google Drive의 디자인 문서를 읽고, Jira의 티켓을 업데이트하고, Slack에서 데이터를 가져오거나, 자신의 커스텀 도구를 사용할 수 있습니다.

`CLAUDE.md` 는 프로젝트 루트에 추가하는 마크다운 파일로, Claude Code가 모든 세션의 시작 부분에서 읽습니다. 이를 사용하여 코딩 표준, 아키텍처 결정, 선호하는 라이브러리 및 검토 체크리스트를 설정합니다. Claude는 또한 작업할 때 자동 메모리를 구축하여 빌드 명령 및 디버깅 인사이트와 같은 학습 내용을 저장하므로 아무것도 작성할 필요가 없습니다.

커스텀 명령을 생성하여 팀이 공유할 수 있는 반복 가능한 워크플로우를 패키징합니다(예: `/review-pr` 또는 `/deploy-staging`).

Hooks를 사용하면 Claude Code 작업 전후에 셸 명령을 실행할 수 있습니다(예: 모든 파일 편집 후 자동 포맷팅 또는 커밋 전 lint 실행).

작업의 다른 부분에서 동시에 작동하는 여러 Claude Code 에이전트를 생성합니다. 리드 에이전트가 작업을 조정하고, 하위 작업을 할당하고, 결과를 병합합니다.

완전히 커스텀 워크플로우의 경우, Agent SDK를 사용하면 Claude Code의 도구 및 기능으로 구동되는 자신의 에이전트를 구축할 수 있으며, 오케스트레이션, 도구 액세스 및 권한에 대한 완전한 제어가 가능합니다.

Claude Code는 구성 가능하며 Unix 철학을 따릅니다. 로그를 파이프하고, CI에서 실행하거나, 다른 도구와 연결합니다:

```
## Monitor logs and get alerted
tail -f app.log | claude -p "Slack me if you see any anomalies"

## Automate translations in CI
claude -p "translate new strings into French and raise a PR for review"

## Bulk operations across files
git diff main --name-only | claude -p "review these changed files for security
issues"
```

전체 명령 및 플래그 세트는 CLI 참조를 참조하세요.

세션은 단일 환경에 연결되지 않습니다. 컨텍스트가 변경되면 환경 간에 작업을 이동합니다:

- 원격 제어를 사용하여 책상에서 떠나 휴대폰이나 모든 브라우저에서 계속 작업합니다
- 웹 또는 iOS 앱에서 오래 실행되는 작업을 시작한 다음 `/teleport`를 사용하여 터미널로 가져옵니다
- `/desktop`을 사용하여 터미널 세션을 데스크톱 앱으로 전달하여 시각적 diff 검토를 수행합니다
- Slack에서 `@Claude`를 언급하여 팀 채팅에서 작업을 라우팅하고 버그 보고서를 받으면 풀 요청을 다시 받습니다

모든 곳에서 Claude Code 사용

각 환경은 동일한 기본 Claude Code 엔진에 연결되므로 CLAUDE.md 파일, 설정 및 MCP 서버가 모든 환경에서 작동합니다.

위의 Terminal, VS Code, JetBrains, Desktop 및 Web 환경 외에도 Claude Code는 CI/CD, 채팅 및 브라우저 워크플로우와 통합됩니다:

원하는 작업	최적의 옵션
휴대폰이나 다른 기기에서 로컬 세션 계속하기	원격 제어
로컬에서 작업 시작, 모바일에서 계속하기	웹 또는 Claude iOS 앱
PR 검토 및 이슈 분류 자동화	GitHub Actions 또는 GitLab CI/CD
모든 PR에서 자동 코드 검토 받기	GitHub Code Review
Slack의 버그 보고서를 풀 요청으로 라우팅	Slack
라이브 웹 애플리케이션 디버깅	Chrome
자신의 워크플로우를 위한 커스텀 에이전트 구축	Agent SDK

다음 단계

Claude Code를 설치한 후 이 가이드를 통해 더 깊이 있게 학습할 수 있습니다.

- 빠른 시작: 코드베이스 탐색에서 수정 사항 커밋까지 첫 번째 실제 작업을 진행합니다
- 지침 및 메모리 저장: CLAUDE.md 파일 및 자동 메모리를 사용하여 Claude에 영구적인 지침을 제공합니다
- 일반적인 워크플로우 및 모범 사례: Claude Code에서 최대한의 이점을 얻기 위한 패턴
- 설정: Claude Code를 워크플로우에 맞게 사용자 정의합니다
- 문제 해결: 일반적인 문제에 대한 솔루션
- code.claude.com: 데모, 가격 책정 및 제품 세부 정보

빠른 시작

> Claude Code에 오신 것을 환영합니다!

이 빠른 시작 가이드를 통해 몇 분 안에 AI 기반 코딩 지원을 사용할 수 있습니다. 이 가이드를 마치면 일반적인 개발 작업에 Claude Code를 사용하는 방법을 이해하게 됩니다.

시작하기 전에

다음을 확인하십시오:

- 열려 있는 터미널 또는 명령 프롬프트
  - 터미널을 처음 사용하는 경우 터미널 가이드를 확인하십시오
- 작업할 코드 프로젝트
- Claude 구독(Pro, Max, Teams 또는 Enterprise), Claude Console 계정 또는 지원되는 클라우드 제공자를 통한 액세스

> **Note:**
>
> 이 가이드는 터미널 CLI를 다룹니다. Claude Code는 웹, 데스크톱 앱, VS Code 및 JetBrains IDE, Slack, GitHub Actions 및 GitLab의 CI/CD에서도 사용할 수 있습니다. 모든 인터페이스를 참조하십시오.

단계 1: Claude Code 설치

To install Claude Code, use one of the following methods:

Native Install (Recommended)

macOS, Linux, WSL:

```
curl -fsSL https://claude.ai/install.sh | bash
```

Windows PowerShell:

```
irm https://claude.ai/install.ps1 | iex
```

Windows CMD:

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd && del install.cmd
```

Windows requires Git for Windows. Install it first if you don't have it.

> **Info:**
>
> Native installations automatically update in the background to keep you on the latest version.

Homebrew

```
brew install --cask claude-code
```

> **Info:**
>
> Homebrew installations do not auto-update. Run `brew upgrade claude-code` periodically to get the latest features and security fixes.

WinGet

```
winget install Anthropic.ClaudeCode
```

> **Info:**
>
> WinGet installations do not auto-update. Run `winget upgrade Anthropic.ClaudeCode` periodically to get the latest features and security fixes.

단계 2: 계정에 로그인

Claude Code를 사용하려면 계정이 필요합니다. `claude` 명령으로 대화형 세션을 시작할 때 로그인해야 합니다:

```
claude
## 처음 사용할 때 로그인하라는 메시지가 표시됩니다
```

```
/login
## 프롬프트를 따라 계정으로 로그인하십시오
```

다음 계정 유형 중 하나로 로그인할 수 있습니다:

- Claude Pro, Max, Teams 또는 Enterprise(권장)
- Claude Console(선불 크레딧이 있는 API 액세스). 처음 로그인할 때 "Claude Code" 작업 공간이 Console에서 자동으로 생성되어 비용 추적을 중앙화합니다.
- Amazon Bedrock, Google Vertex AI 또는 Microsoft Foundry(엔터프라이즈 클라우드 제공자)

로그인하면 자격 증명이 저장되고 다시 로그인할 필요가 없습니다. 나중에 계정을 전환하려면 `/login` 명령을 사용하십시오.

단계 3: 첫 번째 세션 시작

프로젝트 디렉토리에서 터미널을 열고 Claude Code를 시작하십시오:

```
cd /path/to/your/project
claude
```

세션 정보, 최근 대화 및 최신 업데이트가 포함된 Claude Code 환영 화면이 표시됩니다. 사용 가능한 명령을 보려면 `/help`를 입력하거나 이전 대화를 계속하려면 `/resume`을 입력하십시오.

> **Tip:**
>
> 로그인(단계 2) 후 자격 증명이 시스템에 저장됩니다. 자격 증명 관리에서 자세히 알아보십시오.

단계 4: 첫 번째 질문 하기

코드베이스를 이해하는 것부터 시작하겠습니다. 다음 명령 중 하나를 시도하십시오:

```
이 프로젝트는 무엇을 하나요?
```

Claude가 파일을 분석하고 요약을 제공합니다. 더 구체적인 질문을 할 수도 있습니다:

```
이 프로젝트는 어떤 기술을 사용하나요?
```

```
주요 진입점은 어디인가요?
```

```
폴더 구조를 설명해주세요
```

Claude의 기능에 대해 물어볼 수도 있습니다:

```
Claude Code는 무엇을 할 수 있나요?
```

```
Claude Code에서 사용자 정의 skills를 만드는 방법은?
```

```
Claude Code는 Docker와 함께 작동할 수 있나요?
```

> **Note:**
>
> Claude Code는 필요에 따라 프로젝트 파일을 읽습니다. 수동으로 컨텍스트를 추가할 필요가 없습니다.

단계 5: 첫 번째 코드 변경 수행

이제 Claude Code가 실제 코딩을 하도록 해봅시다. 간단한 작업을 시도하십시오:

```
주 파일에 hello world 함수 추가
```

Claude Code는 다음을 수행합니다:

1. 적절한 파일 찾기
2. 제안된 변경 사항 표시
3. 승인 요청
4. 편집 수행

> **Note:**
>
> Claude Code는 파일을 수정하기 전에 항상 권한을 요청합니다. 개별 변경 사항을 승인하거나 세션에 대해 "모두 수락" 모드를 활성화할 수 있습니다.

단계 6: Claude Code와 함께 Git 사용

Claude Code는 Git 작업을 대화형으로 만듭니다:

```
어떤 파일을 변경했나요?
```

```
설명적인 메시지로 변경 사항 커밋
```

더 복잡한 Git 작업을 요청할 수도 있습니다:

```
feature/quickstart라는 새 브랜치 생성
```

```
마지막  5개의 커밋 표시
```

```
병합 충돌을 해결하는 데 도움을 주세요
```

단계 7: 버그 수정 또는 기능 추가

Claude는 디버깅 및 기능 구현에 능숙합니다.

자연어로 원하는 것을 설명하십시오:

```
사용자 등록 양식에 입력 유효성 검사 추가
```

또는 기존 문제를 수정하십시오:

```
사용자가 빈 양식을 제출할 수 있는 버그가 있습니다  -  수정하세요
```

Claude Code는 다음을 수행합니다:

- 관련 코드 찾기
- 컨텍스트 이해
- 솔루션 구현
- 사용 가능한 경우 테스트 실행

단계 8: 다른 일반적인 워크플로우 시도

Claude와 함께 작업하는 방법은 여러 가지입니다:

코드 리팩토링

```
인증 모듈을 콜백 대신 async/await를 사용하도록 리팩토링
```

테스트 작성

```
계산기 함수에 대한 단위 테스트 작성
```

문서 업데이트

```
설치 지침으로 README 업데이트
```

코드 검토

```
내 변경 사항을 검토하고 개선 사항을 제안해주세요
```

> **Tip:**
>
> 도움이 되는 동료와 대화하듯이 Claude와 대화하십시오. 달성하고 싶은 것을 설명하면 도움을 드릴 것입니다.

필수 명령

일상적인 사용을 위한 가장 중요한 명령은 다음과 같습니다:

명령	기능	예시
`claude`	대화형 모드 시작	`claude`
`claude "task"`	일회성 작업 실행	`claude "fix the build error"`
`claude -p "query"`	일회성 쿼리 실행 후 종료	`claude -p "explain this function"`
`claude -c`	현재 디렉토리에서 가장 최근 대화 계속	`claude -c`
`claude -r`	이전 대화 재개	`claude -r`
`claude commit`	Git 커밋 생성	`claude commit`
`/clear`	대화 기록 지우기	`/clear`
`/help`	사용 가능한 명령 표시	`/help`
`exit` 또는 Ctrl+C	Claude Code 종료	`exit`

전체 명령 목록은 CLI 참조를 참조하십시오.

초보자를 위한 팁

자세한 내용은 모범 사례 및 일반적인 워크플로우를 참조하십시오.

대신: “버그 수정”

시도: “사용자가 잘못된 자격 증명을 입력한 후 빈 화면을 보는 로그인 버그 수정”

복잡한 작업을 단계로 나누기:

```
1. 사용자 프로필을 위한 새 데이터베이스 테이블 생성
2. 사용자 프로필을 가져오고 업데이트하는 API 엔드포인트 생성
3. 사용자가 자신의 정보를 보고 편집할 수 있는 웹페이지 구축
```

변경하기 전에 Claude가 코드를 이해하도록 하기:

```
데이터베이스 스키마 분석
```

영국 고객이 가장 자주 반품하는 제품을 보여주는 대시보드 구축

- `?`를 눌러 사용 가능한 모든 키보드 바로가기 보기
- 명령 완성을 위해 Tab 사용
- ↑를 눌러 명령 기록 보기
- `/`를 입력하여 모든 명령 및 skills 보기

다음은?

기본 사항을 배웠으므로 더 고급 기능을 살펴보십시오:

- Claude Code 작동 방식 에이전트 루프, 기본 제공 도구 및 Claude Code가 프로젝트와 상호 작용하는 방식 이해
- 모범 사례 효과적인 프롬프팅 및 프로젝트 설정으로 더 나은 결과 얻기
- 일반적인 워크플로우 일반적인 작업에 대한 단계별 가이드
- Claude Code 확장 CLAUDE.md, skills, hooks, MCP 등으로 사용자 정의

도움 받기

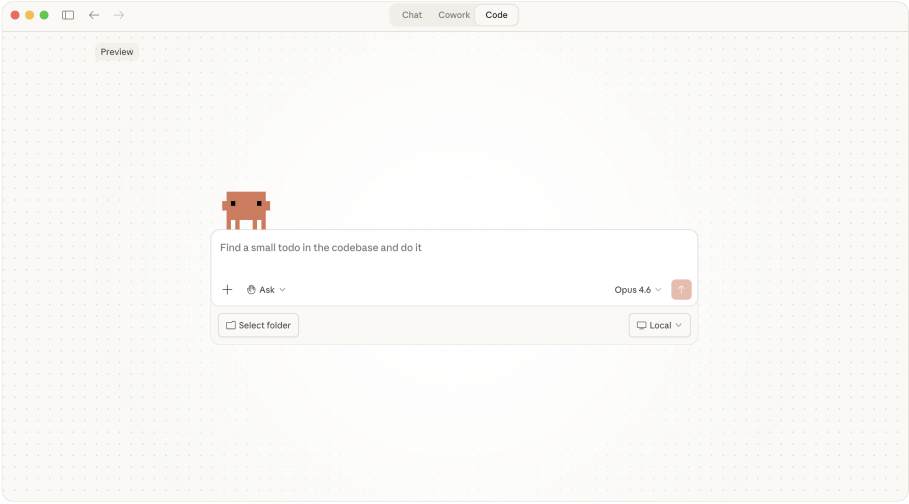
- **Claude Code에서**: `/help`를 입력하거나 "어떻게…"를 물어보기
- **문서**: 여기 있습니다! 다른 가이드 찾아보기
- **커뮤니티**: 팁과 지원을 위해 Discord에 참여하기

데스크톱 앱 시작하기

> 데스크톱에 Claude Code를 설치하고 첫 번째 코딩 세션을 시작합니다

데스크톱 앱은 Claude Code를 그래픽 인터페이스와 함께 제공합니다: 시각적 diff 검토, 라이브 앱 미리보기, GitHub PR 모니터링 및 자동 병합, Git worktree 격리를 통한 병렬 세션, 예약된 작업, 그리고 작업을 원격으로 실행할 수 있는 기능입니다. 터미널이 필요하지 않습니다.

이 페이지는 앱 설치 및 첫 번째 세션 시작을 안내합니다. 이미 설정되어 있다면 전체 참조는 Claude Code Desktop 사용을 참조하십시오.



Code 탭이 선택된 Claude Code Desktop 인터페이스로, 프롬프트 상자, 권한 모드 선택기(Ask permissions로 설정됨), 모델 선택기, 폴더 선택기, 그리고 Local 환경 옵션을 보여줍니다

데스크톱 앱에는 세 개의 탭이 있습니다:

- **Chat**: 파일 접근이 없는 일반 대화로, claude.ai와 유사합니다.
- **Cowork**: 자신의 환경을 가진 클라우드 VM에서 작업을 수행하는 자율 백그라운드 에이전트입니다. 사용자가 다른 작업을 하는 동안 독립적으로 실행될 수 있습니다.
- **Code**: 로컬 파일에 직접 접근할 수 있는 대화형 코딩 어시스턴트입니다. 실시간으로 각 변경 사항을 검토하고 승인합니다.

Chat과 Cowork는 Claude Desktop 지원 문서에서 다룹니다. 이 페이지는 **Code** 탭에 중점을 둡니다.

> **Note:**
>
> Claude Code는 Pro, Max, Teams, 또는 Enterprise 구독이 필요합니다.

설치

Step 1: 앱 다운로드

플랫폼에 맞는 Claude를 다운로드합니다.

- macOS Intel 및 Apple Silicon용 범용 빌드
- Windows x64 프로세서용

Windows ARM64의 경우 여기에서 다운로드하십시오.

Linux는 현재 지원되지 않습니다.

Step 2: 로그인

Applications 폴더(macOS) 또는 Start 메뉴(Windows)에서 Claude를 실행합니다. Anthropic 계정으로 로그인합니다.

Step 3: Code 탭 열기

상단 중앙의 **Code** 탭을 클릭합니다. Code를 클릭하면 업그레이드하라는 메시지가 표시되면 먼저 유료 요금제를 구독해야 합니다. 온라인으로 로그인하라는 메시지가 표시되면 로그인을 완료하고 앱을 다시 시작합니다. 403 오류가 표시되면 인증 문제 해결을 참조하십시오.

데스크톱 앱에는 Claude Code가 포함되어 있습니다. Node.js나 CLI를 별도로 설치할 필요가 없습니다. 터미널에서 `claude` 를 사용하려면 CLI를 별도로 설치하십시오. CLI 시작하기를 참조하십시오.

첫 번째 세션 시작

Code 탭이 열려 있으면 프로젝트를 선택하고 Claude에게 할 일을 지시합니다.

Step 1: 환경 및 폴더 선택

Local을 선택하여 파일을 직접 사용하여 머신에서 Claude를 실행합니다. **Select folder**를 클릭하고 프로젝트 디렉토리를 선택합니다.

> **Tip:**

> 잘 알고 있는 작은 프로젝트부터 시작하십시오. Claude Code가 할 수 있는 일을 보는 가장 빠른 방법입니다. Windows에서는 로컬 세션이 작동하려면 Git이 설치되어 있어야 합니다. 대부분의 Mac에는 기본적으로 Git이 포함되어 있습니다.

다음을 선택할 수도 있습니다:

- **Remote**: Anthropic의 클라우드 인프라에서 세션을 실행하며, 앱을 닫아도 계속됩니다. 원격 세션은 웹의 Claude Code와 동일한 인프라를 사용합니다.
- **SSH**: SSH를 통해 원격 머신(자신의 서버, 클라우드 VM 또는 dev 컨테이너)에 연결합니다. Claude Code는 원격 머신에 설치되어야 합니다.

Step 2: 모델 선택

전송 버튼 옆의 드롭다운에서 모델을 선택합니다. Opus, Sonnet, Haiku의 비교는 모델을 참조하십시오. 세션이 시작된 후에는 모델을 변경할 수 없습니다.

Step 3: Claude에게 할 일 지시

Claude가 할 일을 입력합니다:

- `TODO 주석을 찾아서 수정하기`
- `main 함수에 대한 테스트 추가`
- `이 코드베이스에 대한 지침이 포함된 CLAUDE.md 생성`

세션은 코드에 대한 Claude와의 대화입니다. 각 세션은 자신의 컨텍스트와 변경 사항을 추적하므로 여러 작업을 동시에 수행할 수 있으며 서로 간섭하지 않습니다.

Step 4: 변경 사항 검토 및 수락

기본적으로 Code 탭은 Ask permissions 모드에서 시작되며, Claude는 변경 사항을 제안하고 적용하기 전에 승인을 기다립니다. 다음을 보게 됩니다:

1. 각 파일에서 정확히 무엇이 변경될지 보여주는 diff 보기
2. 각 변경 사항을 승인하거나 거부하는 Accept/Reject 버튼
3. Claude가 요청을 처리하면서 실시간 업데이트

변경 사항을 거부하면 Claude는 다르게 진행하고 싶은 방법을 묻습니다. 승인할 때까지 파일이 수정되지 않습니다.

이제 뭘 할까요?

첫 번째 편집을 완료했습니다. Desktop이 할 수 있는 모든 것에 대한 전체 참조는 Claude Code Desktop 사용을 참조하십시오. 다음으로 시도할 수 있는 몇 가지 사항입니다.

중단 및 조정. 언제든지 Claude를 중단할 수 있습니다. 잘못된 방향으로 가고 있다면 중지 버튼을 클릭하거나 수정 사항을 입력하고 **Enter**를 누릅니다. Claude는 작업을 중지하고 입력에 따라 조정합니다. 완료될 때까지 기다리거나 다시 시작할 필요가 없습니다.

Claude에게 더 많은 컨텍스트 제공. 프롬프트 상자에 `@filename`을 입력하여 특정 파일을 대화에 가져오거나, 첨부 버튼을 사용하여 이미지 및 PDF를 첨부하거나, 파일을 프롬프트에 직접 드래그 앤 드롭합니다. Claude가 더 많은 컨텍스트를 가질수록 결과가 더 좋습니다. 파일 및 컨텍스트 추가를 참조하십시오.

반복 가능한 작업에 skills 사용. `/`를 입력하거나 **+** → **Slash commands**를 클릭하여 기본 제공 명령, 사용자 정의 skills, 그리고 플러그인 skills를 찾아봅니다. Skills는 코드 검토 체크리스트나 배포 단계와 같이 필요할 때마다 호출할 수 있는 재사용 가능한 프롬프트입니다.

커밋하기 전에 변경 사항 검토. Claude가 파일을 편집한 후 `+12 -1` 표시기가 나타납니다. 이를 클릭하여 diff 보기를 열고, 파일별로 수정 사항을 검토하고, 특정 줄에 대해 댓글을 달 수 있습니다. Claude는 댓글을 읽고 수정합니다. **Review code**를 클릭하여 Claude가 diff를 평가하고 인라인 제안을 남기도록 합니다.

제어 수준 조정. 권한 모드는 균형을 제어합니다. Ask permissions(기본값)는 모든 편집 전에 승인이 필요합니다. Auto accept edits는 파일 편집을 자동으로 수락하여 더 빠른 반복을 가능하게 합니다. Plan mode는 Claude가 파일을 건드리지 않고 접근 방식을 계획하도록 하며, 이는 큰 리팩토링 전에 유용합니다.

더 많은 기능을 위해 플러그인 추가. 프롬프트 상자 옆의 **+** 버튼을 클릭하고 **Plugins**를 선택하여 skills, 에이전트, MCP servers 등을 추가하는 플러그인을 찾아보고 설치합니다.

앱 미리보기. Preview 드롭다운을 클릭하여 dev 서버를 데스크톱에서 직접 실행합니다. Claude는 실행 중인 앱을 보고, 엔드포인트를 테스트하고, 로그를 검사하고, 보는 것에 대해 반복할 수 있습니다. 앱 미리보기를 참조하십시오.

pull request 추적. PR을 열면 Claude Code는 CI 확인 결과를 모니터링하고 실패를 자동으로 수정하거나 모든 확인이 통과되면 PR을 자동으로 병합할 수 있습니다. pull request 상태 모니터링을 참조하십시오.

Claude를 일정에 따라 실행. 예약된 작업을 설정하여 Claude를 정기적으로 자동으로 실행합니다: 매일 아침 일일 코드 검토, 주간 종속성 감사, 또는 연결된 도구에서 정보를 가져오는 브리핑입니다.

준비가 되면 확장. 사이드바에서 병렬 세션을 열어 여러 작업을 동시에 수행하며, 각각 자신의 Git worktree에서 실행합니다. 장기 실행 작업을 클라우드로 보내 앱을 닫아도 계속되도록 하거나, 작업이 예상보다 오래 걸리면 웹이나 IDE에서 세션을 계속합니다. GitHub, Slack, Linear와 같은 외부 도구를 연결하여 워크플로우를 통합합니다.

CLI에서 오셨나요?

Desktop은 CLI와 동일한 엔진을 그래픽 인터페이스와 함께 실행합니다. 동일한 프로젝트에서 둘 다 동시에 실행할 수 있으며, 구성(CLAUDE.md 파일, MCP servers, hooks, skills, 그리고 설정)을 공유합니다. 기능, 플래그 동등물, 그리고 Desktop에서 사용할 수 없는 것의 전체 비교는 CLI 비교를 참조하십시오.

다음 단계

- Claude Code Desktop 사용: 권한 모드, 병렬 세션, diff 보기, 커넥터, 그리고 엔터프라이즈 구성
- 문제 해결: 일반적인 오류 및 설정 문제에 대한 해결책
- 모범 사례: 효과적인 프롬프트 작성 및 Claude Code 활용을 위한 팁
- 일반적인 워크플로우: 디버깅, 리팩토링, 테스트 등에 대한 튜토리얼

Claude Code 설정

> 개발 머신에 Claude Code를 설치, 인증하고 사용을 시작합니다.

시스템 요구사항

- **운영 체제**:
  - macOS 13.0+
  - Windows 10 1809+ 또는 Windows Server 2019+ (설정 참고 사항 참조)
  - Ubuntu 20.04+
  - Debian 10+
  - Alpine Linux 3.19+ (추가 종속성 필요)
- **하드웨어**: 4GB 이상 RAM
- **네트워크**: 인터넷 연결 필수 (네트워크 구성 참조)
- **Shell**: Bash 또는 Zsh에서 최적으로 작동합니다
- **위치**: Anthropic 지원 국가

추가 종속성

- **ripgrep**: 일반적으로 Claude Code에 포함됩니다. 검색이 실패하면 검색 문제 해결을 참조하십시오.
- **Node.js 18+**: 더 이상 사용되지 않는 npm 설치에만 필요합니다

설치

To install Claude Code, use one of the following methods:

Native Install (Recommended)

macOS, Linux, WSL:

```
curl -fsSL https://claude.ai/install.sh | bash
```

Windows PowerShell:

```
irm https://claude.ai/install.ps1 | iex
```

Windows CMD:

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd && del install.cmd
```

Windows requires Git for Windows. Install it first if you don't have it.

> **Info:**
>
> Native installations automatically update in the background to keep you on the latest version.

Homebrew

```
brew install --cask claude-code
```

> **Info:**
>
> Homebrew installations do not auto-update. Run `brew upgrade claude-code` periodically to get the latest features and security fixes.

WinGet

```
winget install Anthropic.ClaudeCode
```

> **Info:**
>
> WinGet installations do not auto-update. Run `winget upgrade Anthropic.ClaudeCode` periodically to get the latest features and security fixes.

설치 프로세스가 완료된 후 프로젝트로 이동하여 Claude Code를 시작합니다:

```
cd your-awesome-project
claude
```

설치 중에 문제가 발생하면 문제 해결 가이드를 참조하십시오.

> **Tip:**
>
> 설치 후 `claude doctor` 를 실행하여 설치 유형 및 버전을 확인합니다.

플랫폼별 설정

Windows: Claude Code를 기본적으로 실행하거나 (Git Bash 필요) WSL 내에서 실행합니다. WSL 1과 WSL 2 모두 지원되지만 WSL 1은 제한된 지원을 받으며 Bash 도구 sandboxing과 같은 기능을 지원하지 않습니다.

Alpine Linux 및 기타 musl/uClibc 기반 배포판:

Alpine 및 기타 musl/uClibc 기반 배포판의 기본 설치 프로그램에는 `libgcc`, `libstdc++` 및 `ripgrep` 이 필요합니다. 배포판의 패키지 관리자를 사용하여 이를 설치한 후 `USE_BUILTIN_RIPGREP=0` 을 설정합니다.

Alpine에서:

```
apk add libgcc libstdc++ ripgrep
```

인증

개인 사용자의 경우

1. **Claude Pro 또는 Max 플랜** (권장): Claude의 Pro 또는 Max 플랜을 구독하여 Claude Code와 웹의 Claude를 모두 포함하는 통합 구독을 받습니다. 한 곳에서 계정을 관리하고 Claude.ai 계정으로 로그인합니다.
2. **Claude Console**: Claude Console을 통해 연결하고 OAuth 프로세스를 완료합니다. Anthropic Console에서 활성 청구가 필요합니다. “Claude Code” 워크스페이스가 사용 추적 및 비용 관리를 위해 자동으로 생성됩니다. Claude Code 워크스페이스에 대한 API 키를 만들 수 없습니다. 이는 Claude Code 사용 전용입니다.

팀 및 조직의 경우

1. **Claude for Teams 또는 Enterprise** (권장): Claude for Teams 또는 Claude for Enterprise를 구독하여 중앙 집중식 청구, 팀 관리 및 Claude Code와 웹의 Claude에 대한 액세스를 받습니다. 팀 멤버는 Claude.ai 계정으로 로그인합니다.
2. **팀 청구를 포함한 Claude Console**: 팀 청구를 사용하여 공유 Claude Console 조직을 설정합니다. 팀 멤버를 초대하고 사용 추적을 위한 역할을 할당합니다.
3. **클라우드 제공자**: 기존 클라우드 인프라를 사용한 배포를 위해 Claude Code를 Amazon Bedrock, Google Vertex AI 또는 Microsoft Foundry를 사용하도록 구성합니다.

특정 버전 설치

기본 설치 프로그램은 특정 버전 번호 또는 릴리스 채널(`latest` 또는 `stable`)을 허용합니다. 설치 시 선택한 채널이 자동 업데이트의 기본값이 됩니다. 자세한 내용은 릴리스 채널 구성을 참조하십시오.

최신 버전을 설치하려면 (기본값):

macOS, Linux, WSL

```
curl -fsSL https://claude.ai/install.sh | bash
```

Windows PowerShell

```
irm https://claude.ai/install.ps1 | iex
```

Windows CMD

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd && del
install.cmd
```

안정 버전을 설치하려면:

macOS, Linux, WSL

```
curl -fsSL https://claude.ai/install.sh | bash -s stable
```

Windows PowerShell

```
& ([scriptblock]::Create((irm https://claude.ai/install.ps1))) stable
```

Windows CMD

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd stable &&
del install.cmd
```

특정 버전 번호를 설치하려면:

macOS, Linux, WSL

```
curl -fsSL https://claude.ai/install.sh | bash -s 1.0.58
```

Windows PowerShell

```
& ([scriptblock]::Create((irm https://claude.ai/install.ps1))) 1.0.58
```

Windows CMD

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd 1.0.58 &&
del install.cmd
```

바이너리 무결성 및 코드 서명

- 모든 플랫폼의 SHA256 체크섬은 릴리스 매니페스트에 게시되며, 현재 `https://storage.googleapis.com/claude-code-dist-86c565f3-f756-42ad-8dfa-d59b1c096819/claude-code-releases/{VERSION}/manifest.json` 에 위치합니다 (예: `{VERSION}` 을 `2.0.30` 으로 바꿈).
- 서명된 바이너리는 다음 플랫폼에 배포됩니다:
  - macOS: “Anthropic PBC”에서 서명하고 Apple에서 공증
  - Windows: “Anthropic, PBC”에서 서명

NPM 설치 (더 이상 사용되지 않음)

NPM 설치는 더 이상 사용되지 않습니다. 가능하면 기본 설치 방법을 사용하십시오. 기존 npm 설치를 기본으로 마이그레이션하려면 `claude install` 을 실행합니다.

전역 npm 설치

```
npm install -g @anthropic-ai/claude-code
```

> **Warning:**
>
> `sudo npm install -g` 를 사용하지 마십시오. 이는 권한 문제 및 보안 위험으로 이어질 수 있습니다. 권한 오류가 발생하면 권장 솔루션에 대해 권한 오류 문제 해결을 참조하십시오.

Windows 설정

옵션 1: WSL 내 Claude Code

- WSL 1과 WSL 2 모두 지원됩니다
- WSL 2는 향상된 보안을 위해 sandboxing을 지원합니다. WSL 1은 sandboxing을 지원하지 않습니다.

옵션 2: Git Bash를 사용한 기본 Windows의 Claude Code

- Git for Windows 필요
- 휴대용 Git 설치의 경우 `bash.exe` 의 경로를 지정합니다:

```
$env:CLAUDE_CODE_GIT_BASH_PATH="C:\Program Files\Git\bin\bash.exe"
```

Claude Code 업데이트

자동 업데이트

Claude Code는 최신 기능과 보안 수정 사항을 보유하도록 자동으로 최신 상태를 유지합니다.

- **업데이트 확인**: 시작 시 및 실행 중 주기적으로 수행됨
- **업데이트 프로세스**: 백그라운드에서 자동으로 다운로드 및 설치
- **알림**: 업데이트가 설치되면 알림이 표시됨
- **업데이트 적용**: 업데이트는 다음 번 Claude Code를 시작할 때 적용됨

> **Note:**
>
> Homebrew 및 WinGet 설치는 자동 업데이트되지 않습니다. `brew upgrade claude-code` 또는 `winget upgrade Anthropic.ClaudeCode` 를 사용하여 수동으로 업데이트합니다.

> **알려진 문제:** Claude Code는 새 버전이 이러한 패키지 관리자에서 사용 가능하기 전에 업데이트를 알릴 수 있습니다. 업그레이드가 실패하면 잠시 기다렸다가 나중에 다시 시도하십시오.

릴리스 채널 구성

자동 업데이트 및 `claude update`에 대해 Claude Code가 따르는 릴리스 채널을 `autoUpdatesChannel` 설정으로 구성합니다:

- `"latest"` (기본값): 새 기능이 릴리스되는 즉시 받습니다
- `"stable"`: 일반적으로 약 1주일 된 버전을 사용하여 주요 회귀가 있는 릴리스를 건너뜁니다

`/config` → **자동 업데이트 채널**을 통해 또는 settings.json 파일에 추가하여 구성합니다:

```
{
  "autoUpdatesChannel": "stable"
}
```

엔터프라이즈 배포의 경우 관리되는 설정을 사용하여 조직 전체에서 일관된 릴리스 채널을 적용할 수 있습니다.

자동 업데이트 비활성화

셸 또는 settings.json 파일에서 `DISABLE_AUTOUPDATER` 환경 변수를 설정합니다:

```
export DISABLE_AUTOUPDATER=1
```

수동으로 업데이트

```
claude update
```

Claude Code 제거

Claude Code를 제거해야 하는 경우 설치 방법에 따른 지침을 따릅니다.

기본 설치

Claude Code 바이너리 및 버전 파일을 제거합니다:

macOS, Linux, WSL:

```
rm -f ~/.local/bin/claude
rm -rf ~/.local/share/claude
```

Windows PowerShell:

```
Remove-Item -Path "$env:USERPROFILE\.local\bin\claude.exe" -Force
Remove-Item -Path "$env:USERPROFILE\.local\share\claude" -Recurse -Force
```

Windows CMD:

```
del "%USERPROFILE%\.local\bin\claude.exe"
rmdir /s /q "%USERPROFILE%\.local\share\claude"
```

Homebrew 설치

```
brew uninstall --cask claude-code
```

WinGet 설치

```
winget uninstall Anthropic.ClaudeCode
```

NPM 설치

```
npm uninstall -g @anthropic-ai/claude-code
```

구성 파일 정리 (선택 사항)

> **Warning:**
> 구성 파일을 제거하면 모든 설정, 허용된 도구, MCP 서버 구성 및 세션 기록이 삭제됩니다.

Claude Code 설정 및 캐시된 데이터를 제거하려면:

macOS, Linux, WSL:

```
## 사용자 설정 및 상태 제거
rm -rf ~/.claude
rm ~/.claude.json

## 프로젝트별 설정 제거 (프로젝트 디렉토리에서 실행)
rm -rf .claude
rm -f .mcp.json
```

Windows PowerShell:

```
## 사용자 설정 및 상태 제거
Remove-Item -Path "$env:USERPROFILE\.claude" -Recurse -Force
Remove-Item -Path "$env:USERPROFILE\.claude.json" -Force

## 프로젝트별 설정 제거 (프로젝트 디렉토리에서 실행)
Remove-Item -Path ".claude" -Recurse -Force
Remove-Item -Path ".mcp.json" -Force
```

Windows CMD:

```
REM 사용자 설정 및 상태 제거
rmdir /s /q "%USERPROFILE%\.claude"
del "%USERPROFILE%\.claude.json"

REM 프로젝트별 설정 제거 (프로젝트 디렉토리에서 실행)
rmdir /s /q ".claude"
del ".mcp.json"
```

Claude Code의 작동 방식

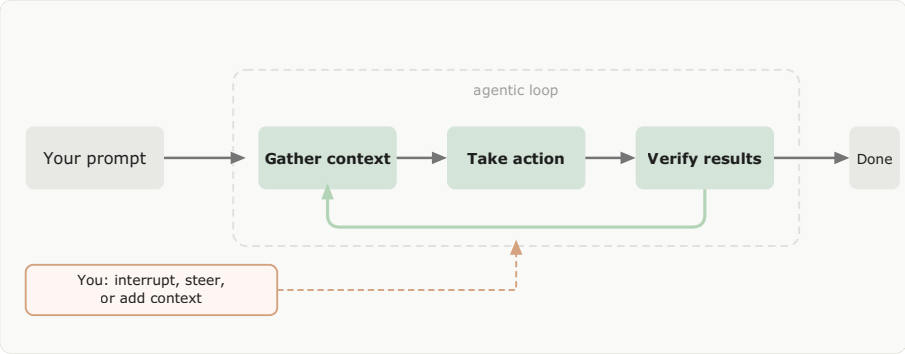
> 에이전트 루프, 내장 도구, Claude Code가 프로젝트와 상호작용하는 방식을 이해합니다.

Claude Code는 터미널에서 실행되는 에이전트 어시스턴트입니다. 코딩에 탁월하지만 명령줄에서 할 수 있는 모든 작업을 도와줄 수 있습니다: 문서 작성, 빌드 실행, 파일 검색, 주제 조사 등.

이 가이드는 핵심 아키텍처, 내장 기능, 그리고 Claude Code를 효과적으로 사용하기 위한 팁을 다룹니다. 단계별 설명서는 일반적인 워크플로우를 참조하세요. skills, MCP, hooks와 같은 확장 기능은 Claude Code 확장을 참조하세요.

에이전트 루프

Claude에게 작업을 주면 세 가지 단계를 거칩니다: **컨텍스트 수집**, **작업 수행**, **결과 검증**. 이 단계들은 함께 진행됩니다. Claude는 코드를 이해하기 위해 파일을 검색하든, 변경을 위해 편집하든, 작업을 확인하기 위해 테스트를 실행하든 전반적으로 도구를 사용합니다.



에이전트 루프: 프롬프트가 Claude가 컨텍스트를 수집하고, 작업을 수행하고, 결과를 검증하고, 작업이 완료될 때까지 반복하도록 합니다. 언제든지 중단할 수 있습니다.

루프는 사용자가 요청한 내용에 맞게 조정됩니다. 코드베이스에 대한 질문은 컨텍스트 수집만 필요할 수 있습니다. 버그 수정은 세 단계를 반복적으로 거칩니다. 리팩토링은 광범위한 검증을 포함할 수 있습니다. Claude는 이전 단계에서 배운 내용을 바탕으로 각 단계에서 필요한 것을 결정하고, 수십 개의 작업을 연결하며 그 과정에서 방향을 수정합니다.

사용자도 이 루프의 일부입니다. 언제든지 중단하여 Claude를 다른 방향으로 유도하거나, 추가 컨텍스트를 제공하거나, 다른 접근 방식을 시도하도록 요청할 수 있습니다. Claude는 자율적으로 작동하지만 사용자의 입력에 반응합니다.

에이전트 루프는 두 가지 구성 요소로 구동됩니다: 추론하는 모델과 작용하는 도구. Claude Code는 Claude 주변의 **에이전트 하네스** 역할을 합니다: 언어 모델을 능력 있는 코딩 에이전트로 변환하는 도구, 컨텍스트 관리, 실행 환경을 제공합니다.

모델

Claude Code는 Claude 모델을 사용하여 코드를 이해하고 작업에 대해 추론합니다. Claude는 모든 언어의 코드를 읽을 수 있고, 구성 요소가 어떻게 연결되는지 이해하며, 목표를 달성하기 위해 무엇을 변경해야 하는지 파악할 수 있습니다. 복잡한 작업의 경우 작업을 단계로 나누고, 실행하고, 배운 내용을 바탕으로 조정합니다.

여러 모델을 사용할 수 있으며 각각 다른 장단점이 있습니다. Sonnet은 대부분의 코딩 작업을 잘 처리합니다. Opus는 복잡한 아키텍처 결정을 위한 더 강력한 추론을 제공합니다. 세션 중에 `/model`로 전환하거나 `claude --model <name>`으로 시작하세요.

이 가이드에서 "Claude가 선택한다" 또는 "Claude가 결정한다"고 할 때, 모델이 추론을 수행하는 것입니다.

도구

도구는 Claude Code를 에이전트로 만드는 것입니다. 도구가 없으면 Claude는 텍스트로만 응답할 수 있습니다. 도구가 있으면 Claude는 작용할 수 있습니다: 코드를 읽고, 파일을 편집하고, 명령을 실행하고, 웹을 검색하고, 외부 서비스와 상호작용합니다. 각 도구 사용은 루프에 다시 피드백되는 정보를 반환하여 Claude의 다음 결정을 알립니다.

내장 도구는 일반적으로 다섯 가지 범주로 나뉘며, 각각은 다른 종류의 에이전시를 나타냅니다.

범주	Claude가 할 수 있는 것
파일 작업	파일 읽기, 코드 편집, 새 파일 생성, 이름 변경 및 재구성
검색	패턴으로 파일 찾기, 정규식으로 콘텐츠 검색, 코드베이스 탐색
실행	셸 명령 실행, 서버 시작, 테스트 실행, git 사용
웹	웹 검색, 문서 가져오기, 오류 메시지 조회
코드 인텔리전스	편집 후 타입 오류 및 경고 확인, 정의로 이동, 참조 찾기(코드 인텔리전스 플러그인 필요)

이것이 주요 기능입니다. Claude는 또한 subagents를 생성하고, 질문을 하고, 다른 오케스트레이션 작업을 위한 도구를 가지고 있습니다. 전체 목록은 Claude가 사용할 수 있는 도구를 참조하세요.

Claude는 프롬프트와 그 과정에서 배운 내용을 바탕으로 사용할 도구를 선택합니다. "실패한 테스트를 수정해"라고 말하면 Claude는 다음을 수행할 수 있습니다:

1. 테스트 스위트를 실행하여 무엇이 실패하는지 확인
2. 오류 출력 읽기
3. 관련 소스 파일 검색
4. 해당 파일을 읽어 코드 이해
5. 파일을 편집하여 문제 수정
6. 테스트를 다시 실행하여 검증

각 도구 사용은 Claude에게 다음 단계를 알리는 새로운 정보를 제공합니다. 이것이 에이전트 루프의 작동입니다.

기본 기능 확장: 내장 도구는 기초입니다. skills로 Claude가 알 수 있는 것을 확장하고, MCP로 외부 서비스에 연결하고, hooks로 워크플로우를 자동화하고, subagents로 작업을 위임할 수 있습니다. 이러한 확장은 핵심 에이전트 루프 위에 계층을 형성합니다. 필요에 맞는 확장을 선택하는 방법은 Claude Code 확장을 참조하세요.

Claude가 접근할 수 있는 것

이 가이드는 터미널에 중점을 둡니다. Claude Code는 또한 VS Code, JetBrains IDE, 및 기타 환경에서 실행됩니다.

디렉토리에서 `claude` 를 실행하면 Claude Code는 다음에 접근할 수 있습니다:

- **프로젝트.** 디렉토리 및 하위 디렉토리의 파일, 그리고 허가를 받은 다른 곳의 파일.
- **터미널.** 실행할 수 있는 모든 명령: 빌드 도구, git, 패키지 관리자, 시스템 유틸리티, 스크립트. 명령줄에서 할 수 있는 것이면 Claude도 할 수 있습니다.
- **git 상태.** 현재 브랜치, 커밋되지 않은 변경 사항, 최근 커밋 기록.
- **CLAUDE.md.** 프로젝트별 지침, 규칙, Claude가 매 세션마다 알아야 할 컨텍스트를 저장하는 마크다운 파일.
- **자동 메모리.** Claude가 작업하면서 자동으로 저장하는 학습 내용(프로젝트 패턴 및 사용자 선호도 등). MEMORY.md의 처음 200줄이 각 세션 시작 시 로드됩니다.
- **구성한 확장.** 외부 서비스를 위한 MCP servers, 워크플로우를 위한 skills, 위임된 작업을 위한 subagents, 브라우저 상호작용을 위한 Claude in Chrome.

Claude가 전체 프로젝트를 보기 때문에 전체 프로젝트에서 작업할 수 있습니다. "인증 버그를 수정해"라고 Claude에게 요청하면 관련 파일을 검색하고, 컨텍스트를 이해하기 위해 여러 파일을 읽고, 여러 파일에 걸쳐 조정된 편집을 수행하고, 수정을 검증하기 위해 테스트를 실행하고, 요청하면 변경 사항을 커밋합니다. 이는 현재 파일만 보는 인라인 코드 어시스턴트와 다릅니다.

환경 및 인터페이스

위에서 설명한 에이전트 루프, 도구, 기능은 Claude Code를 사용하는 모든 곳에서 동일합니다. 변하는 것은 코드가 실행되는 위치와 상호작용하는 방식입니다.

실행 환경

Claude Code는 세 가지 환경에서 실행되며, 각각은 코드가 실행되는 위치에 대해 다른 장단점이 있습니다.

환경	코드 실행 위치	사용 사례
로컬	사용자 머신	기본값. 파일, 도구, 환경에 대한 전체 접근
클라우드	Anthropic 관리 VM	작업 오프로드, 로컬에 없는 리포지토리에서 작업
원격 제어	사용자 머신, 브라우저에서 제어	웹 UI를 사용하면서 모든 것을 로컬로 유지

인터페이스

터미널, 데스크톱 앱, IDE 확장, claude.ai/code, 원격 제어, Slack, CI/CD 파이프라인을 통해 Claude Code에 접근할 수 있습니다. 인터페이스는 Claude를 보고 상호작용하는 방식을 결정하지만, 기본 에이전트 루프는 동일합니다. 전체 목록은 Claude Code를 어디서나 사용을 참조하세요.

세션으로 작업

Claude Code는 작업하면서 대화를 로컬에 저장합니다. 각 메시지, 도구 사용, 결과가 저장되어 되돌리기, 재개 및 포크 세션을 활성화합니다. Claude가 코드를 변경하기 전에 영향을 받는 파일의 스냅샷을 만들어 필요하면 되돌릴 수 있습니다.

세션은 독립적입니다. 각 새 세션은 이전 세션의 대화 기록 없이 새로운 컨텍스트 윈도우로 시작합니다. Claude는 자동 메모리를 사용하여 세션 간에 학습을 유지할 수 있으며, CLAUDE.md에 자신의 지속적인 지침을 추가할 수 있습니다.

브랜치 간 작업

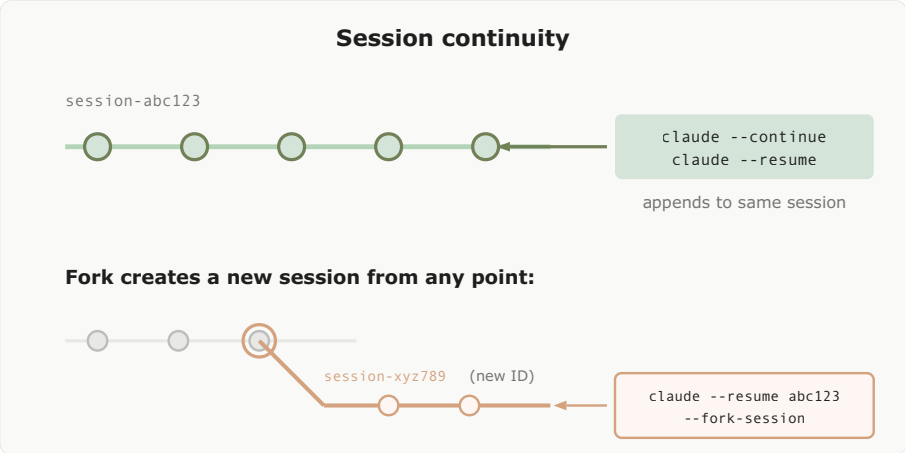
각 Claude Code 대화는 현재 디렉토리에 연결된 세션입니다. 재개할 때 해당 디렉토리의 세션만 표시됩니다.

Claude는 현재 브랜치의 파일을 봅니다. 브랜치를 전환하면 Claude는 새 브랜치의 파일을 보지만 대화 기록은 동일하게 유지됩니다. Claude는 전환 후에도 논의한 내용을 기억합니다.

세션이 디렉토리에 연결되어 있으므로 git worktrees를 사용하여 개별 브랜치에 대한 별도 디렉토리를 만들어 병렬 Claude 세션을 실행할 수 있습니다.

세션 재개 또는 포크

`claude --continue` 또는 `claude --resume`으로 세션을 재개하면 동일한 세션 ID를 사용하여 중단한 지점부터 시작합니다. 새 메시지는 기존 대화에 추가됩니다. 전체 대화 기록이 복원되지만 세션 범위 권한은 복원되지 않습니다. 다시 승인해야 합니다.



세션 연속성: 재개는 동일한 세션을 계속하고, 포크는 새 ID로 새 브랜치를 만듭니다.

원본 세션에 영향을 주지 않고 다른 접근 방식을 시도하려면 `--fork-session` 플래그를 사용하세요:

```
claude --continue --fork-session
```

이는 그 시점까지의 대화 기록을 유지하면서 새 세션 ID를 만듭니다. 원본 세션은 변경되지 않습니다. 재개와 마찬가지로 포크된 세션은 세션 범위 권한을 상속하지 않습니다.

여러 터미널에서 동일한 세션: 여러 터미널에서 동일한 세션을 재개하면 두 터미널 모두 동일한 세션 파일에 쓰기를 수행합니다. 두 터미널의 메시지가 같은 노트북에 두 사람이 쓰는 것처럼 인터리브됩니다. 아무것도 손상되지 않지만 대화가 뒤섞입니다. 각 터미널은 세션 중에 자신의 메시지만 보지만, 나중에 해당 세션을 재개하면 모든 것이 인터리브된 것을 볼 수 있습니다. 동일한 시작점에서 병렬 작업을 하려면 `--fork-session`을 사용하여 각 터미널에 자신의 깨끗한 세션을 제공하세요.

컨텍스트 윈도우

Claude의 컨텍스트 윈도우는 대화 기록, 파일 콘텐츠, 명령 출력, CLAUDE.md, 로드된 skills, 시스템 지침을 보유합니다. 작업하면서 컨텍스트가 채워집니다. Claude는 자동으로 압축하지만 대화 초반의 지침이 손실될 수 있습니다. 지속적인 규칙을 CLAUDE.md에 넣고 `/context`를 실행하여 공간을 사용하는 것을 확인하세요.

컨텍스트가 가득 찰 때

Claude Code는 한계에 접근할 때 컨텍스트를 자동으로 관리합니다. 먼저 오래된 도구 출력을 지우고, 필요하면 대화를 요약합니다. 요청과 주요 코드 스니펫은 유지되지만 대화 초반의 자세한 지침이 손실될 수 있습니다. 대화 기록에 의존하기보다는 지속적인 규칙을 CLAUDE.md에 넣으세요.

압축 중에 보존되는 것을 제어하려면 CLAUDE.md에 "Compact Instructions" 섹션을 추가하거나 `/compact`를 포커스와 함께 실행하세요(예: `/compact focus on the API changes`).

`/context`를 실행하여 공간을 사용하는 것을 확인하세요. MCP servers는 모든 요청에 도구 정의를 추가하므로 몇 개의 서버가 작업을 시작하기 전에 상당한 컨텍스트를 소비할 수 있습니다. `/mcp`를 실행하여 서버별 비용을 확인하세요.

skills 및 subagents로 컨텍스트 관리

압축 외에도 다른 기능을 사용하여 컨텍스트에 로드되는 것을 제어할 수 있습니다.

Skills는 요청 시 로드됩니다. Claude는 세션 시작 시 skill 설명을 보지만 전체 콘텐츠는 skill이 사용될 때만 로드됩니다. 수동으로 호출하는 skills의 경우 `disable-model-invocation: true`를 설정하여 필요할 때까지 설명을 컨텍스트 밖으로 유지하세요.

Subagents는 주 대화와 완전히 분리된 자신의 새로운 컨텍스트를 얻습니다. 그들의 작업은 컨텍스트를 부풀리지 않습니다. 완료되면 요약을 반환합니다. 이 격리가 긴 세션에서 subagents를 도움이 되는 이유입니다.

각 기능의 비용은 컨텍스트 비용을 참조하고, 컨텍스트 관리 팁은 토큰 사용 감소를 참조하세요.

체크포인트 및 권한으로 안전 유지

Claude는 두 가지 안전 메커니즘을 가지고 있습니다: 체크포인트는 파일 변경을 취소할 수 있게 하고, 권한은 Claude가 요청 없이 할 수 있는 것을 제어합니다.

체크포인트로 변경 취소

모든 파일 편집은 되돌릴 수 있습니다. Claude가 파일을 편집하기 전에 현재 콘텐츠의 스냅샷을 만듭니다. 문제가 발생하면 `Esc`를 두 번 눌러 이전 상태로 되돌리거나 Claude에게 취소하도록 요청하세요.

체크포인트는 세션에 로컬이며 git과 분리되어 있습니다. 파일 변경만 다룹니다. 원격 시스템(데이터베이스, API, 배포)에 영향을 주는 작업은 체크포인트할 수 없으므로 Claude는 외부 부작용이 있는 명령을 실행하기 전에 요청합니다.

Claude가 할 수 있는 것 제어

`Shift+Tab`을 눌러 권한 모드를 순환하세요:

- **기본값**: Claude는 파일 편집 및 셸 명령 전에 요청
- **자동 수락 편집**: Claude는 파일을 편집하지만 명령은 여전히 요청
- **계획 모드**: Claude는 읽기 전용 도구만 사용하여 실행 전에 승인할 수 있는 계획을 만듭니다

`.claude/settings.json`에서 특정 명령을 허용하여 Claude가 매번 요청하지 않도록 할 수 있습니다. 이는 `npm test` 또는 `git status`와 같은 신뢰할 수 있는 명령에 유용합니다. 설정은 조직 전체 정책에서 개인 선호도까지 범위를 지정할 수 있습니다. 자세한 내용은 권한을 참조하세요.

Claude Code를 효과적으로 사용

이 팁은 Claude Code에서 더 나은 결과를 얻는 데 도움이 됩니다.

Claude Code에 도움을 요청

Claude Code는 사용 방법을 가르칠 수 있습니다. “hooks를 설정하려면 어떻게 하나요?” 또는 “CLAUDE.md를 구조화하는 최선의 방법은 무엇인가요?” 와 같은 질문을 하면 Claude가 설명합니다.

내장 명령도 설정을 안내합니다:

- `/init`은 프로젝트를 위한 CLAUDE.md 생성을 안내합니다
- `/agents`는 사용자 정의 subagents 구성을 도와줍니다
- `/doctor`는 설치의 일반적인 문제를 진단합니다

대화입니다

Claude Code는 대화형입니다. 완벽한 프롬프트가 필요하지 않습니다. 원하는 것으로 시작한 다음 개선하세요:

```
로그인 버그 수정
```

[Claude가 조사하고 시도]

```
정확하지 않습니다. 문제는 세션 처리에 있습니다.
```

[Claude가 접근 방식 조정]

첫 번째 시도가 맞지 않으면 다시 시작할 필요가 없습니다. 반복합니다.

중단 및 조종

언제든지 Claude를 중단할 수 있습니다. 잘못된 경로로 가고 있으면 수정 사항을 입력하고 Enter를 누르세요. Claude는 작업을 중지하고 입력을 바탕으로 접근 방식을 조정합니다. 완료될 때까지 기다리거나 다시 시작할 필요가 없습니다.

처음부터 구체적으로

초기 프롬프트가 정확할수록 필요한 수정이 적습니다. 특정 파일을 참조하고, 제약 조건을 언급하고, 예제 패턴을 지적하세요.

```
체크아웃 흐름이 만료된 카드를 가진 사용자에게 손상되었습니다.
문제를 찾기 위해 src/payments/를 확인하세요. 특히 토큰 새로고침.
먼저 실패하는 테스트를 작성한 다음 수정하세요.
```

모호한 프롬프트는 작동하지만 더 많은 시간을 조종하는 데 소비합니다. 위와 같은 구체적인 프롬프트는 종종 첫 번째 시도에서 성공합니다.

Claude가 검증할 수 있는 것을 제공

Claude는 자신의 작업을 확인할 수 있을 때 더 잘 수행합니다. 테스트 케이스를 포함하고, 예상 UI의 스크린샷을 붙여넣거나, 원하는 출력을 정의하세요.

```
validateEmail을 구현하세요. 테스트 케이스: 'user@example.com' → true,
'invalid' → false, 'user@.com' → false. 후에 테스트를 실행하세요.
```

시각적 작업의 경우 디자인의 스크린샷을 붙여넣고 Claude에게 구현을 비교하도록 요청하세요.

구현 전에 탐색

복잡한 문제의 경우 연구와 코딩을 분리하세요. 계획 모드(`Shift+Tab` 두 번)를 사용하여 먼저 코드베이스를 분석하세요:

```
src/auth/를 읽고 세션 처리 방식을 이해하세요.
그런 다음 OAuth 지원 추가를 위한 계획을 만드세요.
```

계획을 검토하고 대화를 통해 개선한 다음 Claude가 구현하도록 하세요. 이 2단계 접근 방식은 코드로 바로 뛰어드는 것보다 더 나은 결과를 생성합니다.

지시하지 말고 위임

능력 있는 동료에게 위임하는 것처럼 생각하세요. 컨텍스트와 방향을 제공한 다음 Claude가 세부 사항을 파악하도록 신뢰하세요:

```
체크아웃 흐름이 만료된 카드를 가진 사용자에게 손상되었습니다.
관련 코드는 src/payments/에 있습니다. 조사하고 수정할 수 있나요?
```

읽을 파일이나 실행할 명령을 지정할 필요가 없습니다. Claude가 파악합니다.

다음 단계

- 기능으로 확장 Skills, MCP 연결, 사용자 정의 명령 추가
- 일반적인 워크플로우 일반적인 작업을 위한 단계별 가이드

Part 2: Core Usage

Claude Code 확장하기

> CLAUDE.md, Skills, subagents, hooks, MCP, 플러그인을 언제 사용할지 이해합니다.

Claude Code는 코드를 추론하는 모델과 파일 작업, 검색, 실행 및 웹 접근을 위한 내장 도구를 결합합니다. 내장 도구는 대부분의 코딩 작업을 다룹니다. 이 가이드는 확장 계층을 다룹니다. Claude가 알아야 할 내용을 사용자 정의하고, 외부 서비스에 연결하고, 워크플로우를 자동화하기 위해 추가하는 기능입니다.

> **Note:**
>
> 핵심 에이전트 루프가 어떻게 작동하는지 알아보려면 Claude Code 작동 방식을 참조하세요.

Claude Code를 처음 사용하시나요? 프로젝트 규칙을 위해 CLAUDE.md로 시작하세요. 필요에 따라 다른 확장을 추가하세요.

개요

확장은 에이전트 루프의 다양한 부분에 연결됩니다.

- **CLAUDE.md**는 Claude가 모든 세션에서 보는 지속적인 컨텍스트를 추가합니다.
- **Skills**는 재사용 가능한 지식과 호출 가능한 워크플로우를 추가합니다.
- **MCP**는 Claude를 외부 서비스 및 도구에 연결합니다.
- **Subagents**는 격리된 컨텍스트에서 자신의 루프를 실행하고 요약을 반환합니다.
- **Agent teams**는 공유 작업 및 피어 투 피어 메시징으로 여러 독립적인 세션을 조정합니다.
- **Hooks**는 결정론적 스크립트로 루프 외부에서 완전히 실행됩니다.
- **Plugins** 및 **marketplaces**는 이러한 기능을 패키징하고 배포합니다.

Skills는 가장 유연한 확장입니다. Skill은 지식, 워크플로우 또는 지침을 포함하는 마크다운 파일입니다. `/deploy` 와 같은 명령으로 skill을 호출하거나, Claude가 관련이 있을 때 자동으로 로드할 수 있습니다. Skill은 현재 대화에서 실행되거나 subagents를 통해 격리된 컨텍스트에서 실행될 수 있습니다.

기능을 목표에 맞추기

기능은 Claude가 모든 세션에서 보는 항상 켜진 컨텍스트부터 사용자나 Claude가 호출할 수 있는 온디맨드 기능, 특정 이벤트에서 실행되는 백그라운드 자동화까지 다양합니다. 아래 표는 사용 가능한 기능과 각 기능이 언제 적절한지 보여줍니다.

기능	수행 작업	사용 시기	예시
CLAUDE.md	모든 대화에서 로드되는 지속적인 컨텍스트	프로젝트 규칙, “항상 X를 수행” 규칙	“npm이 아닌 pnpm을 사용하세요. 커밋하기 전에 테스트를 실행하세요.”
Skill	Claude가 사용할 수 있는 지침, 지식 및 워크플로우	재사용 가능한 콘텐츠, 참조 문서, 반복 가능한 작업	`/deploy`는 배포 체크리스트를 실행합니다. 엔드포인트 패턴이 있는 API 문서 skill
Subagent	요약된 결과를 반환하는 격리된 실행 컨텍스트	컨텍스트 격리, 병렬 작업, 특화된 워커	많은 파일을 읽지만 주요 결과만 반환하는 연구 작업
Agent teams	여러 독립적인 Claude Code 세션 조정	병렬 연구, 새로운 기능 개발, 경쟁하는 가설로 디버깅	보안, 성능 및 테스트를 동시에 확인하는 검토자 생성
MCP	외부 서비스에 연결	외부 데이터 또는 작업	데이터베이스 쿼리, Slack에 게시, 브라우저 제어
Hook	이벤트에서 실행되는 결정론적 스크립트	예측 가능한 자동화, LLM 없음	모든 파일 편집 후 ESLint 실행

Plugins는 패키징 계층입니다. 플러그인은 skill, hook, subagent 및 MCP 서버를 단일 설치 가능한 단위로 번들합니다. 플러그인 skill은 네임스페이스됩니다(예: `/my-plugin:review`). 따라서 여러 플러그인이 공존할 수 있습니다. 여러 저장소에서 동일한 설정을 재사용하거나 **marketplace**를 통해 다른 사용자에게 배포하려는 경우 플러그인을 사용하세요.

유사한 기능 비교

일부 기능은 유사해 보일 수 있습니다. 구별하는 방법은 다음과 같습니다.

Skill vs Subagent

Skill과 subagent는 다양한 문제를 해결합니다.

- **Skills**는 모든 컨텍스트에 로드할 수 있는 재사용 가능한 콘텐츠입니다.
- **Subagents**는 주 대화와 별도로 실행되는 격리된 워커입니다.

측면	Skill	Subagent
정의	재사용 가능한 지침, 지식 또는 워크플로우	자신의 컨텍스트를 가진 격리된 워커
주요 이점	컨텍스트 간 콘텐츠 공유	컨텍스트 격리. 작업은 별도로 발생하고 요약만 반환됩니다.
최적 용도	참조 자료, 호출 가능한 워크플로우	많은 파일을 읽는 작업, 병렬 작업, 특화된 워커

Skill은 참조 또는 작업일 수 있습니다. 참조 skill은 Claude가 세션 전체에서 사용하는 지식을 제공합니다(API 스타일 가이드처럼). 작업 skill은 Claude에게 특정 작업을 수행하도록 지시합니다(배포 워크플로우를 실행하는 `/deploy` 처럼).

컨텍스트 격리가 필요하거나 컨텍스트 윈도우가 가득 찰 때 subagent를 사용하세요. Subagent는 수십 개의 파일을 읽거나 광범위한 검색을 실행할 수 있지만, 주 대화는 요약만 받습니다. Subagent 작업이 주 컨텍스트를 소비하지 않으므로, 중간 작업이 표시되어야 할 필요가 없을 때도 유용합니다. 사용자 정의 subagent는 자신의 지침을 가질 수 있고 skill을 미리 로드할 수 있습니다.

결합할 수 있습니다. Subagent는 특정 skill을 미리 로드할 수 있습니다(`skills:` 필드). Skill은 `context: fork` 를 사용하여 격리된 컨텍스트에서 실행될 수 있습니다. 자세한 내용은 Skills를 참조하세요.

CLAUDE.md vs Skill

둘 다 지침을 저장하지만 로드 방식과 목적이 다릅니다.

측면	CLAUDE.md	Skill
로드	모든 세션, 자동으로	온디맨드
파일 포함 가능	예, `@path` 가져오기 사용	예, `@path` 가져오기 사용
워크플로우 트리거 가능	아니요	예, `/<name>` 사용
최적 용도	“항상 X를 수행” 규칙	참조 자료, 호출 가능한 워크플로우

CLAUDE.md에 넣으세요. Claude가 항상 알아야 할 경우: 코딩 규칙, 빌드 명령, 프로젝트 구조, "X를 하지 마세요" 규칙.

Skill에 넣으세요. 참조 자료인 경우 Claude가 때때로 필요합니다(API 문서, 스타일 가이드) 또는 `/<name>` 으로 트리거하는 워크플로우입니다(배포, 검토, 릴리스).

경험 법칙: CLAUDE.md를 200줄 이하로 유지하세요. 증가하면 참조 콘텐츠를 skill로 이동하거나 `.claude/rules/` 파일로 분할하세요.

CLAUDE.md vs Rules vs Skills

세 가지 모두 지침을 저장하지만 로드 방식이 다릅니다.

측면	CLAUDE.md	`.claude/rules/`	Skill
로드	모든 세션	모든 세션, 또는 일치하는 파일이 열릴 때	온디맨드, 호출되거나 관련이 있을 때
범위	전체 프로젝트	파일 경로로 범위 지정 가능	작업별
최적 용도	핵심 규칙 및 빌드 명령	언어별 또는 디렉토리별 가이드라인	참조 자료, 반복 가능한 워크플로우

CLAUDE.md를 사용하세요. 모든 세션이 필요한 지침: 빌드 명령, 테스트 규칙, 프로젝트 아키텍처.

규칙을 사용하세요. CLAUDE.md를 집중시키기 위해. `paths` frontmatter가 있는 규칙은 Claude가 일치하는 파일로 작업할 때만 로드되어 컨텍스트를 절약합니다.

Skill을 사용하세요. Claude가 때때로만 필요한 콘텐츠, API 문서 또는 `/<name>` 으로 트리거하는 배포 체크리스트.

Subagent vs Agent team

둘 다 작업을 병렬화하지만 아키텍처가 다릅니다.

- **Subagents**는 세션 내에서 실행되고 결과를 주 컨텍스트에 보고합니다.
- **Agent teams**는 서로 통신하는 독립적인 Claude Code 세션입니다.

측면	Subagent	Agent team
컨텍스트	자신의 컨텍스트 윈도우; 결과는 호출자에게 반환됨	자신의 컨텍스트 윈도우; 완전히 독립적
통신	주 에이전트에게만 결과 보고	팀원이 서로 직접 메시지

측면	Subagent	Agent team
조정	주 에이전트가 모든 작업 관리	공유 작업 목록과 자체 조정
최적 용도	결과만 중요한 집중된 작업	논의 및 협력이 필요한 복잡한 작업
토큰 비용	낮음: 결과가 주 컨텍스트로 요약됨	높음: 각 팀원은 별도의 Claude 인스턴스

빠르고 집중된 워커가 필요할 때 subagent를 사용하세요. 질문을 연구하고, 주장을 확인하고, 파일을 검토하세요. Subagent는 작업을 수행하고 요약을 반환합니다. 주 대화는 깔끔하게 유지됩니다.

팀원이 결과를 공유하고, 서로 도전하고, 독립적으로 조정해야 할 때 agent team을 사용하세요. Agent team은 경쟁하는 가설이 있는 연구, 병렬 코드 검토, 각 팀원이 별도의 부분을 소유하는 새로운 기능 개발에 최적입니다.

전환점: 병렬 subagent를 실행하지만 컨텍스트 제한에 도달하거나, subagent가 서로 통신해야 할 경우, agent team이 자연스러운 다음 단계입니다.

> **Note:**
>
> Agent team은 실험적이며 기본적으로 비활성화됩니다. 설정 및 현재 제한 사항은 agent teams를 참조하세요.

MCP vs Skill

MCP는 Claude를 외부 서비스에 연결합니다. Skill은 Claude가 알아야 할 내용을 확장하며, 이러한 서비스를 효과적으로 사용하는 방법도 포함합니다.

측면	MCP	Skill
정의	외부 서비스 연결 프로토콜	지식, 워크플로우 및 참조 자료
제공	도구 및 데이터 접근	지식, 워크플로우, 참조 자료
예시	Slack 통합, 데이터베이스 쿼리, 브라우저 제어	코드 검토 체크리스트, 배포 워크플로우, API 스타일 가이드

이들은 다양한 문제를 해결하며 함께 잘 작동합니다.

MCP는 Claude에게 외부 시스템과 상호 작용할 수 있는 능력을 제공합니다. MCP 없이는 Claude가 데이터베이스를 쿼리하거나 Slack에 게시할 수 없습니다.

Skill은 Claude에게 이러한 도구를 효과적으로 사용하는 방법에 대한 지식을 제공하며, `/<name>`으로 트리거할 수 있는 워크플로우도 포함합니다. Skill에는 팀의 데이터베이스 스키마 및 쿼리 패턴, 또는 팀의 메시지 형식 규칙이 있는 `/post-to-slack` 워크플로우가 포함될 수 있습니다.

예: MCP 서버는 Claude를 데이터베이스에 연결합니다. Skill은 Claude에게 데이터 모델, 일반적인 쿼리 패턴, 다양한 작업에 사용할 테이블을 가르칩니다.

기능이 어떻게 계층화되는지 이해하기

기능은 여러 수준에서 정의될 수 있습니다. 사용자 전체, 프로젝트별, 플러그인을 통해, 또는 관리 정책을 통해. 또한 CLAUDE.md 파일을 하위 디렉토리에 중첩하거나 monorepo의 특정 패키지에 skill을 배치할 수 있습니다. 동일한 기능이 여러 수준에 존재할 때, 계층화 방식은 다음과 같습니다.

- **CLAUDE.md 파일**은 추가적입니다. 모든 수준이 동시에 Claude의 컨텍스트에 콘텐츠를 제공합니다. 작업 디렉토리 및 위의 파일은 시작 시 로드되고, 하위 디렉토리는 작업할 때 로드됩니다. 지침이 충돌할 때, Claude는 판단을 사용하여 조정하며, 더 구체적인 지침이 일반적으로 우선합니다. CLAUDE.md 파일이 로드되는 방식을 참조하세요.
- **Skill과 subagent**는 이름으로 재정의됩니다. 동일한 이름이 여러 수준에 존재할 때, 우선순위에 따라 하나의 정의가 승리합니다(skill의 경우 관리 > 사용자 > 프로젝트; subagent의 경우 관리 > CLI 플래그 > 프로젝트 > 사용자 > 플러그인). 플러그인 skill은 네임스페이스됩니다. 충돌을 피하기 위해. Skill 검색 및 subagent 범위를 참조하세요.
- **MCP 서버**는 이름으로 재정의됩니다. 로컬 > 프로젝트 > 사용자. MCP 범위를 참조하세요.
- **Hooks**는 병합됩니다. 모든 등록된 hook은 소스에 관계없이 일치하는 이벤트에 대해 실행됩니다. Hooks를 참조하세요.

기능 결합하기

각 확장은 다양한 문제를 해결합니다. CLAUDE.md는 항상 켜진 컨텍스트를 처리하고, skill은 온디맨드 지식과 워크플로우를 처리하고, MCP는 외부 연결을 처리하고, subagent는 격리를 처리하고, hook은 자동화를 처리합니다. 실제 설정은 워크플로우에 따라 이들을 결합합니다.

예를 들어, CLAUDE.md를 프로젝트 규칙에 사용하고, skill을 배포 워크플로우에 사용하고, MCP를 데이터베이스에 연결하고, hook을 모든 편집 후 린팅을 실행하는 데 사용할 수 있습니다. 각 기능은 최적의 작업을 처리합니다.

패턴	작동 방식	예시
Skill + MCP	MCP는 연결을 제공하고, skill은 Claude에게 잘 사용하는 방법을 가르칩니다.	MCP는 데이터베이스에 연결하고, skill은 스키마 및 쿼리 패턴을 문서화합니다.
Skill + Subagent	Skill은 병렬 작업을 위해 subagent를 생성합니다.	`/audit` skill은 보안, 성능 및 스타일 subagent를 시작하여 격리된 컨텍스트에서 작동합니다.
CLAUDE.md + Skill	CLAUDE.md는 항상 켜진 규칙을 보유하고, skill은 온디맨드로 로드되는 참조 자료를 보유합니다.	CLAUDE.md는 "API 규칙을 따르세요"라고 말하고, skill은 전체 API 스타일 가이드를 포함합니다.
Hook + MCP	Hook은 MCP를 통해 외부 작업을 트리거합니다.	편집 후 hook은 Claude가 중요한 파일을 수정할 때 Slack 알림을 보냅니다.

컨텍스트 비용 이해하기

추가하는 모든 기능은 Claude의 컨텍스트를 소비합니다. 너무 많으면 컨텍스트 윈도우를 채울 수 있지만, 노이즈를 추가하여 Claude를 덜 효과적으로 만들 수도 있습니다. Skill이 올바르게 트리거되지 않거나 Claude가 규칙을 잃을 수 있습니다. 이러한 트레이드오프를 이해하면 효과적인 설정을 구축하는 데 도움이 됩니다.

기능별 컨텍스트 비용

각 기능은 다양한 로딩 전략과 컨텍스트 비용을 가집니다.

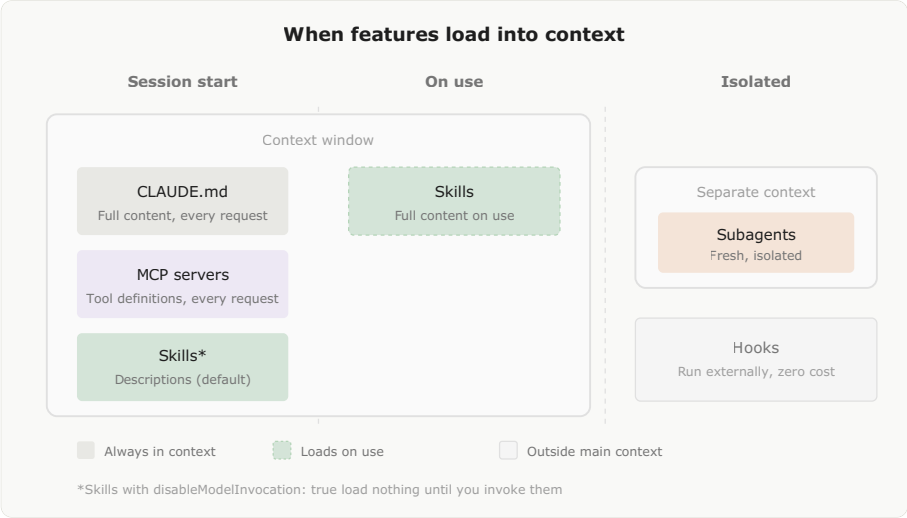
기능	로드 시기	로드되는 내용	컨텍스트 비용
CLAUDE.md	세션 시작	전체 콘텐츠	모든 요청
Skill	세션 시작 + 사용 시	시작 시 설명, 사용 시 전체 콘텐츠	낮음(모든 요청마다 설명)*
MCP 서버	세션 시작	모든 도구 정의 및 스키마	모든 요청
Subagent	생성 시	지정된 skill이 있는 신선한 컨텍스트	주 세션에서 격리됨

기능	로드 시기	로드되는 내용	컨텍스트 비용
Hooks	트리거 시	없음(외부에서 실행)	0, hook이 추가 컨텍스트를 반환하지 않는 한

*기본적으로 skill 설명은 세션 시작 시 로드되므로 Claude가 사용할 시기를 결정할 수 있습니다. Skill의 frontmatter에서 `disable-model-invocation: true`를 설정하여 수동으로 호출할 때까지 Claude에서 완전히 숨깁니다. 이는 skill의 컨텍스트 비용을 0으로 줄입니다.

기능이 어떻게 로드되는지 이해하기

각 기능은 세션의 다양한 지점에서 로드됩니다. 아래 탭은 각 기능이 언제 로드되고 무엇이 컨텍스트에 들어가는지 설명합니다.



컨텍스트 로딩: CLAUDE.md와 MCP는 세션 시작 시 로드되고 모든 요청에 유지됩니다. Skill은 시작 시 설명을 로드하고 호출 시 전체 콘텐츠를 로드합니다. Subagent는 격리된 컨텍스트를 받습니다. Hook은 외부에서 실행됩니다.

CLAUDE.md

시기: 세션 시작

로드되는 내용: 모든 CLAUDE.md 파일의 전체 콘텐츠(관리, 사용자 및 프로젝트 수준).

상속: Claude는 작업 디렉토리에서 루트까지 CLAUDE.md 파일을 읽고, 해당 파일에 접근할 때 하위 디렉토리에서 중첩된 파일을 검색합니다. 자세한 내용은 CLAUDE.md 파일이 로드되는 방식을 참조하세요.

> **Tip:**
> CLAUDE.md를 약 500줄 이하로 유지하세요. 참조 자료를 skill로 이동하면 온디맨드로 로드됩니다.

Skills

Skill은 Claude의 도구 키트에 있는 추가 기능입니다. 참조 자료(API 스타일 가이드처럼) 또는 `/<name>`으로 트리거하는 호출 가능한 워크플로우(배포처럼)일 수 있습니다. Claude Code는 기본적으로 작동하는 `/simplify`, `/batch`, `/debug`와 같은 번들 skill과 함께 제공됩니다. 자신의 것을 만들 수도 있습니다. Claude는 적절할 때 skill을 사용하거나 직접 호출할 수 있습니다.

시기: Skill의 구성에 따라 다릅니다. 기본적으로 설명은 세션 시작 시 로드되고 전체 콘텐츠는 사용 시 로드됩니다. 사용자 전용 skill(`disable-model-invocation: true`)의 경우, 호출할 때까지 아무것도 로드되지 않습니다.

로드되는 내용: 모델 호출 가능 skill의 경우, Claude는 모든 요청에서 이름과 설명을 봅니다. `/<name>`으로 skill을 호출하거나 Claude가 자동으로 로드할 때, 전체 콘텐츠가 대화에 로드됩니다.

Claude가 skill을 선택하는 방식: Claude는 작업을 skill 설명과 비교하여 관련성이 있는지 결정합니다. 설명이 모호하거나 겹치면, Claude가 잘못된 skill을 로드하거나 도움이 될 skill을 놓칠 수 있습니다. Claude에게 특정 skill을 사용하도록 지시하려면 `/<name>`으로 호출하세요. `disable-model-invocation: true`가 있는 Skill은 호출할 때까지 Claude에게 보이지 않습니다.

컨텍스트 비용: 사용할 때까지 낮음. 사용자 전용 skill은 호출할 때까지 0 비용입니다.

Subagent에서: Skill은 subagent에서 다르게 작동합니다. 온디맨드 로딩 대신, subagent에 전달된 skill은 시작 시 컨텍스트에 완전히 미리 로드됩니다. Subagent는 주 세션에서 skill을 상속하지 않습니다. 명시적으로 지정해야 합니다.

> **Tip:**
> 부작용이 있는 skill에 `disable-model-invocation: true`를 사용하세요. 이는 컨텍스트를 절약하고 오직 사용자만 트리거하도록 보장합니다.

MCP servers

시기: 세션 시작.

로드되는 내용: 연결된 서버의 모든 도구 정의 및 JSON 스키마.

컨텍스트 비용: 도구 검색(기본적으로 활성화)은 MCP 도구를 컨텍스트의 최대 10%까지 로드하고 나머지는 필요할 때까지 연기합니다.

신뢰성 참고: MCP 연결은 세션 중간에 조용히 실패할 수 있습니다. 서버가 연결 해제되면 도구가 경고 없이 사라집니다. Claude가 이전에 접근할 수 있었던 MCP 도구를 사용하지 못하는 경우, `/mcp` 로 연결을 확인하세요.

> **Tip:**
>
> 서버당 토큰 비용을 보려면 `/mcp` 를 실행하세요. 적극적으로 사용하지 않는 서버를 연결 해제하세요.

Subagents

시기: 온디맨드, 작업을 위해 사용자나 Claude가 생성할 때.

로드되는 내용: 신선한, 격리된 컨텍스트 포함:

- 시스템 프롬프트(캐시 효율성을 위해 부모와 공유)
- 에이전트의 `skills:` 필드에 나열된 skill의 전체 콘텐츠
- CLAUDE.md 및 git 상태(부모에서 상속)
- 리드 에이전트가 프롬프트에서 전달하는 모든 컨텍스트

컨텍스트 비용: 주 세션에서 격리됨. Subagent는 대화 기록이나 호출된 skill을 상속하지 않습니다.

> **Tip:**
>
> 전체 대화 컨텍스트가 필요하지 않은 작업에 subagent를 사용하세요. 격리는 주 세션이 부풀어지는 것을 방지합니다.

Hooks

시기: 트리거 시. Hook은 도구 실행, 세션 경계, 프롬프트 제출, 권한 요청 및 압축과 같은 특정 라이프사이클 이벤트에서 실행됩니다. 전체 목록은 Hooks를 참조하세요.

로드되는 내용: 기본적으로 없음. Hook은 외부 스크립트로 실행됩니다.

컨텍스트 비용: 0, hook이 대화에 메시지로 추가되는 출력을 반환하지 않는 한.

> **Tip:**
>
> Hook은 Claude의 컨텍스트에 영향을 주지 않아야 하는 부작용(린팅, 로깅)에 이상적입니다.

더 알아보기

각 기능에는 설정 지침, 예시 및 구성 옵션이 있는 자신의 가이드가 있습니다.

- CLAUDE.md 프로젝트 컨텍스트, 규칙 및 지침 저장
- Skills Claude에게 도메인 전문성 및 재사용 가능한 워크플로우 제공
- Subagents 격리된 컨텍스트로 작업 오프로드
- Agent teams 병렬로 작동하는 여러 세션 조정
- MCP Claude를 외부 서비스에 연결
- Hooks Hook으로 워크플로우 자동화
- Plugins 기능 세트 번들 및 공유
- Marketplaces 플러그인 컬렉션 호스트 및 배포

대화형 모드

Claude Code 세션의 키보드 단축키, 입력 모드 및 대화형 기능에 대한 완전한 참조입니다.

키보드 단축키

Note:

키보드 단축키는 플랫폼 및 터미널에 따라 다를 수 있습니다. `?` 를 눌러 사용자 환경에서 사용 가능한 단축키를 확인하세요.

macOS 사용자: Option/Alt 키 단축키(`Alt+B` , `Alt+F` , `Alt+Y` , `Alt+M` , `Alt+P`)를 사용하려면 터미널에서 Option을 Meta로 구성해야 합니다:

- **iTerm2**: 설정 → 프로필 → 키 → Left/Right Option 키를 "Esc+"로 설정
- **Terminal.app**: 설정 → 프로필 → 키보드 → "Option을 Meta 키로 사용" 확인
- **VS Code**: 설정 → 프로필 → 키 → Left/Right Option 키를 "Esc+"로 설정

자세한 내용은 터미널 구성을 참조하세요.

일반 제어

단축키	설명	컨텍스트
`Ctrl+C`	현재 입력 또는 생성 취소	표준 중단
`Ctrl+F`	모든 백그라운드 에이전트 종료. 3초 이내에 두 번 누르면 확인	백그라운드 에이전트 제어
`Ctrl+D`	Claude Code 세션 종료	EOF 신호
`Ctrl+G`	기본 텍스트 편집기에서 열기	기본 텍스트 편집기에서 프롬프트 또는 사용자 정의 응답 편집
`Ctrl+L`	터미널 화면 지우기	대화 기록 유지
`Ctrl+O`	상세 출력 토글	자세한 도구 사용 및 실행 표시
`Ctrl+R`	역방향 검색 명령 기록	이전 명령을 대화형으로 검색

단축키	설명	컨텍스트
`Ctrl+V` 또는 `Cmd+V` (iTerm2) 또는 `Alt+V` (Windows)	클립보드에서 이미지 붙여넣기	이미지 또는 이미지 파일 경로 붙여넣기
`Ctrl+B`	백그라운드 실행 작업	bash 명령 및 에이전트를 백그라운드로 실행. Tmux 사용자는 두 번 누르기
`Ctrl+T`	작업 목록 토글	터미널 상태 영역에서 작업 목록 표시 또는 숨기기
`Left/Right 화살표`	대화 상자 탭 순환	권한 대화 상자 및 메뉴의 탭 간 탐색
`Up/Down 화살표`	명령 기록 탐색	이전 입력 회상
`Esc` + `Esc`	되돌리기 또는 요약	코드 및/또는 대화를 이전 지점으로 복원하거나 선택한 메시지에서 요약
`Shift+Tab` 또는 `Alt+M` (일부 구성)	권한 모드 토글	자동 수락 모드, Plan Mode 및 일반 모드 간 전환.
`Option+P` (macOS) 또는 `Alt+P` (Windows/Linux)	모델 전환	프롬프트를 지우지 않고 모델 전환
`Option+T` (macOS) 또는 `Alt+T` (Windows/Linux)	확장 사고 토글	확장 사고 모드 활성화 또는 비활성화. 이 단축키를 활성화하려면 먼저 `/terminal-setup` 실행

텍스트 편집

단축키	설명	컨텍스트
`Ctrl+K`	줄 끝까지 삭제	삭제된 텍스트를 붙여넣기용으로 저장
`Ctrl+U`	전체 줄 삭제	삭제된 텍스트를 붙여넣기용으로 저장
`Ctrl+Y`	삭제된 텍스트 붙여넣기	`Ctrl+K` 또는 `Ctrl+U` 로 삭제한 텍스트 붙여넣기

단축키	설명	컨텍스트
`Alt+Y` (`Ctrl+Y` 이후)	붙여넣기 기록 순환	붙여넣은 후 이전에 삭제한 텍스트를 순환합니다. macOS에서 Option을 Meta로 필요
`Alt+B`	커서를 한 단어 뒤로 이동	단어 탐색. macOS에서 Option을 Meta로 필요
`Alt+F`	커서를 한 단어 앞으로 이동	단어 탐색. macOS에서 Option을 Meta로 필요

테마 및 표시

단축키	설명	컨텍스트
`Ctrl+T`	코드 블록의 구문 강조 토글	`/theme` 선택기 메뉴 내에서만 작동합니다. Claude의 응답에서 코드가 구문 색상을 사용하는지 여부를 제어합니다

> **Note:**
> 구문 강조는 Claude Code의 네이티브 빌드에서만 사용 가능합니다.

여러 줄 입력

방법	단축키	컨텍스트
빠른 이스케이프	`\` + `Enter`	모든 터미널에서 작동
macOS 기본값	`Option+Enter`	macOS의 기본값
Shift+Enter	`Shift+Enter`	iTerm2, WezTerm, Ghostty, Kitty에서 기본적으로 작동
제어 시퀀스	`Ctrl+J`	여러 줄의 라인 피드 문자
붙여넣기 모드	직접 붙여넣기	코드 블록, 로그의 경우

> **Tip:**
>
> Shift+Enter는 iTerm2, WezTerm, Ghostty 및 Kitty에서 구성 없이 작동합니다. 다른 터미널(VS Code, Alacritty, Zed, Warp)의 경우 `/terminal-setup` 을 실행하여 바인딩을 설치하세요.

빠른 명령

단축키	설명	참고
`/` 시작	명령 또는 skill	기본 제공 명령 및 skills 참조
`!` 시작	Bash 모드	명령을 직접 실행하고 실행 출력을 세션에 추가
`@`	파일 경로 언급	파일 경로 자동 완성 트리거

기본 제공 명령

Claude Code에서 `/` 를 입력하여 사용 가능한 모든 명령을 보거나, `/` 다음에 문자를 입력하여 필터링하세요. 모든 명령이 모든 사용자에게 표시되지는 않습니다. 일부는 플랫폼, 요금제 또는 환경에 따라 다릅니다. 예를 들어, `/desktop` 은 macOS 및 Windows에서만 나타나고, `/upgrade` 및 `/privacy-settings` 는 Pro 및 Max 요금제에서만 사용 가능하며, `/terminal-setup` 은 터미널이 기본적으로 키바인딩을 지원할 때 숨겨집니다.

Claude Code는 또한 `/` 를 입력할 때 기본 제공 명령과 함께 나타나는 `/simplify` , `/batch` 및 `/debug` 와 같은 번들 skills와 함께 제공됩니다. 자신의 명령을 만들려면 skills를 참조하세요.

아래 표에서 `<arg>` 는 필수 인수를 나타내고 `[arg]` 는 선택적 인수를 나타냅니다.

명령	목적
`/add-dir <path>`	현재 세션에 새 작업 디렉토리 추가
`/agents`	agent 구성 관리
`/btw <question>`	대화에 추가하지 않고 빠른 부가 질문 하기
`/chrome`	Chrome의 Claude 설정 구성
`/clear`	대화 기록을 지우고 컨텍스트 해제. 별칭: `/reset` , `/new`

명령	목적
`/compact [instructions]`	선택적 포커스 지침으로 대화 압축
`/config`	설정 인터페이스를 열어 테마, 모델, 출력 스타일 및 기타 기본 설정 조정. 별칭: `/settings`
`/context`	현재 컨텍스트 사용을 색상 그리드로 시각화
`/copy`	마지막 어시스턴트 응답을 클립보드에 복사. 코드 블록이 있을 때 개별 블록 또는 전체 응답을 선택할 수 있는 대화형 선택기 표시
`/cost`	토큰 사용 통계 표시. 구독별 세부 정보는 비용 추적 가이드 참조
`/desktop`	현재 세션을 Claude Code 데스크톱 앱에서 계속. macOS 및 Windows만 해당. 별칭: `/app`
`/diff`	커밋되지 않은 변경 사항 및 턴별 diff를 보여주는 대화형 diff 뷰어 열기. 왼쪽/오른쪽 화살표를 사용하여 현재 git diff와 개별 Claude 턴 간 전환, 위/아래로 파일 탐색
`/doctor`	Claude Code 설치 및 설정 진단 및 확인
`/exit`	CLI 종료. 별칭: `/quit`
`/export [filename]`	현재 대화를 일반 텍스트로 내보내기. 파일 이름이 있으면 해당 파일에 직접 작성. 없으면 클립보드에 복사하거나 파일에 저장할 대화 열기
`/extra-usage`	속도 제한에 도달했을 때 계속 작동하도록 추가 사용 구성
`/fast [on|off]`	빠른 모드 켜기 또는 끄기
`/feedback [report]`	Claude Code에 대한 피드백 제출. 별칭: `/bug`
`/fork [name]`	이 지점에서 현재 대화의 포크 생성
`/help`	도움말 및 사용 가능한 명령 표시
`/hooks`	hook 구성 관리
`/ide`	IDE 통합 관리 및 상태 표시
`/init`	`CLAUDE.md` 가이드로 프로젝트 초기화

명령	목적
`/insights`	프로젝트 영역, 상호 작용 패턴 및 마찰 지점을 포함하여 Claude Code 세션을 분석하는 보고서 생성
`/install-github-app`	리포지토리에 대한 Claude GitHub Actions 앱 설정. 리포지토리 선택 및 통합 구성을 안내합니다
`/install-slack-app`	Claude Slack 앱 설치. OAuth 흐름을 완료하기 위해 브라우저 열기
`/keybindings`	키바인딩 구성 파일 열기 또는 생성
`/login`	Anthropic 계정에 로그인
`/logout`	Anthropic 계정에서 로그아웃
`/mcp`	MCP 서버 연결 및 OAuth 인증 관리
`/memory`	`CLAUDE.md` 메모리 파일 편집, 자동 메모리 활성화 또는 비활성화, 자동 메모리 항목 보기
`/mobile`	Claude 모바일 앱 다운로드 QR 코드 표시. 별칭: `/ios`, `/android`
`/model [model]`	AI 모델 선택 또는 변경. 지원하는 모델의 경우 왼쪽/오른쪽 화살표를 사용하여 노력 수준 조정. 변경 사항은 현재 응답이 완료될 때까지 기다리지 않고 즉시 적용됩니다
`/passes`	친구와 Claude Code의 무료 1주일 공유. 계정이 적격인 경우에만 표시
`/permissions`	권한 보기 또는 업데이트. 별칭: `/allowed-tools`
`/plan`	프롬프트에서 직접 Plan Mode 입력
`/plugin`	Claude Code plugins 관리
`/pr-comments [PR]`	GitHub 풀 요청에서 댓글 가져오기 및 표시. 현재 분기에 대한 PR을 자동으로 감지하거나 PR URL 또는 번호 전달. `gh` CLI 필요
`/privacy-settings`	개인 정보 보호 설정 보기 및 업데이트. Pro 및 Max 요금제 구독자만 사용 가능
`/release-notes`	가장 최근 버전이 프롬프트에 가장 가까운 전체 변경 로그 보기
`/reload-plugins`	모든 활성 plugins를 다시 로드하여 재시작 없이 보류 중인 변경 사항 적용. 로드된 내용을 보고하고 재시작이 필요한 변경 사항 기록

명령	목적
`/remote-control`	이 세션을 claude.ai에서 원격 제어할 수 있도록 설정. 별칭: `/rc`
`/remote-env`	teleport 세션의 기본 원격 환경 구성
`/rename [name]`	현재 세션 이름 바꾸기. 이름 없이 대화 기록에서 자동 생성
`/resume [session]`	ID 또는 이름으로 대화 재개 또는 세션 선택기 열기. 별칭: `/continue`
`/review`	더 이상 사용되지 않음. 대신 `code-review` plugin 설치: `claude plugin install code-review@claude-code-marketplace`
`/rewind`	대화 및/또는 코드를 이전 지점으로 되돌리거나 선택한 메시지에서 요약. checkpointing 참조. 별칭: `/checkpoint`
`/sandbox`	sandbox 모드 토글. 지원되는 플랫폼에서만 사용 가능
`/security-review`	현재 분기의 보류 중인 변경 사항을 보안 취약점에 대해 분석. git diff를 검토하고 주입, 인증 문제 및 데이터 노출과 같은 위험을 식별합니다
`/skills`	사용 가능한 skills 나열
`/stats`	일일 사용, 세션 기록, 연속 기록 및 모델 기본 설정 시각화
`/status`	버전, 모델, 계정 및 연결성을 보여주는 설정 인터페이스(상태 탭) 열기
`/statusline`	Claude Code의 상태 줄 구성. 원하는 내용을 설명하거나 인수 없이 실행하여 셸 프롬프트에서 자동 구성
`/stickers`	Claude Code 스티커 주문
`/tasks`	백그라운드 작업 나열 및 관리
`/terminal-setup`	Shift+Enter 및 기타 단축키에 대한 터미널 키바인딩 구성. VS Code, Alacritty 또는 Warp와 같이 필요한 터미널에서만 표시
`/theme`	색상 테마 변경. 밝은 색과 어두운 색 변형, 색맹 접근 가능(daltonized) 테마 및 터미널의 색상 팔레트를 사용하는 ANSI 테마 포함
`/upgrade`	업그레이드 페이지를 열어 더 높은 요금제로 전환
`/usage`	요금제 사용 제한 및 속도 제한 상태 표시
`/vim`	Vim 및 일반 편집 모드 간 토글

MCP 프롬프트

MCP 서버는 명령으로 나타나는 프롬프트를 노출할 수 있습니다. 이들은 `/mcp__<server>__<prompt>` 형식을 사용하며 연결된 서버에서 동적으로 발견됩니다. 자세한 내용은 MCP 프롬프트를 참조하세요.

Vim 편집기 모드

`/vim` 명령으로 vim 스타일 편집을 활성화하거나 `/config` 를 통해 영구적으로 구성하세요.

모드 전환

명령	작업	모드에서
`Esc`	NORMAL 모드 입력	INSERT
`i`	커서 앞에 삽입	NORMAL
`I`	줄의 시작에 삽입	NORMAL
`a`	커서 뒤에 삽입	NORMAL
`A`	줄의 끝에 삽입	NORMAL
`o`	아래 줄 열기	NORMAL
`O`	위 줄 열기	NORMAL

탐색 (NORMAL 모드)

명령	작업
`h` / `j` / `k` / `l`	왼쪽/아래/위/오른쪽 이동
`w`	다음 단어
`e`	단어 끝
`b`	이전 단어
`0`	줄의 시작
`$`	줄의 끝
`^`	첫 번째 공백이 아닌 문자
`gg`	입력의 시작

명령	작업
`G`	입력의 끝
`f{char}`	다음 문자 발생으로 점프
`F{char}`	이전 문자 발생으로 점프
`t{char}`	다음 문자 발생 직전으로 점프
`T{char}`	이전 문자 발생 직후로 점프
`;`	마지막 f/F/t/T 모션 반복
`,`	마지막 f/F/t/T 모션을 역순으로 반복

> **Note:**
>
> vim 일반 모드에서 커서가 입력의 시작 또는 끝에 있고 더 이상 이동할 수 없으면 화살표 키가 명령 기록을 탐색합니다.

편집 (NORMAL 모드)

명령	작업
`x`	문자 삭제
`dd`	줄 삭제
`D`	줄 끝까지 삭제
`dw` / `de` / `db`	단어 삭제/끝까지/뒤로
`cc`	줄 변경
`C`	줄 끝까지 변경
`cw` / `ce` / `cb`	단어 변경/끝까지/뒤로
`yy` / `Y`	줄 복사
`yw` / `ye` / `yb`	단어 복사/끝까지/뒤로
`p`	커서 뒤에 붙여넣기
`P`	커서 앞에 붙여넣기

명령	작업
`>>`	줄 들여쓰기
`<<`	줄 내어쓰기
`J`	줄 결합
`.`	마지막 변경 반복

텍스트 객체 (NORMAL 모드)

텍스트 객체는 `d`, `c` 및 `y` 와 같은 연산자와 함께 작동합니다:

명령	작업
`iw` / `aw`	내부/주변 단어
`iW` / `aW`	내부/주변 WORD (공백 구분)
`i"` / `a"`	내부/주변 큰따옴표
`i'` / `a'`	내부/주변 작은따옴표
`i(` / `a(`	내부/주변 괄호
`i[` / `a[`	내부/주변 대괄호
`i{` / `a{`	내부/주변 중괄호

명령 기록

Claude Code는 현재 세션의 명령 기록을 유지합니다:

- 입력 기록은 작업 디렉토리별로 저장됩니다
- 입력 기록은 `/clear` 를 실행하여 새 세션을 시작할 때 재설정됩니다. 이전 세션의 대화는 보존되며 재개할 수 있습니다.
- Up/Down 화살표를 사용하여 탐색 (위의 키보드 단축키 참조)
- **참고**: 기록 확장(`!`)은 기본적으로 비활성화됩니다

Ctrl+R을 사용한 역방향 검색

`Ctrl+R` 을 눌러 명령 기록을 대화형으로 검색합니다:

1. **검색 시작**: `Ctrl+R` 을 눌러 역방향 기록 검색 활성화

2. **쿼리 입력**: 이전 명령에서 검색할 텍스트 입력. 검색 용어는 일치하는 결과에서 강조 표시됩니다
3. **일치 탐색**: `Ctrl+R` 을 다시 눌러 더 오래된 일치 항목을 순환합니다
4. **일치 수락**:
   - `Tab` 또는 `Esc` 를 눌러 현재 일치 항목을 수락하고 편집 계속
   - `Enter` 를 눌러 명령을 수락하고 즉시 실행
5. **검색 취소**:
   - `Ctrl+C` 를 눌러 취소하고 원래 입력 복원
   - 빈 검색에서 `Backspace` 를 눌러 취소

검색은 검색 용어가 강조된 일치하는 명령을 표시하므로 이전 입력을 찾아 재사용할 수 있습니다.

백그라운드 bash 명령

Claude Code는 bash 명령을 백그라운드에서 실행하여 장시간 실행되는 프로세스가 실행되는 동안 계속 작업할 수 있도록 지원합니다.

백그라운드 실행 작동 방식

Claude Code가 명령을 백그라운드에서 실행하면 명령을 비동기적으로 실행하고 즉시 백그라운드 작업 ID를 반환합니다. Claude Code는 명령이 백그라운드에서 계속 실행되는 동안 새 프롬프트에 응답할 수 있습니다.

명령을 백그라운드에서 실행하려면 다음 중 하나를 수행할 수 있습니다:

- Claude Code에 명령을 백그라운드에서 실행하도록 프롬프트
- Ctrl+B를 눌러 일반 Bash 도구 호출을 백그라운드로 이동. (Tmux 사용자는 tmux의 접두사 키로 인해 Ctrl+B를 두 번 눌러야 합니다.)

주요 기능:

- 출력은 버퍼링되고 Claude는 TaskOutput 도구를 사용하여 검색할 수 있습니다
- 백그라운드 작업에는 추적 및 출력 검색을 위한 고유 ID가 있습니다
- 백그라운드 작업은 Claude Code가 종료될 때 자동으로 정리됩니다

모든 백그라운드 작업 기능을 비활성화하려면 `CLAUDE_CODE_DISABLE_BACKGROUND_TASKS` 환경 변수를 `1` 로 설정하세요. 자세한 내용은 환경 변수를 참조하세요.

일반적인 백그라운드 명령:

- 빌드 도구 (webpack, vite, make)
- 패키지 관리자 (npm, yarn, pnpm)

- 테스트 러너 (jest, pytest)
- 개발 서버
- 장시간 실행 프로세스 (docker, terraform)

`!` 접두사를 사용한 Bash 모드

입력 앞에 `!`를 붙여 Claude를 거치지 않고 bash 명령을 직접 실행합니다:

```
! npm test
! git status
! ls -la
```

Bash 모드:

- 명령 및 출력을 대화 컨텍스트에 추가합니다
- 실시간 진행 상황 및 출력 표시
- 장시간 실행 명령에 대해 동일한 `Ctrl+B` 백그라운드 실행 지원
- Claude가 명령을 해석하거나 승인할 필요가 없습니다
- 기록 기반 자동 완성 지원: 부분 명령을 입력하고 **Tab**을 눌러 현재 프로젝트의 이전 `!` 명령에서 완성
- `Escape`, `Backspace` 또는 빈 프롬프트에서 `Ctrl+U`로 종료

이는 대화 컨텍스트를 유지하면서 빠른 셸 작업에 유용합니다.

프롬프트 제안

세션을 처음 열 때 시작하는 데 도움이 되는 회색 예제 명령이 프롬프트 입력에 나타납니다.
Claude Code는 프로젝트의 git 기록에서 이를 선택하므로 최근에 작업한 파일을 반영합니다.

Claude가 응답한 후 제안은 대화 기록을 기반으로 계속 나타나며, 예를 들어 다중 부분 요청의 후속 단계 또는 워크플로우의 자연스러운 연속입니다.

- **Tab**을 눌러 제안을 수락하거나 **Enter**를 눌러 수락하고 제출
- 입력을 시작하여 제안 해제

제안은 부모 대화의 프롬프트 캐시를 재사용하는 백그라운드 요청으로 실행되므로 추가 비용은 최소입니다. Claude Code는 불필요한 비용을 피하기 위해 캐시가 콜드일 때 제안 생성을 건너뜁니다.

제안은 대화의 첫 번째 턴 후, 비대화형 모드에서 및 Plan Mode에서 자동으로 건너뜁니다.

프롬프트 제안을 완전히 비활성화하려면 환경 변수를 설정하거나 `/config`에서 설정을 토글하세요:

```
export CLAUDE_CODE_ENABLE_PROMPT_SUGGESTION=false
```

/btw를 사용한 부가 질문

`/btw`를 사용하여 현재 작업에 대한 빠른 질문을 대화 기록에 추가하지 않고 합니다. 이는 빠른 답변을 원하지만 주 컨텍스트를 복잡하게 하거나 Claude를 장시간 실행 작업에서 벗어나게 하고 싶지 않을 때 유용합니다.

```
/btw what was the name of that config file again?
```

부가 질문은 현재 대화에 완전히 표시되므로 Claude가 이미 읽은 코드, 이전에 내린 결정 또는 세션의 다른 항목에 대해 질문할 수 있습니다. 질문과 답변은 임시입니다: 해제 가능한 오버레이에 나타나며 대화 기록에 절대 입력되지 않습니다.

- **Claude가 작업하는 동안 사용 가능**: Claude가 응답을 처리하는 동안에도 `/btw`를 실행할 수 있습니다. 부가 질문은 독립적으로 실행되며 주 턴을 중단하지 않습니다.
- **도구 접근 없음**: 부가 질문은 이미 컨텍스트에 있는 것에서만 답변합니다. Claude는 부가 질문에 답할 때 파일을 읽거나 명령을 실행하거나 검색할 수 없습니다.
- **단일 응답**: 후속 턴이 없습니다. 왕복이 필요하면 일반 프롬프트를 대신 사용하세요.
- **낮은 비용**: 부가 질문은 부모 대화의 프롬프트 캐시를 재사용하므로 추가 비용은 최소입니다.

Space, **Enter** 또는 **Escape**를 눌러 답변을 해제하고 프롬프트로 돌아갑니다.

`/btw`는 subagent의 역입니다: 전체 대화를 보지만 도구가 없는 반면, subagent는 전체 도구를 가지지만 빈 컨텍스트로 시작합니다. `/btw`를 사용하여 Claude가 이 세션에서 이미 알고 있는 것에 대해 질문하세요; subagent를 사용하여 새로운 것을 찾아보세요.

작업 목록

복잡한 다단계 작업을 수행할 때 Claude는 진행 상황을 추적하기 위해 작업 목록을 만듭니다. 작업은 터미널의 상태 영역에 보류 중, 진행 중 또는 완료를 나타내는 표시기와 함께 나타납니다.

- `Ctrl+T`를 눌러 작업 목록 보기를 토글합니다. 디스플레이는 한 번에 최대 10개의 작업을 표시합니다
- 모든 작업을 보거나 지우려면 Claude에 직접 요청하세요: “show me all tasks” 또는 “clear all tasks”

- 작업은 컨텍스트 압축 전체에서 지속되어 Claude가 더 큰 프로젝트에서 조직화된 상태를 유지하도록 도와줍니다
- 세션 간 작업 목록을 공유하려면 `CLAUDE_CODE_TASK_LIST_ID`를 `~/.claude/tasks/`의 명명된 디렉토리로 사용하도록 설정하세요:
  `CLAUDE_CODE_TASK_LIST_ID=my-project claude`
- 이전 TODO 목록으로 되돌리려면 `CLAUDE_CODE_ENABLE_TASKS=false`를 설정하세요.

PR 검토 상태

열린 풀 요청이 있는 분기에서 작업할 때 Claude Code는 바닥글에 클릭 가능한 PR 링크를 표시합니다 (예: "PR #446"). 링크에는 검토 상태를 나타내는 색상 밑줄이 있습니다:

- 녹색: 승인됨
- 노란색: 검토 대기 중
- 빨간색: 변경 요청됨
- 회색: 초안
- 보라색: 병합됨

`Cmd+click` (Mac) 또는 `Ctrl+click` (Windows/Linux)로 링크를 클릭하여 브라우저에서 풀 요청을 엽니다. 상태는 60초마다 자동으로 업데이트됩니다.

> **Note:**
>
> PR 상태는 `gh` CLI가 설치되고 인증되어야 합니다 (`gh auth login`).

참고 항목

- Skills - 사용자 정의 프롬프트 및 워크플로우
- Checkpointing - Claude의 편집 되돌리기 및 이전 상태 복원
- CLI 참조 - 명령줄 플래그 및 옵션
- 설정 - 구성 옵션
- 메모리 관리 - CLAUDE.md 파일 관리

Claude Code 모범 사례

> 환경 구성부터 병렬 세션 확장까지 Claude Code를 최대한 활용하기 위한 팁과 패턴입니다.

Claude Code는 에이전트 코딩 환경입니다. 질문에 답하고 기다리는 챗봇과 달리 Claude Code는 파일을 읽고, 명령을 실행하고, 변경을 수행하며, 당신이 지켜보거나 방향을 바꾸거나 완전히 떠나 있는 동안에도 자율적으로 문제를 해결할 수 있습니다.

이는 작업 방식을 바꿉니다. 직접 코드를 작성하고 Claude에게 검토를 요청하는 대신, 원하는 것을 설명하면 Claude가 어떻게 구축할지 파악합니다. Claude는 탐색하고, 계획하고, 구현합니다.

하지만 이러한 자율성에도 학습 곡선이 있습니다. Claude는 이해해야 할 특정 제약 조건 내에서 작동합니다.

이 가이드는 Anthropic의 내부 팀과 다양한 코드베이스, 언어, 환경에서 Claude Code를 사용하는 엔지니어들 사이에서 효과적으로 입증된 패턴을 다룹니다. 에이전트 루프가 내부적으로 어떻게 작동하는지에 대해서는 Claude Code 작동 방식을 참조하십시오.

대부분의 모범 사례는 하나의 제약 조건을 기반으로 합니다: Claude의 context window가 빠르게 채워지고, 채워질수록 성능이 저하됩니다.

Claude의 context window는 모든 메시지, Claude가 읽은 모든 파일, 모든 명령 출력을 포함한 전체 대화를 보유합니다. 그러나 이는 빠르게 채워질 수 있습니다. 단일 디버깅 세션이나 코드베이스 탐색만으로도 수만 개의 토큰을 생성하고 소비할 수 있습니다.

LLM 성능이 context가 채워질수록 저하되기 때문에 이는 중요합니다. context window가 거의 가득 차면 Claude는 이전 지시사항을 "잊기" 시작하거나 더 많은 실수를 할 수 있습니다. context window는 관리해야 할 가장 중요한 리소스입니다. 사용자 정의 상태 줄로 context 사용량을 지속적으로 추적하고, 토큰 사용량 감소에서 토큰 사용량을 줄이기 위한 전략을 참조하십시오.

Claude에게 작업을 검증할 방법 제공하기

> **Tip:**

> Claude가 자신의 작업을 확인할 수 있도록 테스트, 스크린샷 또는 예상 출력을 포함하십시오. 이것이 할 수 있는 가장 높은 영향력의 단일 작업입니다.

Claude는 테스트를 실행하고, 스크린샷을 비교하고, 출력을 검증하는 등 자신의 작업을 검증할 수 있을 때 훨씬 더 잘 수행합니다.

명확한 성공 기준이 없으면 올바르게 보이지만 실제로는 작동하지 않는 것을 생성할 수 있습니다. 당신이 유일한 피드백 루프가 되고, 모든 실수가 당신의 주의를 필요로 합니다.

전략	이전	이후
검증 기준 제공	“이메일 주소를 검증하는 함수를 구현하세요”	“validateEmail 함수를 작성하세요. 예제 테스트 케이스: user@example.com은 true, invalid는 false, user@.com은 false입니다. 구현 후 테스트를 실행하세요”
UI 변경 사항을 시각적으로 검증	“대시보드를 더 좋게 보이게 하세요”	“[스크린샷 붙여넣기] 이 디자인을 구현하세요. 결과의 스크린샷을 찍고 원본과 비교하세요. 차이점을 나열하고 수정하세요”
증상이 아닌 근본 원인 해결	“빌드가 실패하고 있습니다”	“빌드가 이 오류로 실패합니다: [오류 붙여넣기]. 수정하고 빌드가 성공하는지 확인하세요. 근본 원인을 해결하고 오류를 억제하지 마세요”

UI 변경 사항은 Chrome 확장 프로그램의 Claude를 사용하여 검증할 수 있습니다. 브라우저에서 새 탭을 열고, UI를 테스트하고, 코드가 작동할 때까지 반복합니다.

검증은 테스트 스위트, linter 또는 출력을 확인하는 Bash 명령일 수도 있습니다. 검증을 견고하게 만드는 데 투자하십시오.

먼저 탐색하고, 그 다음 계획하고, 그 다음 코드 작성하기

> **Tip:**
>
> 연구 및 계획을 구현과 분리하여 잘못된 문제를 해결하는 것을 피하십시오.

Claude가 바로 코딩으로 뛰어들도록 하면 잘못된 문제를 해결하는 코드가 생성될 수 있습니다. Plan Mode를 사용하여 탐색을 실행과 분리하십시오.

권장 워크플로우에는 4가지 단계가 있습니다:

Step 1: 탐색

Plan Mode를 입력하십시오. Claude는 파일을 읽고 변경을 수행하지 않고 질문에 답합니다.

```
/src/auth를 읽고 세션 및 로그인을 어떻게 처리하는지 이해하세요.
또한 비밀에 대한 환경 변수를 어떻게 관리하는지 살펴보세요.
```

Step 2: 계획

Claude에게 상세한 구현 계획을 작성하도록 요청하십시오.

```
Google OAuth를 추가하고 싶습니다. 어떤 파일을 변경해야 합니까?
세션 흐름은 무엇입니까? 계획을 작성하세요.
```

`Ctrl+G` 를 눌러 Claude가 진행하기 전에 텍스트 편집기에서 계획을 열어 직접 편집하십시오.

Step 3: 구현

Normal Mode로 전환하고 Claude가 코드를 작성하도록 하여 계획에 대해 검증하십시오.

```
계획에서 OAuth 흐름을 구현하세요. 콜백 핸들러에 대한 테스트를 작성하고,
테스트 스위트를 실행하고 실패를 수정하세요.
```

Step 4: 커밋

Claude에게 설명적인 메시지로 커밋하고 PR을 생성하도록 요청하십시오.

```
설명적인 메시지로 커밋하고 PR을 열기
```

Plan Mode는 유용하지만 오버헤드도 추가합니다.

범위가 명확하고 수정이 작은 작업(예: 오타 수정, 로그 줄 추가 또는 변수 이름 바꾸기)의 경우 Claude에게 직접 수행하도록 요청하십시오.

계획은 접근 방식이 불확실할 때, 변경이 여러 파일을 수정할 때, 또는 수정 중인 코드에 익숙하지 않을 때 가장 유용합니다. diff를 한 문장으로 설명할 수 있다면 계획을 건너뛰십시오.

프롬프트에서 구체적인 컨텍스트 제공하기

> **Tip:**
> 지시사항이 정확할수록 필요한 수정이 적습니다.

Claude는 의도를 추론할 수 있지만 마음을 읽을 수는 없습니다. 특정 파일을 참조하고, 제약 조건을 언급하고, 예제 패턴을 지적하십시오.

전략	이전	이후
작업 범위 지정. 어떤 파일, 어떤 시나리오, 테스트 선호도를 지정하십시오.	“foo.py에 대한 테스트 추가”	“사용자가 로그아웃된 경우의 엣지 케이스를 다루는 foo.py에 대한 테스트를 작성하세요. 모의 객체를 피하세요.”
소스 지적. Claude를 질문에 답할 수 있는 소스로 안내하십시오.	“ExecutionFactory가 왜 그렇게 이상한 API를 가지고 있습니까?”	“ExecutionFactory의 git 히스토리를 살펴보고 API가 어떻게 되었는지 요약하세요”
기존 패턴 참조. Claude를 코드베이스의 패턴으로 지적하십시오.	“캘린더 위젯 추가”	“홈 페이지에서 기존 위젯이 어떻게 구현되는지 살펴보고 패턴을 이해하세요. HotDogWidget.php는 좋은 예입니다. 패턴을 따라 사용자가 월을 선택하고 앞뒤로 이동하여 연도를 선택할 수 있는 새로운 캘린더 위젯을 구현하세요. 코드베이스에서 이미 사용 중인 라이브러리 외에는 라이브러리를 사용하지 않고 처음부터 빌드하세요.”
증상 설명. 증상, 가능한 위치, “수정됨”의 모습을 제공하십시오.	“로그인 버그 수정”	“사용자가 세션 시간 초과 후 로그인이 실패한다고 보고합니다. src/auth/의 인증 흐름, 특히 토큰 새로 고침을 확인하세요. 문제를 재현하는 실패한 테스트를 작성한 다음 수정하세요”

모호한 프롬프트는 탐색 중이고 과정을 수정할 여유가 있을 때 유용할 수 있습니다. `"이 파일에서 무엇을 개선하시겠습니까?"` 와 같은 프롬프트는 당신이 생각하지 못했을 것들을 드러낼 수 있습니다.

풍부한 콘텐츠 제공하기

> **Tip:**
>
> `@` 를 사용하여 파일을 참조하거나, 스크린샷/이미지를 붙여넣거나, 데이터를 직접 파이프하십시오.

여러 가지 방법으로 Claude에게 풍부한 데이터를 제공할 수 있습니다:

- **`@` 로 파일 참조** 코드가 어디에 있는지 설명하는 대신. Claude는 응답하기 전에 파일을 읽습니다.
- **이미지를 직접 붙여넣기**. 프롬프트에 이미지를 복사/붙여넣기 또는 드래그 앤 드롭하십시오.
- **문서 및 API 참조에 URL 제공**. `/permissions` 를 사용하여 자주 사용되는 도메인을 허용 목록에 추가하십시오.
- **데이터 파이프** `cat error.log | claude` 를 실행하여 파일 내용을 직접 전송하십시오.
- **Claude가 필요한 것을 가져오도록 하기**. Bash 명령, MCP 도구 또는 파일 읽기를 사용하여 Claude가 자체적으로 컨텍스트를 가져오도록 하십시오.

환경 구성하기

몇 가지 설정 단계를 통해 Claude Code를 모든 세션에서 훨씬 더 효과적으로 만들 수 있습니다. 확장 기능의 전체 개요 및 각 기능을 사용할 시기에 대해서는 Claude Code 확장을 참조하십시오.

효과적인 CLAUDE.md 작성하기

> **Tip:**
>
> `/init` 을 실행하여 현재 프로젝트 구조를 기반으로 시작 CLAUDE.md 파일을 생성한 다음 시간이 지남에 따라 개선하십시오.

CLAUDE.md는 Claude가 모든 대화의 시작 부분에서 읽는 특수 파일입니다. Bash 명령, 코드 스타일 및 워크플로우 규칙을 포함하십시오. 이는 Claude에게 코드만으로는 추론할 수 없는 지속적인 컨텍스트를 제공합니다.

`/init` 명령은 코드베이스를 분석하여 빌드 시스템, 테스트 프레임워크 및 코드 패턴을 감지하여 개선할 수 있는 견고한 기초를 제공합니다.

CLAUDE.md 파일에 필수 형식은 없지만 짧고 인간이 읽을 수 있도록 유지하십시오. 예를 들어:

```
## 코드 스타일
- ES 모듈(import/export) 구문을 사용하고, CommonJS(require)는 사용하지 마세요
- 가능하면 import를 구조 분해하세요 (예: import { foo } from 'bar')

## 워크플로우
- 일련의 코드 변경을 완료했을 때 타입 체크를 확인하세요
- 성능상 이유로 전체 테스트 스위트가 아닌 단일 테스트를 실행하는 것을 선호하세요
```

CLAUDE.md는 모든 세션에서 로드되므로 광범위하게 적용되는 것만 포함하십시오. 도메인 지식이나 때때만 관련된 워크플로우의 경우 대신 skills를 사용하십시오. Claude는 필요에 따라 로드하므로 모든 대화를 복잡하게 하지 않습니다.

간결하게 유지하십시오. 각 줄에 대해 다음을 물어보십시오: "이것을 제거하면 Claude가 실수를 할까?" 그렇지 않으면 삭제하십시오. 부풀려진 CLAUDE.md 파일은 Claude가 실제 지시사항을 무시하게 합니다!

✅ 포함	❌ 제외
Claude가 추측할 수 없는 Bash 명령	Claude가 코드를 읽어서 파악할 수 있는 것
기본값과 다른 코드 스타일 규칙	Claude가 이미 알고 있는 표준 언어 규칙
테스트 지시사항 및 선호하는 테스트 러너	상세한 API 문서(대신 문서 링크)
저장소 에티켓(분기 이름 지정, PR 규칙)	자주 변경되는 정보
프로젝트에 특정한 아키텍처 결정	긴 설명 또는 튜토리얼
개발자 환경 특이성(필수 환경 변수)	자명한 관행(예: "깨끗한 코드 작성")
일반적인 함정 또는 명백하지 않은 동작	파일별 코드베이스 설명

Claude가 규칙에도 불구하고 계속 원하지 않는 작업을 수행하면 파일이 너무 길어서 규칙이 손실되고 있을 가능성이 있습니다. Claude가 CLAUDE.md에서 답변된 질문을 하면 표현이 모호할 수 있습니다. CLAUDE.md를 코드처럼 취급하십시오: 문제가 발생하면 검토하고, 정기적으로 정리하고, 변경 사항을 관찰하여 Claude의 동작이 실제로 변경되는지 테스트하십시오.

강조(예: "IMPORTANT" 또는 "YOU MUST")를 추가하여 지시사항을 조정하면 준수를 개선할 수 있습니다. CLAUDE.md를 git에 체크인하여 팀이 기여할 수 있도록 하십시오. 파일은 시간이 지남에 따라 가치가 증가합니다.

CLAUDE.md 파일은 `@path/to/import` 구문을 사용하여 추가 파일을 가져올 수 있습니다:

```
프로젝트 개요는 @README.md를 참조하고 사용 가능한 npm 명령은 @package.json을 참조하세요.

## 추가 지시사항
- Git 워크플로우: @docs/git-instructions.md
- 개인 재정의: @~/.claude/my-project-instructions.md
```

CLAUDE.md 파일을 여러 위치에 배치할 수 있습니다:

- **홈 폴더(`~/.claude/CLAUDE.md`)**: 모든 Claude 세션에 적용됨
- **프로젝트 루트(`./CLAUDE.md`)**: git에 체크인하여 팀과 공유
- **상위 디렉토리**: 모노레포에 유용하며, `root/CLAUDE.md`와 `root/foo/CLAUDE.md` 모두 자동으로 가져와집니다
- **하위 디렉토리**: Claude는 해당 디렉토리의 파일로 작업할 때 필요에 따라 하위 CLAUDE.md 파일을 가져옵니다

권한 구성하기

> **Tip:**
>
> `/permissions`를 사용하여 안전한 명령을 허용 목록에 추가하거나 `/sandbox`를 사용하여 OS 수준 격리를 수행하십시오. 이는 제어를 유지하면서 중단을 줄입니다.

기본적으로 Claude Code는 시스템을 수정할 수 있는 작업에 대한 권한을 요청합니다: 파일 쓰기, Bash 명령, MCP 도구 등. 이는 안전하지만 번거롭습니다. 10번째 승인 후에는 실제로 검토하지 않고 클릭만 하고 있습니다. 이러한 중단을 줄이는 두 가지 방법이 있습니다:

- **권한 허용 목록**: 안전하다고 알고 있는 특정 도구 허용(예: `npm run lint` 또는 `git commit`)
- **샌드박싱**: Claude가 정의된 경계 내에서 더 자유롭게 작동할 수 있도록 하는 OS 수준 격리를 활성화하여 파일 시스템 및 네트워크 액세스를 제한합니다

또는 `--dangerously-skip-permissions`를 사용하여 lint 오류 수정 또는 보일러플레이트 생성과 같은 포함된 워크플로우에 대한 모든 권한 확인을 무시하십시오.

> **Warning:**
>
> Claude가 임의의 명령을 실행하도록 하면 데이터 손실, 시스템 손상 또는 프롬프트 주입을 통한 데이터 유출이 발생할 수 있습니다. 인터넷 액세스가 없는 샌드박스에서만 `--dangerously-skip-permissions`를 사용하십시오.

권한 구성 및 샌드박싱 활성화에 대해 자세히 읽어보십시오.

CLI 도구 사용하기

> **Tip:**
>
> Claude Code에 `gh`, `aws`, `gcloud`, `sentry-cli` 와 같은 CLI 도구를 사용하여 외부 서비스와 상호 작용하도록 하십시오.

CLI 도구는 외부 서비스와 상호 작용하는 가장 context 효율적인 방법입니다. GitHub를 사용하면 `gh` CLI를 설치하십시오. Claude는 이슈 생성, pull 요청 열기, 댓글 읽기에 사용하는 방법을 알고 있습니다. `gh` 없으면 Claude는 여전히 GitHub API를 사용할 수 있지만 인증되지 않은 요청은 종종 속도 제한에 도달합니다.

Claude는 또한 아직 알지 못하는 CLI 도구를 배우는 데 효과적입니다. `Use 'foo-cli-tool --help' to learn about foo tool, then use it to solve A, B, C.` 와 같은 프롬프트를 시도해보십시오.

MCP 서버 연결하기

> **Tip:**
>
> `claude mcp add` 를 실행하여 Notion, Figma 또는 데이터베이스와 같은 외부 도구를 연결하십시오.

MCP 서버를 사용하면 이슈 추적기에서 기능을 구현하고, 데이터베이스를 쿼리하고, 모니터링 데이터를 분석하고, Figma에서 디자인을 통합하고, 워크플로우를 자동화하도록 Claude에게 요청할 수 있습니다.

hooks 설정하기

> **Tip:**
>
> 예외 없이 매번 발생해야 하는 작업에 hooks를 사용하십시오.

Hooks는 Claude의 워크플로우의 특정 지점에서 자동으로 스크립트를 실행합니다. 권고적인 CLAUDE.md 지시사항과 달리 hooks는 결정론적이며 작업이 발생함을 보장합니다.

Claude가 hooks를 작성할 수 있습니다. “모든 파일 편집 후 eslint를 실행하는 hook 작성” 또는 *“마이그레이션 폴더에 대한 쓰기를 차단하는 hook 작성”*과 같은 프롬프트를 시도해보십시오. `/hooks` 를 실행하여 대화형 구성을 하거나 `.claude/settings.json` 을 직접 편집하십시오.

skills 생성하기

> **Tip:**
>
> `.claude/skills/` 에 `SKILL.md` 파일을 생성하여 Claude에게 도메인 지식과 재사용 가능한 워크플로우를 제공하십시오.

Skills는 프로젝트, 팀 또는 도메인에 특정한 정보로 Claude의 지식을 확장합니다. Claude는 관련이 있을 때 자동으로 적용하거나 `/skill-name` 으로 직접 호출할 수 있습니다.

`.claude/skills/` 에 `SKILL.md` 가 있는 디렉토리를 추가하여 skill을 생성하십시오:

```
---
name: api-conventions
description: 우리 서비스의 REST API 설계 규칙
---
## API 규칙
- URL 경로에 kebab-case 사용
- JSON 속성에 camelCase 사용
- 항상 목록 엔드포인트에 페이지 매김 포함
- URL 경로에서 API 버전 지정 (/v1/, /v2/)
```

Skills는 또한 직접 호출하는 반복 가능한 워크플로우를 정의할 수 있습니다:

```
---
name: fix-issue
description: GitHub 이슈 수정
disable-model-invocation: true
---
GitHub 이슈를 분석하고 수정하세요: $ARGUMENTS.

1. `gh issue view`를 사용하여 이슈 세부 정보 가져오기
2. 이슈에 설명된 문제 이해
3. 관련 파일에 대한 코드베이스 검색
4. 이슈를 수정하기 위해 필요한 변경 사항 구현
5. 수정을 확인하기 위해 테스트 작성 및 실행
6. 코드가 linting 및 타입 체크를 통과하는지 확인
7. 설명적인 커밋 메시지 생성
8. 푸시 및 PR 생성
```

`/fix-issue 1234`를 실행하여 호출하십시오. 부작용이 있는 워크플로우의 경우 `disable-model-invocation: true`를 사용하여 수동으로 트리거하려고 합니다.

사용자 정의 subagents 생성하기

> **Tip:**
>
> `.claude/agents/`에서 전문화된 어시스턴트를 정의하여 Claude가 격리된 작업에 위임할 수 있도록 하십시오.

Subagents는 자신의 context와 자신의 허용된 도구 집합으로 실행됩니다. 많은 파일을 읽거나 주요 대화를 복잡하게 하지 않고 전문화된 초점이 필요한 작업에 유용합니다.

```
---
name: security-reviewer
description: 보안 취약점에 대한 코드 검토
tools: Read, Grep, Glob, Bash
model: opus
---
당신은 선임 보안 엔지니어입니다. 다음에 대해 코드를 검토하세요:
- 주입 취약점(SQL, XSS, 명령 주입)
- 인증 및 권한 부여 결함
- 코드의 비밀 또는 자격 증명
- 안전하지 않은 데이터 처리

특정 줄 참조 및 제안된 수정 사항을 제공하세요.
```

Claude에게 명시적으로 subagents를 사용하도록 하십시오: "subagent를 사용하여 이 코드를 보안 문제에 대해 검토하세요."

plugins 설치하기

> **Tip:**
>
> `/plugin` 을 실행하여 마켓플레이스를 탐색하십시오. Plugins는 구성 없이 skills, tools, integrations를 추가합니다.

Plugins는 커뮤니티 및 Anthropic의 마켓플레이스에서 설치 가능한 단일 단위로 skills, hooks, subagents, MCP 서버를 번들로 제공합니다. 타입이 지정된 언어로 작업하면 코드 인텔리전스 plugin을 설치하여 Claude에게 정확한 기호 탐색 및 편집 후 자동 오류 감지를 제공하십시오.

skills, subagents, hooks, MCP 중에서 선택하는 방법에 대한 지침은 Claude Code 확장을 참조하십시오.

효과적으로 소통하기

Claude Code와의 소통 방식은 결과의 품질에 크게 영향을 미칩니다.

코드베이스 질문 하기

> **Tip:**

> 선임 엔지니어에게 물어볼 질문을 Claude에게 하십시오.

새로운 코드베이스에 온보딩할 때 Claude Code를 학습 및 탐색에 사용하십시오. Claude에게 다른 엔지니어에게 물어볼 것과 같은 종류의 질문을 할 수 있습니다:

- 로깅은 어떻게 작동합니까?
- 새로운 API 엔드포인트를 어떻게 만듭니까?
- `foo.rs`의 134번 줄에서 `async move { ... }`는 무엇을 합니까?
- `CustomerOnboardingFlowImpl`은 어떤 엣지 케이스를 처리합니까?
- 이 코드가 333번 줄에서 `bar()` 대신 `foo()`를 호출하는 이유는 무엇입니까?

이런 방식으로 Claude Code를 사용하는 것은 효과적인 온보딩 워크플로우이며, 램프업 시간을 개선하고 다른 엔지니어의 부담을 줄입니다. 특별한 프롬프팅이 필요하지 않습니다: 직접 질문하십시오.

Claude가 당신을 인터뷰하도록 하기

> **Tip:**
>
> 더 큰 기능의 경우 Claude가 먼저 당신을 인터뷰하도록 하십시오. 최소한의 프롬프트로 시작하고 Claude에게 `AskUserQuestion` 도구를 사용하여 당신을 인터뷰하도록 요청하십시오.

Claude는 기술 구현, UI/UX, 엣지 케이스, 트레이드오프를 포함하여 아직 고려하지 않은 것들에 대해 질문합니다.

```
[간단한 설명]을 빌드하고 싶습니다. AskUserQuestion 도구를 사용하여 자세히 인터뷰해주세요.

기술 구현, UI/UX, 엣지 케이스, 우려 사항, 트레이드오프에 대해 질문하세요. 명백한 질문을 하지 마세요, 당신이 고려하지 않았을 수 있는 어려운 부분을 파고드세요.

모든 것을 다룰 때까지 인터뷰를 계속한 다음 SPEC.md에 완전한 사양을 작성하세요.
```

사양이 완료되면 새 세션을 시작하여 실행하십시오. 새 세션은 구현에만 집중하는 깨끗한 context를 가지고 있으며, 참조할 수 있는 작성된 사양이 있습니다.

세션 관리하기

대화는 지속적이고 되돌릴 수 있습니다. 이를 활용하십시오!

조기에 자주 방향 수정하기

> **Tip:**
>
> Claude가 궤도를 벗어나는 것을 알아차리면 즉시 수정하십시오.

최고의 결과는 긴밀한 피드백 루프에서 나옵니다. Claude가 때때로 첫 시도에서 문제를 완벽하게 해결하지만, 빠르게 수정하면 일반적으로 더 빠르게 더 나은 솔루션을 생성합니다.

- `Esc` : `Esc` 키로 Claude의 작업을 중간에 중지하십시오. Context는 보존되므로 방향을 바꿀 수 있습니다.
- `Esc + Esc` **또는** `/rewind` : `Esc` 를 두 번 누르거나 `/rewind` 를 실행하여 rewind 메뉴를 열고 이전 대화 및 코드 상태를 복원하거나 선택한 메시지에서 요약하십시오.
- `"Undo that"` : Claude에게 변경 사항을 되돌리도록 하십시오.
- `/clear` : 관련 없는 작업 간에 context를 재설정하십시오. 관련 없는 context가 있는 긴 세션은 성능을 줄일 수 있습니다.

한 세션에서 같은 문제에 대해 Claude를 두 번 이상 수정했다면 context는 실패한 접근 방식으로 복잡해져 있습니다. `/clear` 를 실행하고 배운 내용을 통합하는 더 구체적인 프롬프트로 새로 시작하십시오. 누적된 수정이 있는 긴 세션보다 더 나은 프롬프트가 있는 깨끗한 세션이 거의 항상 더 나은 성능을 발휘합니다.

context 적극적으로 관리하기

> **Tip:**
>
> 관련 없는 작업 간에 `/clear` 를 자주 실행하여 context window를 재설정하십시오.

Claude Code는 context 제한에 접근할 때 대화 기록을 자동으로 압축하여 중요한 코드와 결정을 보존하면서 공간을 확보합니다.

긴 세션 동안 Claude의 context window는 관련 없는 대화, 파일 내용, 명령으로 채워질 수 있습니다. 이는 성능을 줄이고 때때로 Claude를 산만하게 할 수 있습니다.

- 작업 간에 자주 `/clear` 를 사용하여 context window를 완전히 재설정하십시오
- 자동 압축이 트리거되면 Claude는 코드 패턴, 파일 상태, 주요 결정을 포함하여 가장 중요한 것을 요약합니다
- 더 많은 제어를 위해 `/compact <instructions>` 를 실행하십시오(예: `/compact Focus on the API changes`)

- 대화의 일부만 압축하려면 `Esc + Esc` 또는 `/rewind`를 사용하고, 메시지 체크포인트를 선택하고, **Summarize from here**를 선택하십시오. 이는 해당 지점부터의 메시지를 압축하면서 이전 context를 유지합니다.
- CLAUDE.md에서 `"When compacting, always preserve the full list of modified files and any test commands"`와 같은 지시사항으로 압축 동작을 사용자 정의하여 중요한 context가 요약을 통해 유지되도록 하십시오
- 빠른 질문의 경우 context에 들어가지 않아야 하므로 `/btw`를 사용하십시오. 답변은 해제 가능한 오버레이에 나타나고 대화 기록에 들어가지 않으므로 context를 증가시키지 않고 세부 정보를 확인할 수 있습니다.

subagents를 사용하여 조사하기

> **Tip:**
>
> `"use subagents to investigate X"`로 연구를 위임하십시오. 그들은 별도의 context에서 탐색하여 구현을 위해 주요 대화를 깨끗하게 유지합니다.

context가 기본 제약 조건이므로 subagents는 사용 가능한 가장 강력한 도구 중 하나입니다. Claude가 코드베이스를 연구할 때 많은 파일을 읽으며, 모두 context를 소비합니다. Subagents는 별도의 context window에서 실행되고 요약을 보고합니다:

```
subagents를 사용하여 인증 시스템이 토큰 새로 고침을 어떻게 처리하는지,
그리고 재사용해야 할 기존 OAuth 유틸리티가 있는지 조사하세요.
```

subagent는 코드베이스를 탐색하고, 관련 파일을 읽고, 주요 대화를 복잡하게 하지 않고 발견 사항을 보고합니다.

Claude가 구현한 후 검증을 위해 subagents를 사용할 수도 있습니다:

```
subagent를 사용하여 이 코드를 엣지 케이스에 대해 검토하세요
```

체크포인트로 rewind하기

> **Tip:**
>
> Claude가 수행하는 모든 작업은 체크포인트를 생성합니다. 이전 체크포인트로 대화, 코드 또는 둘 다를 복원할 수 있습니다.

Claude는 변경 전에 자동으로 체크포인트합니다. `Escape` 를 두 번 누르거나 `/rewind` 를 실행하여 rewind 메뉴를 열기. 대화만 복원하거나, 코드만 복원하거나, 둘 다 복원하거나, 선택한 메시지에서 요약할 수 있습니다. 자세한 내용은 Checkpointing을 참조하십시오.

모든 움직임을 신중하게 계획하는 대신 Claude에게 위험한 것을 시도하도록 할 수 있습니다. 작동하지 않으면 rewind하고 다른 접근 방식을 시도하십시오. 체크포인트는 세션 간에 유지되므로 터미널을 닫아도 나중에 rewind할 수 있습니다.

> **Warning:**
>
> 체크포인트는 Claude가 수행한 변경만 추적하며, 외부 프로세스는 추적하지 않습니다. 이는 git의 대체품이 아닙니다.

대화 재개하기

> **Tip:**
>
> `claude --continue` 를 실행하여 중단한 곳에서 계속하거나, `--resume` 을 사용하여 최근 세션에서 선택하십시오.

Claude Code는 대화를 로컬로 저장합니다. 작업이 여러 세션에 걸쳐 있을 때 context를 다시 설명할 필요가 없습니다:

```
claude --continue    # 가장 최근 대화 재개
claude --resume      # 최근 대화에서 선택
```

`/rename` 을 사용하여 세션에 `"oauth-migration"` 또는 `"debugging-memory-leak"` 과 같은 설명적인 이름을 지정하여 나중에 찾을 수 있도록 하십시오. 세션을 분기처럼 취급하십시오: 다양한 작업 스트림은 별도의 지속적인 context를 가질 수 있습니다.

자동화 및 확장하기

한 Claude로 효과적이 되면 병렬 세션, 비대화형 모드, fan-out 패턴으로 출력을 곱하십시오.

지금까지 모든 것은 한 명의 인간, 한 명의 Claude, 한 개의 대화를 가정합니다. 하지만 Claude Code는 수평으로 확장됩니다. 이 섹션의 기술은 더 많은 작업을 수행하는 방법을 보여줍니다.

비대화형 모드 실행하기

> **Tip:**
>
> CI, pre-commit hooks 또는 스크립트에서 `claude -p "prompt"` 를 사용하십시오. 스트리밍 JSON 출력의 경우 `--output-format stream-json` 을 추가하십시오.

`claude -p "your prompt"` 를 사용하면 세션 없이 비대화형으로 Claude를 실행할 수 있습니다. 비대화형 모드는 Claude를 CI 파이프라인, pre-commit hooks 또는 자동화된 워크플로우에 통합하는 방법입니다. 출력 형식을 사용하면 결과를 프로그래밍 방식으로 구문 분석할 수 있습니다: 일반 텍스트, JSON 또는 스트리밍 JSON.

```
## 일회성 쿼리
claude -p "이 프로젝트가 무엇을 하는지 설명하세요"

## 스크립트를 위한 구조화된 출력
claude -p "모든 API 엔드포인트 나열" --output-format json

## 실시간 처리를 위한 스트리밍
claude -p "이 로그 파일 분석" --output-format stream-json
```

여러 Claude 세션 실행하기

> **Tip:**
>
> 개발 속도를 높이거나, 격리된 실험을 실행하거나, 복잡한 워크플로우를 시작하기 위해 여러 Claude 세션을 병렬로 실행하십시오.

병렬 세션을 실행하는 세 가지 주요 방법이 있습니다:

- Claude Code 데스크톱 앱: 여러 로컬 세션을 시각적으로 관리하십시오. 각 세션은 자신의 격리된 worktree를 가집니다.
- 웹의 Claude Code: Anthropic의 안전한 클라우드 인프라에서 격리된 VM에서 실행하십시오.
- Agent teams: 공유 작업, 메시징, 팀 리더를 사용한 여러 세션의 자동 조정.

작업을 병렬화하는 것 외에도 여러 세션은 품질 중심 워크플로우를 활성화합니다. 새로운 context는 Claude가 방금 작성한 코드에 편향되지 않으므로 코드 검토를 개선합니다.

예를 들어 Writer/Reviewer 패턴을 사용하십시오:

세션 A (작성자)	세션 B (검토자)
`API 엔드포인트에 대한 속도 제한기 구현`	
	`@src/middleware/rateLimiter.ts의 속도 제한기 구현을 검토하세요. 엣지 케이스, 경쟁 조건, 기존 미들웨어 패턴과의 일관성을 찾으세요.`
`검토 피드백은 다음과 같습니다: [세션 B 출력]. 이 문제들을 해결하세요.`	

테스트로 비슷한 작업을 수행할 수 있습니다: 한 Claude가 테스트를 작성하고 다른 Claude가 테스트를 통과하는 코드를 작성합니다.

파일 전체에 fan out하기

> **Tip:**
>
> 각각에 대해 `claude -p` 를 호출하는 루프를 통해 작업을 분배하십시오. 배치 작업의 경우 `--allowedTools` 를 사용하여 권한을 범위 지정하십시오.

대규모 마이그레이션 또는 분석의 경우 많은 병렬 Claude 호출 전체에 작업을 분배할 수 있습니다:

Step 1: 작업 목록 생성

Claude가 마이그레이션이 필요한 모든 파일을 나열하도록 하십시오(예: `마이그레이션이 필요한 모든 2,000개의 Python 파일 나열`)

Step 2: 목록을 통해 루프하는 스크립트 작성

```
for file in $(cat files.txt); do
  claude -p "React에서 Vue로 $file 마이그레이션. OK 또는 FAIL 반환." \
    --allowedTools "Edit,Bash(git commit *)"
done
```

Step 3: 몇 개 파일에서 테스트한 다음 규모에 맞게 실행

처음 2-3개 파일에서 잘못된 것을 기반으로 프롬프트를 개선한 다음 전체 집합에서 실행하십시오. `--allowedTools` 플래그는 Claude가 할 수 있는 작업을 제한하며, 이는 무인 상태에서 실행할 때 중요합니다.

Claude를 기존 데이터/처리 파이프라인에 통합할 수도 있습니다:

```
claude -p "<your prompt>" --output-format json | your_command
```

개발 중에 디버깅을 위해 `--verbose`를 사용하고 프로덕션에서는 끄십시오.

일반적인 실패 패턴 피하기

이는 일반적인 실수입니다. 조기에 인식하면 시간을 절약합니다:

- **주방 싱크 세션.** 한 작업으로 시작한 다음 Claude에게 관련 없는 것을 물어본 다음 첫 번째 작업으로 돌아갑니다. Context는 관련 없는 정보로 가득 찹니다. > **수정**: 관련 없는 작업 간에 `/clear`.
- **반복적으로 수정.** Claude가 뭔가 잘못하고, 당신이 수정하고, 여전히 잘못되고, 다시 수정합니다. Context는 실패한 접근 방식으로 오염됩니다. > **수정**: 두 번의 실패한 수정 후 `/clear`를 하고 배운 내용을 통합하는 더 나은 초기 프롬프트를 작성하십시오.
- **과도하게 지정된 CLAUDE.md.** CLAUDE.md가 너무 길면 Claude는 중요한 규칙이 노이즈에 손실되기 때문에 절반을 무시합니다. > **수정**: 무자비하게 정리하십시오. Claude가 지시사항 없이 이미 올바르게 수행하면 삭제하거나 hook으로 변환하십시오.
- **신뢰-검증 간격.** Claude는 그럴듯해 보이지만 엣지 케이스를 처리하지 않는 구현을 생성합니다. > **수정**: 항상 검증(테스트, 스크립트, 스크린샷)을 제공하십시오. 검증할 수 없으면 배포하지 마십시오.
- **무한 탐색.** 범위를 지정하지 않고 Claude에게 뭔가를 "조사"하도록 요청합니다. Claude는 수백 개의 파일을 읽으며 context를 채웁니다. > **수정**: 조사를 좁게 범위 지정하거나 subagents를 사용하여 탐색이 주요 context를 소비하지 않도록 하십시오.

직관 개발하기

이 가이드의 패턴은 정해진 것이 아닙니다. 일반적으로 잘 작동하지만 모든 상황에 최적일 수는 없는 시작점입니다.

때로는 복잡한 문제에 깊이 있고 기록이 가치 있기 때문에 context가 누적되도록 해야 합니다. 때로는 작업이 탐색적이기 때문에 계획을 건너뛰고 Claude가 파악하도록 해야 합니다. 때로는 모호한 프롬프트가 정확히 맞기 때문에 Claude가 문제를 해석하는 방식을 보고 싶을 때입니다.

작동하는 것에 주의를 기울이십시오. Claude가 훌륭한 출력을 생성할 때 당신이 한 것을 주목하십시오: 프롬프트 구조, 제공한 context, 당신이 있던 모드. Claude가 어려움을 겪을 때 왜인지 물어보십시오. Context가 너무 시끄러웠습니까? 프롬프트가 너무 모호했습니까? 작업이 한 번에 너무 컸습니까?

시간이 지남에 따라 어떤 가이드도 포착할 수 없는 직관을 개발할 것입니다. 구체적일 때와 개방적일 때, 계획할 때와 탐색할 때, context를 지울 때와 누적하도록 할 때를 알게 될 것입니다.

관련 리소스

- Claude Code 작동 방식: 에이전트 루프, 도구, context 관리
- Claude Code 확장: skills, hooks, MCP, subagents, plugins
- 일반적인 워크플로우: 디버깅, 테스트, PR 등에 대한 단계별 레시피
- CLAUDE.md: 프로젝트 규칙 및 지속적인 context 저장

일반적인 워크플로우

> Claude Code를 사용하여 코드베이스 탐색, 버그 수정, 리팩토링, 테스트 및 기타 일상적인 작업을 위한 단계별 가이드입니다.

이 페이지는 일상적인 개발을 위한 실용적인 워크플로우를 다룹니다: 낯선 코드 탐색, 디버깅, 리팩토링, 테스트 작성, PR 생성 및 세션 관리. 각 섹션에는 자신의 프로젝트에 맞게 조정할 수 있는 예제 프롬프트가 포함되어 있습니다. 더 높은 수준의 패턴과 팁은 모범 사례를 참조하십시오.

새로운 코드베이스 이해하기

코드베이스의 빠른 개요 얻기

새로운 프로젝트에 방금 참여했고 그 구조를 빠르게 이해해야 한다고 가정해봅시다.

Step 1: 프로젝트 루트 디렉토리로 이동

```
cd /path/to/project
```

Step 2: Claude Code 시작

```
claude
```

Step 3: 높은 수준의 개요 요청

```
give me an overview of this codebase
```

Step 4: 특정 구성 요소에 대해 더 깊이 있게 살펴보기

```
explain the main architecture patterns used here
```

```
what are the key data models?
```

```
how is authentication handled?
```

> **Tip:**
>
> 팁:
>
> - 광범위한 질문으로 시작한 다음 특정 영역으로 좁혀나가기
> - 프로젝트에서 사용되는 코딩 규칙과 패턴에 대해 질문하기
> - 프로젝트별 용어의 용어집 요청하기

관련 코드 찾기

특정 기능이나 기능과 관련된 코드를 찾아야 한다고 가정해봅시다.

Step 1: Claude에게 관련 파일을 찾도록 요청

```
find the files that handle user authentication
```

Step 2: 구성 요소가 어떻게 상호작용하는지에 대한 컨텍스트 얻기

```
how do these authentication files work together?
```

Step 3: 실행 흐름 이해하기

```
trace the login process from front-end to database
```

> **Tip:**
>
> 팁:
>
> - 찾고 있는 것에 대해 구체적으로 설명하기
> - 프로젝트의 도메인 언어 사용하기
> - 언어에 대한 코드 인텔리전스 플러그인을 설치하여 Claude에게 정확한 “정의로 이동” 및 “참조 찾기” 네비게이션 제공하기

효율적으로 버그 수정하기

오류 메시지가 나타났고 그 원인을 찾아 수정해야 한다고 가정해봅시다.

Step 1: Claude와 오류 공유하기

```
I'm seeing an error when I run npm test
```

Step 2: 수정 권장사항 요청하기

```
suggest a few ways to fix the @ts-ignore in user.ts
```

Step 3: 수정 적용하기

```
update user.ts to add the null check you suggested
```

> **Tip:**
>
> 팁:
>
> - Claude에게 문제를 재현하는 명령과 스택 추적을 알려주기
> - 오류를 재현하는 단계 언급하기
> - 오류가 간헐적인지 일관적인지 Claude에게 알려주기

코드 리팩토링

오래된 코드를 최신 패턴과 관행을 사용하도록 업데이트해야 한다고 가정해봅시다.

Step 1: 리팩토링할 레거시 코드 식별

```
find deprecated API usage in our codebase
```

Step 2: 리팩토링 권장사항 얻기

```
suggest how to refactor utils.js to use modern JavaScript features
```

Step 3: 안전하게 변경사항 적용하기

```
refactor utils.js to use ES2024 features while maintaining the same behavior
```

Step 4: 리팩토링 검증하기

```
run tests for the refactored code
```

> **Tip:**
>
> 팁:
>
> - Claude에게 최신 접근 방식의 이점을 설명하도록 요청하기
> - 필요할 때 변경사항이 하위 호환성을 유지하도록 요청하기
> - 작고 테스트 가능한 증분으로 리팩토링 수행하기

특화된 subagent 사용하기

특정 작업을 더 효과적으로 처리하기 위해 특화된 AI subagent를 사용하고 싶다고 가정해봅시다.

Step 1: 사용 가능한 subagent 보기

```
/agents
```

이것은 모든 사용 가능한 subagent를 표시하고 새로운 것을 만들 수 있게 해줍니다.

Step 2: 자동으로 subagent 사용하기

Claude Code는 자동으로 적절한 작업을 특화된 subagent에게 위임합니다:

```
review my recent code changes for security issues
```

```
run all tests and fix any failures
```

Step 3: 명시적으로 특정 subagent 요청하기

```
use the code-reviewer subagent to check the auth module
```

```
have the debugger subagent investigate why users can't log in
```

Step 4: 워크플로우를 위한 사용자 정의 subagent 만들기

```
/agents
```

그런 다음 “Create New subagent”를 선택하고 프롬프트를 따라 다음을 정의합니다:

- subagent의 목적을 설명하는 고유 식별자 (예: `code-reviewer`, `api-designer`).
- Claude가 이 agent를 사용해야 할 때
- 액세스할 수 있는 도구
- agent의 역할과 동작을 설명하는 시스템 프롬프트

> **Tip:**
>
> 팁:
>
> - 팀 공유를 위해 `.claude/agents/` 에 프로젝트별 subagent 만들기
> - 자동 위임을 활성화하기 위해 설명적인 `description` 필드 사용하기
> - 각 subagent가 실제로 필요한 것으로 도구 액세스 제한하기
> - 자세한 예제는 subagent 문서를 확인하기

안전한 코드 분석을 위해 Plan Mode 사용하기

Plan Mode는 Claude에게 읽기 전용 작업으로 코드베이스를 분석하여 계획을 세우도록 지시하며, 코드베이스 탐색, 복잡한 변경 계획 또는 코드 안전한 검토에 완벽합니다. Plan Mode에서 Claude는 `AskUserQuestion` 을 사용하여 계획을 제안하기 전에 요구사항을 수집하고 목표를 명확히 합니다.

Plan Mode를 사용할 때

- **다단계 구현**: 기능이 많은 파일을 편집해야 할 때
- **코드 탐색**: 무엇이든 변경하기 전에 코드베이스를 철저히 조사하고 싶을 때
- **대화형 개발**: Claude와 방향을 반복하고 싶을 때

Plan Mode 사용 방법

세션 중에 Plan Mode 켜기

Shift+Tab을 사용하여 세션 중에 Plan Mode로 전환할 수 있습니다.

Normal Mode에 있으면 **Shift+Tab**은 먼저 Auto-Accept Mode로 전환되며, 터미널 하단에 `▶▶ accept edits on`으로 표시됩니다. 그 다음 **Shift+Tab**은 Plan Mode로 전환되며, `⏸ plan mode on`으로 표시됩니다.

Plan Mode에서 새 세션 시작하기

Plan Mode에서 새 세션을 시작하려면 `--permission-mode plan` 플래그를 사용합니다:

```
claude --permission-mode plan
```

Plan Mode에서 "headless" 쿼리 실행하기

`-p`를 사용하여 Plan Mode에서 직접 쿼리를 실행할 수도 있습니다 (즉, "headless mode"에서):

```
claude --permission-mode plan -p "Analyze the authentication system and suggest improvements"
```

예제: 복잡한 리팩토링 계획하기

```
claude --permission-mode plan
```

```
I need to refactor our authentication system to use OAuth2. Create a detailed migration plan.
```

Claude는 현재 구현을 분석하고 포괄적인 계획을 만듭니다. 후속 질문으로 정제합니다:

```
What about backward compatibility?
```

```
How should we handle database migration?
```

> **Tip:**
>
> 계획을 기본 텍스트 편집기에서 열고 Claude가 진행하기 전에 직접 편집하려면 `Ctrl+G`를 누르십시오.

Plan Mode를 기본값으로 구성하기

```
// .claude/settings.json
{
  "permissions": {
    "defaultMode": "plan"
  }
}
```

더 많은 구성 옵션은 설정 문서를 참조하십시오.

테스트 작업하기

테스트되지 않은 코드에 대한 테스트를 추가해야 한다고 가정해봅시다.

Step 1: 테스트되지 않은 코드 식별

```
find functions in NotificationsService.swift that are not covered by tests
```

Step 2: 테스트 스캐폴딩 생성

```
add tests for the notification service
```

Step 3: 의미 있는 테스트 케이스 추가

```
add test cases for edge conditions in the notification service
```

Step 4: 테스트 실행 및 검증

```
run the new tests and fix any failures
```

Claude는 프로젝트의 기존 패턴과 규칙을 따르는 테스트를 생성할 수 있습니다. 테스트를 요청할 때 검증하고 싶은 동작에 대해 구체적으로 설명하십시오. Claude는 기존 테스트 파일을 검토하여 이미 사용 중인 스타일, 프레임워크 및 어설션 패턴을 일치시킵니다.

포괄적인 커버리지를 위해 Claude에게 놓쳤을 수 있는 엣지 케이스를 식별하도록 요청하십시오. Claude는 코드 경로를 분석하고 오류 조건, 경계값 및 쉽게 간과할 수 있는 예상치 못한 입력에 대한 테스트를 제안할 수 있습니다.

풀 요청 만들기

Claude에게 직접 풀 요청을 만들도록 요청하거나 (“create a pr for my changes”), 단계별로 Claude를 안내할 수 있습니다:

Step 1: 변경사항 요약하기

```
summarize the changes I've made to the authentication module
```

Step 2: 풀 요청 생성하기

```
create a pr
```

Step 3: 검토 및 정제하기

```
enhance the PR description with more context about the security improvements
```

`gh pr create`를 사용하여 PR을 만들면 세션이 자동으로 해당 PR에 연결됩니다. 나중에 `claude --from-pr <number>`로 재개할 수 있습니다.

> **Tip:**
>
> Claude가 생성한 PR을 제출하기 전에 검토하고 Claude에게 잠재적 위험이나 고려사항을 강조하도록 요청하십시오.

문서 처리하기

코드에 대한 문서를 추가하거나 업데이트해야 한다고 가정해봅시다.

Step 1: 문서화되지 않은 코드 식별

```
find functions without proper JSDoc comments in the auth module
```

Step 2: 문서 생성하기

```
add JSDoc comments to the undocumented functions in auth.js
```

Step 3: 검토 및 개선하기

```
improve the generated documentation with more context and examples
```

Step 4: 문서 검증하기

```
check if the documentation follows our project standards
```

> **Tip:**
>
> 팁:
>
> - 원하는 문서 스타일 지정하기 (JSDoc, docstrings 등)
> - 문서에 예제 요청하기
> - 공개 API, 인터페이스 및 복잡한 로직에 대한 문서 요청하기

이미지 작업하기

코드베이스에서 이미지로 작업해야 하고 Claude의 이미지 콘텐츠 분석 도움을 원한다고 가정해 봅시다.

Step 1: 대화에 이미지 추가하기

다음 방법 중 하나를 사용할 수 있습니다:

1. Claude Code 창으로 이미지를 드래그 앤 드롭하기
2. 이미지를 복사하고 ctrl+v로 CLI에 붙여넣기 (cmd+v 사용하지 않기)
3. Claude에 이미지 경로 제공하기. 예: “Analyze this image: /path/to/your/image.png”

Step 2: Claude에게 이미지 분석 요청하기

```
What does this image show?
```

```
Describe the UI elements in this screenshot
```

```
Are there any problematic elements in this diagram?
```

Step 3: 컨텍스트를 위해 이미지 사용하기

```
Here's a screenshot of the error. What's causing it?
```

```
This is our current database schema. How should we modify it for the new feature?
```

Step 4: 시각적 콘텐츠에서 코드 제안 얻기

```
Generate CSS to match this design mockup
```

```
What HTML structure would recreate this component?
```

> **Tip:**
>
> 팁:
>
> - 텍스트 설명이 불명확하거나 번거로울 때 이미지 사용하기
> - 더 나은 컨텍스트를 위해 오류, UI 디자인 또는 다이어그램의 스크린샷 포함하기
> - 대화에서 여러 이미지로 작업할 수 있습니다
> - 이미지 분석은 다이어그램, 스크린샷, 목업 등과 함께 작동합니다
> - Claude가 이미지를 참조할 때 (예: `[Image #1]`), `Cmd+Click` (Mac) 또는 `Ctrl+Click` (Windows/Linux)을 링크에 클릭하여 기본 뷰어에서 이미지를 엽니다

파일 및 디렉토리 참조하기

@를 사용하여 Claude가 읽을 때까지 기다리지 않고 파일이나 디렉토리를 빠르게 포함합니다.

Step 1: 단일 파일 참조하기

```
Explain the logic in @src/utils/auth.js
```

이것은 대화에 파일의 전체 내용을 포함합니다.

Step 2: 디렉토리 참조하기

```
What's the structure of @src/components?
```

이것은 파일 정보가 있는 디렉토리 목록을 제공합니다.

Step 3: MCP 리소스 참조하기

```
Show me the data from @github:repos/owner/repo/issues
```

이것은 @server:resource 형식을 사용하여 연결된 MCP 서버에서 데이터를 가져옵니다. 자세한 내용은 MCP 리소스를 참조하십시오.

> **Tip:**
>
> 팁:
>
> - 파일 경로는 상대 또는 절대 경로일 수 있습니다
> - @ 파일 참조는 파일의 디렉토리 및 상위 디렉토리에 `CLAUDE.md` 를 추가합니다
> - 디렉토리 참조는 내용이 아닌 파일 목록을 표시합니다
> - 단일 메시지에서 여러 파일을 참조할 수 있습니다 (예: “@file1.js and @file2.js”)

확장된 사고 사용하기 (thinking mode)

확장된 사고는 기본적으로 활성화되어 있으며, Claude가 복잡한 문제를 단계별로 추론할 수 있는 공간을 제공합니다. 이 추론은 `Ctrl+O` 로 전환할 수 있는 자세한 모드에서 볼 수 있습니다.

또한 Opus 4.6은 적응형 추론을 도입합니다: 고정된 사고 토큰 예산 대신 모델은 노력 수준 설정에 따라 동적으로 사고를 할당합니다. 확장된 사고와 적응형 추론은 함께 작동하여 Claude가 응답하기 전에 얼마나 깊이 있게 추론할지에 대한 제어를 제공합니다.

확장된 사고는 복잡한 아키텍처 결정, 어려운 버그, 다단계 구현 계획 및 다양한 접근 방식 간의 트레이드오프 평가에 특히 유용합니다.

> **Note:**
>
> “think”, “think hard”, “think more”와 같은 구문은 일반 프롬프트 지시로 해석되며 사고 토큰을 할당하지 않습니다.

thinking mode 구성하기

사고는 기본적으로 활성화되어 있지만 조정하거나 비활성화할 수 있습니다.

범위	구성 방법	세부사항
노력 수준	`/model` 에서 조정하거나 `CLAUDE_CODE_EFFORT_LEVEL` 설정	Opus 4.6 및 Sonnet 4.6에 대한 사고 깊이 제어: low, medium, high. 노력 수준 조정 참조
`ultrathink` **키워드**	프롬프트의 어디든 "ultrathink" 포함	Opus 4.6 및 Sonnet 4.6에서 해당 턴에 대해 노력을 높음으로 설정합니다. 노력 설정을 영구적으로 변경하지 않고 깊은 추론이 필요한 일회성 작업에 유용합니다
토글 단축키	`Option+T` (macOS) 또는 `Alt+T` (Windows/Linux) 누르기	현재 세션에 대해 사고 켜기/끄기 (모든 모델). 터미널 구성이 필요할 수 있습니다
전역 기본값	`/config` 를 사용하여 thinking mode 토글	모든 프로젝트에서 기본값 설정 (모든 모델). `~/.claude/settings.json` 에 `alwaysThinkingEnabled` 로 저장됩니다
토큰 예산 제한	`MAX_THINKING_TOKENS` 환경 변수 설정	사고 예산을 특정 토큰 수로 제한 (Opus 4.6에서는 0으로 설정하지 않으면 무시됨). 예: `export MAX_THINKING_TOKENS=10000`

Claude의 사고 과정을 보려면 `Ctrl+O` 를 눌러 자세한 모드를 전환하고 회색 이탤릭 텍스트로 표시된 내부 추론을 확인하십시오.

확장된 사고 작동 방식

확장된 사고는 Claude가 응답하기 전에 수행하는 내부 추론의 양을 제어합니다. 더 많은 사고는 솔루션을 탐색하고, 엣지 케이스를 분석하고, 실수를 자체 수정할 수 있는 더 많은 공간을 제공합니다.

Opus 4.6의 경우, 사고는 적응형 추론을 사용합니다: 모델은 선택한 노력 수준 (low, medium, high)에 따라 동적으로 사고 토큰을 할당합니다. 이것은 속도와 추론 깊이 간의 트레이드오프를 조정하는 권장 방법입니다.

다른 모델의 경우, 사고는 출력 예산에서 최대 31,999개 토큰의 고정 예산을 사용합니다. `MAX_THINKING_TOKENS` 환경 변수로 이를 제한하거나 `/config` 또는 `Option+T` / `Alt+T` 토글을 통해 사고를 완전히 비활성화할 수 있습니다.

`MAX_THINKING_TOKENS` 는 Opus 4.6 및 Sonnet 4.6에서 무시되며, 적응형 추론이 사고 깊이를 제어하기 때문입니다. 한 가지 예외: `MAX_THINKING_TOKENS=0` 을 설정하면 여전히 모든 모델에서 사고를 완전히 비활성화합니다. 적응형 사고를 비활성화하고 고정 사고 예산으로 되돌리려면 `CLAUDE_CODE_DISABLE_ADAPTIVE_THINKING=1` 을 설정하십시오. 환경 변수를 참조하십시오.

> **Warning:**
> Claude 4 모델이 요약된 사고를 표시하더라도 사용된 모든 사고 토큰에 대해 청구됩니다

이전 대화 재개하기

Claude Code를 시작할 때 이전 세션을 재개할 수 있습니다:

- `claude --continue` 는 현재 디렉토리에서 가장 최근 대화를 계속합니다
- `claude --resume` 은 대화 선택기를 열거나 이름으로 재개합니다
- `claude --from-pr 123` 은 특정 풀 요청에 연결된 세션을 재개합니다

활성 세션 내에서 `/resume` 을 사용하여 다른 대화로 전환합니다.

세션은 프로젝트 디렉토리별로 저장됩니다. `/resume` 선택기는 worktree를 포함한 동일한 git 저장소의 세션을 표시합니다.

세션 이름 지정하기

나중에 찾기 위해 세션에 설명적인 이름을 지정합니다. 이것은 여러 작업이나 기능을 작업할 때 모범 사례입니다.

Step 1: 현재 세션 이름 지정하기

세션 중에 `/rename` 을 사용하여 기억하기 쉬운 이름을 지정합니다:

```
/rename auth-refactor
```

선택기에서 세션 이름을 바꿀 수도 있습니다: `/resume`을 실행하고 세션으로 이동한 다음 `R`을 누릅니다.

Step 2: 나중에 이름으로 재개하기

명령줄에서:

```
claude --resume auth-refactor
```

또는 활성 세션 내에서:

```
/resume auth-refactor
```

세션 선택기 사용하기

`/resume` 명령 (또는 인수 없이 `claude --resume`)은 다음 기능이 있는 대화형 세션 선택기를 엽니다:

선택기의 키보드 단축키:

단축키	작업
`↑` / `↓`	세션 간 이동
`→` / `←`	그룹화된 세션 확장 또는 축소
`Enter`	강조 표시된 세션 선택 및 재개
`P`	세션 콘텐츠 미리보기
`R`	강조 표시된 세션 이름 바꾸기
`/`	검색하여 세션 필터링
`A`	현재 디렉토리와 모든 프로젝트 간 전환
`B`	현재 git 분기의 세션으로 필터링
`Esc`	선택기 또는 검색 모드 종료

세션 구성:

선택기는 유용한 메타데이터가 있는 세션을 표시합니다:

- 세션 이름 또는 초기 프롬프트

- 마지막 활동 이후 경과 시간
- 메시지 수
- Git 분기 (해당하는 경우)

포크된 세션 (`/rewind` 또는 `--fork-session` 으로 생성됨)은 루트 세션 아래에 그룹화되어 관련 대화를 더 쉽게 찾을 수 있습니다.

Tip:

팁:

- **세션 조기 이름 지정**: 고유한 작업을 시작할 때 `/rename` 을 사용합니다—나중에 "payment-integration" 을 찾는 것이 "explain this function"보다 훨씬 쉽습니다
- 현재 디렉토리에서 가장 최근 대화에 빠르게 액세스하려면 `--continue` 사용
- 필요한 세션을 알 때 `--resume session-name` 사용
- 검색하고 선택해야 할 때 `--resume` (이름 없이) 사용
- 스크립트의 경우 `claude --continue --print "prompt"` 를 사용하여 비대화형 모드에서 재개
- 선택기에서 `P` 를 눌러 재개하기 전에 세션을 미리봅니다
- 재개된 대화는 원본과 동일한 모델 및 구성으로 시작됩니다

작동 방식:

1. **대화 저장소**: 모든 대화는 전체 메시지 기록과 함께 로컬에 자동으로 저장됩니다
2. **메시지 역직렬화**: 재개할 때 전체 메시지 기록이 복원되어 컨텍스트를 유지합니다
3. **도구 상태**: 이전 대화의 도구 사용 및 결과가 보존됩니다
4. **컨텍스트 복원**: 대화는 모든 이전 컨텍스트와 함께 재개됩니다

Git worktree를 사용하여 병렬 Claude Code 세션 실행하기

여러 작업을 동시에 수행할 때 각 Claude 세션이 변경사항이 충돌하지 않도록 코드베이스의 자체 복사본을 가져야 합니다. Git worktree는 각각 자체 파일과 분기를 가지면서 동일한 저장소 기록 및 원격 연결을 공유하는 별도의 작업 디렉토리를 만들어 이를 해결합니다. 이는 한 worktree에서 기능을 작업하는 동안 Claude가 다른 worktree에서 버그를 수정할 수 있으며 어느 세션도 다른 세션을 방해하지 않음을 의미합니다.

`--worktree` (`-w`) 플래그를 사용하여 격리된 worktree를 만들고 Claude를 시작합니다. 전달하는 값은 worktree 디렉토리 이름과 분기 이름이 됩니다:

```
## "feature-auth"라는 worktree에서 Claude 시작
## 새 분기를 사용하여 .claude/worktrees/feature-auth/ 생성
claude --worktree feature-auth

## 별도의 worktree에서 다른 세션 시작
claude --worktree bugfix-123
```

이름을 생략하면 Claude가 자동으로 임의의 이름을 생성합니다:

```
## "bright-running-fox"와 같은 이름 자동 생성
claude --worktree
```

Worktree는 `<repo>/.claude/worktrees/<name>`에 생성되고 기본 원격 분기에서 분기됩니다. worktree 분기는 `worktree-<name>`으로 이름이 지정됩니다.

세션 중에 Claude에게 "work in a worktree" 또는 "start a worktree"를 요청할 수도 있으며, 자동으로 하나를 만듭니다.

Subagent worktree

Subagent도 worktree 격리를 사용하여 충돌 없이 병렬로 작업할 수 있습니다. Claude에게 "use worktrees for your agents" 를 요청하거나 agent의 frontmatter에 `isolation: worktree` 를 추가하여 사용자 정의 subagent에서 구성합니다. 각 subagent는 변경사항 없이 완료되면 자동으로 정리되는 자체 worktree를 가집니다.

Worktree 정리

worktree 세션을 종료할 때 Claude는 변경사항이 있는지 여부에 따라 정리를 처리합니다:

- **변경사항 없음**: worktree 및 해당 분기가 자동으로 제거됩니다
- **변경사항 또는 커밋 존재**: Claude는 worktree를 유지할지 제거할지 묻습니다. 유지하면 디렉토리와 분기가 보존되어 나중에 돌아올 수 있습니다. 제거하면 worktree 디렉토리와 해당 분기가 삭제되어 모든 커밋되지 않은 변경사항과 커밋이 버려집니다

Claude 세션 외부에서 worktree를 정리하려면 수동 worktree 관리를 사용합니다.

> **Tip:**
>
> `.gitignore`에 `.claude/worktrees/`를 추가하여 worktree 콘텐츠가 주 저장소에 추적되지 않은 파일로 나타나지 않도록 합니다.

수동으로 worktree 관리하기

worktree 위치 및 분기 구성에 대한 더 많은 제어를 위해 Git을 사용하여 직접 worktree를 만듭니다. 특정 기존 분기를 체크아웃하거나 worktree를 저장소 외부에 배치해야 할 때 유용합니다.

```
## 새 분기를 사용하여 worktree 만들기
git worktree add ../project-feature-a -b feature-a

## 기존 분기를 사용하여 worktree 만들기
git worktree add ../project-bugfix bugfix-123

## worktree에서 Claude 시작
cd ../project-feature-a && claude

## 완료되면 정리
git worktree list
git worktree remove ../project-feature-a
```

공식 Git worktree 문서에서 자세히 알아봅니다.

> **Tip:**
>
> 프로젝트의 설정에 따라 각 새 worktree에서 개발 환경을 초기화해야 합니다. 스택에 따라 여기에는 종속성 설치 (`npm install` , `yarn`), 가상 환경 설정 또는 프로젝트의 표준 설정 프로세스 따르기가 포함될 수 있습니다.

Git이 아닌 버전 제어

Worktree 격리는 기본적으로 git과 함께 작동합니다. SVN, Perforce 또는 Mercurial과 같은 다른 버전 제어 시스템의 경우 WorktreeCreate 및 WorktreeRemove hook을 구성하여 사용자 정의 worktree 생성 및 정리 로직을 제공합니다. 구성되면 이러한 hook은 `--worktree` 를 사용할 때 기본 git 동작을 대체합니다.

공유 작업 및 메시징을 사용한 병렬 세션의 자동 조정을 위해 agent team을 참조하십시오.

Claude가 주의가 필요할 때 알림 받기

오래 실행되는 작업을 시작하고 다른 창으로 전환할 때 Claude가 완료되거나 입력이 필요할 때 알 수 있도록 데스크톱 알림을 설정할 수 있습니다. 이것은 Claude가 권한을 기다리거나, 유휴 상태이고 새 프롬프트를 기다리거나, 인증을 완료할 때마다 발생하는 `Notification` hook 이벤트를 사용합니다.

Step 1: hook 메뉴 열기

`/hooks`를 입력하고 이벤트 목록에서 `Notification`을 선택합니다.

Step 2: matcher 구성하기

모든 알림 유형에 대해 발생하도록 `+ Match all (no filter)`을 선택합니다. 특정 이벤트에만 알림을 받으려면 `+ Add new matcher…`를 선택하고 다음 값 중 하나를 입력합니다:

Matcher	발생 시기
`permission_prompt`	Claude가 도구 사용을 승인하도록 요청할 때
`idle_prompt`	Claude가 완료되고 다음 프롬프트를 기다릴 때
`auth_success`	인증이 완료될 때
`elicitation_dialog`	Claude가 질문을 할 때

Step 3: 알림 명령 추가하기

`+ Add new hook…`을 선택하고 OS에 대한 명령을 입력합니다:

macOS

AppleScript를 통해 네이티브 macOS 알림을 트리거하기 위해 `osascript`를 사용합니다:

```
osascript -e 'display notification "Claude Code needs your attention" with title "Claude Code"'
```

Linux

대부분의 Linux 데스크톱에 알림 데몬과 함께 사전 설치된 `notify-send`를 사용합니다:

```
notify-send 'Claude Code' 'Claude Code needs your attention'
```

Windows (PowerShell)

PowerShell을 사용하여 .NET의 Windows Forms를 통해 네이티브 메시지 상자를 표시합니다:

```
powershell.exe -Command
"[System.Reflection.Assembly]::LoadWithPartialName('System.Windows.Forms');
[System.Windows.Forms.MessageBox]::Show('Claude Code needs your attention',
'Claude Code')"
```

Step 4: 사용자 설정에 저장하기

`User settings`을 선택하여 모든 프로젝트에 알림을 적용합니다.

JSON 구성 예제가 있는 전체 연습은 hook으로 워크플로우 자동화하기를 참조하십시오. 전체 이벤트 스키마 및 알림 유형은 Notification 참조를 참조하십시오.

Claude를 unix 스타일 유틸리티로 사용하기

검증 프로세스에 Claude 추가하기

Claude Code를 linter 또는 코드 검토자로 사용하고 싶다고 가정해봅시다.

빌드 스크립트에 Claude 추가하기:

```
// package.json
{
    ...
    "scripts": {
        ...
        "lint:claude": "claude -p 'you are a linter. please look at the changes
vs. main and report any issues related to typos. report the filename and line
number on one line, and a description of the issue on the second line. do not
return any other text.'"
    }
}
```

> **Tip:**
>
> 팁:
>
> - CI/CD 파이프라인에서 자동 코드 검토를 위해 Claude 사용하기
> - 프롬프트를 사용자 정의하여 프로젝트와 관련된 특정 문제 확인하기

- 다양한 유형의 검증을 위해 여러 스크립트 만들기 고려하기

파이프 인, 파이프 아웃

Claude로 데이터를 파이프하고 구조화된 형식으로 데이터를 다시 받고 싶다고 가정해봅시다.

Claude를 통해 데이터 파이프하기:

```
cat build-error.txt | claude -p 'concisely explain the root cause of this build error' > output.txt
```

Tip:

팁:

- 기존 셸 스크립트에 Claude를 통합하기 위해 파이프 사용하기
- 강력한 워크플로우를 위해 다른 Unix 도구와 결합하기
- 구조화된 출력을 위해 –output-format 사용 고려하기

출력 형식 제어하기

특히 Claude Code를 스크립트나 다른 도구에 통합할 때 특정 형식의 Claude 출력이 필요하다고 가정해봅시다.

Step 1: 텍스트 형식 사용 (기본값)

```
cat data.txt | claude -p 'summarize this data' --output-format text > summary.txt
```

이것은 Claude의 일반 텍스트 응답만 출력합니다 (기본 동작).

Step 2: JSON 형식 사용

```
cat code.py | claude -p 'analyze this code for bugs' --output-format json > analysis.json
```

이것은 비용 및 기간을 포함한 메타데이터가 있는 메시지의 JSON 배열을 출력합니다.

Step 3: 스트리밍 JSON 형식 사용

```
cat log.txt | claude -p 'parse this log file for errors' --output-format stream-
json
```

이것은 Claude가 요청을 처리할 때 실시간으로 일련의 JSON 객체를 출력합니다. 각 메시지는 유효한 JSON 객체이지만 연결된 전체 출력은 유효한 JSON이 아닙니다.

> **Tip:**
>
> 팁:
>
> - Claude의 응답만 필요한 간단한 통합을 위해 `--output-format text` 사용하기
> - 전체 대화 로그가 필요할 때 `--output-format json` 사용하기
> - 각 대화 턴의 실시간 출력을 위해 `--output-format stream-json` 사용하기

Claude의 기능에 대해 Claude에게 물어보기

Claude는 자신의 문서에 대한 기본 제공 액세스 권한을 가지고 있으며 자신의 기능과 제한사항에 대한 질문에 답할 수 있습니다.

예제 질문

```
can Claude Code create pull requests?
```

```
how does Claude Code handle permissions?
```

```
what skills are available?
```

```
how do I use MCP with Claude Code?
```

```
how do I configure Claude Code for Amazon Bedrock?
```

```
what are the limitations of Claude Code?
```

Note:

Claude는 이러한 질문에 대해 문서 기반 답변을 제공합니다. 실행 가능한 예제 및 실습 시연을 위해 위의 특정 워크플로우 섹션을 참조하십시오.

Tip:

팁:

- Claude는 사용 중인 버전에 관계없이 항상 최신 Claude Code 문서에 액세스할 수 있습니다
- 자세한 답변을 얻으려면 구체적인 질문하기
- Claude는 MCP 통합, 엔터프라이즈 구성 및 고급 워크플로우와 같은 복잡한 기능을 설명할 수 있습니다

다음 단계

- 모범 사례 Claude Code에서 최대한 활용하기 위한 패턴
- Claude Code 작동 방식 agentic 루프 및 컨텍스트 관리 이해하기
- Claude Code 확장하기 skill, hook, MCP, subagent 및 플러그인 추가하기
- 참조 구현 개발 컨테이너 참조 구현 복제하기

Part 3: Commands & Reference

CLI 참조

> Claude Code 명령줄 인터페이스의 완전한 참조로, 명령어와 플래그를 포함합니다.

CLI 명령어

이러한 명령어를 사용하여 세션을 시작하고, 콘텐츠를 파이프하고, 대화를 재개하고, 업데이트를 관리할 수 있습니다:

명령어	설명	예시
`claude`	대화형 세션 시작	`claude`
`claude "query"`	초기 프롬프트로 대화형 세션 시작	`claude "explain this project"`
`claude -p "query"`	SDK를 통해 쿼리하고 종료	`claude -p "explain this function"`
`cat file | claude -p "query"`	파이프된 콘텐츠 처리	`cat logs.txt | claude -p "explain"`
`claude -c`	현재 디렉토리에서 가장 최근 대화 계속	`claude -c`
`claude -c -p "query"`	SDK를 통해 계속	`claude -c -p "Check for type errors"`
`claude -r "<session>" "query"`	ID 또는 이름으로 세션 재개	`claude -r "auth-refactor" "Finish this PR"`
`claude update`	최신 버전으로 업데이트	`claude update`
`claude auth login`	Anthropic 계정에 로그인합니다. `--email` 을 사용하여 이메일 주소를 미리 입력하고 `--sso` 를 사용하여 SSO 인증을 강제할 수 있습니다	`claude auth login --email user@example.com --sso`
`claude auth logout`	Anthropic 계정에서 로그아웃합니다	`claude auth logout`

명령어	설명	예시
`claude auth status`	인증 상태를 JSON으로 표시합니다. 사람이 읽을 수 있는 출력을 위해 `--text`를 사용합니다. 로그인되어 있으면 코드 0으로, 로그인되어 있지 않으면 1로 종료됩니다	`claude auth status`
`claude agents`	모든 구성된 subagents를 소스별로 그룹화하여 나열합니다	`claude agents`
`claude mcp`	Model Context Protocol (MCP) 서버 구성	Claude Code MCP 문서를 참조하세요.
`claude remote-control`	로컬에서 실행하는 동안 Claude.ai 또는 Claude 앱에서 Claude Code를 제어하기 위한 Remote Control 세션을 시작합니다. 플래그는 Remote Control을 참조하세요	`claude remote-control`

CLI 플래그

이러한 명령줄 플래그를 사용하여 Claude Code의 동작을 사용자 정의합니다:

플래그	설명	예시
`--add-dir`	Claude가 액세스할 추가 작업 디렉토리 추가(각 경로가 디렉토리로 존재하는지 검증)	`claude --add-dir ../apps ../lib`
`--agent`	현재 세션에 대한 에이전트 지정 (`agent` 설정 재정의)	`claude --agent my-custom-agent`
`--agents`	JSON을 통해 사용자 정의 subagents를 동적으로 정의(형식은 아래 참조)	`claude --agents '{"reviewer":{"description":"Reviews code","prompt":"You are a code reviewer"}}'`

플래그	설명	예시
`--allow-dangerously-skip-permissions`	권한 우회를 옵션으로 활성화하되 즉시 활성화하지 않습니다. `--permission-mode`와 함께 구성 가능(주의하여 사용)	`claude --permission-mode plan --allow-dangerously-skip-permissions`
`--allowedTools`	권한 프롬프트 없이 실행되는 도구입니다. 패턴 매칭을 위해 권한 규칙 구문을 참조하세요. 사용 가능한 도구를 제한하려면 `--tools`를 대신 사용하세요	`"Bash(git log *)" "Bash(git diff *)" "Read"`
`--append-system-prompt`	기본 시스템 프롬프트의 끝에 사용자 정의 텍스트 추가	`claude --append-system-prompt "Always use TypeScript"`
`--append-system-prompt-file`	파일에서 추가 시스템 프롬프트 텍스트를 로드하고 기본 프롬프트에 추가	`claude --append-system-prompt-file ./extra-rules.txt`
`--betas`	API 요청에 포함할 베타 헤더(API 키 사용자만 해당)	`claude --betas interleaved-thinking`
`--chrome`	웹 자동화 및 테스트를 위한 Chrome 브라우저 통합 활성화	`claude --chrome`
`--continue`, `-c`	현재 디렉토리에서 가장 최근 대화 로드	`claude --continue`
`--dangerously-skip-permissions`	모든 권한 프롬프트 건너뛰기(주의하여 사용)	`claude --dangerously-skip-permissions`
`--debug`	선택적 카테고리 필터링을 사용하여 디버그 모드 활성화(예: `"api,hooks"` 또는 `"!statsig,!file"`)	`claude --debug "api,mcp"`
`--disable-slash-commands`	이 세션에 대한 모든 skills 및 명령어 비활성화	`claude --disable-slash-commands`
`--disallowedTools`	모델의 컨텍스트에서 제거되고 사용할 수 없는 도구	`"Bash(git log *)" "Bash(git diff *)" "Edit"`
`--fallback-model`	기본 모델이 과부하일 때 지정된 모델로 자동 폴백 활성화(인쇄 모드만 해당)	`claude -p --fallback-model sonnet "query"`

플래그	설명	예시
`--fork-session`	재개할 때 원본을 재사용하는 대신 새 세션 ID 생성(`--resume` 또는 `--continue` 와 함께 사용)	`claude --resume abc123 --fork-session`
`--from-pr`	특정 GitHub PR에 연결된 세션 재개. PR 번호 또는 URL을 허용합니다. `gh pr create` 를 통해 생성된 세션은 자동으로 연결됩니다	`claude --from-pr 123`
`--ide`	정확히 하나의 유효한 IDE를 사용할 수 있는 경우 시작 시 IDE에 자동으로 연결	`claude --ide`
`--init`	초기화 hooks를 실행하고 대화형 모드 시작	`claude --init`
`--init-only`	초기화 hooks를 실행하고 종료(대화형 세션 없음)	`claude --init-only`
`--include-partial-messages`	부분 스트리밍 이벤트를 출력에 포함(`--print` 와 `--output-format=stream-json` 필요)	`claude -p --output-format stream-json --include-partial-messages "query"`
`--input-format`	인쇄 모드의 입력 형식 지정(옵션: `text` , `stream-json`)	`claude -p --output-format json --input-format stream-json`
`--json-schema`	에이전트가 워크플로우를 완료한 후 JSON Schema와 일치하는 검증된 JSON 출력 가져오기(인쇄 모드만 해당, 구조화된 출력 참조)	`claude -p --json-schema '{"type":"object","properties":{...}}' "query"`
`--maintenance`	유지 관리 hooks를 실행하고 종료	`claude --maintenance`
`--max-budget-usd`	중지하기 전에 API 호출에 소비할 최대 달러 금액(인쇄 모드만 해당)	`claude -p --max-budget-usd 5.00 "query"`
`--max-turns`	에이전트 턴의 수 제한(인쇄 모드만 해당). 제한에 도달하면 오류로 종료됩니다. 기본적으로 제한 없음	`claude -p --max-turns 3 "query"`

플래그	설명	예시
`--mcp-config`	JSON 파일 또는 문자열에서 MCP 서버 로드(공백으로 구분)	`claude --mcp-config ./mcp.json`
`--model`	최신 모델의 별칭(`sonnet` 또는 `opus`) 또는 모델의 전체 이름으로 현재 세션의 모델 설정	`claude --model claude-sonnet-4-6`
`--no-chrome`	이 세션에 대한 Chrome 브라우저 통합 비활성화	`claude --no-chrome`
`--no-session-persistence`	세션 지속성 비활성화하여 세션이 디스크에 저장되지 않고 재개할 수 없음(인쇄 모드만 해당)	`claude -p --no-session-persistence "query"`
`--output-format`	인쇄 모드의 출력 형식 지정(옵션: `text` , `json` , `stream-json`)	`claude -p "query" --output-format json`
`--permission-mode`	지정된 권한 모드에서 시작	`claude --permission-mode plan`
`--permission-prompt-tool`	비대화형 모드에서 권한 프롬프트를 처리할 MCP 도구 지정	`claude -p --permission-prompt-tool mcp_auth_tool "query"`
`--plugin-dir`	이 세션에만 플러그인 디렉토리에서 플러그인 로드(반복 가능)	`claude --plugin-dir ./my-plugins`
`--print` , `-p`	대화형 모드 없이 응답 인쇄(Agent SDK 문서에서 프로그래밍 방식 사용 세부 정보 참조)	`claude -p "query"`
`--remote`	제공된 작업 설명으로 claude.ai에서 새 웹 세션 생성	`claude --remote "Fix the login bug"`
`--resume` , `-r`	ID 또는 이름으로 특정 세션 재개하거나 세션을 선택할 대화형 선택기 표시	`claude --resume auth-refactor`
`--session-id`	대화에 특정 세션 ID 사용(유효한 UUID 여야 함)	`claude --session-id "550e8400-e29b-41d4-a716-446655440000"`

플래그	설명	예시
`--setting-sources`	로드할 설정 소스의 쉼표로 구분된 목록 (`user`, `project`, `local`)	`claude --setting-sources user,project`
`--settings`	추가 설정을 로드할 설정 JSON 파일 또는 JSON 문자열의 경로	`claude --settings ./settings.json`
`--strict-mcp-config`	`--mcp-config`의 MCP 서버만 사용하고 다른 모든 MCP 구성 무시	`claude --strict-mcp-config --mcp-config ./mcp.json`
`--system-prompt`	전체 시스템 프롬프트를 사용자 정의 텍스트로 바꾸기	`claude --system-prompt "You are a Python expert"`
`--system-prompt-file`	파일에서 시스템 프롬프트를 로드하여 기본 프롬프트 바꾸기	`claude --system-prompt-file ./custom-prompt.txt`
`--teleport`	로컬 터미널에서 웹 세션 재개	`claude --teleport`
`--teammate-mode`	에이전트 팀 팀원 표시 방식 설정: `auto`(기본값), `in-process`, 또는 `tmux`. 에이전트 팀 설정 참조	`claude --teammate-mode in-process`
`--tools`	Claude가 사용할 수 있는 기본 제공 도구 제한. `""`를 사용하여 모두 비활성화, `"default"`를 사용하여 모두 활성화, 또는 `"Bash,Edit,Read"`와 같은 도구 이름 사용	`claude --tools "Bash,Edit,Read"`
`--verbose`	자세한 로깅 활성화, 전체 턴별 출력 표시	`claude --verbose`
`--version`, `-v`	버전 번호 출력	`claude -v`
`--worktree`, `-w`	`<repo>/.claude/worktrees/<name>`에서 격리된 git worktree에서 Claude 시작. 이름이 지정되지 않으면 자동으로 생성됨	`claude -w feature-auth`

Tip:

`--output-format json` 플래그는 스크립팅 및 자동화에 특히 유용하며, Claude의 응답을 프로그래밍 방식으로 구문 분석할 수 있습니다.

Agents 플래그 형식

`--agents` 플래그는 하나 이상의 사용자 정의 subagents를 정의하는 JSON 객체를 허용합니다. 각 subagent는 고유한 이름(키로 사용)과 다음 필드를 포함하는 정의 객체가 필요합니다:

필드	필수	설명
`description`	예	subagent를 호출해야 할 때를 설명하는 자연어 설명
`prompt`	예	subagent의 동작을 안내하는 시스템 프롬프트
`tools`	아니오	subagent가 사용할 수 있는 특정 도구의 배열(예: `["Read", "Edit", "Bash"]`). 생략하면 모든 도구를 상속합니다. `Agent(agent_type)` 구문 지원
`disallowedTools`	아니오	이 subagent에 대해 명시적으로 거부할 도구 이름의 배열
`model`	아니오	사용할 모델 별칭: `sonnet`, `opus`, `haiku`, 또는 `inherit`. 생략하면 `inherit`로 기본 설정됨
`skills`	아니오	subagent의 컨텍스트에 미리 로드할 skill 이름의 배열
`mcpServers`	아니오	이 subagent에 대한 MCP servers의 배열. 각 항목은 서버 이름 문자열 또는 `{name: config}` 객체
`maxTurns`	아니오	subagent가 중지되기 전의 최대 에이전트 턴 수

예시:

```
claude --agents '{
  "code-reviewer": {
    "description": "Expert code reviewer. Use proactively after code changes.",
    "prompt": "You are a senior code reviewer. Focus on code quality, security,
and best practices.",
    "tools": ["Read", "Grep", "Glob", "Bash"],
    "model": "sonnet"
  },
  "debugger": {
    "description": "Debugging specialist for errors and test failures.",
    "prompt": "You are an expert debugger. Analyze errors, identify root causes,
and provide fixes."
  }
}'
```

subagents 생성 및 사용에 대한 자세한 내용은 subagents 문서를 참조하세요.

시스템 프롬프트 플래그

Claude Code는 시스템 프롬프트를 사용자 정의하기 위한 4가지 플래그를 제공합니다. 4가지 모두 대화형 및 비대화형 모드에서 작동합니다.

플래그	동작	사용 사례
`--system-prompt`	**전체** 기본 프롬프트 **바꾸기**	Claude의 동작 및 지침에 대한 완전한 제어
`--system-prompt-file`	**파일 내용으로 바꾸기**	재현성 및 버전 제어를 위해 파일에서 프롬프트 로드
`--append-system-prompt`	**기본 프롬프트에 추가**	기본 Claude Code 동작을 유지하면서 특정 지침 추가
`--append-system-prompt-file`	**기본 프롬프트에 파일 내용 추가**	기본값을 유지하면서 파일에서 추가 지침 로드

각각을 사용할 때:

- `--system-prompt` : Claude의 시스템 프롬프트를 완전히 제어해야 할 때 사용합니다. 이는 모든 기본 Claude Code 지침을 제거하여 백지 상태를 제공합니다.

```
claude --system-prompt "You are a Python expert who only writes type-annotated code"
```

- `--system-prompt-file` : 파일에서 사용자 정의 프롬프트를 로드하려고 할 때 사용합니다. 팀 일관성 또는 버전 제어 프롬프트 템플릿에 유용합니다.

```
claude --system-prompt-file ./prompts/code-review.txt
```

- `--append-system-prompt` : Claude Code의 기본 기능을 유지하면서 특정 지침을 추가하려고 할 때 사용합니다. 대부분의 사용 사례에서 가장 안전한 옵션입니다.

```
claude --append-system-prompt "Always use TypeScript and include JSDoc comments"
```

- `--append-system-prompt-file` : Claude Code의 기본값을 유지하면서 파일에서 지침을 추가하려고 할 때 사용합니다. 버전 제어 추가에 유용합니다.

```
claude --append-system-prompt-file ./prompts/style-rules.txt
```

`--system-prompt`와 `--system-prompt-file`은 상호 배타적입니다. 추가 플래그는 바꾸기 플래그 중 하나와 함께 사용할 수 있습니다.

대부분의 사용 사례에서 `--append-system-prompt` 또는 `--append-system-prompt-file`을 권장합니다. 이들은 Claude Code의 기본 제공 기능을 유지하면서 사용자 정의 요구 사항을 추가합니다. 시스템 프롬프트를 완전히 제어해야 할 때만 `--system-prompt` 또는 `--system-prompt-file`을 사용하세요.

참고 항목

- Chrome 확장 프로그램 - 브라우저 자동화 및 웹 테스트
- 대화형 모드 - 단축키, 입력 모드 및 대화형 기능
- 빠른 시작 가이드 - Claude Code 시작하기
- 일반적인 워크플로우 - 고급 워크플로우 및 패턴
- 설정 - 구성 옵션
- Agent SDK 문서 - 프로그래밍 방식 사용 및 통합

키보드 단축키 사용자 정의

> keybindings 구성 파일을 사용하여 Claude Code에서 키보드 단축키를 사용자 정의합니다.

Note:

사용자 정의 가능한 키보드 단축키는 Claude Code v2.1.18 이상이 필요합니다. `claude --version` 으로 버전을 확인하세요.

Claude Code는 사용자 정의 가능한 키보드 단축키를 지원합니다. `/keybindings` 를 실행하여 `~/.claude/keybindings.json` 에서 구성 파일을 만들거나 열 수 있습니다.

구성 파일

keybindings 구성 파일은 `bindings` 배열이 있는 객체입니다. 각 블록은 컨텍스트와 키 입력을 작업에 매핑하는 맵을 지정합니다.

Note:

keybindings 파일의 변경 사항은 자동으로 감지되고 Claude Code를 다시 시작하지 않고도 적용됩니다.

필드	설명
`$schema`	편집기 자동 완성을 위한 선택적 JSON Schema URL
`$docs`	선택적 설명서 URL
`bindings`	컨텍스트별 바인딩 블록 배열

이 예제는 채팅 컨텍스트에서 `Ctrl+E` 를 외부 편집기를 열기에 바인딩하고 `Ctrl+U` 를 바인딩 해제합니다:

```
{
  "$schema": "https://www.schemastore.org/claude-code-keybindings.json",
  "$docs": "https://code.claude.com/docs/ko/keybindings",
  "bindings": [
    {
      "context": "Chat",
      "bindings": {
        "ctrl+e": "chat:externalEditor",
        "ctrl+u": null
      }
    }
  ]
}
```

컨텍스트

각 바인딩 블록은 바인딩이 적용되는 **컨텍스트**를 지정합니다:

컨텍스트	설명
`Global`	앱의 모든 곳에 적용됨
`Chat`	주 채팅 입력 영역
`Autocomplete`	자동 완성 메뉴가 열려 있음
`Settings`	설정 메뉴(Escape로만 닫기)
`Confirmation`	권한 및 확인 대화 상자
`Tabs`	탭 네비게이션 구성 요소
`Help`	도움말 메뉴가 표시됨
`Transcript`	트랜스크립트 뷰어
`HistorySearch`	기록 검색 모드(Ctrl+R)
`Task`	백그라운드 작업이 실행 중
`ThemePicker`	테마 선택기 대화 상자
`Attachments`	이미지/첨부 파일 바 네비게이션

컨텍스트	설명
`Footer`	바닥글 표시기 네비게이션(작업, 팀, diff)
`MessageSelector`	되돌리기 및 요약 대화 상자 메시지 선택
`DiffDialog`	Diff 뷰어 네비게이션
`ModelPicker`	모델 선택기 노력 수준
`Select`	일반 선택/목록 구성 요소
`Plugin`	플러그인 대화 상자(찾아보기, 발견, 관리)

사용 가능한 작업

작업은 `namespace:action` 형식을 따릅니다. 예를 들어 `chat:submit` 은 메시지를 보내고 `app:toggleTodos` 는 작업 목록을 표시합니다. 각 컨텍스트에는 사용 가능한 특정 작업이 있습니다.

앱 작업

`Global` 컨텍스트에서 사용 가능한 작업:

작업	기본값	설명
`app:interrupt`	Ctrl+C	현재 작업 취소
`app:exit`	Ctrl+D	Claude Code 종료
`app:toggleTodos`	Ctrl+T	작업 목록 표시 여부 전환
`app:toggleTranscript`	Ctrl+O	상세 트랜스크립트 전환

기록 작업

명령 기록을 탐색하기 위한 작업:

작업	기본값	설명
`history:search`	Ctrl+R	기록 검색 열기
`history:previous`	Up	이전 기록 항목
`history:next`	Down	다음 기록 항목

채팅 작업

`Chat` 컨텍스트에서 사용 가능한 작업:

작업	기본값	설명
`chat:cancel`	Escape	현재 입력 취소
`chat:cycleMode`	Shift+Tab*	권한 모드 순환
`chat:modelPicker`	Cmd+P / Meta+P	모델 선택기 열기
`chat:thinkingToggle`	Cmd+T / Meta+T	확장 사고 전환
`chat:submit`	Enter	메시지 제출
`chat:undo`	Ctrl+_	마지막 작업 실행 취소
`chat:externalEditor`	Ctrl+G	외부 편집기에서 열기
`chat:stash`	Ctrl+S	현재 프롬프트 숨기기
`chat:imagePaste`	Ctrl+V (Windows에서 Alt+V)	이미지 붙여넣기

*VT 모드가 없는 Windows에서(Node <24.2.0/<22.17.0, Bun <1.2.23), 기본값은 Meta+M입니다.

자동 완성 작업

`Autocomplete` 컨텍스트에서 사용 가능한 작업:

작업	기본값	설명
`autocomplete:accept`	Tab	제안 수락
`autocomplete:dismiss`	Escape	메뉴 닫기
`autocomplete:previous`	Up	이전 제안
`autocomplete:next`	Down	다음 제안

확인 작업

`Confirmation` 컨텍스트에서 사용 가능한 작업:

작업	기본값	설명
`confirm:yes`	Y, Enter	작업 확인

작업	기본값	설명
`confirm:no`	N, Escape	작업 거부
`confirm:previous`	Up	이전 옵션
`confirm:next`	Down	다음 옵션
`confirm:nextField`	Tab	다음 필드
`confirm:previousField`	(바인딩 해제됨)	이전 필드
`confirm:cycleMode`	Shift+Tab	권한 모드 순환
`confirm:toggleExplanation`	Ctrl+E	권한 설명 전환

권한 작업

권한 대화 상자의 `Confirmation` 컨텍스트에서 사용 가능한 작업:

작업	기본값	설명
`permission:toggleDebug`	Ctrl+D	권한 디버그 정보 전환

트랜스크립트 작업

`Transcript` 컨텍스트에서 사용 가능한 작업:

작업	기본값	설명
`transcript:toggleShowAll`	Ctrl+E	모든 콘텐츠 표시 전환
`transcript:exit`	Ctrl+C, Escape	트랜스크립트 보기 종료

기록 검색 작업

`HistorySearch` 컨텍스트에서 사용 가능한 작업:

작업	기본값	설명
`historySearch:next`	Ctrl+R	다음 일치 항목
`historySearch:accept`	Escape, Tab	선택 수락
`historySearch:cancel`	Ctrl+C	검색 취소
`historySearch:execute`	Enter	선택한 명령 실행

작업 작업

`Task` 컨텍스트에서 사용 가능한 작업:

작업	기본값	설명
`task:background`	Ctrl+B	현재 작업을 백그라운드로 이동

테마 작업

`ThemePicker` 컨텍스트에서 사용 가능한 작업:

작업	기본값	설명
`theme:toggleSyntaxHighlighting`	Ctrl+T	구문 강조 전환

도움말 작업

`Help` 컨텍스트에서 사용 가능한 작업:

작업	기본값	설명
`help:dismiss`	Escape	도움말 메뉴 닫기

탭 작업

`Tabs` 컨텍스트에서 사용 가능한 작업:

작업	기본값	설명
`tabs:next`	Tab, Right	다음 탭
`tabs:previous`	Shift+Tab, Left	이전 탭

첨부 파일 작업

`Attachments` 컨텍스트에서 사용 가능한 작업:

작업	기본값	설명
`attachments:next`	Right	다음 첨부 파일
`attachments:previous`	Left	이전 첨부 파일
`attachments:remove`	Backspace, Delete	선택한 첨부 파일 제거
`attachments:exit`	Down, Escape	첨부 파일 바 종료

바닥글 작업

`Footer` 컨텍스트에서 사용 가능한 작업:

작업	기본값	설명
`footer:next`	Right	다음 바닥글 항목
`footer:previous`	Left	이전 바닥글 항목
`footer:openSelected`	Enter	선택한 바닥글 항목 열기
`footer:clearSelection`	Escape	바닥글 선택 지우기

메시지 선택기 작업

`MessageSelector` 컨텍스트에서 사용 가능한 작업:

작업	기본값	설명
`messageSelector:up`	Up, K, Ctrl+P	목록에서 위로 이동
`messageSelector:down`	Down, J, Ctrl+N	목록에서 아래로 이동
`messageSelector:top`	Ctrl+Up, Shift+Up, Meta+Up, Shift+K	맨 위로 이동
`messageSelector:bottom`	Ctrl+Down, Shift+Down, Meta+Down, Shift+J	맨 아래로 이동
`messageSelector:select`	Enter	메시지 선택

Diff 작업

`DiffDialog` 컨텍스트에서 사용 가능한 작업:

작업	기본값	설명
`diff:dismiss`	Escape	Diff 뷰어 닫기
`diff:previousSource`	Left	이전 diff 소스
`diff:nextSource`	Right	다음 diff 소스
`diff:previousFile`	Up	Diff의 이전 파일

작업	기본값	설명
`diff:nextFile`	Down	Diff의 다음 파일
`diff:viewDetails`	Enter	Diff 세부 정보 보기
`diff:back`	(컨텍스트별)	Diff 뷰어에서 뒤로 이동

모델 선택기 작업

`ModelPicker` 컨텍스트에서 사용 가능한 작업:

작업	기본값	설명
`modelPicker:decreaseEffort`	Left	노력 수준 감소
`modelPicker:increaseEffort`	Right	노력 수준 증가

선택 작업

`Select` 컨텍스트에서 사용 가능한 작업:

작업	기본값	설명
`select:next`	Down, J, Ctrl+N	다음 옵션
`select:previous`	Up, K, Ctrl+P	이전 옵션
`select:accept`	Enter	선택 수락
`select:cancel`	Escape	선택 취소

플러그인 작업

`Plugin` 컨텍스트에서 사용 가능한 작업:

작업	기본값	설명
`plugin:toggle`	Space	플러그인 선택 전환
`plugin:install`	I	선택한 플러그인 설치

설정 작업

`Settings` 컨텍스트에서 사용 가능한 작업:

작업	기본값	설명
`settings:search`	/	검색 모드 진입
`settings:retry`	R	사용량 데이터 다시 로드(오류 시)

키 입력 구문

수정자

`+` 구분자로 수정자 키를 사용합니다:

- `ctrl` 또는 `control` - Control 키
- `alt`, `opt`, 또는 `option` - Alt/Option 키
- `shift` - Shift 키
- `meta`, `cmd`, 또는 `command` - Meta/Command 키

예를 들어:

```
ctrl+k          수정자가 있는 단일 키
shift+tab       Shift + Tab
meta+p          Command/Meta + P
ctrl+shift+c    여러 수정자
```

대문자

독립 실행형 대문자는 Shift를 의미합니다. 예를 들어 `K` 는 `shift+k` 와 동일합니다. 이는 대문자와 소문자 키가 다른 의미를 갖는 vim 스타일 바인딩에 유용합니다.

수정자가 있는 대문자(예: `ctrl+K`)는 스타일 지정으로 처리되며 Shift를 의미하지 **않습니다** — `ctrl+K` 는 `ctrl+k` 와 동일합니다.

코드

코드는 공백으로 구분된 키 입력 시퀀스입니다:

```
ctrl+k ctrl+s    Ctrl+K를 누르고 놓은 다음 Ctrl+S를 누릅니다
```

특수 키

- `escape` 또는 `esc` - Escape 키
- `enter` 또는 `return` - Enter 키

- `tab` - Tab 키
- `space` - 스페이스바
- `up`, `down`, `left`, `right` - 화살표 키
- `backspace`, `delete` - Delete 키

기본 단축키 바인딩 해제

작업을 `null`로 설정하여 기본 단축키를 바인딩 해제합니다:

```
{
  "bindings": [
    {
      "context": "Chat",
      "bindings": {
        "ctrl+s": null
      }
    }
  ]
}
```

예약된 단축키

이러한 단축키는 다시 바인딩할 수 없습니다:

단축키	이유
Ctrl+C	하드코딩된 중단/취소
Ctrl+D	하드코딩된 종료

터미널 충돌

일부 단축키는 터미널 멀티플렉서와 충돌할 수 있습니다:

단축키	충돌
Ctrl+B	tmux 접두사(두 번 눌러서 보내기)
Ctrl+A	GNU screen 접두사
Ctrl+Z	Unix 프로세스 일시 중단(SIGTSTP)

Vim 모드 상호 작용

vim 모드가 활성화되면(`/vim`), 키바인딩과 vim 모드는 독립적으로 작동합니다:

- **Vim 모드**는 텍스트 입력 수준에서 입력을 처리합니다(커서 이동, 모드, 동작).
- **키바인딩**은 구성 요소 수준에서 작업을 처리합니다(작업 전환, 제출 등).
- vim 모드의 Escape 키는 INSERT를 NORMAL 모드로 전환합니다. `chat:cancel` 을 트리거하지 않습니다.
- 대부분의 Ctrl+key 단축키는 vim 모드를 통과하여 키바인딩 시스템으로 이동합니다.
- vim NORMAL 모드에서 `?` 는 도움말 메뉴를 표시합니다(vim 동작).

유효성 검사

Claude Code는 키바인딩을 검증하고 다음에 대한 경고를 표시합니다:

- 구문 분석 오류(잘못된 JSON 또는 구조)
- 잘못된 컨텍스트 이름
- 예약된 단축키 충돌
- 터미널 멀티플렉서 충돌
- 동일한 컨텍스트의 중복 바인딩

`/doctor` 를 실행하여 키바인딩 경고를 확인합니다.

Part 4: Configuration

권한 구성

> 세분화된 권한 규칙, 모드 및 관리형 정책을 통해 Claude Code가 액세스하고 수행할 수 있는 작업을 제어합니다.

Claude Code는 에이전트가 수행할 수 있는 작업과 수행할 수 없는 작업을 정확하게 지정할 수 있도록 세분화된 권한을 지원합니다. 권한 설정은 버전 제어에 체크인할 수 있으며 조직의 모든 개발자에게 배포할 수 있을 뿐만 아니라 개별 개발자가 사용자 정의할 수 있습니다.

권한 시스템

Claude Code는 강력함과 안전성의 균형을 맞추기 위해 계층화된 권한 시스템을 사용합니다:

도구 유형	예시	승인 필요	"예, 다시 묻지 않기" 동작
읽기 전용	파일 읽기, Grep	아니오	해당 없음
Bash 명령	셸 실행	예	프로젝트 디렉토리 및 명령당 영구적
파일 수정	Edit/Write 파일	예	세션 종료까지

권한 관리

`/permissions` 를 사용하여 Claude Code의 도구 권한을 보고 관리할 수 있습니다. 이 UI는 모든 권한 규칙과 이들이 출처한 settings.json 파일을 나열합니다.

- **Allow** 규칙을 사용하면 Claude Code가 수동 승인 없이 지정된 도구를 사용할 수 있습니다.
- **Ask** 규칙은 Claude Code가 지정된 도구를 사용하려고 할 때마다 확인을 요청합니다.
- **Deny** 규칙은 Claude Code가 지정된 도구를 사용하지 못하도록 방지합니다.

규칙은 순서대로 평가됩니다: **deny -> ask -> allow**. 첫 번째 일치하는 규칙이 우선이므로 deny 규칙이 항상 우선합니다.

권한 모드

Claude Code는 도구 승인 방식을 제어하는 여러 권한 모드를 지원합니다. 설정 파일에서 `defaultMode` 를 설정합니다:

모드	설명
`default`	표준 동작: 각 도구를 처음 사용할 때 권한을 요청합니다
`acceptEdits`	세션에 대해 파일 편집 권한을 자동으로 수락합니다
`plan`	Plan Mode: Claude는 파일을 분석할 수 있지만 수정하거나 명령을 실행할 수 없습니다
`dontAsk`	`/permissions` 또는 `permissions.allow` 규칙을 통해 사전 승인되지 않은 한 도구를 자동으로 거부합니다
`bypassPermissions`	모든 권한 프롬프트를 건너뜁니다(안전한 환경 필요, 아래 경고 참조)

> **Warning:**
>
> `bypassPermissions` 모드는 모든 권한 검사를 비활성화합니다. 컨테이너나 VM과 같은 Claude Code가 손상을 일으킬 수 없는 격리된 환경에서만 사용합니다. 관리자는 관리형 설정에서 `disableBypassPermissionsMode` 를 `"disable"` 로 설정하여 이 모드를 방지할 수 있습니다.

권한 규칙 구문

권한 규칙은 `Tool` 또는 `Tool(specifier)` 형식을 따릅니다.

도구의 모든 사용 일치

도구의 모든 사용을 일치시키려면 괄호 없이 도구 이름만 사용합니다:

규칙	효과
`Bash`	모든 Bash 명령과 일치합니다
`WebFetch`	모든 웹 가져오기 요청과 일치합니다
`Read`	모든 파일 읽기와 일치합니다

`Bash(*)` 는 `Bash` 와 동등하며 모든 Bash 명령과 일치합니다.

세분화된 제어를 위해 지정자 사용

괄호 안에 지정자를 추가하여 특정 도구 사용과 일치시킵니다:

규칙	효과
`Bash(npm run build)`	정확한 명령 `npm run build`와 일치합니다
`Read(./.env)`	현재 디렉토리의 `.env` 파일 읽기와 일치합니다
`WebFetch(domain:example.com)`	example.com으로의 가져오기 요청과 일치합니다

와일드카드 패턴

Bash 규칙은 `*`를 사용한 glob 패턴을 지원합니다. 와일드카드는 명령의 어느 위치에나 나타날 수 있습니다. 이 구성은 npm 및 git commit 명령을 허용하면서 git push를 차단합니다:

```
{
  "permissions": {
    "allow": [
      "Bash(npm run *)",
      "Bash(git commit *)",
      "Bash(git * main)",
      "Bash(* --version)",
      "Bash(* --help *)"
    ],
    "deny": [
      "Bash(git push *)"
    ]
  }
}
```

`*` 앞의 공백이 중요합니다: `Bash(ls *)`는 `ls -la`와 일치하지만 `lsof`와는 일치하지 않으며, `Bash(ls*)`는 둘 다 일치합니다. 레거시 `:*` 접미사 구문은 `*`와 동등하지만 더 이상 사용되지 않습니다.

도구별 권한 규칙

Bash

Bash 권한 규칙은 `*`를 사용한 와일드카드 일치를 지원합니다. 와일드카드는 명령의 시작, 중간 또는 끝을 포함하여 어느 위치에나 나타날 수 있습니다:

- `Bash(npm run build)`는 정확한 Bash 명령 `npm run build`와 일치합니다

- `Bash(npm run test *)` 는 `npm run test` 로 시작하는 Bash 명령과 일치합니다
- `Bash(npm *)` 는 `npm` 로 시작하는 모든 명령과 일치합니다
- `Bash(* install)` 은 `install` 로 끝나는 모든 명령과 일치합니다
- `Bash(git * main)` 은 `git checkout main` , `git merge main` 과 같은 명령과 일치합니다

`*` 가 앞에 공백이 있는 끝에 나타날 때(예: `Bash(ls *)`), 단어 경계를 적용하여 접두사 뒤에 공백이나 문자열 끝이 필요합니다. 예를 들어, `Bash(ls *)` 는 `ls -la` 와 일치하지만 `lsof` 와는 일치하지 않습니다. 반대로, 공백이 없는 `Bash(ls*)` 는 단어 경계 제약이 없으므로 `ls -la` 와 `lsof` 모두와 일치합니다.

> **Tip:**
>
> Claude Code는 셸 연산자(예: `&&`)를 인식하므로 `Bash(safe-cmd *)` 와 같은 접두사 일치 규칙은 `safe-cmd && other-cmd` 명령을 실행할 권한을 부여하지 않습니다.

> **Warning:**
>
> 명령 인수를 제약하려고 시도하는 Bash 권한 패턴은 취약합니다. 예를 들어, `Bash(curl http://github.com/ *)` 는 curl을 GitHub URL로 제한하려고 하지만 다음과 같은 변형과는 일치하지 않습니다:
>
> - URL 앞의 옵션: `curl -X GET http://github.com/...`
> - 다른 프로토콜: `curl https://github.com/...`
> - 리다이렉트: `curl -L http://bit.ly/xyz` (github로 리다이렉트)
> - 변수: `URL=http://github.com && curl $URL`
> - 추가 공백: `curl http://github.com`
>
> 더 안정적인 URL 필터링을 위해 다음을 고려합니다:
>
> - **Bash 네트워크 도구 제한**: deny 규칙을 사용하여 `curl` , `wget` 및 유사한 명령을 차단한 다음 허용된 도메인에 대해 `WebFetch(domain:github.com)` 권한으로 WebFetch 도구를 사용합니다
> - **PreToolUse 훅 사용**: Bash 명령의 URL을 검증하고 허용되지 않은 도메인을 차단하는 훅을 구현합니다
> - CLAUDE.md를 통해 Claude Code에 허용된 curl 패턴에 대해 지시합니다
>
> WebFetch만 사용하는 것은 네트워크 액세스를 방지하지 않습니다. Bash가 허용되면 Claude는 여전히 `curl` , `wget` 또는 다른 도구를 사용하여 모든 URL에 도달할 수 있습니다.

Read 및 Edit

`Edit` 규칙은 파일을 편집하는 모든 기본 제공 도구에 적용됩니다. Claude는 Grep 및 Glob과 같이 파일을 읽는 모든 기본 제공 도구에 `Read` 규칙을 적용하기 위해 최선을 다합니다.

Read 및 Edit 규칙은 모두 gitignore 사양을 따르며 4가지 고유한 패턴 유형이 있습니다:

패턴	의미	예시	일치
`//path`	파일 시스템 루트의 **절대** 경로	`Read(//Users/alice/secrets/**)`	`/Users/alice/secrets/**`
`~/path`	**홈** 디렉토리의 경로	`Read(~/Documents/*.pdf)`	`/Users/alice/Documents/*.pdf`
`/path`	프로젝트 루트에 **상대적인** 경로	`Edit(/src/**/*.ts)`	`<project root>/src/**/*.ts`
`path` 또는 `./path`	현재 디렉토리에 **상대적인** 경로	`Read(*.env)`	`<cwd>/*.env`

> **Warning:**
>
> `/Users/alice/file` 과 같은 패턴은 절대 경로가 아닙니다. 프로젝트 루트에 상대적입니다. 절대 경로의 경우 `//Users/alice/file` 을 사용합니다.

예시:

- `Edit(/docs/**)` : `<project>/docs/` 의 편집 (NOT `/docs/` and NOT `<project>/.claude/docs/`)
- `Read(~/.zshrc)` : 홈 디렉토리의 `.zshrc` 읽기
- `Edit(//tmp/scratch.txt)` : 절대 경로 `/tmp/scratch.txt` 편집
- `Read(src/**)` : `<current-directory>/src/` 에서 읽기

> **Note:**
>
> gitignore 패턴에서 `*` 는 단일 디렉토리의 파일과 일치하고 `**` 는 디렉토리 전체에서 재귀적으로 일치합니다. 모든 파일 액세스를 허용하려면 괄호 없이 도구 이름만 사용합니다: `Read` , `Edit` 또는 `Write` .

WebFetch

- `WebFetch(domain:example.com)` 은 example.com으로의 가져오기 요청과 일치합니다

MCP

- `mcp__puppeteer` 는 `puppeteer` 서버(Claude Code에서 구성된 이름)에서 제공하는 모든 도구와 일치합니다
- `mcp__puppeteer__*` 와일드카드 구문은 `puppeteer` 서버의 모든 도구와도 일치합니다
- `mcp__puppeteer__puppeteer_navigate` 는 `puppeteer` 서버에서 제공하는 `puppeteer_navigate` 도구와 일치합니다

Agent (subagents)

`Agent(AgentName)` 규칙을 사용하여 Claude가 사용할 수 있는 subagents를 제어합니다:

- `Agent(Explore)` 는 Explore subagent와 일치합니다
- `Agent(Plan)` 은 Plan subagent와 일치합니다
- `Agent(my-custom-agent)` 는 `my-custom-agent` 라는 사용자 정의 subagent와 일치합니다

이러한 규칙을 설정의 `deny` 배열에 추가하거나 `--disallowedTools` CLI 플래그를 사용하여 특정 에이전트를 비활성화합니다. Explore 에이전트를 비활성화하려면:

```
{
  "permissions": {
    "deny": ["Agent(Explore)"]
  }
}
```

훅으로 권한 확장

Claude Code 훅은 런타임에 권한 평가를 수행하기 위해 사용자 정의 셸 명령을 등록하는 방법을 제공합니다. Claude Code가 도구 호출을 할 때, PreToolUse 훅은 권한 시스템 전에 실행되며, 훅 출력은 권한 시스템 대신 도구 호출을 승인하거나 거부할지 여부를 결정할 수 있습니다.

작업 디렉토리

기본적으로 Claude는 시작된 디렉토리의 파일에 액세스할 수 있습니다. 이 액세스를 확장할 수 있습니다:

- **시작 중**: `--add-dir <path>` CLI 인수 사용
- **세션 중**: `/add-dir` 명령 사용
- **영구 구성**: 설정 파일의 `additionalDirectories` 에 추가

추가 디렉토리의 파일은 원래 작업 디렉토리와 동일한 권한 규칙을 따릅니다: 프롬프트 없이 읽을 수 있게 되며, 파일 편집 권한은 현재 권한 모드를 따릅니다.

권한이 샌드박싱과 상호 작용하는 방식

권한과 샌드박싱은 상호 보완적인 보안 계층입니다:

- **권한**은 Claude Code가 사용할 수 있는 도구와 액세스할 수 있는 파일 또는 도메인을 제어합니다. 모든 도구(Bash, Read, Edit, WebFetch, MCP 등)에 적용됩니다.
- **샌드박싱**은 Bash 도구의 파일 시스템 및 네트워크 액세스를 제한하는 OS 수준 적용을 제공합니다. Bash 명령 및 해당 자식 프로세스에만 적용됩니다.

심층 방어를 위해 둘 다 사용합니다:

- 권한 deny 규칙은 Claude가 제한된 리소스에 액세스하려고 시도하는 것을 차단합니다
- 샌드박스 제한은 프롬프트 주입이 Claude의 의사 결정을 우회하더라도 Bash 명령이 정의된 경계 외부의 리소스에 도달하는 것을 방지합니다
- 샌드박스의 파일 시스템 제한은 Read 및 Edit deny 규칙을 사용하며, 별도의 샌드박스 구성은 사용하지 않습니다
- 네트워크 제한은 WebFetch 권한 규칙과 샌드박스의 `allowedDomains` 목록을 결합합니다

관리형 설정

Claude Code 구성에 대한 중앙 집중식 제어가 필요한 조직의 경우, 관리자는 사용자 또는 프로젝트 설정으로 재정의할 수 없는 관리형 설정을 배포할 수 있습니다. 이러한 정책 설정은 일반 설정 파일과 동일한 형식을 따르며 MDM/OS 수준 정책, 관리형 설정 파일 또는 서버 관리형 설정을 통해 전달될 수 있습니다. 전달 메커니즘 및 파일 위치는 설정 파일을 참조합니다.

관리형 전용 설정

일부 설정은 관리형 설정에서만 효과적입니다:

설정	설명
`disableBypassPermissionsMode`	`bypassPermissions` 모드 및 `--dangerously-skip-permissions` 플래그를 방지하려면 `"disable"`로 설정합니다
`allowManagedPermissionRulesOnly`	`true`일 때, 사용자 및 프로젝트 설정이 `allow`, `ask` 또는 `deny` 권한 규칙을 정의하는 것을 방지합니다. 관리형 설정의 규칙만 적용됩니다
`allowManagedHooksOnly`	`true`일 때, 사용자, 프로젝트 및 플러그인 훅의 로드를 방지합니다. 관리형 훅 및 SDK 훅만 허용됩니다

설정	설명
`allowManagedMcpServersOnly`	`true` 일 때, 관리형 설정의 `allowedMcpServers` 만 존중됩니다. `deniedMcpServers` 는 여전히 모든 소스에서 병합됩니다. 관리형 MCP 구성 참조
`blockedMarketplaces`	마켓플레이스 소스의 차단 목록입니다. 차단된 소스는 다운로드 전에 확인되므로 파일 시스템에 닿지 않습니다. 관리형 마켓플레이스 제한 참조
`sandbox.network.allowManagedDomainsOnly`	`true` 일 때, 관리형 설정의 `allowedDomains` 및 `WebFetch(domain:...)` allow 규칙만 존중됩니다. 허용되지 않은 도메인은 사용자에게 프롬프트하지 않고 자동으로 차단됩니다. 거부된 도메인은 여전히 모든 소스에서 병합됩니다
`strictKnownMarketplaces`	사용자가 추가할 수 있는 플러그인 마켓플레이스를 제어합니다. 관리형 마켓플레이스 제한 참조
`allow_remote_sessions`	`true` 일 때, 사용자가 Remote Control 및 웹 세션을 시작할 수 있습니다. 기본값은 `true` 입니다. 원격 세션 액세스를 방지하려면 `false` 로 설정합니다

설정 우선순위

권한 규칙은 다른 모든 Claude Code 설정과 동일한 설정 우선순위를 따릅니다:

1. **관리형 설정**: 명령줄 인수를 포함한 다른 수준으로 재정의할 수 없습니다
2. **명령줄 인수**: 임시 세션 재정의
3. **로컬 프로젝트 설정** (`.claude/settings.local.json`)
4. **공유 프로젝트 설정** (`.claude/settings.json`)
5. **사용자 설정** (`~/.claude/settings.json`)

도구가 어느 수준에서든 거부되면 다른 수준은 이를 허용할 수 없습니다. 예를 들어, 관리형 설정 deny는 `--allowedTools` 로 재정의할 수 없으며, `--disallowedTools` 는 관리형 설정이 정의하는 것 이상의 제한을 추가할 수 있습니다.

권한이 사용자 설정에서 허용되지만 프로젝트 설정에서 거부되면, 프로젝트 설정이 우선이며 권한이 차단됩니다.

예시 구성

이 저장소에는 일반적인 배포 시나리오에 대한 시작 설정 구성이 포함되어 있습니다. 이를 시작점으로 사용하고 필요에 맞게 조정합니다.

참고 항목

- 설정: 권한 설정 테이블을 포함한 완전한 구성 참조
- 샌드박싱: Bash 명령에 대한 OS 수준 파일 시스템 및 네트워크 격리
- 인증: Claude Code에 대한 사용자 액세스 설정
- 보안: 보안 보호 및 모범 사례
- 훅: 워크플로우 자동화 및 권한 평가 확장

Claude가 프로젝트를 기억하는 방법

> CLAUDE.md 파일로 Claude에 지속적인 지침을 제공하고, 자동 메모리를 통해 Claude가 자동으로 학습을 축적하도록 합니다.

각 Claude Code 세션은 새로운 컨텍스트 윈도우로 시작됩니다. 두 가지 메커니즘이 세션 간에 지식을 전달합니다:

- **CLAUDE.md 파일**: Claude에 지속적인 컨텍스트를 제공하기 위해 작성하는 지침
- **자동 메모리**: 사용자의 수정 및 선호도에 따라 Claude가 자신을 위해 작성하는 노트

이 페이지에서는 다음을 다룹니다:

- CLAUDE.md 파일 작성 및 구성
- `.claude/rules/` 를 사용하여 특정 파일 유형에 규칙 범위 지정
- 자동 메모리 구성하여 Claude가 자동으로 노트를 작성하도록 함
- 지침이 따라지지 않을 때 문제 해결

CLAUDE.md vs 자동 메모리

Claude Code에는 두 가지 상호 보완적인 메모리 시스템이 있습니다. 둘 다 모든 대화의 시작 시 로드됩니다. Claude는 이들을 강제된 구성이 아닌 컨텍스트로 취급합니다. 지침이 더 구체적이고 간결할수록 Claude가 더 일관되게 따릅니다.

	CLAUDE.md 파일	자동 메모리
작성자	사용자	Claude
포함 내용	지침 및 규칙	학습 및 패턴
범위	프로젝트, 사용자 또는 조직	작업 트리당
로드 대상	모든 세션	모든 세션(처음 200줄)
사용 목적	코딩 표준, 워크플로우, 프로젝트 아키텍처	빌드 명령, 디버깅 인사이트, Claude가 발견한 선호도

Claude의 동작을 안내하려면 CLAUDE.md 파일을 사용합니다. 자동 메모리를 통해 Claude는 수동 작업 없이 사용자의 수정으로부터 학습할 수 있습니다.

Subagent도 자신의 자동 메모리를 유지할 수 있습니다. 자세한 내용은 subagent 구성을 참조하세요.

CLAUDE.md 파일

CLAUDE.md 파일은 프로젝트, 개인 워크플로우 또는 전체 조직에 대해 Claude에 지속적인 지침을 제공하는 마크다운 파일입니다. 이러한 파일을 일반 텍스트로 작성하면 Claude가 모든 세션의 시작 시 이를 읽습니다.

CLAUDE.md 파일을 어디에 배치할지 선택

CLAUDE.md 파일은 여러 위치에 있을 수 있으며, 각 위치는 다른 범위를 가집니다. 더 구체적인 위치가 더 광범위한 위치보다 우선합니다.

범위	위치	목적	사용 사례	공유 대상
관리 정책	• macOS: `/Library/Application Support/ClaudeCode/CLAUDE.md` • Linux 및 WSL: `/etc/claude-code/CLAUDE.md` • Windows: `C:\Program Files\ClaudeCode\CLAUDE.md`	IT/DevOps에서 관리하는 조직 전체 지침	회사 코딩 표준, 보안 정책, 규정 준수 요구사항	조직의 모든 사용자
프로젝트 지침	`./CLAUDE.md` 또는 `./.claude/CLAUDE.md`	프로젝트에 대한 팀 공유 지침	프로젝트 아키텍처, 코딩 표준, 일반적인 워크플로우	소스 제어를 통한 팀 멤버
사용자 지침	`~/.claude/CLAUDE.md`	모든 프로젝트에 대한 개인 선호도	코드 스타일 선호도, 개인 도구 단축키	본인만(모든 프로젝트)

작업 디렉토리 위의 디렉토리 계층 구조에 있는 CLAUDE.md 파일은 시작 시 전체가 로드됩니다. 하위 디렉토리의 CLAUDE.md 파일은 Claude가 해당 디렉토리의 파일을 읽을 때 필요에 따라 로드됩니다. 전체 해석 순서는 CLAUDE.md 파일이 로드되는 방식을 참조하세요.

대규모 프로젝트의 경우 프로젝트 규칙을 사용하여 지침을 주제별 파일로 나눌 수 있습니다. 규칙을 통해 특정 파일 유형 또는 하위 디렉토리에 지침의 범위를 지정할 수 있습니다.

프로젝트 CLAUDE.md 설정

프로젝트 CLAUDE.md는 `./CLAUDE.md` 또는 `./.claude/CLAUDE.md`에 저장할 수 있습니다. 이 파일을 만들고 프로젝트에서 작업하는 모든 사람에게 적용되는 지침을 추가합니다: 빌드 및 테스트 명령, 코딩 표준, 아키텍처 결정, 명명 규칙 및 일반적인 워크플로우. 이러한 지침은 버전 제어를 통해 팀과 공유되므로 개인 선호도보다는 프로젝트 수준의 표준에 중점을 두세요.

> **Tip:**
>
> `/init`을 실행하여 CLAUDE.md를 자동으로 생성합니다. Claude가 코드베이스를 분석하고 발견한 빌드 명령, 테스트 지침 및 프로젝트 규칙이 포함된 파일을 만듭니다. CLAUDE.md가 이미 존재하면 `/init`은 덮어쓰지 않고 개선 사항을 제안합니다. 거기서부터 Claude가 자신에게 발견하지 못할 지침으로 개선합니다.

효과적인 지침 작성

CLAUDE.md 파일은 모든 세션의 시작 시 컨텍스트 윈도우에 로드되어 대화와 함께 토큰을 소비합니다. 강제된 구성이 아닌 컨텍스트이기 때문에 지침을 작성하는 방식이 Claude가 이를 따르는 신뢰성에 영향을 미칩니다. 구체적이고 간결하며 잘 구조화된 지침이 가장 잘 작동합니다.

크기: CLAUDE.md 파일당 200줄 이하를 목표로 합니다. 더 긴 파일은 더 많은 컨텍스트를 소비하고 준수를 줄입니다. 지침이 커지면 가져오기나 `.claude/rules/` 파일을 사용하여 분할합니다.

구조: 마크다운 헤더와 글머리 기호를 사용하여 관련 지침을 그룹화합니다. Claude는 독자와 같은 방식으로 구조를 스캔합니다: 구성된 섹션이 조밀한 단락보다 따르기 쉽습니다.

구체성: 검증할 수 있을 정도로 구체적인 지침을 작성합니다. 예를 들어:

- "코드를 제대로 포맷하세요" 대신 "2칸 들여쓰기 사용"
- "변경 사항을 테스트하세요" 대신 "커밋하기 전에 `npm test` 실행"
- "파일을 정리하세요" 대신 "API 핸들러는 `src/api/handlers/`에 위치"

일관성: 두 규칙이 서로 모순되면 Claude가 하나를 임의로 선택할 수 있습니다. CLAUDE.md 파일, 하위 디렉토리의 중첩된 CLAUDE.md 파일 및 `.claude/rules/`를 주기적으로 검토하여 오래되었거나 충돌하는 지침을 제거합니다. 모노레포에서는 `claudeMdExcludes`를 사용하여 작업과 관련이 없는 다른 팀의 CLAUDE.md 파일을 건너뜁니다.

추가 파일 가져오기

CLAUDE.md 파일은 `@path/to/import` 구문을 사용하여 추가 파일을 가져올 수 있습니다. 가져온 파일은 확장되어 참조하는 CLAUDE.md와 함께 시작 시 컨텍스트에 로드됩니다.

상대 경로와 절대 경로 모두 허용됩니다. 상대 경로는 작업 디렉토리가 아닌 가져오기를 포함하는 파일을 기준으로 해석됩니다. 가져온 파일은 최대 5개 홉의 깊이로 다른 파일을 재귀적으로 가져올 수 있습니다.

README, package.json 및 워크플로우 가이드를 가져오려면 CLAUDE.md의 어디든지 `@` 구문으로 참조합니다:

```
프로젝트 개요는 @README를 참조하고 이 프로젝트의 사용 가능한 npm 명령은 @package.json을 참조하세요.

## 추가 지침
- git 워크플로우 @docs/git-instructions.md
```

체크인하지 않으려는 개인 선호도의 경우 홈 디렉토리에서 파일을 가져옵니다. 가져오기는 공유 CLAUDE.md에 있지만 가리키는 파일은 컴퓨터에 남아 있습니다:

```
## 개인 선호도
- @~/.claude/my-project-instructions.md
```

> **Warning:**
>
> Claude Code가 처음으로 프로젝트에서 외부 가져오기를 만나면 파일을 나열하는 승인 대화를 표시합니다. 거부하면 가져오기가 비활성화된 상태로 유지되고 대화가 다시 나타나지 않습니다.

지침을 구성하는 더 구조화된 접근 방식은 `.claude/rules/` 을 참조하세요.

CLAUDE.md 파일이 로드되는 방식

Claude Code는 현재 작업 디렉토리에서 디렉토리 트리를 따라 올라가며 CLAUDE.md 파일을 읽고 각 디렉토리를 확인합니다. 즉, `foo/bar/` 에서 Claude Code를 실행하면 `foo/bar/CLAUDE.md` 와 `foo/CLAUDE.md` 모두에서 지침을 로드합니다.

Claude는 또한 현재 작업 디렉토리 아래의 하위 디렉토리에서 CLAUDE.md 파일을 발견합니다. 시작 시 로드하는 대신 Claude가 해당 하위 디렉토리의 파일을 읽을 때 포함됩니다.

대규모 모노레포에서 작업하고 다른 팀의 CLAUDE.md 파일이 선택되는 경우 `claudeMdExcludes` 를 사용하여 건너뜁니다.

추가 디렉토리에서 로드

`--add-dir` 플래그는 Claude에 주 작업 디렉토리 외부의 추가 디렉토리에 대한 액세스를 제공합니다. 기본적으로 이러한 디렉토리의 CLAUDE.md 파일은 로드되지 않습니다.

추가 디렉토리에서 CLAUDE.md 파일을 로드하려면 `CLAUDE.md` , `.claude/CLAUDE.md` 및 `.claude/rules/*.md` 를 포함하여 `CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD` 환경 변수를 설정합니다:

```
CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD=1 claude --add-dir ../shared-config
```

`.claude/rules/` 로 규칙 구성

더 큰 프로젝트의 경우 `.claude/rules/` 디렉토리를 사용하여 지침을 여러 파일로 구성할 수 있습니다. 이렇게 하면 지침이 모듈식이 되고 팀이 유지 관리하기 쉬워집니다. 규칙을 특정 파일 경로로 범위 지정할 수도 있으므로 Claude가 일치하는 파일로 작업할 때만 컨텍스트에 로드되어 노이즈를 줄이고 컨텍스트 공간을 절약합니다.

> **Note:**
>
> 규칙은 모든 세션에 로드되거나 일치하는 파일이 열릴 때 로드됩니다. 항상 컨텍스트에 있을 필요가 없는 작업별 지침의 경우 대신 skills를 사용하세요. 이는 호출할 때 또는 Claude가 프롬프트와 관련이 있다고 판단할 때만 로드됩니다.

규칙 설정

프로젝트의 `.claude/rules/` 디렉토리에 마크다운 파일을 배치합니다. 각 파일은 `testing.md` 또는 `api-design.md` 와 같은 설명적인 파일명으로 한 가지 주제를 다루어야 합니다. 모든 `.md` 파일은 재귀적으로 발견되므로 `frontend/` 또는 `backend/` 와 같은 하위 디렉토리로 규칙을 구성할 수 있습니다:

```
your-project/
├── .claude/
│   ├── CLAUDE.md           # 주 프로젝트 지침
│   └── rules/
│       ├── code-style.md   # 코드 스타일 가이드라인
│       ├── testing.md      # 테스트 규칙
│       └── security.md     # 보안 요구사항
```

`paths` frontmatter가 없는 규칙은 `.claude/CLAUDE.md` 와 동일한 우선순위로 시작 시 로드됩니다.

경로별 규칙

규칙은 `paths` 필드가 있는 YAML frontmatter를 사용하여 특정 파일로 범위를 지정할 수 있습니다. 이러한 조건부 규칙은 Claude가 지정된 패턴과 일치하는 파일로 작업할 때만 적용됩니다.

```
---
paths:
  - "src/api/**/*.ts"
---

## API 개발 규칙

- 모든 API 엔드포인트는 입력 검증을 포함해야 합니다
- 표준 오류 응답 형식 사용
- OpenAPI 문서 주석 포함
```

`paths` 필드가 없는 규칙은 무조건 로드되며 모든 파일에 적용됩니다. 경로 범위 규칙은 모든 도구 사용 시가 아닌 패턴과 일치하는 파일을 읽을 때 트리거됩니다.

`paths` 필드에서 glob 패턴을 사용하여 확장명, 디렉토리 또는 조합으로 파일을 일치시킵니다:

패턴	일치
`**/*.ts`	모든 디렉토리의 모든 TypeScript 파일
`src/**/*`	`src/` 디렉토리 아래의 모든 파일
`*.md`	프로젝트 루트의 마크다운 파일
`src/components/*.tsx`	특정 디렉토리의 React 컴포넌트

여러 패턴을 지정하고 중괄호 확장을 사용하여 한 패턴에서 여러 확장명을 일치시킬 수 있습니다:

```
---
paths:
  - "src/**/*.{ts,tsx}"
  - "lib/**/*.ts"
  - "tests/**/*.test.ts"
---
```

심볼릭 링크로 프로젝트 간 규칙 공유

`.claude/rules/` 디렉토리는 심볼릭 링크를 지원하므로 공유 규칙 세트를 유지하고 여러 프로젝트에 링크할 수 있습니다. 심볼릭 링크는 해석되어 정상적으로 로드되며 순환 심볼릭 링크는 감지되고 우아하게 처리됩니다.

이 예제는 공유 디렉토리와 개별 파일을 모두 링크합니다:

```
ln -s ~/shared-claude-rules .claude/rules/shared
ln -s ~/company-standards/security.md .claude/rules/security.md
```

사용자 수준 규칙

`~/.claude/rules/` 의 개인 규칙은 컴퓨터의 모든 프로젝트에 적용됩니다. 프로젝트별이 아닌 선호도에 사용합니다:

```
~/.claude/rules/
├── preferences.md    # 개인 코딩 선호도
└── workflows.md      # 선호하는 워크플로우
```

사용자 수준 규칙은 프로젝트 규칙 전에 로드되어 프로젝트 규칙에 더 높은 우선순위를 부여합니다.

대규모 팀을 위한 CLAUDE.md 관리

조직에서 Claude Code를 팀 전체에 배포하는 경우 지침을 중앙 집중식으로 관리하고 로드되는 CLAUDE.md 파일을 제어할 수 있습니다.

조직 전체 CLAUDE.md 배포

조직은 컴퓨터의 모든 사용자에게 적용되는 중앙 집중식으로 관리되는 CLAUDE.md를 배포할 수 있습니다. 이 파일은 개별 설정으로 제외될 수 없습니다.

Step 1: 관리 정책 위치에 파일 만들기

- macOS: `/Library/Application Support/ClaudeCode/CLAUDE.md`
- Linux 및 WSL: `/etc/claude-code/CLAUDE.md`
- Windows: `C:\Program Files\ClaudeCode\CLAUDE.md`

Step 2: 구성 관리 시스템으로 배포

MDM, 그룹 정책, Ansible 또는 유사한 도구를 사용하여 개발자 컴퓨터 전체에 파일을 배포합니다. 다른 조직 전체 구성 옵션은 관리 설정을 참조하세요.

특정 CLAUDE.md 파일 제외

대규모 모노레포에서 상위 CLAUDE.md 파일에는 작업과 관련이 없는 지침이 포함될 수 있습니다. `claudeMdExcludes` 설정을 통해 경로 또는 glob 패턴으로 특정 파일을 건너뛸 수 있습니다.

이 예제는 상위 폴더의 최상위 CLAUDE.md 및 규칙 디렉토리를 제외합니다. 제외가 컴퓨터에 로컬로 유지되도록 `.claude/settings.local.json`에 추가합니다:

```
{
  "claudeMdExcludes": [
    "**/monorepo/CLAUDE.md",
    "/home/user/monorepo/other-team/.claude/rules/**"
  ]
}
```

패턴은 glob 구문을 사용하여 절대 파일 경로와 일치합니다. `claudeMdExcludes`를 설정 레이어: 사용자, 프로젝트, 로컬 또는 관리 정책에서 구성할 수 있습니다. 배열은 레이어 전체에서 병합됩니다.

관리 정책 CLAUDE.md 파일은 제외될 수 없습니다. 이렇게 하면 개별 설정에 관계없이 조직 전체 지침이 항상 적용됩니다.

자동 메모리

자동 메모리를 통해 Claude는 아무것도 작성하지 않고도 세션 간에 지식을 축적할 수 있습니다. Claude는 작업할 때 자신을 위해 노트를 저장합니다: 빌드 명령, 디버깅 인사이트, 아키텍처 노트, 코드 스타일 선호도 및 워크플로우 습관. Claude는 모든 세션마다 뭔가를 저장하지 않습니다. 정보가 향후 대화에서 유용할지 여부에 따라 기억할 가치가 있는지 결정합니다.

> **Note:**
>
> 자동 메모리는 Claude Code v2.1.59 이상이 필요합니다. `claude --version`으로 버전을 확인합니다.

자동 메모리 활성화 또는 비활성화

자동 메모리는 기본적으로 켜져 있습니다. 토글하려면 세션에서 `/memory`를 열고 자동 메모리 토글을 사용하거나 프로젝트 설정에서 `autoMemoryEnabled`를 설정합니다:

```
{
  "autoMemoryEnabled": false
}
```

환경 변수를 통해 자동 메모리를 비활성화하려면 `CLAUDE_CODE_DISABLE_AUTO_MEMORY=1`을 설정합니다.

저장 위치

각 프로젝트는 `~/.claude/projects/<project>/memory/` 에 자신의 메모리 디렉토리를 가집니다. `<project>` 경로는 git 저장소에서 파생되므로 동일한 저장소 내의 모든 worktree 및 하위 디렉토리는 하나의 자동 메모리 디렉토리를 공유합니다. git 저장소 외부에서는 프로젝트 루트가 대신 사용됩니다.

자동 메모리를 다른 위치에 저장하려면 사용자 또는 로컬 설정에서 `autoMemoryDirectory` 를 설정합니다:

```
{
  "autoMemoryDirectory": "~/my-custom-memory-dir"
}
```

이 설정은 정책, 로컬 및 사용자 설정에서 허용됩니다. 공유 프로젝트가 자동 메모리 쓰기를 민감한 위치로 리디렉션하는 것을 방지하기 위해 프로젝트 설정(`.claude/settings.json`)에서는 허용되지 않습니다.

디렉토리에는 `MEMORY.md` 진입점과 선택적 주제 파일이 포함됩니다:

```
~/.claude/projects/<project>/memory/
├── MEMORY.md          # 간결한 인덱스, 모든 세션에 로드됨
├── debugging.md       # 디버깅 패턴에 대한 상세 노트
├── api-conventions.md # API 설계 결정
└── ...                # Claude가 만드는 다른 주제 파일
```

`MEMORY.md` 는 메모리 디렉토리의 인덱스 역할을 합니다. Claude는 세션 전체에서 이 디렉토리의 파일을 읽고 쓰며 `MEMORY.md` 를 사용하여 저장된 내용을 추적합니다.

자동 메모리는 컴퓨터 로컬입니다. 동일한 git 저장소 내의 모든 worktree 및 하위 디렉토리는 하나의 자동 메모리 디렉토리를 공유합니다. 파일은 컴퓨터 또는 클라우드 환경 간에 공유되지 않습니다.

작동 방식

`MEMORY.md` 의 처음 200줄은 모든 대화의 시작 시 로드됩니다. 200줄을 초과하는 콘텐츠는 세션 시작 시 로드되지 않습니다. Claude는 상세한 노트를 별도의 주제 파일로 이동하여 `MEMORY.md` 를 간결하게 유지합니다.

이 200줄 제한은 `MEMORY.md` 에만 적용됩니다. CLAUDE.md 파일은 길이에 관계없이 전체가 로드되지만 더 짧은 파일이 더 나은 준수를 생성합니다.

`debugging.md` 또는 `patterns.md`와 같은 주제 파일은 시작 시 로드되지 않습니다. Claude는 필요할 때 표준 파일 도구를 사용하여 필요에 따라 읽습니다.

Claude는 세션 중에 메모리 파일을 읽고 씁니다. Claude Code 인터페이스에서 "Writing memory" 또는 "Recalled memory"를 보면 Claude가 `~/.claude/projects/<project>/memory/`에서 활발히 업데이트하거나 읽고 있습니다.

메모리 감사 및 편집

자동 메모리 파일은 언제든지 편집하거나 삭제할 수 있는 일반 마크다운입니다. `/memory`를 실행하여 세션 내에서 메모리 파일을 찾아보고 엽니다.

`/memory`로 보기 및 편집

`/memory` 명령은 현재 세션에 로드된 모든 CLAUDE.md 및 규칙 파일을 나열하고, 자동 메모리를 켜거나 끌 수 있으며, 자동 메모리 폴더를 열 수 있는 링크를 제공합니다. 파일을 선택하여 편집기에서 엽니다.

Claude에게 "항상 npm이 아닌 pnpm 사용" 또는 "API 테스트에 로컬 Redis 인스턴스가 필요함을 기억"과 같이 뭔가를 기억하도록 요청하면 Claude가 자동 메모리에 저장합니다. 대신 CLAUDE.md에 지침을 추가하려면 Claude에게 직접 "이것을 CLAUDE.md에 추가"라고 요청하거나 `/memory`를 통해 파일을 직접 편집합니다.

메모리 문제 해결

이들은 CLAUDE.md 및 자동 메모리의 가장 일반적인 문제와 디버깅 단계입니다.

Claude가 CLAUDE.md를 따르지 않음

CLAUDE.md는 컨텍스트이지 강제가 아닙니다. Claude가 이를 읽고 따르려고 하지만 특히 모호하거나 충돌하는 지침의 경우 엄격한 준수를 보장하지 않습니다.

디버깅하려면:

- `/memory`를 실행하여 CLAUDE.md 파일이 로드되는지 확인합니다. 파일이 나열되지 않으면 Claude가 볼 수 없습니다.
- 관련 CLAUDE.md가 세션에 대해 로드되는 위치에 있는지 확인합니다(CLAUDE.md 파일을 어디에 배치할지 선택 참조).
- 지침을 더 구체적으로 만듭니다. "코드를 제대로 포맷하세요" 대신 "2칸 들여쓰기 사용"이 더 잘 작동합니다.
- CLAUDE.md 파일 전체에서 충돌하는 지침을 찾습니다. 두 파일이 동일한 동작에 대해 다른 지침을 제공하면 Claude가 하나를 임의로 선택할 수 있습니다.

> **Tip:**
>
> `InstructionsLoaded` hook을 사용하여 로드된 정확한 지침 파일, 로드 시기 및 이유를 기록합니다. 이는 경로별 규칙 또는 하위 디렉토리의 지연 로드 파일을 디버깅하는 데 유용합니다.

자동 메모리가 저장한 내용을 모름

`/memory`를 실행하고 자동 메모리 폴더를 선택하여 Claude가 저장한 내용을 찾아봅니다. 모든 것이 읽고, 편집하거나 삭제할 수 있는 일반 마크다운입니다.

CLAUDE.md가 너무 큼

200줄을 초과하는 파일은 더 많은 컨텍스트를 소비하고 준수를 줄일 수 있습니다. 상세한 콘텐츠를 `@path` 가져오기로 참조되는 별도 파일로 이동(추가 파일 가져오기 참조)하거나 `.claude/rules/` 파일 전체에서 지침을 분할합니다.

`/compact` 후 지침이 손실된 것 같음

CLAUDE.md는 압축을 완전히 견딥니다. `/compact` 후 Claude는 디스크에서 CLAUDE.md를 다시 읽고 세션에 새로 다시 주입합니다. 압축 후 지침이 사라지면 대화에서만 제공되었고 CLAUDE.md에 작성되지 않았습니다. 세션 간에 지속되도록 CLAUDE.md에 추가합니다.

효과적인 지침에 대한 지침은 효과적인 지침 작성을 참조하세요.

관련 리소스

- Skills: 필요에 따라 로드되는 반복 가능한 워크플로우 패키징
- 설정: 설정 파일로 Claude Code 동작 구성
- 세션 관리: 컨텍스트 관리, 대화 재개 및 병렬 세션 실행
- Subagent 메모리: subagent가 자신의 자동 메모리를 유지하도록 허용

Claude Code 설정

> 전역 및 프로젝트 수준 설정과 환경 변수로 Claude Code를 구성합니다.

Claude Code는 사용자의 필요에 맞게 동작을 구성할 수 있는 다양한 설정을 제공합니다. 대화형 REPL을 사용할 때 `/config` 명령을 실행하여 Claude Code를 구성할 수 있으며, 이는 상태 정보를 보고 구성 옵션을 수정할 수 있는 탭 형식의 설정 인터페이스를 엽니다.

구성 범위

Claude Code는 **범위 시스템**을 사용하여 구성이 적용되는 위치와 공유 대상을 결정합니다. 범위를 이해하면 개인 사용, 팀 협업 또는 엔터프라이즈 배포를 위해 Claude Code를 구성하는 방법을 결정하는 데 도움이 됩니다.

사용 가능한 범위

범위	위치	영향을 받는 대상	팀과 공유?
Managed	서버 관리 설정, plist / 레지스트리 또는 시스템 수준 `managed-settings.json`	머신의 모든 사용자	예 (IT에서 배포)
User	`~/.claude/` 디렉토리	모든 프로젝트에서 사용자	아니오
Project	저장소의 `.claude/`	이 저장소의 모든 협업자	예 (git에 커밋됨)
Local	`.claude/settings.local.json`	이 저장소에서만 사용자	아니오 (gitignored)

각 범위를 사용할 시기

Managed 범위는 다음을 위한 것입니다:

- 조직 전체에서 적용해야 하는 보안 정책
- 재정의할 수 없는 규정 준수 요구사항
- IT/DevOps에서 배포한 표준화된 구성

User 범위는 다음에 가장 적합합니다:

- 모든 곳에서 원하는 개인 설정 (테마, 편집기 설정)

- 모든 프로젝트에서 사용하는 도구 및 플러그인
- API 키 및 인증 (안전하게 저장됨)

Project 범위는 다음에 가장 적합합니다:

- 팀 공유 설정 (권한, hooks, MCP servers)
- 전체 팀이 가져야 할 플러그인
- 협업자 간 도구 표준화

Local 범위는 다음에 가장 적합합니다:

- 특정 프로젝트에 대한 개인 재정의
- 팀과 공유하기 전에 구성 테스트
- 다른 사용자에게는 작동하지 않을 머신 특정 설정

범위 상호작용 방식

동일한 설정이 여러 범위에서 구성되면 더 구체적인 범위가 우선합니다:

1. **Managed** (최상위) - 아무것도 재정의할 수 없음
2. **명령줄 인수** - 임시 세션 재정의
3. **Local** - 프로젝트 및 사용자 설정 재정의
4. **Project** - 사용자 설정 재정의
5. **User** (최하위) - 다른 것이 설정을 지정하지 않을 때 적용

예를 들어, 사용자 설정에서는 권한이 허용되지만 프로젝트 설정에서는 거부되면, 프로젝트 설정이 우선하고 권한이 차단됩니다.

범위를 사용하는 것

범위는 많은 Claude Code 기능에 적용됩니다:

기능	사용자 위치	프로젝트 위치	Local 위치
Settings	`~/.claude/settings.json`	`.claude/settings.json`	`.claude/settings.local.json`
Subagents	`~/.claude/agents/`	`.claude/agents/`	—
MCP servers	`~/.claude.json`	`.mcp.json`	`~/.claude.json` (프로젝트별)

기능	사용자 위치	프로젝트 위치	Local 위치
Plugins	`~/.claude/settings.json`	`.claude/settings.json`	`.claude/settings.local.json`
CLAUDE.md	`~/.claude/CLAUDE.md`	`CLAUDE.md` 또는 `.claude/CLAUDE.md`	—

설정 파일

`settings.json` 파일은 계층적 설정을 통해 Claude Code를 구성하기 위한 공식 메커니즘입니다:

- **사용자 설정**은 `~/.claude/settings.json`에 정의되며 모든 프로젝트에 적용됩니다.
- **프로젝트 설정**은 프로젝트 디렉토리에 저장됩니다:
  - `.claude/settings.json` - 소스 제어에 체크인되고 팀과 공유되는 설정
  - `.claude/settings.local.json` - 체크인되지 않는 설정으로, 개인 설정 및 실험에 유용합니다. Claude Code는 생성될 때 `.claude/settings.local.json`을 무시하도록 git을 구성합니다.
- **Managed 설정**: 중앙 집중식 제어가 필요한 조직의 경우, Claude Code는 managed 설정을 위한 여러 전달 메커니즘을 지원합니다. 모두 동일한 JSON 형식을 사용하며 사용자 또는 프로젝트 설정으로 재정의할 수 없습니다:
  - **서버 관리 설정**: Anthropic의 서버에서 Claude.ai 관리 콘솔을 통해 전달됩니다. 서버 관리 설정을 참조하세요.
  - **MDM/OS 수준 정책**: macOS 및 Windows의 기본 장치 관리를 통해 전달됩니다:
  - macOS: `com.anthropic.claudecode` managed preferences domain (Jamf, Kandji 또는 기타 MDM 도구의 구성 프로필을 통해 배포)
  - Windows: `HKLM\SOFTWARE\Policies\ClaudeCode` 레지스트리 키와 JSON을 포함하는 `Settings` 값 (REG_SZ 또는 REG_EXPAND_SZ) (그룹 정책 또는 Intune을 통해 배포)
  - Windows (사용자 수준): `HKCU\SOFTWARE\Policies\ClaudeCode` (최하위 정책 우선순위, 관리자 수준 소스가 없을 때만 사용)
  - **파일 기반**: `managed-settings.json` 및 `managed-mcp.json`이 시스템 디렉토리에 배포됩니다:
  - macOS: `/Library/Application Support/ClaudeCode/`
  - Linux 및 WSL: `/etc/claude-code/`

  - Windows: `C:\Program Files\ClaudeCode\`

  자세한 내용은 managed 설정 및 Managed MCP 구성을 참조하세요.

> **Note:**
>
> Managed 배포는 `strictKnownMarketplaces` 를 사용하여 **플러그인 마켓플레이스 추가**를 제한할 수도 있습니다. 자세한 내용은 Managed 마켓플레이스 제한을 참조하세요.

- **기타 구성**은 `~/.claude.json` 에 저장됩니다. 이 파일에는 사용자의 설정 (테마, 알림 설정, 편집기 모드), OAuth 세션, 사용자 및 local 범위에 대한 MCP server 구성, 프로젝트별 상태 (허용된 도구, 신뢰 설정) 및 다양한 캐시가 포함됩니다. 프로젝트 범위 MCP 서버는 `.mcp.json` 에 별도로 저장됩니다.

> **Note:**
>
> Claude Code는 자동으로 구성 파일의 타임스탬프가 지정된 백업을 생성하고 데이터 손실을 방지하기 위해 가장 최근의 5개 백업을 유지합니다.

```
{
  "$schema": "https://json.schemastore.org/claude-code-settings.json",
  "permissions": {
    "allow": [
      "Bash(npm run lint)",
      "Bash(npm run test *)",
      "Read(~/.zshrc)"
    ],
    "deny": [
      "Bash(curl *)",
      "Read(./.env)",
      "Read(./.env.*)",
      "Read(./secrets/**)"
    ]
  },
  "env": {
    "CLAUDE_CODE_ENABLE_TELEMETRY": "1",
    "OTEL_METRICS_EXPORTER": "otlp"
  },
  "companyAnnouncements": [
    "Welcome to Acme Corp! Review our code guidelines at docs.acme.com",
    "Reminder: Code reviews required for all PRs",
    "New security policy in effect"
  ]
}
```

위의 예제에서 `$schema` 줄은 Claude Code 설정에 대한 공식 JSON 스키마를 가리킵니다. 이를 `settings.json`에 추가하면 VS Code, Cursor 및 JSON 스키마 검증을 지원하는 다른 편집기에서 자동 완성 및 인라인 검증이 활성화됩니다.

사용 가능한 설정

`settings.json`은 여러 옵션을 지원합니다:

키	설명	예제
`apiKeyHelper`	`/bin/sh` 에서 실행할 사용자 정의 스크립트로 인증 값을 생성합니다. 이 값은 모델 요청에 대해 `X-Api-Key` 및 `Authorization: Bearer` 헤더로 전송됩니다	`/bin/generate_temp_api_key.sh`
`cleanupPeriodDays`	이 기간보다 오래 비활성 상태인 세션은 시작 시 삭제됩니다. `0` 으로 설정하면 모든 세션이 즉시 삭제됩니다. (기본값: 30일)	`20`
`companyAnnouncements`	시작 시 사용자에게 표시할 공지사항입니다. 여러 공지사항이 제공되면 무작위로 순환됩니다.	`["Welcome to Acme Corp! Review our code guidelines at docs.acme.com"]`
`env`	모든 세션에 적용될 환경 변수	`{"FOO": "bar"}`
`attribution`	git 커밋 및 풀 요청에 대한 속성을 사용자 정의합니다. 속성 설정을 참조하세요	`{"commit": "🤖 Generated with Claude Code", "pr": ""}`
`includeCoAuthoredBy`	**더 이상 사용되지 않음**: 대신 `attribution` 을 사용하세요. git 커밋 및 풀 요청에 `co-authored-by Claude` 바이라인을 포함할지 여부 (기본값: `true`)	`false`
`includeGitInstructions`	Claude의 시스템 프롬프트에 기본 제공 커밋 및 PR 워크플로우 지침을 포함합니다 (기본값: `true`). 예를 들어 자신의 git 워크플로우 skills을 사용할 때 이러한 지침을 제거하려면 `false` 로 설정하세요. `CLAUDE_CODE_DISABLE_GIT_INSTRUCTIONS` 환경 변수가 설정되면 이 설정보다 우선합니다	`false`
`permissions`	권한 구조는 아래 표를 참조하세요.	
`hooks`	라이프사이클 이벤트에서 실행할 사용자 정의 명령을 구성합니다. hooks 문서에서 형식을 참조하세요	hooks 참조

키	설명	예제
`disableAllHooks`	모든 hooks 및 사용자 정의 status line 비활성화	`true`
`allowManagedHooksOnly`	(Managed 설정만) 사용자, 프로젝트 및 플러그인 hooks 로드 방지. Managed hooks 및 SDK hooks만 허용합니다. Hook 구성 참조	`true`
`allowedHttpHookUrls`	HTTP hooks가 대상으로 할 수 있는 URL 패턴의 허용 목록입니다. `*`를 와일드카드로 지원합니다. 설정되면 일치하지 않는 URL을 가진 hooks는 차단됩니다. 정의되지 않음 = 제한 없음, 빈 배열 = 모든 HTTP hooks 차단. 배열은 설정 소스 전체에서 병합됩니다. Hook 구성 참조	`["https://hooks.example.com/*"]`
`httpHookAllowedEnvVars`	HTTP hooks가 헤더에 보간할 수 있는 환경 변수 이름의 허용 목록입니다. 설정되면 각 hook의 유효한 `allowedEnvVars`는 이 목록과의 교집합입니다. 정의되지 않음 = 제한 없음. 배열은 설정 소스 전체에서 병합됩니다. Hook 구성 참조	`["MY_TOKEN", "HOOK_SECRET"]`
`allowManagedPermissionRulesOnly`	(Managed 설정만) 사용자 및 프로젝트 설정이 `allow`, `ask` 또는 `deny` 권한 규칙을 정의하는 것을 방지합니다. Managed 설정의 규칙만 적용됩니다. Managed 전용 설정 참조	`true`
`allowManagedMcpServersOnly`	(Managed 설정만) Managed 설정의 `allowedMcpServers`만 존중됩니다. `deniedMcpServers`는 여전히 모든 소스에서 병합됩니다. 사용자는 여전히 MCP 서버를 추가할 수 있지만 관리자 정의 허용 목록만 적용됩니다. Managed MCP 구성 참조	`true`
`model`	Claude Code에 사용할 기본 모델을 재정의합니다	`"claude-sonnet-4-6"`

키	설명	예제
`availableModels`	`/model`, `--model`, Config 도구 또는 `ANTHROPIC_MODEL`을 통해 사용자가 선택할 수 있는 모델을 제한합니다. 기본 옵션에는 영향을 주지 않습니다. 모델 선택 제한 참조	`["sonnet", "haiku"]`
`modelOverrides`	Anthropic 모델 ID를 Bedrock 추론 프로필 ARN과 같은 공급자 특정 모델 ID로 매핑합니다. 각 모델 선택기 항목은 공급자 API를 호출할 때 매핑된 값을 사용합니다. 버전별 모델 ID 재정의 참조	`{"claude-opus-4-6": "arn:aws:bedrock:..."}`
`otelHeadersHelper`	동적 OpenTelemetry 헤더를 생성하는 스크립트입니다. 시작 시 및 주기적으로 실행됩니다 (동적 헤더 참조)	`/bin/generate_otel_headers.sh`
`statusLine`	컨텍스트를 표시하는 사용자 정의 상태 줄을 구성합니다. `statusLine` 문서 참조	`{"type": "command", "command": "~/.claude/statusline.sh"}`
`fileSuggestion`	`@` 파일 자동 완성을 위한 사용자 정의 스크립트를 구성합니다. 파일 제안 설정 참조	`{"type": "command", "command": "~/.claude/file-suggestion.sh"}`
`respectGitignore`	`@` 파일 선택기가 `.gitignore` 패턴을 존중할지 여부를 제어합니다. `true`(기본값)일 때 `.gitignore` 패턴과 일치하는 파일은 제안에서 제외됩니다	`false`
`outputStyle`	시스템 프롬프트를 조정하는 출력 스타일을 구성합니다. 출력 스타일 문서 참조	`"Explanatory"`
`forceLoginMethod`	`claudeai`를 사용하여 Claude.ai 계정으로만 로그인을 제한하거나, `console`을 사용하여 Claude Console (API 사용 청구) 계정으로만 로그인을 제한합니다	`claudeai`

키	설명	예제
`forceLoginOrgUUID`	로그인 중에 자동으로 선택할 조직의 UUID를 지정하여 조직 선택 단계를 건너뜁니다. `forceLoginMethod`가 설정되어야 합니다	`"xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"`
`enableAllProjectMcpServers`	프로젝트 `.mcp.json` 파일에 정의된 모든 MCP 서버를 자동으로 승인합니다	`true`
`enabledMcpjsonServers`	`.mcp.json` 파일에서 승인할 특정 MCP 서버 목록	`["memory", "github"]`
`disabledMcpjsonServers`	`.mcp.json` 파일에서 거부할 특정 MCP 서버 목록	`["filesystem"]`
`allowedMcpServers`	Managed 설정에서 설정되면 사용자가 구성할 수 있는 MCP 서버의 허용 목록입니다. 정의되지 않음 = 제한 없음, 빈 배열 = 잠금. 모든 범위에 적용됩니다. 거부 목록이 우선합니다. Managed MCP 구성 참조	`[{ "serverName": "github" }]`
`deniedMcpServers`	Managed 설정에서 설정되면 명시적으로 차단된 MCP 서버의 거부 목록입니다. Managed 서버를 포함한 모든 범위에 적용됩니다. 거부 목록이 허용 목록보다 우선합니다. Managed MCP 구성 참조	`[{ "serverName": "filesystem" }]`
`strictKnownMarketplaces`	Managed 설정에서 설정되면 사용자가 추가할 수 있는 플러그인 마켓플레이스의 허용 목록입니다. 정의되지 않음 = 제한 없음, 빈 배열 = 잠금. 마켓플레이스 추가에만 적용됩니다. Managed 마켓플레이스 제한 참조	`[{ "source": "github", "repo": "acme-corp/plugins" }]`
`blockedMarketplaces`	(Managed 설정만) 마켓플레이스 소스의 차단 목록입니다. 차단된 소스는 다운로드 전에 확인되므로 파일 시스템에 닿지 않습니다. Managed 마켓플레이스 제한 참조	`[{ "source": "github", "repo": "untrusted/plugins" }]`

키	설명	예제
`pluginTrustMessage`	(Managed 설정만) 설치 전에 표시되는 플러그인 신뢰 경고에 추가되는 사용자 정의 메시지입니다. 이를 사용하여 조직 특정 컨텍스트를 추가합니다. 예를 들어 내부 마켓플레이스의 플러그인이 검증되었음을 확인합니다.	`"All plugins from our marketplace are approved by IT"`
`awsAuthRefresh`	`.aws` 디렉토리를 수정하는 사용자 정의 스크립트 (고급 자격증명 구성 참조)	`aws sso login --profile myprofile`
`awsCredentialExport`	AWS 자격증명이 포함된 JSON을 출력하는 사용자 정의 스크립트 (고급 자격증명 구성 참조)	`/bin/generate_aws_grant.sh`
`alwaysThinkingEnabled`	모든 세션에 대해 기본적으로 확장 사고를 활성화합니다. 일반적으로 직접 편집하기보다는 `/config` 명령을 통해 구성됩니다	`true`
`plansDirectory`	계획 파일이 저장되는 위치를 사용자 정의합니다. 경로는 프로젝트 루트에 상대적입니다. 기본값: `~/.claude/plans`	`"./plans"`
`showTurnDuration`	응답 후 턴 지속 시간 메시지를 표시합니다 (예: “Cooked for 1m 6s”). 이러한 메시지를 숨기려면 `false` 로 설정하세요	`true`
`spinnerVerbs`	스피너 및 턴 지속 시간 메시지에 표시되는 작업 동사를 사용자 정의합니다. `mode` 를 `"replace"` 로 설정하여 동사만 사용하거나 `"append"` 로 설정하여 기본값에 추가합니다	`{"mode": "append", "verbs": ["Pondering", "Crafting"]}`
`language`	Claude의 선호 응답 언어를 구성합니다 (예: `"japanese"`, `"spanish"`, `"french"`). Claude는 기본적으로 이 언어로 응답합니다	`"japanese"`

키	설명	예제
`autoUpdatesChannel`	업데이트를 따를 릴리스 채널입니다. 일반적으로 약 1주일 된 버전이고 주요 회귀가 있는 버전을 건너뛰는 `"stable"` 을 사용하거나 가장 최근 릴리스인 `"latest"` (기본값)를 사용합니다	`"stable"`
`spinnerTipsEnabled`	Claude가 작업 중일 때 스피너에 팁을 표시합니다. 팁을 비활성화하려면 `false` 로 설정하세요 (기본값: `true`)	`false`
`spinnerTipsOverride`	스피너 팁을 사용자 정의 문자열로 재정의합니다. `tips` : 팁 문자열 배열. `excludeDefault` : `true` 이면 사용자 정의 팁만 표시하고, `false` 이거나 없으면 사용자 정의 팁이 기본 제공 팁과 병합됩니다	`{ "excludeDefault": true, "tips": ["Use our internal tool X"] }`
`terminalProgressBarEnabled`	Windows Terminal 및 iTerm2와 같은 지원되는 터미널에서 진행률을 표시하는 터미널 진행률 표시줄을 활성화합니다 (기본값: `true`)	`false`
`prefersReducedMotion`	접근성을 위해 UI 애니메이션 (스피너, shimmer, flash 효과) 감소 또는 비활성화	`true`
`fastModePerSessionOptIn`	`true` 일 때 빠른 모드는 세션 간에 지속되지 않습니다. 각 세션은 빠른 모드가 꺼진 상태로 시작되며 사용자가 `/fast` 로 활성화해야 합니다. 사용자의 빠른 모드 설정은 여전히 저장됩니다. 세션별 옵트인 필요 참조	`true`
`teammateMode`	에이전트 팀 팀원이 표시되는 방식: `auto` (tmux 또는 iTerm2에서 분할 창 선택, 그 외에는 in-process), `in-process` 또는 `tmux` . 에이전트 팀 설정 참조	`"in-process"`

권한 설정

키	설명	예제
`allow`	도구 사용을 허용하는 권한 규칙 배열입니다. 패턴 매칭 세부사항은 아래 권한 규칙 구문을 참조하세요	`[ "Bash(git diff *)" ]`
`ask`	도구 사용 시 확인을 요청하는 권한 규칙 배열입니다. 패턴 매칭 세부사항은 아래 권한 규칙 구문을 참조하세요	`[ "Bash(git push *)" ]`
`deny`	도구 사용을 거부하는 권한 규칙 배열입니다. 이를 사용하여 Claude Code 액세스에서 민감한 파일을 제외합니다. 권한 규칙 구문 및 Bash 권한 제한 참조	`[ "WebFetch", "Bash(curl *)", "Read(./.env)", "Read(./secrets/**)" ]`
`additionalDirectories`	Claude가 액세스할 수 있는 추가 작업 디렉토리	`[ "../docs/" ]`
`defaultMode`	Claude Code를 열 때 기본 권한 모드	`"acceptEdits"`
`disableBypassPermissionsMode`	`bypassPermissions` 모드가 활성화되는 것을 방지하려면 `"disable"` 로 설정합니다. 이는 `--dangerously-skip-permissions` 명령줄 플래그를 비활성화합니다. managed 설정 참조	`"disable"`

권한 규칙 구문

권한 규칙은 `Tool` 또는 `Tool(specifier)` 형식을 따릅니다. 규칙은 순서대로 평가됩니다: 먼저 거부 규칙, 그 다음 요청, 그 다음 허용. 첫 번째 일치 규칙이 우선합니다.

빠른 예제:

규칙	효과
`Bash`	모든 Bash 명령과 일치
`Bash(npm run *)`	`npm run` 으로 시작하는 명령과 일치
`Read(./.env)`	`.env` 파일 읽기와 일치
`WebFetch(domain:example.com)`	example.com으로의 fetch 요청과 일치

와일드카드 동작, Read, Edit, WebFetch, MCP 및 Agent 규칙에 대한 도구 특정 패턴, Bash 패턴의 보안 제한을 포함한 완전한 규칙 구문 참조는 권한 규칙 구문을 참조하세요.

Sandbox 설정

고급 샌드박싱 동작을 구성합니다. 샌드박싱은 bash 명령을 파일 시스템 및 네트워크에서 격리합니다. 자세한 내용은 Sandboxing을 참조하세요.

키	설명	예제
`enabled`	bash 샌드박싱 활성화 (macOS, Linux 및 WSL2). 기본값: false	`true`
`autoAllowBashIfSandboxed`	샌드박싱되면 bash 명령 자동 승인. 기본값: true	`true`
`excludedCommands`	샌드박스 외부에서 실행해야 할 명령	`["git", "docker"]`
`allowUnsandboxedCommands`	`dangerouslyDisableSandbox` 매개변수를 통해 샌드박스 외부에서 명령을 실행하도록 허용합니다. `false` 로 설정하면 `dangerouslyDisableSandbox` 이스케이프 해치가 완전히 비활성화되고 모든 명령은 샌드박싱되거나 `excludedCommands` 에 있어야 합니다. 엄격한 샌드박싱을 요구하는 엔터프라이즈 정책에 유용합니다. 기본값: true	`false`
`filesystem.allowWrite`	샌드박싱된 명령이 쓸 수 있는 추가 경로입니다. 배열은 모든 설정 범위에서 병합됩니다: 사용자, 프로젝트 및 managed 경로가 결합되고 대체되지 않습니다. `Edit(...)` 허용 권한 규칙의 경로와도 병합됩니다. 경로 접두사 아래를 참조하세요.	`["//tmp/build", "~/.kube"]`
`filesystem.denyWrite`	샌드박싱된 명령이 쓸 수 없는 경로입니다. 배열은 모든 설정 범위에서 병합됩니다. `Edit(...)` 거부 권한 규칙의 경로와도 병합됩니다.	`["//etc", "//usr/local/bin"]`

키	설명	예제
`filesystem.denyRead`	샌드박싱된 명령이 읽을 수 없는 경로입니다. 배열은 모든 설정 범위에서 병합됩니다. `Read(...)` 거부 권한 규칙의 경로와도 병합됩니다.	`["~/.aws/credentials"]`
`network.allowUnixSockets`	샌드박스에서 액세스 가능한 Unix 소켓 경로 (SSH 에이전트 등)	`["~/.ssh/agent-socket"]`
`network.allowAllUnixSockets`	샌드박스에서 모든 Unix 소켓 연결을 허용합니다. 기본값: false	`true`
`network.allowLocalBinding`	localhost 포트에 바인딩 허용 (macOS만). 기본값: false	`true`
`network.allowedDomains`	아웃바운드 네트워크 트래픽을 허용할 도메인 배열입니다. 와일드카드를 지원합니다 (예: `*.example.com`).	`["github.com", "*.npmjs.org"]`
`network.allowManagedDomainsOnly`	(Managed 설정만) Managed 설정의 `allowedDomains` 및 `WebFetch(domain:...)` 허용 규칙만 존중됩니다. 사용자, 프로젝트 및 local 설정의 도메인은 무시됩니다. 허용되지 않은 도메인은 사용자에게 메시지를 표시하지 않고 자동으로 차단됩니다. 거부된 도메인은 여전히 모든 소스에서 존중됩니다. 기본값: false	`true`
`network.httpProxyPort`	자신의 프록시를 가져오려는 경우 사용할 HTTP 프록시 포트입니다. 지정하지 않으면 Claude가 자신의 프록시를 실행합니다.	`8080`
`network.socksProxyPort`	자신의 프록시를 가져오려는 경우 사용할 SOCKS5 프록시 포트입니다. 지정하지 않으면 Claude가 자신의 프록시를 실행합니다.	`8081`

키	설명	예제
`enableWeakerNestedSandbox`	권한이 없는 Docker 환경에서 더 약한 샌드박스를 활성화합니다 (Linux 및 WSL2만). **보안을 감소시킵니다.** 기본값: false	`true`
`enableWeakerNetworkIsolation`	(macOS만) 샌드박스에서 시스템 TLS 신뢰 서비스 (`com.apple.trustd.agent`)에 대한 액세스를 허용합니다. `httpProxyPort` 와 함께 MITM 프록시 및 사용자 정의 CA를 사용할 때 `gh` , `gcloud` 및 `terraform` 과 같은 Go 기반 도구가 TLS 인증서를 확인하는 데 필요합니다. **보안을 감소시킵니다** 잠재적 데이터 유출 경로를 열어서. 기본값: false	`true`

Sandbox 경로 접두사

`filesystem.allowWrite` , `filesystem.denyWrite` 및 `filesystem.denyRead` 의 경로는 다음 접두사를 지원합니다:

접두사	의미	예제
`//`	파일 시스템 루트의 절대 경로	`//tmp/build` 는 `/tmp/build` 가 됨
`~/`	홈 디렉토리에 상대적	`~/.kube` 는 `$HOME/.kube` 가 됨
`/`	설정 파일의 디렉토리에 상대적	`/build` 는 `$SETTINGS_DIR/build` 가 됨
`./` 또는 접두사 없음	상대 경로 (샌드박스 런타임에서 해석됨)	`./output`

구성 예제:

```
{
  "sandbox": {
    "enabled": true,
    "autoAllowBashIfSandboxed": true,
    "excludedCommands": ["docker"],
    "filesystem": {
      "allowWrite": ["//tmp/build", "~/.kube"],
      "denyRead": ["~/.aws/credentials"]
    },
    "network": {
      "allowedDomains": ["github.com", "*.npmjs.org", "registry.yarnpkg.com"],
      "allowUnixSockets": [
        "/var/run/docker.sock"
      ],
      "allowLocalBinding": true
    }
  }
}
```

파일 시스템 및 네트워크 제한은 함께 병합되는 두 가지 방식으로 구성할 수 있습니다:

- `sandbox.filesystem` **설정** (위에 표시됨): OS 수준 샌드박스 경계에서 경로를 제어합니다. 이러한 제한은 Claude의 파일 도구뿐만 아니라 모든 서브프로세스 명령 (예: `kubectl`, `terraform`, `npm`)에 적용됩니다.
- **권한 규칙**: `Edit` 허용/거부 규칙을 사용하여 Claude의 파일 도구 액세스를 제어하고, `Read` 거부 규칙을 사용하여 읽기를 차단하고, `WebFetch` 허용/거부 규칙을 사용하여 네트워크 도메인을 제어합니다. 이러한 규칙의 경로도 샌드박스 구성에 병합됩니다.

속성 설정

Claude Code는 git 커밋 및 풀 요청에 속성을 추가합니다. 이들은 별도로 구성됩니다:

- 커밋은 기본적으로 git trailers (예: `Co-Authored-By`)를 사용하며 사용자 정의하거나 비활성화할 수 있습니다
- 풀 요청 설명은 일반 텍스트입니다

키	설명
`commit`	git 커밋에 대한 속성 (모든 trailers 포함). 빈 문자열은 커밋 속성을 숨깁니다

키	설명
`pr`	풀 요청 설명에 대한 속성입니다. 빈 문자열은 풀 요청 속성을 숨깁니다

기본 커밋 속성:

```
🤖 Generated with [Claude Code](https://claude.com/claude-code)

   Co-Authored-By: Claude Sonnet 4.6 <noreply@anthropic.com>
```

기본 풀 요청 속성:

```
🤖 Generated with [Claude Code](https://claude.com/claude-code)
```

예제:

```
{
  "attribution": {
    "commit": "Generated with AI\n\nCo-Authored-By: AI <ai@example.com>",
    "pr": ""
  }
}
```

> **Note:**
>
> `attribution` 설정은 더 이상 사용되지 않는 `includeCoAuthoredBy` 설정보다 우선합니다. 모든 속성을 숨기려면 `commit` 및 `pr`을 빈 문자열로 설정하세요.

파일 제안 설정

`@` 파일 경로 자동 완성을 위한 사용자 정의 명령을 구성합니다. 기본 제공 파일 제안은 빠른 파일 시스템 순회를 사용하지만 대규모 모노레포는 사전 구축된 파일 인덱스 또는 사용자 정의 도구와 같은 프로젝트 특정 인덱싱의 이점을 얻을 수 있습니다.

```
{
  "fileSuggestion": {
    "type": "command",
    "command": "~/.claude/file-suggestion.sh"
  }
}
```

명령은 `CLAUDE_PROJECT_DIR`을 포함한 hooks와 동일한 환경 변수로 실행됩니다. stdin을 통해 `query` 필드가 있는 JSON을 수신합니다:

```
{"query": "src/comp"}
```

stdout에 줄 바꿈으로 구분된 파일 경로를 출력합니다 (현재 15개로 제한됨):

```
src/components/Button.tsx
src/components/Modal.tsx
src/components/Form.tsx
```

예제:

```
#!/bin/bash
query=$(cat | jq -r '.query')
your-repo-file-index --query "$query" | head -20
```

Hook 구성

이러한 설정은 실행할 수 있는 hooks와 HTTP hooks가 액세스할 수 있는 것을 제어합니다. `allowManagedHooksOnly` 설정은 managed 설정에서만 구성할 수 있습니다. URL 및 env var 허용 목록은 모든 설정 수준에서 설정할 수 있으며 소스 전체에서 병합됩니다.

`allowManagedHooksOnly`가 `true`일 때의 동작:

- Managed hooks 및 SDK hooks가 로드됨
- 사용자 hooks, 프로젝트 hooks 및 플러그인 hooks가 차단됨

HTTP hook URL 제한:

HTTP hooks가 대상으로 할 수 있는 URL을 제한합니다. 일치를 위해 `*`를 와일드카드로 지원합니다. 배열이 정의되면 일치하지 않는 URL을 대상으로 하는 HTTP hooks는 자동으로 차단됩니다.

```
{
  "allowedHttpHookUrls": ["https://hooks.example.com/*", "http://localhost:*"]
}
```

HTTP hook 환경 변수 제한:

HTTP hooks가 헤더 값에 보간할 수 있는 환경 변수 이름을 제한합니다. 각 hook의 유효한 `allowedEnvVars`는 자신의 목록과 이 설정의 교집합입니다.

```
{
  "httpHookAllowedEnvVars": ["MY_TOKEN", "HOOK_SECRET"]
}
```

설정 우선순위

설정은 우선순위 순서대로 적용됩니다. 최상위에서 최하위로:

1. **Managed 설정** (서버 관리, MDM/OS 수준 정책 또는 managed 설정)
   - IT에서 서버 전달, MDM 구성 프로필, 레지스트리 정책 또는 managed 설정 파일을 통해 배포한 정책
   - 명령줄 인수를 포함한 다른 수준으로 재정의할 수 없음
   - Managed 계층 내에서 우선순위: 서버 관리 > MDM/OS 수준 정책 > `managed-settings.json` > HKCU 레지스트리 (Windows만). 하나의 managed 소스만 사용되며 소스는 병합되지 않습니다.
2. **명령줄 인수**
   - 특정 세션에 대한 임시 재정의
3. **Local 프로젝트 설정** (`.claude/settings.local.json`)
   - 개인 프로젝트 특정 설정
4. **공유 프로젝트 설정** (`.claude/settings.json`)
   - 소스 제어의 팀 공유 프로젝트 설정
5. **사용자 설정** (`~/.claude/settings.json`)
   - 개인 전역 설정

이 계층 구조는 조직 정책이 항상 적용되면서도 팀과 개인이 자신의 경험을 사용자 정의할 수 있도록 보장합니다.

예를 들어, 사용자 설정에서 `Bash(npm run *)` 을 허용하지만 프로젝트의 공유 설정에서 거부하면, 프로젝트 설정이 우선하고 명령이 차단됩니다.

> **Note:**
>
> **배열 설정은 범위 전체에서 병합됩니다.** 동일한 배열 값 설정 (예: `sandbox.filesystem.allowWrite` 또는 `permissions.allow`)이 여러 범위에 나타나면 배열은 **연결되고 중복 제거되며** 대체되지 않습니다. 이는 낮은 우선순위 범위가 높은 우선순위 범위에서 설정한 항목을 재정의하지 않고 항목을 추가할 수 있음을 의미합니다. 예를 들어, managed 설정이 `allowWrite` 를 `["//opt/company-tools"]` 로 설정하고 사용자가 `["~/.kube"]` 를 추가하면 두 경로 모두 최종 구성에 포함됩니다.

활성 설정 확인

Claude Code 내에서 `/status` 를 실행하여 활성 설정 소스와 출처를 확인합니다. 출력은 각 구성 계층 (managed, user, project)과 `Enterprise managed settings (remote)` , `Enterprise managed settings (plist)` , `Enterprise managed settings (HKLM)` 또는 `Enterprise managed settings (file)` 과 같은 출처를 표시합니다. 설정 파일에 오류가 있으면 `/status` 는 문제를 보고하여 수정할 수 있습니다.

구성 시스템의 핵심 사항

- **메모리 파일 (`CLAUDE.md`)**: Claude가 시작 시 로드하는 지침 및 컨텍스트를 포함합니다
- **설정 파일 (JSON)**: 권한, 환경 변수 및 도구 동작을 구성합니다
- **Skills**: `/skill-name` 으로 호출하거나 Claude가 자동으로 로드할 수 있는 사용자 정의 프롬프트
- **MCP servers**: 추가 도구 및 통합으로 Claude Code를 확장합니다
- **우선순위**: 높은 수준 구성 (Managed)이 낮은 수준 (User/Project)을 재정의합니다
- **상속**: 설정은 병합되며 더 구체적인 설정이 더 광범위한 설정을 추가하거나 재정의합니다

시스템 프롬프트

Claude Code의 내부 시스템 프롬프트는 게시되지 않습니다. 사용자 정의 지침을 추가하려면 `CLAUDE.md` 파일 또는 `--append-system-prompt` 플래그를 사용하세요.

민감한 파일 제외

API 키, 비밀 및 환경 파일과 같은 민감한 정보가 포함된 파일에서 Claude Code가 액세스하는 것을 방지하려면 `.claude/settings.json` 파일에서 `permissions.deny` 설정을 사용하세요:

```
{
  "permissions": {
    "deny": [
      "Read(./.env)",
      "Read(./.env.*)",
      "Read(./secrets/**)",
      "Read(./config/credentials.json)",
      "Read(./build)"
    ]
  }
}
```

이는 더 이상 사용되지 않는 `ignorePatterns` 구성을 대체합니다. 이러한 패턴과 일치하는 파일은 파일 검색 및 검색 결과에서 제외되며 이러한 파일에 대한 읽기 작업이 거부됩니다.

Subagent 구성

Claude Code는 사용자 및 프로젝트 수준에서 구성할 수 있는 사용자 정의 AI subagents를 지원합니다. 이러한 subagents는 YAML frontmatter가 있는 Markdown 파일로 저장됩니다:

- **사용자 subagents**: `~/.claude/agents/` - 모든 프로젝트에서 사용 가능
- **프로젝트 subagents**: `.claude/agents/` - 프로젝트에 특정이며 팀과 공유할 수 있음

Subagent 파일은 사용자 정의 프롬프트 및 도구 권한이 있는 특화된 AI 어시스턴트를 정의합니다. subagents 문서에서 subagents를 만들고 사용하는 방법에 대해 자세히 알아보세요.

플러그인 구성

Claude Code는 skills, agents, hooks 및 MCP servers로 기능을 확장할 수 있는 플러그인 시스템을 지원합니다. 플러그인은 마켓플레이스를 통해 배포되며 사용자 및 저장소 수준에서 구성할 수 있습니다.

플러그인 설정

`settings.json`의 플러그인 관련 설정:

```
{
  "enabledPlugins": {
    "formatter@acme-tools": true,
    "deployer@acme-tools": true,
    "analyzer@security-plugins": false
  },
  "extraKnownMarketplaces": {
    "acme-tools": {
      "source": "github",
      "repo": "acme-corp/claude-plugins"
    }
  }
}
```

`enabledPlugins`

활성화된 플러그인을 제어합니다. 형식: `"plugin-name@marketplace-name": true/false`

범위:

- **사용자 설정** (`~/.claude/settings.json`): 개인 플러그인 설정
- **프로젝트 설정** (`.claude/settings.json`): 팀과 공유되는 프로젝트 특정 플러그인
- **Local 설정** (`.claude/settings.local.json`): 머신별 재정의 (커밋되지 않음)

예제:

```
{
  "enabledPlugins": {
    "code-formatter@team-tools": true,
    "deployment-tools@team-tools": true,
    "experimental-features@personal": false
  }
}
```

`extraKnownMarketplaces`

저장소에서 사용 가능하게 해야 할 추가 마켓플레이스를 정의합니다. 일반적으로 팀 멤버가 필요한 플러그인 소스에 액세스할 수 있도록 저장소 수준 설정에서 사용됩니다.

저장소에 `extraKnownMarketplaces`가 포함되면:

1. 팀 멤버는 폴더를 신뢰할 때 마켓플레이스를 설치하라는 메시지를 받습니다
2. 그 다음 팀 멤버는 해당 마켓플레이스에서 플러그인을 설치하라는 메시지를 받습니다
3. 사용자는 원하지 않는 마켓플레이스 또는 플러그인을 건너뛸 수 있습니다 (사용자 설정에 저장됨)
4. 설치는 신뢰 경계를 존중하고 명시적 동의가 필요합니다

예제:

```
{
  "extraKnownMarketplaces": {
    "acme-tools": {
      "source": {
        "source": "github",
        "repo": "acme-corp/claude-plugins"
      }
    },
    "security-plugins": {
      "source": {
        "source": "git",
        "url": "https://git.example.com/security/plugins.git"
      }
    }
  }
}
```

마켓플레이스 소스 유형:

- `github`: GitHub 저장소 (`repo` 사용)
- `git`: 모든 git URL (`url` 사용)
- `directory`: 로컬 파일 시스템 경로 (`path` 사용, 개발 전용)
- `hostPattern`: 마켓플레이스 호스트와 일치하는 정규식 패턴 (`hostPattern` 사용)

`strictKnownMarketplaces`

Managed 설정만: 사용자가 추가할 수 있는 플러그인 마켓플레이스를 제어합니다. 이 설정은 managed 설정에서만 구성할 수 있으며 관리자에게 마켓플레이스 소스에 대한 엄격한 제어를 제공합니다.

Managed 설정 파일 위치:

- **macOS**: `/Library/Application Support/ClaudeCode/managed-settings.json`
- **Linux 및 WSL**: `/etc/claude-code/managed-settings.json`
- **Windows**: `C:\Program Files\ClaudeCode\managed-settings.json`

주요 특성:

- Managed 설정 (`managed-settings.json`)에서만 사용 가능
- 사용자 또는 프로젝트 설정으로 재정의할 수 없음 (최상위 우선순위)
- 네트워크/파일 시스템 작업 전에 적용됨 (차단된 소스는 실행되지 않음)
- `hostPattern` 을 제외한 소스 사양에 대해 정확한 일치를 사용합니다. `hostPattern` 은 정규식 일치를 사용합니다

허용 목록 동작:

- `undefined` (기본값): 제한 없음 - 사용자는 모든 마켓플레이스를 추가할 수 있음
- 빈 배열 `[]` : 완전 잠금 - 사용자는 새 마켓플레이스를 추가할 수 없음
- 소스 목록: 사용자는 정확히 일치하는 마켓플레이스만 추가할 수 있음

지원되는 모든 소스 유형:

허용 목록은 7가지 마켓플레이스 소스 유형을 지원합니다. 대부분의 소스는 정확한 일치를 사용하는 반면 `hostPattern` 은 마켓플레이스 호스트에 대해 정규식 일치를 사용합니다.

1. **GitHub 저장소**:

```
{ "source": "github", "repo": "acme-corp/approved-plugins" }
{ "source": "github", "repo": "acme-corp/security-tools", "ref": "v2.0" }
{ "source": "github", "repo": "acme-corp/plugins", "ref": "main", "path": "marketp
lace" }
```

필드: `repo` (필수), `ref` (선택: 분기/태그/SHA), `path` (선택: 하위 디렉토리)

1. **Git 저장소**:

```
{ "source": "git", "url": "https://gitlab.example.com/tools/plugins.git" }
{ "source": "git", "url": "https://bitbucket.org/acme-corp/plugins.git", "ref": "p
roduction" }
{ "source": "git", "url": "ssh://git@git.example.com/plugins.git", "ref": "v3.1",
"path": "approved" }
```

필드: `url` (필수), `ref` (선택: 분기/태그/SHA), `path` (선택: 하위 디렉토리)

1. **URL 기반 마켓플레이스**:

```
{ "source": "url", "url": "https://plugins.example.com/marketplace.json" }
{ "source": "url", "url": "https://cdn.example.com/marketplace.json", "headers":
{ "Authorization": "Bearer ${TOKEN}" } }
```

필드: `url` (필수), `headers` (선택: 인증된 액세스를 위한 HTTP 헤더)

> **Note:**
>
> URL 기반 마켓플레이스는 `marketplace.json` 파일만 다운로드합니다. 서버에서 플러그인 파일을 다운로드하지 않습니다. URL 기반 마켓플레이스의 플러그인은 상대 경로가 아닌 외부 소스 (GitHub, npm 또는 git URL)를 사용해야 합니다. 상대 경로가 있는 플러그인의 경우 대신 Git 기반 마켓플레이스를 사용하세요. 문제 해결에서 자세한 내용을 참조하세요.

1. **NPM 패키지**:

```
{ "source": "npm", "package": "@acme-corp/claude-plugins" }
{ "source": "npm", "package": "@acme-corp/approved-marketplace" }
```

필드: `package` (필수, 범위가 지정된 패키지 지원)

1. **파일 경로**:

```
{ "source": "file", "path": "/usr/local/share/claude/acme-marketplace.json" }
{ "source": "file", "path": "/opt/acme-corp/plugins/marketplace.json" }
```

필드: `path` (필수: marketplace.json 파일의 절대 경로)

1. **디렉토리 경로**:

```
{ "source": "directory", "path": "/usr/local/share/claude/acme-plugins" }
{ "source": "directory", "path": "/opt/acme-corp/approved-marketplaces" }
```

필드: `path` (필수: `.claude-plugin/marketplace.json`을 포함하는 디렉토리의 절대 경로)

1. **호스트 패턴 일치**:

```
{ "source": "hostPattern", "hostPattern": "^github\\.example\\.com$" }
{ "source": "hostPattern", "hostPattern": "^gitlab\\.internal\\.example\\.com$" }
```

필드: `hostPattern` (필수: 마켓플레이스 호스트와 일치하는 정규식 패턴)

각 저장소를 열거하지 않고 특정 호스트의 모든 마켓플레이스를 허용하려면 호스트 패턴 일치를 사용하세요. 이는 개발자가 자신의 마켓플레이스를 만드는 내부 GitHub Enterprise 또는 GitLab 서버가 있는 조직에 유용합니다.

소스 유형별 호스트 추출:

- `github` : 항상 `github.com` 에 대해 일치
- `git` : URL에서 호스트명 추출 (HTTPS 및 SSH 형식 지원)
- `url` : URL에서 호스트명 추출
- `npm` , `file` , `directory` : 호스트 패턴 일치에 지원되지 않음

구성 예제:

예제: 특정 마켓플레이스만 허용:

```
{
  "strictKnownMarketplaces": [
    {
      "source": "github",
      "repo": "acme-corp/approved-plugins"
    },
    {
      "source": "github",
      "repo": "acme-corp/security-tools",
      "ref": "v2.0"
    },
    {
      "source": "url",
      "url": "https://plugins.example.com/marketplace.json"
    },
    {
      "source": "npm",
      "package": "@acme-corp/compliance-plugins"
    }
  ]
}
```

예제 - 모든 마켓플레이스 추가 비활성화:

```
{
  "strictKnownMarketplaces": []
}
```

예제: 내부 git 서버의 모든 마켓플레이스 허용:

```
{
  "strictKnownMarketplaces": [
    {
      "source": "hostPattern",
      "hostPattern": "^github\\.example\\.com$"
    }
  ]
}
```

정확한 일치 요구사항:

마켓플레이스 소스는 사용자의 추가가 허용되려면 **정확히** 일치해야 합니다. Git 기반 소스(`github` 및 `git`)의 경우 모든 선택적 필드를 포함합니다:

- `repo` 또는 `url`이 정확히 일치해야 함
- `ref` 필드가 정확히 일치해야 함 (또는 둘 다 정의되지 않음)
- `path` 필드가 정확히 일치해야 함 (또는 둘 다 정의되지 않음)

일치하지 **않는** 소스의 예:

```
// 이들은 다른 소스입니다:
{ "source": "github", "repo": "acme-corp/plugins" }
{ "source": "github", "repo": "acme-corp/plugins", "ref": "main" }

// 이것도 다릅니다:
{ "source": "github", "repo": "acme-corp/plugins", "path": "marketplace" }
{ "source": "github", "repo": "acme-corp/plugins" }
```

`extraKnownMarketplaces`**와의 비교**:

측면	`strictKnownMarketplaces`	`extraKnownMarketplaces`
목적	조직 정책 적용	팀 편의
설정 파일	`managed-settings.json`만	모든 설정 파일
동작	허용 목록에 없는 추가 차단	누락된 마켓플레이스 자동 설치

측면	`strictKnownMarketplaces`	`extraKnownMarketplaces`
적용 시기	네트워크/파일 시스템 작업 전	사용자 신뢰 프롬프트 후
재정의 가능	아니오 (최상위 우선순위)	예 (높은 우선순위 설정으로)
소스 형식	직접 소스 객체	중첩된 소스가 있는 명명된 마켓플레이스
사용 사례	규정 준수, 보안 제한	온보딩, 표준화

형식 차이:

`strictKnownMarketplaces`는 직접 소스 객체를 사용합니다:

```
{
  "strictKnownMarketplaces": [
    { "source": "github", "repo": "acme-corp/plugins" }
  ]
}
```

`extraKnownMarketplaces`는 명명된 마켓플레이스가 필요합니다:

```
{
  "extraKnownMarketplaces": {
    "acme-tools": {
      "source": { "source": "github", "repo": "acme-corp/plugins" }
    }
  }
}
```

중요 참고사항:

- 제한은 네트워크 요청 또는 파일 시스템 작업 전에 확인됨
- 차단되면 사용자는 소스가 managed 정책으로 차단되었음을 나타내는 명확한 오류 메시지를 봅니다
- 제한은 새 마켓플레이스 추가에만 적용되며 이전에 설치된 마켓플레이스는 계속 액세스 가능합니다
- Managed 설정은 최상위 우선순위를 가지며 재정의할 수 없습니다

Managed 마켓플레이스 제한에서 사용자 대면 문서를 참조하세요.

플러그인 관리

`/plugin` 명령을 사용하여 플러그인을 대화형으로 관리합니다:

- 마켓플레이스에서 사용 가능한 플러그인 찾아보기
- 플러그인 설치/제거
- 플러그인 활성화/비활성화
- 플러그인 세부사항 보기 (제공되는 명령, agents, hooks)
- 마켓플레이스 추가/제거

플러그인 문서에서 플러그인 시스템에 대해 자세히 알아보세요.

환경 변수

Claude Code는 동작을 제어하기 위해 다음 환경 변수를 지원합니다:

> **Note:**
>
> 모든 환경 변수는 `settings.json` 에서도 구성할 수 있습니다. 이는 각 세션에 대해 환경 변수를 자동으로 설정하거나 전체 팀 또는 조직에 대해 환경 변수 집합을 배포하는 방법으로 유용합니다.

변수	목적	
`ANTHROPIC_API_KEY`	API 키로 `X-Api-Key` 헤더로 전송됨, 일반적으로 Claude SDK용 (대화형 사용의 경우 `/login` 실행)	
`ANTHROPIC_AUTH_TOKEN`	`Authorization` 헤더의 사용자 정의 값 (여기서 설정한 값은 `Bearer` 접두사가 붙음)	
`ANTHROPIC_CUSTOM_HEADERS`	요청에 추가할 사용자 정의 헤더 (`Name: Value` 형식, 여러 헤더의 경우 줄 바꿈으로 구분)	
`ANTHROPIC_DEFAULT_HAIKU_MODEL`	모델 구성 참조	
`ANTHROPIC_DEFAULT_OPUS_MODEL`	모델 구성 참조	

변수	목적	
`ANTHROPIC_DEFAULT_SONNET_MODEL`	모델 구성 참조	
`ANTHROPIC_FOUNDRY_API_KEY`	Microsoft Foundry 인증을 위한 API 키 (Microsoft Foundry 참조)	
`ANTHROPIC_FOUNDRY_BASE_URL`	Foundry 리소스의 전체 기본 URL (예: `https://my-resource.services.ai.azure.com/anthropic`). `ANTHROPIC_FOUNDRY_RESOURCE`의 대안 (Microsoft Foundry 참조)	
`ANTHROPIC_FOUNDRY_RESOURCE`	Foundry 리소스 이름 (예: `my-resource`). `ANTHROPIC_FOUNDRY_BASE_URL`이 설정되지 않은 경우 필수 (Microsoft Foundry 참조)	
`ANTHROPIC_MODEL`	사용할 모델 설정의 이름 (모델 구성 참조)	
`ANTHROPIC_SMALL_FAST_MODEL`	[더 이상 사용되지 않음] 백그라운드 작업을 위한 Haiku 클래스 모델의 이름	
`ANTHROPIC_SMALL_FAST_MODEL_AWS_REGION`	Bedrock을 사용할 때 Haiku 클래스 모델의 AWS 지역 재정의	
`AWS_BEARER_TOKEN_BEDROCK`	Bedrock API 인증을 위한 API 키 (Bedrock API 키 참조)	
`BASH_DEFAULT_TIMEOUT_MS`	장시간 실행되는 bash 명령의 기본 타임아웃	
`BASH_MAX_OUTPUT_LENGTH`	bash 출력이 중간 잘림되기 전의 최대 문자 수	
`BASH_MAX_TIMEOUT_MS`	모델이 장시간 실행되는 bash 명령에 대해 설정할 수 있는 최대 타임아웃	

변수	목적	
`CLAUDE_AUTOCOMPACT_PCT_OVERRIDE`	자동 압축이 트리거되는 컨텍스트 용량의 백분율 (1-100)을 설정합니다. 기본적으로 자동 압축은 약 95% 용량에서 트리거됩니다. 더 빨리 압축하려면 `50` 과 같은 낮은 값을 사용하세요. 기본 임계값보다 높은 값은 효과가 없습니다. 주 대화 및 subagents에 적용됩니다. 이 백분율은 status line에서 사용 가능한 `context_window.used_percentage` 필드와 일치합니다	
`CLAUDE_BASH_MAINTAIN_PROJECT_WORKING_DIR`	각 Bash 명령 후 원래 작업 디렉토리로 돌아갑니다	
`CLAUDE_CODE_ACCOUNT_UUID`	인증된 사용자의 계정 UUID입니다. SDK 호출자가 계정 정보를 동기적으로 제공하여 초기 원격 분석 이벤트가 계정 메타데이터를 갖지 않는 경쟁 조건을 피하는 데 사용됩니다. `CLAUDE_CODE_USER_EMAIL` 및 `CLAUDE_CODE_ORGANIZATION_UUID` 도 설정되어야 합니다	
`CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD`	`--add-dir` 로 지정된 디렉토리에서 CLAUDE.md 파일을 로드하려면 `1` 로 설정합니다. 기본적으로 추가 디렉토리는 메모리 파일을 로드하지 않습니다	`1`
`CLAUDE_CODE_API_KEY_HELPER_TTL_MS`	자격증명을 새로 고쳐야 하는 간격 (밀리초) (`apiKeyHelper` 사용 시)	
`CLAUDE_CODE_CLIENT_CERT`	mTLS 인증을 위한 클라이언트 인증서 파일의 경로	
`CLAUDE_CODE_CLIENT_KEY`	mTLS 인증을 위한 클라이언트 개인 키 파일의 경로	
`CLAUDE_CODE_CLIENT_KEY_PASSPHRASE`	암호화된 CLAUDE_CODE_CLIENT_KEY의 암호 (선택사항)	

변수	목적	
`CLAUDE_CODE_DISABLE_1M_CONTEXT`	1M 컨텍스트 윈도우 지원을 비활성화하려면 `1` 로 설정합니다. 설정되면 1M 모델 변형은 모델 선택기에서 사용할 수 없습니다. 규정 준수 요구사항이 있는 엔터프라이즈 환경에 유용합니다	
`CLAUDE_CODE_DISABLE_ADAPTIVE_THINKING`	Opus 4.6 및 Sonnet 4.6에 대해 적응형 추론을 비활성화하려면 `1` 로 설정합니다. 비활성화되면 이러한 모델은 `MAX_THINKING_TOKENS` 로 제어되는 고정 사고 예산으로 돌아갑니다	
`CLAUDE_CODE_DISABLE_AUTO_MEMORY`	자동 메모리를 비활성화하려면 `1` 로 설정합니다. 점진적 롤아웃 중에 자동 메모리를 강제로 켜려면 `0` 으로 설정합니다. 비활성화되면 Claude는 자동 메모리 파일을 생성하거나 로드하지 않습니다	
`CLAUDE_CODE_DISABLE_GIT_INSTRUCTIONS`	Claude의 시스템 프롬프트에서 기본 제공 커밋 및 PR 워크플로우 지침을 제거하려면 `1` 로 설정합니다. 자신의 git 워크플로우 skills을 사용할 때 유용합니다. 설정되면 `includeGitInstructions` 설정보다 우선합니다	
`CLAUDE_CODE_DISABLE_BACKGROUND_TASKS`	모든 백그라운드 작업 기능을 비활성화하려면 `1` 로 설정합니다. Bash 및 subagent 도구의 `run_in_background` 매개변수, 자동 백그라운드 처리 및 Ctrl+B 단축키 포함	
`CLAUDE_CODE_DISABLE_CRON`	예약된 작업을 비활성화하려면 `1` 로 설정합니다. `/loop` skill 및 cron 도구를 사용할 수 없게 되고 이미 예약된 작업은 중지되며, 세션 중에 이미 실행 중인 작업도 포함됩니다	

변수	목적	
`CLAUDE_CODE_DISABLE_EXPERIMENTAL_BETAS`	Anthropic API 특정 `anthropic-beta` 헤더를 비활성화하려면 `1` 로 설정합니다. LLM 게이트웨이와 함께 타사 공급자를 사용할 때 "Unexpected value(s) for the `anthropic-beta` header" 와 같은 문제가 발생하는 경우 사용하세요	
`CLAUDE_CODE_DISABLE_FAST_MODE`	빠른 모드를 비활성화하려면 `1` 로 설정합니다	
`CLAUDE_CODE_DISABLE_FEEDBACK_SURVEY`	"Claude가 어떻게 하고 있나요?" 세션 품질 설문조사를 비활성화하려면 `1` 로 설정합니다. 타사 공급자를 사용하거나 원격 분석이 비활성화되면 자동으로 비활성화됩니다. 세션 품질 설문조사 참조	
`CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC`	`DISABLE_AUTOUPDATER` , `DISABLE_BUG_COMMAND` , `DISABLE_ERROR_REPORTING` 및 `DISABLE_TELEMETRY` 설정과 동일	
`CLAUDE_CODE_DISABLE_TERMINAL_TITLE`	대화 컨텍스트를 기반으로 자동 터미널 제목 업데이트를 비활성화하려면 `1` 로 설정합니다	
`CLAUDE_CODE_EFFORT_LEVEL`	지원되는 모델의 노력 수준을 설정합니다. 값: `low` , `medium` , `high` . 낮은 노력은 더 빠르고 저렴하며, 높은 노력은 더 깊은 추론을 제공합니다. Opus 4.6 및 Sonnet 4.6에서 지원됩니다. 노력 수준 조정 참조	
`CLAUDE_CODE_ENABLE_PROMPT_SUGGESTION`	프롬프트 제안을 비활성화하려면 `false` 로 설정합니다 (`/config` 의 "프롬프트 제안" 토글). 이들은 Claude가 응답한 후 프롬프트 입력에 나타나는 회색으로 표시된 예측입니다. 프롬프트 제안 참조	

변수	목적	
`CLAUDE_CODE_ENABLE_TASKS`	이전 TODO 목록으로 임시로 되돌리려면 `false` 로 설정합니다. 기본값: `true` . 작업 목록 참조	
`CLAUDE_CODE_ENABLE_TELEMETRY`	OpenTelemetry 데이터 수집을 메트릭 및 로깅에 대해 활성화하려면 `1` 로 설정합니다. OTel 내보내기를 구성하기 전에 필요합니다. 모니터링 참조	
`CLAUDE_CODE_EXIT_AFTER_STOP_DELAY`	쿼리 루프가 유휴 상태가 된 후 자동으로 종료되기 전에 대기할 시간 (밀리초)입니다. 자동화된 워크플로우 및 SDK 모드를 사용하는 스크립트에 유용합니다	
`CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS`	에이전트 팀을 활성화하려면 `1` 로 설정합니다. 에이전트 팀은 실험적이며 기본적으로 비활성화됩니다	
`CLAUDE_CODE_FILE_READ_MAX_OUTPUT_TOKENS`	파일 읽기의 기본 토큰 제한을 재정의합니다. 전체 파일을 읽어야 할 때 유용합니다	
`CLAUDE_CODE_HIDE_ACCOUNT_INFO`	Claude Code UI에서 이메일 주소 및 조직 이름을 숨기려면 `1` 로 설정합니다. 스트리밍 또는 녹화할 때 유용합니다	
`CLAUDE_CODE_IDE_SKIP_AUTO_INSTALL`	IDE 확장 자동 설치 건너뛰기	
`CLAUDE_CODE_MAX_OUTPUT_TOKENS`	대부분의 요청에 대한 최대 출력 토큰 수를 설정합니다. 기본값: 32,000. 최대값: 64,000. 이 값을 증가시키면 자동 압축이 트리거되기 전에 사용 가능한 유효 컨텍스트 윈도우가 감소합니다.	
`CLAUDE_CODE_ORGANIZATION_UUID`	인증된 사용자의 조직 UUID입니다. SDK 호출자가 계정 정보를 동기적으로 제공하는 데 사용됩니다. `CLAUDE_CODE_ACCOUNT_UUID` 및 `CLAUDE_CODE_USER_EMAIL` 도 설정되어야 합니다	

변수	목적	
`CLAUDE_CODE_OTEL_HEADERS_HELPER_DEBOUNCE_MS`	동적 OpenTelemetry 헤더를 새로 고치는 간격 (밀리초) (기본값: 1740000 / 29분). 동적 헤더 참조	
`CLAUDE_CODE_PLAN_MODE_REQUIRED`	에이전트 팀 팀원이 계획 승인을 요구할 때 자동으로 `true` 로 설정됩니다. 읽기 전용: Claude Code가 팀원을 생성할 때 설정됩니다. 팀원에 대한 계획 승인 필요 참조	
`CLAUDE_CODE_PLUGIN_GIT_TIMEOUT_MS`	플러그인을 설치하거나 업데이트할 때 git 작업의 타임아웃 (밀리초) (기본값: 120000). 대규모 저장소 또는 느린 네트워크 연결의 경우 이 값을 증가시키세요. Git 작업 시간 초과 참조	
`CLAUDE_CODE_PROXY_RESOLVES_HOSTS`	프록시가 호출자 대신 DNS 해석을 수행하도록 허용하려면 `true` 로 설정합니다. 프록시가 호스트명 해석을 처리해야 하는 환경에서 옵트인합니다	
`CLAUDE_CODE_SHELL`	자동 셸 감지를 재정의합니다. 로그인 셸이 선호하는 작업 셸과 다를 때 유용합니다 (예: `bash` vs `zsh`)	
`CLAUDE_CODE_SHELL_PREFIX`	모든 bash 명령을 래핑할 명령 접두사 (예: 로깅 또는 감사용). 예: `/path/to/logger.sh` 는 `/path/to/logger.sh <command>` 를 실행합니다	
`CLAUDE_CODE_SIMPLE`	최소 시스템 프롬프트 및 Bash, 파일 읽기 및 파일 편집 도구만으로 실행하려면 `1` 로 설정합니다. MCP 도구, 첨부 파일, hooks 및 CLAUDE.md 파일을 비활성화합니다	
`CLAUDE_CODE_SKIP_BEDROCK_AUTH`	Bedrock에 대한 AWS 인증을 건너뜁니다 (예: LLM 게이트웨이를 사용할 때)	
`CLAUDE_CODE_SKIP_FOUNDRY_AUTH`	Microsoft Foundry에 대한 Azure 인증을 건너뜁니다 (예: LLM 게이트웨이를 사용할 때)	

변수	목적	
`CLAUDE_CODE_SKIP_VERTEX_AUTH`	Vertex에 대한 Google 인증을 건너뜁니다 (예: LLM 게이트웨이를 사용할 때)	
`CLAUDE_CODE_SUBAGENT_MODEL`	모델 구성 참조	
`CLAUDE_CODE_TASK_LIST_ID`	세션 간에 작업 목록을 공유합니다. 여러 Claude Code 인스턴스에서 동일한 ID를 설정하여 공유 작업 목록을 조정합니다. 작업 목록 참조	
`CLAUDE_CODE_TEAM_NAME`	이 팀원이 속한 에이전트 팀의 이름입니다. 에이전트 팀 멤버에서 자동으로 설정됩니다	
`CLAUDE_CODE_TMPDIR`	내부 임시 파일에 사용할 임시 디렉토리를 재정의합니다. Claude Code는 이 경로에 `/claude/`를 추가합니다. 기본값: Unix/macOS에서 `/tmp`, Windows에서 `os.tmpdir()`	
`CLAUDE_CODE_USER_EMAIL`	인증된 사용자의 이메일 주소입니다. SDK 호출자가 계정 정보를 동기적으로 제공하는 데 사용됩니다. `CLAUDE_CODE_ACCOUNT_UUID` 및 `CLAUDE_CODE_ORGANIZATION_UUID`도 설정되어야 합니다	
`CLAUDE_CODE_USE_BEDROCK`	Bedrock 사용	
`CLAUDE_CODE_USE_FOUNDRY`	Microsoft Foundry 사용	
`CLAUDE_CODE_USE_VERTEX`	Vertex 사용	
`CLAUDE_CONFIG_DIR`	Claude Code가 구성 및 데이터 파일을 저장하는 위치를 사용자 정의합니다	
`DISABLE_AUTOUPDATER`	자동 업데이트를 비활성화하려면 `1`로 설정합니다.	

변수	목적	
`DISABLE_BUG_COMMAND`	`/bug` 명령을 비활성화하려면 `1` 로 설정합니다	
`DISABLE_COST_WARNINGS`	비용 경고 메시지를 비활성화하려면 `1` 로 설정합니다	
`DISABLE_ERROR_REPORTING`	Sentry 오류 보고를 거부하려면 `1` 로 설정합니다	
`DISABLE_INSTALLATION_CHECKS`	설치 경고를 비활성화하려면 `1` 로 설정합니다. 설치 위치를 수동으로 관리할 때만 사용하세요. 표준 설치의 문제를 숨길 수 있습니다	
`DISABLE_NON_ESSENTIAL_MODEL_CALLS`	맛 텍스트와 같은 중요하지 않은 경로에 대한 모델 호출을 비활성화하려면 `1` 로 설정합니다	
`DISABLE_PROMPT_CACHING`	모든 모델에 대해 prompt caching을 비활성화하려면 `1` 로 설정합니다 (모델별 설정보다 우선)	
`DISABLE_PROMPT_CACHING_HAIKU`	Haiku 모델에 대해 prompt caching을 비활성화하려면 `1` 로 설정합니다	
`DISABLE_PROMPT_CACHING_OPUS`	Opus 모델에 대해 prompt caching을 비활성화하려면 `1` 로 설정합니다	
`DISABLE_PROMPT_CACHING_SONNET`	Sonnet 모델에 대해 prompt caching을 비활성화하려면 `1` 로 설정합니다	
`DISABLE_TELEMETRY`	Statsig 원격 분석을 거부하려면 `1` 로 설정합니다 (Statsig 이벤트는 코드, 파일 경로 또는 bash 명령과 같은 사용자 데이터를 포함하지 않음)	
`ENABLE_CLAUDEAI_MCP_SERVERS`	Claude Code에서 claude.ai MCP servers를 비활성화하려면 `false` 로 설정합니다. 로그인한 사용자에 대해 기본적으로 활성화됨	

변수	목적	
`ENABLE_TOOL_SEARCH`	MCP 도구 검색을 제어합니다. 값: `auto` (기본값, 10% 컨텍스트에서 활성화), `auto:N` (사용자 정의 임계값, 예: 5%의 경우 `auto:5`), `true` (항상 켜짐), `false` (비활성화)	
`FORCE_AUTOUPDATE_PLUGINS`	주 자동 업데이터가 `DISABLE_AUTOUPDATER`를 통해 비활성화된 경우에도 플러그인 자동 업데이트를 강제하려면 `true`로 설정합니다	
`HTTP_PROXY`	네트워크 연결을 위한 HTTP 프록시 서버를 지정합니다	
`HTTPS_PROXY`	네트워크 연결을 위한 HTTPS 프록시 서버를 지정합니다	
`IS_DEMO`	데모 모드를 활성화하려면 `true`로 설정합니다: UI에서 이메일 및 조직을 숨기고, 온보딩을 건너뛰고, 내부 명령을 숨깁니다. 스트리밍 또는 녹화 세션에 유용합니다	
`MAX_MCP_OUTPUT_TOKENS`	MCP 도구 응답에서 허용되는 최대 토큰 수입니다. Claude Code는 출력이 10,000 토큰을 초과할 때 경고를 표시합니다 (기본값: 25000)	
`MAX_THINKING_TOKENS`	확장 사고 토큰 예산을 재정의합니다. 사고는 기본적으로 최대 예산 (31,999 토큰)에서 활성화됩니다. 예산을 제한하려면 (예: `MAX_THINKING_TOKENS=10000`) 또는 사고를 완전히 비활성화하려면 (`MAX_THINKING_TOKENS=0`) 사용하세요. Opus 4.6의 경우 사고 깊이는 대신 노력 수준으로 제어되며 `0`으로 설정하여 사고를 비활성화하지 않는 한 이 변수는 무시됩니다.	

변수	목적	
`MCP_CLIENT_SECRET`	사전 구성된 자격증명이 필요한 MCP 서버에 대한 OAuth 클라이언트 비밀입니다. MCP 서버를 `--client-secret` 으로 추가할 때 대화형 프롬프트를 피합니다	
`MCP_OAUTH_CALLBACK_PORT`	사전 구성된 자격증명으로 MCP 서버를 추가할 때 `--callback-port` 의 대안으로 OAuth 리디렉션 콜백의 고정 포트	
`MCP_TIMEOUT`	MCP 서버 시작의 타임아웃 (밀리초)	
`MCP_TOOL_TIMEOUT`	MCP 도구 실행의 타임아웃 (밀리초)	
`NO_PROXY`	프록시를 우회하여 직접 발급될 요청의 도메인 및 IP 목록	
`SLASH_COMMAND_TOOL_CHAR_BUDGET`	Skill 도구에 표시되는 skill 메타데이터의 문자 예산을 재정의합니다. 예산은 컨텍스트 윈도우의 2%에서 동적으로 확장되며 16,000 문자의 폴백이 있습니다. 이전 버전과의 호환성을 위해 레거시 이름 유지	
`USE_BUILTIN_RIPGREP`	Claude Code에 포함된 `rg` 대신 시스템 설치 `rg` 를 사용하려면 `0` 으로 설정합니다	
`VERTEX_REGION_CLAUDE_3_5_HAIKU`	Vertex AI를 사용할 때 Claude 3.5 Haiku의 지역 재정의	
`VERTEX_REGION_CLAUDE_3_7_SONNET`	Vertex AI를 사용할 때 Claude 3.7 Sonnet의 지역 재정의	
`VERTEX_REGION_CLAUDE_4_0_OPUS`	Vertex AI를 사용할 때 Claude 4.0 Opus의 지역 재정의	
`VERTEX_REGION_CLAUDE_4_0_SONNET`	Vertex AI를 사용할 때 Claude 4.0 Sonnet의 지역 재정의	
`VERTEX_REGION_CLAUDE_4_1_OPUS`	Vertex AI를 사용할 때 Claude 4.1 Opus의 지역 재정의	

Claude가 사용할 수 있는 도구

Claude Code는 코드베이스를 이해하고 수정하는 데 도움이 되는 강력한 도구 집합에 액세스할 수 있습니다:

도구	설명	권한 필요
Agent	작업을 처리하기 위해 자신의 컨텍스트 윈도우가 있는 subagent를 생성합니다	아니오
AskUserQuestion	요구사항을 수집하거나 모호함을 명확히 하기 위해 객관식 질문을 합니다	아니오
Bash	환경에서 셸 명령을 실행합니다. Bash 도구 동작 참조	예
CronCreate	현재 세션 내에서 반복 또는 일회성 프롬프트를 예약합니다 (Claude가 종료되면 사라짐). 예약된 작업 참조	아니오
CronDelete	ID로 예약된 작업을 취소합니다	아니오
CronList	세션의 모든 예약된 작업을 나열합니다	아니오
Edit	특정 파일에 대한 대상 편집을 수행합니다	예
EnterPlanMode	코딩 전에 접근 방식을 설계하기 위해 계획 모드로 전환합니다	아니오
EnterWorktree	격리된 git worktree를 생성하고 전환합니다	아니오
ExitPlanMode	승인을 위해 계획을 제시하고 계획 모드를 종료합니다	예
ExitWorktree	worktree 세션을 종료하고 원래 디렉토리로 돌아갑니다	아니오
Glob	패턴 매칭을 기반으로 파일을 찾습니다	아니오
Grep	파일 내용에서 패턴을 검색합니다	아니오
ListMcpResourcesTool	연결된 MCP servers에서 노출된 리소스를 나열합니다	아니오

도구	설명	권한 필요
LSP	언어 서버를 통한 코드 인텔리전스입니다. 파일 편집 후 유형 오류 및 경고를 자동으로 보고합니다. 또한 탐색 작업을 지원합니다: 정의로 이동, 참조 찾기, 유형 정보 가져오기, 기호 나열, 구현 찾기, 호출 계층 추적. 코드 인텔리전스 플러그인과 해당 언어 서버 바이너리가 필요합니다	아니오
NotebookEdit	Jupyter 노트북 셀을 수정합니다	예
Read	파일의 내용을 읽습니다	아니오
ReadMcpResourceTool	URI로 특정 MCP 리소스를 읽습니다	아니오
Skill	주 대화 내에서 skill을 실행합니다	예
TaskCreate	작업 목록에 새 작업을 생성합니다	아니오
TaskGet	특정 작업의 전체 세부사항을 검색합니다	아니오
TaskList	현재 상태가 있는 모든 작업을 나열합니다	아니오
TaskOutput	백그라운드 작업에서 출력을 검색합니다	아니오
TaskStop	ID로 실행 중인 백그라운드 작업을 중지합니다	아니오
TaskUpdate	작업 상태, 종속성, 세부사항을 업데이트하거나 작업을 삭제합니다	아니오
TodoWrite	세션 작업 체크리스트를 관리합니다. 비대화형 모드 및 Agent SDK에서 사용 가능합니다. 대화형 세션은 대신 TaskCreate, TaskGet, TaskList 및 TaskUpdate를 사용합니다	아니오
ToolSearch	도구 검색이 활성화되면 지연된 도구를 검색하고 로드합니다	아니오
WebFetch	지정된 URL에서 콘텐츠를 가져옵니다	예
WebSearch	웹 검색을 수행합니다	예

도구	설명	권한 필요
Write	파일을 생성하거나 덮어씁니다	예

권한 규칙은 `/allowed-tools`를 사용하거나 권한 설정에서 구성할 수 있습니다. 도구 특정 권한 규칙도 참조하세요.

Bash 도구 동작

Bash 도구는 다음 지속성 동작으로 셸 명령을 실행합니다:

- **작업 디렉토리 지속**: Claude가 작업 디렉토리를 변경하면 (예: `cd /path/to/dir`), 후속 Bash 명령은 해당 디렉토리에서 실행됩니다.
  `CLAUDE_BASH_MAINTAIN_PROJECT_WORKING_DIR=1`을 사용하여 각 명령 후 프로젝트 디렉토리로 재설정할 수 있습니다.
- **환경 변수는 지속되지 않음**: 한 Bash 명령에서 설정된 환경 변수 (예: `export MY_VAR=value`)는 **후속 Bash 명령에서 사용할 수 없습니다**. 각 Bash 명령은 새로운 셸 환경에서 실행됩니다.

Bash 명령에서 환경 변수를 사용 가능하게 하려면 **3가지 옵션**이 있습니다:

옵션 1: Claude Code를 시작하기 전에 환경 활성화 (가장 간단한 접근)

Claude Code를 시작하기 전에 터미널에서 가상 환경을 활성화합니다:

```
conda activate myenv
## 또는: source /path/to/venv/bin/activate
claude
```

이는 셸 환경에서 작동하지만 Claude의 Bash 명령 내에서 설정된 환경 변수는 명령 간에 지속되지 않습니다.

옵션 2: Claude Code를 시작하기 전에 CLAUDE_ENV_FILE 설정 (지속적인 환경 설정)

환경 설정을 포함하는 셸 스크립트의 경로를 내보냅니다:

```
export CLAUDE_ENV_FILE=/path/to/env-setup.sh
claude
```

여기서 `/path/to/env-setup.sh`에는:

```
conda activate myenv
## 또는: source /path/to/venv/bin/activate
## 또는: export MY_VAR=value
```

Claude Code는 각 Bash 명령 전에 이 파일을 소싱하여 모든 명령에서 환경을 지속시킵니다.

옵션 3: SessionStart hook 사용 (프로젝트 특정 구성)

`.claude/settings.json`에서 구성합니다:

```
{
  "hooks": {
    "SessionStart": [{
      "matcher": "startup",
      "hooks": [{
        "type": "command",
        "command": "echo 'conda activate myenv' >> \"$CLAUDE_ENV_FILE\""
      }]
    }]
  }
}
```

Hook은 `$CLAUDE_ENV_FILE`에 쓰며, 이는 각 Bash 명령 전에 소싱됩니다. 이는 팀 공유 프로젝트 구성에 이상적입니다.

SessionStart hooks에서 옵션 3에 대한 자세한 내용을 참조하세요.

hooks로 도구 확장

Claude Code hooks를 사용하여 모든 도구 실행 전후에 사용자 정의 명령을 실행할 수 있습니다.

예를 들어 Claude가 Python 파일을 수정한 후 Python 포매터를 자동으로 실행하거나 특정 경로에 대한 Write 작업을 차단하여 프로덕션 구성 파일 수정을 방지할 수 있습니다.

참고 항목

- 권한: 권한 시스템, 규칙 구문, 도구 특정 패턴 및 managed 정책
- 인증: Claude Code에 대한 사용자 액세스 설정
- 문제 해결: 일반적인 구성 문제에 대한 솔루션

모델 구성

> Claude Code 모델 구성에 대해 알아보기, opusplan과 같은 모델 별칭 포함

사용 가능한 모델

Claude Code의 `model` 설정에서 다음 중 하나를 구성할 수 있습니다:

- **모델 별칭**
- **모델 이름**
  - Anthropic API: 전체 모델 이름
  - Bedrock: 추론 프로필 ARN
  - Foundry: 배포 이름
  - Vertex: 버전 이름

모델 별칭

모델 별칭은 정확한 버전 번호를 기억할 필요 없이 모델 설정을 선택할 수 있는 편리한 방법을 제공합니다:

모델 별칭	동작
`default`	계정 유형에 따른 권장 모델 설정
`sonnet`	일일 코딩 작업을 위해 최신 Sonnet 모델(현재 Sonnet 4.6) 사용
`opus`	복잡한 추론 작업을 위해 최신 Opus 모델(현재 Opus 4.6) 사용
`haiku`	간단한 작업을 위해 빠르고 효율적인 Haiku 모델 사용
`sonnet[1m]`	긴 세션을 위해 100만 토큰 컨텍스트 윈도우를 사용하는 Sonnet 사용
`opusplan`	Plan Mode 중에 `opus` 를 사용한 후 실행을 위해 `sonnet` 으로 전환하는 특수 모드

별칭은 항상 최신 버전을 가리킵니다. 특정 버전으로 고정하려면 전체 모델 이름(예: `claude-opus-4-6`)을 사용하거나 `ANTHROPIC_DEFAULT_OPUS_MODEL` 과 같은 해당 환경 변수를 설정합니다.

모델 설정

다음과 같은 여러 방법으로 모델을 구성할 수 있으며, 우선순위 순서대로 나열되어 있습니다:

1. **세션 중** - `/model <alias|name>` 을 사용하여 세션 중에 모델 전환
2. **시작 시** - `claude --model <alias|name>` 으로 실행
3. **환경 변수** - `ANTHROPIC_MODEL=<alias|name>` 설정
4. **설정** - `model` 필드를 사용하여 설정 파일에서 영구적으로 구성

사용 예시:

```
## Opus로 시작
claude --model opus

## 세션 중에 Sonnet으로 전환
/model sonnet
```

설정 파일 예시:

```
{
    "permissions": {
        ...
    },
    "model": "opus"
}
```

모델 선택 제한

엔터프라이즈 관리자는 관리 또는 정책 설정에서 `availableModels` 을 사용하여 사용자가 선택할 수 있는 모델을 제한할 수 있습니다.

`availableModels` 이 설정되면 사용자는 `/model`, `--model` 플래그, Config 도구 또는 `ANTHROPIC_MODEL` 환경 변수를 통해 목록에 없는 모델로 전환할 수 없습니다.

```
{
  "availableModels": ["sonnet", "haiku"]
}
```

기본 모델 동작

모델 선택기의 Default 옵션은 `availableModels`의 영향을 받지 않습니다. 항상 사용 가능하며 사용자의 구독 계층에 따른 시스템의 런타임 기본값을 나타냅니다.

`availableModels: []`인 경우에도 사용자는 자신의 계층에 대한 Default 모델로 Claude Code를 사용할 수 있습니다.

사용자가 실행하는 모델 제어

모델 경험을 완전히 제어하려면 `availableModels`과 `model` 설정을 함께 사용합니다:

- **availableModels**: 사용자가 전환할 수 있는 항목 제한
- **model**: Default를 재정의하는 명시적 모델 설정

이 예시는 모든 사용자가 Sonnet 4.6을 실행하고 Sonnet과 Haiku 중에서만 선택할 수 있도록 보장합니다:

```
{
  "model": "sonnet",
  "availableModels": ["sonnet", "haiku"]
}
```

병합 동작

`availableModels`이 사용자 설정 및 프로젝트 설정과 같은 여러 수준에서 설정되면 배열이 병합되고 중복이 제거됩니다. 엄격한 허용 목록을 적용하려면 가장 높은 우선순위를 가지는 관리 또는 정책 설정에서 `availableModels`을 설정합니다.

특수 모델 동작

`default` 모델 설정

`default`의 동작은 계정 유형에 따라 다릅니다:

- **Max 및 Team Premium**: Opus 4.6으로 기본 설정
- **Pro 및 Team Standard**: Sonnet 4.6으로 기본 설정
- **Enterprise**: Opus 4.6을 사용할 수 있지만 기본값이 아님

Claude Code는 Opus의 사용 임계값에 도달하면 자동으로 Sonnet으로 폴백할 수 있습니다.

`opusplan` 모델 설정

`opusplan` 모델 별칭은 자동화된 하이브리드 접근 방식을 제공합니다:

- **Plan Mode에서** - 복잡한 추론 및 아키텍처 결정을 위해 `opus` 사용
- **실행 Mode에서** - 코드 생성 및 구현을 위해 자동으로 `sonnet`으로 전환

이는 두 가지 장점을 모두 제공합니다: 계획을 위한 Opus의 우수한 추론 능력과 실행을 위한 Sonnet의 효율성.

노력 수준 조정

노력 수준은 적응형 추론을 제어하며, 작업 복잡도에 따라 동적으로 사고를 할당합니다. 낮은 노력은 간단한 작업의 경우 더 빠르고 저렴하며, 높은 노력은 복잡한 문제에 대해 더 깊은 추론을 제공합니다.

세 가지 수준을 사용할 수 있습니다: **low**, **medium**, **high**. Opus 4.6은 Max 및 Team 구독자를 위해 기본적으로 중간 노력으로 설정됩니다.

노력 설정:

- **`/model` 에서**: 모델을 선택할 때 좌우 화살표 키를 사용하여 노력 슬라이더 조정
- **환경 변수**: `CLAUDE_CODE_EFFORT_LEVEL=low|medium|high` 설정
- **설정**: 설정 파일에서 `effortLevel` 설정

노력은 Opus 4.6 및 Sonnet 4.6에서 지원됩니다. 지원되는 모델이 선택되면 `/model`에 노력 슬라이더가 나타납니다. 현재 노력 수준은 로고 및 스피너 옆에도 표시되므로(예: "with low effort"), `/model`을 열지 않고도 어떤 설정이 활성화되어 있는지 확인할 수 있습니다.

Opus 4.6 및 Sonnet 4.6에서 적응형 추론을 비활성화하고 이전의 고정 사고 예산으로 되돌리려면 `CLAUDE_CODE_DISABLE_ADAPTIVE_THINKING=1`을 설정합니다. 비활성화되면 이러한 모델은 `MAX_THINKING_TOKENS`로 제어되는 고정 예산을 사용합니다. 환경 변수를 참조하세요.

확장 컨텍스트

Opus 4.6 및 Sonnet 4.6은 대규모 코드베이스를 사용한 긴 세션을 위해 100만 토큰 컨텍스트 윈도우를 지원합니다.

> **Note:**
>
> 1M 컨텍스트 윈도우는 현재 베타 버전입니다. 기능, 가격 책정 및 가용성이 변경될 수 있습니다.

확장 컨텍스트는 다음에서 사용 가능합니다:

- **API 및 종량제 사용자**: 1M 컨텍스트에 대한 전체 액세스

- **Pro, Max, Teams 및 Enterprise 구독자**: 추가 사용이 활성화된 경우 사용 가능

1M 컨텍스트를 완전히 비활성화하려면 `CLAUDE_CODE_DISABLE_1M_CONTEXT=1` 을 설정합니다. 이는 모델 선택기에서 1M 모델 변형을 제거합니다. 환경 변수를 참조하세요.

1M 모델을 선택해도 청구가 즉시 변경되지 않습니다. 세션은 컨텍스트가 200K 토큰을 초과할 때까지 표준 요금을 사용합니다. 200K 토큰을 초과하면 요청은 장문 컨텍스트 가격 책정과 전용 속도 제한으로 청구됩니다. 구독자의 경우 200K를 초과하는 토큰은 구독을 통해서가 아니라 추가 사용으로 청구됩니다.

계정이 1M 컨텍스트를 지원하면 최신 버전의 Claude Code에서 모델 선택기(`/model`)에 옵션이 나타납니다. 표시되지 않으면 세션을 다시 시작해 보세요.

모델 별칭 또는 전체 모델 이름과 함께 `[1m]` 접미사를 사용할 수도 있습니다:

```
## sonnet[1m] 별칭 사용
/model sonnet[1m]

## 또는 전체 모델 이름에 [1m] 추가
/model claude-sonnet-4-6[1m]
```

현재 모델 확인

다음과 같은 여러 방법으로 현재 사용 중인 모델을 확인할 수 있습니다:

1. 상태 줄에서(구성된 경우)
2. `/status` 에서, 계정 정보도 표시합니다.

환경 변수

다음 환경 변수를 사용할 수 있으며, 이는 별칭이 매핑되는 모델 이름을 제어하기 위해 전체 **모델 이름**(또는 API 제공자에 해당하는 이름)이어야 합니다.

환경 변수	설명
`ANTHROPIC_DEFAULT_OPUS_MODEL`	`opus` 에 사용할 모델, 또는 Plan Mode가 활성화되었을 때 `opusplan` 에 사용할 모델.
`ANTHROPIC_DEFAULT_SONNET_MODEL`	`sonnet` 에 사용할 모델, 또는 Plan Mode가 활성화되지 않았을 때 `opusplan` 에 사용할 모델.
`ANTHROPIC_DEFAULT_HAIKU_MODEL`	`haiku` 에 사용할 모델, 또는 백그라운드 기능에 사용할 모델

환경 변수	설명
`CLAUDE_CODE_SUBAGENT_MODEL`	subagents에 사용할 모델

참고: `ANTHROPIC_SMALL_FAST_MODEL`은 `ANTHROPIC_DEFAULT_HAIKU_MODEL`을 위해 더 이상 사용되지 않습니다.

타사 배포를 위한 모델 고정

Bedrock, Vertex AI 또는 Foundry를 통해 Claude Code를 배포할 때 사용자에게 롤아웃하기 전에 모델 버전을 고정합니다.

고정하지 않으면 Claude Code는 최신 버전으로 확인되는 모델 별칭(`sonnet`, `opus`, `haiku`)을 사용합니다. Anthropic이 새 모델을 출시할 때 새 버전이 활성화되지 않은 계정의 사용자는 조용히 중단됩니다.

> **Warning:**
> 초기 설정의 일부로 세 가지 모델 환경 변수를 모두 특정 버전 ID로 설정합니다. 이 단계를 건너뛰면 Claude Code 업데이트로 인해 사용자가 중단될 수 있습니다.

제공자에 대한 버전별 모델 ID와 함께 다음 환경 변수를 사용합니다:

제공자	예시
Bedrock	`export ANTHROPIC_DEFAULT_OPUS_MODEL='us.anthropic.claude-opus-4-6-v1'`
Vertex AI	`export ANTHROPIC_DEFAULT_OPUS_MODEL='claude-opus-4-6'`
Foundry	`export ANTHROPIC_DEFAULT_OPUS_MODEL='claude-opus-4-6'`

`ANTHROPIC_DEFAULT_SONNET_MODEL` 및 `ANTHROPIC_DEFAULT_HAIKU_MODEL`에 대해 동일한 패턴을 적용합니다. 모든 제공자의 현재 및 레거시 모델 ID는 모델 개요를 참조하세요. 사용자를 새 모델 버전으로 업그레이드하려면 이러한 환경 변수를 업데이트하고 다시 배포합니다.

> **Note:**
> 타사 제공자를 사용할 때 `settings.availableModels` 허용 목록이 여전히 적용됩니다. 필터링은 제공자별 모델 ID가 아닌 모델 별칭(`opus`, `sonnet`, `haiku`)과 일치합니다.

버전별 모델 ID 재정의

위의 패밀리 수준 환경 변수는 패밀리 별칭당 하나의 모델 ID를 구성합니다. 동일한 패밀리 내의 여러 버전을 서로 다른 제공자 ID에 매핑해야 하는 경우 대신 `modelOverrides` 설정을 사용합니다.

`modelOverrides` 는 개별 Anthropic 모델 ID를 Claude Code가 제공자의 API로 보내는 제공자별 문자열에 매핑합니다. 사용자가 `/model` 선택기에서 매핑된 모델을 선택하면 Claude Code는 기본 제공 기본값 대신 구성된 값을 사용합니다.

이를 통해 엔터프라이즈 관리자는 거버넌스, 비용 할당 또는 지역 라우팅을 위해 각 모델 버전을 특정 Bedrock 추론 프로필 ARN, Vertex AI 버전 이름 또는 Foundry 배포 이름으로 라우팅할 수 있습니다.

설정 파일에서 `modelOverrides` 를 설정합니다:

```
{
  "modelOverrides": {
    "claude-opus-4-6": "arn:aws:bedrock:us-east-2:123456789012:application-inference-profile/opus-prod",
    "claude-opus-4-5-20251101": "arn:aws:bedrock:us-east-2:123456789012:application-inference-profile/opus-45-prod",
    "claude-sonnet-4-6": "arn:aws:bedrock:us-east-2:123456789012:application-inference-profile/sonnet-prod"
  }
}
```

키는 모델 개요에 나열된 Anthropic 모델 ID여야 합니다. 날짜가 지정된 모델 ID의 경우 날짜 접미사를 정확히 표시된 대로 포함합니다. 알 수 없는 키는 무시됩니다.

재정의는 `/model` 선택기의 각 항목을 지원하는 기본 제공 모델 ID를 대체합니다. Bedrock에서 재정의는 Claude Code가 시작 시 자동으로 발견하는 모든 추론 프로필보다 우선합니다. `ANTHROPIC_MODEL` , `--model` 또는 `ANTHROPIC_DEFAULT_*_MODEL` 환경 변수를 통해 직접 제공하는 값은 제공자로 그대로 전달되며 `modelOverrides` 로 변환되지 않습니다.

`modelOverrides` 는 `availableModels` 과 함께 작동합니다. 허용 목록은 재정의 값이 아닌 Anthropic 모델 ID에 대해 평가되므로 `availableModels` 의 `"opus"` 와 같은 항목은 Opus 버전이 ARN에 매핑되어도 계속 일치합니다.

Prompt caching 구성

Claude Code는 자동으로 prompt caching을 사용하여 성능을 최적화하고 비용을 절감합니다. 전역적으로 또는 특정 모델 계층에 대해 prompt caching을 비활성화할 수 있습니다:

환경 변수	설명
`DISABLE_PROMPT_CACHING`	모든 모델에 대해 prompt caching을 비활성화하려면 `1` 로 설정(모델별 설정보다 우선)
`DISABLE_PROMPT_CACHING_HAIKU`	Haiku 모델에 대해서만 prompt caching을 비활성화하려면 `1` 로 설정
`DISABLE_PROMPT_CACHING_SONNET`	Sonnet 모델에 대해서만 prompt caching을 비활성화하려면 `1` 로 설정
`DISABLE_PROMPT_CACHING_OPUS`	Opus 모델에 대해서만 prompt caching을 비활성화하려면 `1` 로 설정

이러한 환경 변수는 prompt caching 동작에 대한 세밀한 제어를 제공합니다. 전역 `DISABLE_PROMPT_CACHING` 설정은 모델별 설정보다 우선하므로 필요할 때 모든 캐싱을 빠르게 비활성화할 수 있습니다. 모델별 설정은 특정 모델 디버깅 또는 다양한 캐싱 구현을 가질 수 있는 클라우드 제공자와 작업할 때와 같은 선택적 제어에 유용합니다.

출력 스타일

> 소프트웨어 엔지니어링 이상의 용도로 Claude Code 적응시키기

출력 스타일을 사용하면 Claude Code의 로컬 스크립트 실행, 파일 읽기/쓰기, TODO 추적과 같은 핵심 기능을 유지하면서 모든 유형의 에이전트로 사용할 수 있습니다.

기본 제공 출력 스타일

Claude Code의 **Default** 출력 스타일은 기존 시스템 프롬프트이며, 소프트웨어 엔지니어링 작업을 효율적으로 완료하도록 설계되었습니다.

코드베이스와 Claude의 작동 방식을 가르치는 데 중점을 두는 두 가지 추가 기본 제공 출력 스타일이 있습니다:

- **Explanatory**: 소프트웨어 엔지니어링 작업을 완료하는 동안 교육용 "Insights"를 제공합니다. 구현 선택 사항과 코드베이스 패턴을 이해하는 데 도움이 됩니다.
- **Learning**: 협업 방식의 학습 모드로, Claude는 코딩하면서 "Insights"를 공유할 뿐만 아니라 사용자가 작은 전략적 코드 조각을 직접 작성하도록 요청합니다. Claude Code는 구현할 코드에 `TODO(human)` 마커를 추가합니다.

출력 스타일의 작동 방식

출력 스타일은 Claude Code의 시스템 프롬프트를 직접 수정합니다.

- 모든 출력 스타일은 효율적인 출력을 위한 지침(간결한 응답 등)을 제외합니다.
- 사용자 정의 출력 스타일은 `keep-coding-instructions`가 true가 아닌 한 코딩 관련 지침(테스트를 통한 코드 검증 등)을 제외합니다.
- 모든 출력 스타일은 시스템 프롬프트 끝에 추가된 자체 사용자 정의 지침을 가집니다.
- 모든 출력 스타일은 대화 중에 Claude가 출력 스타일 지침을 준수하도록 상기시키는 알림을 트리거합니다.

출력 스타일 변경

`/config`를 실행하고 **Output style**을 선택하여 메뉴에서 스타일을 선택합니다. 선택 사항은 로컬 프로젝트 수준의 `.claude/settings.local.json`에 저장됩니다.

메뉴 없이 스타일을 설정하려면 설정 파일에서 `outputStyle` 필드를 직접 편집합니다:

```
{
  "outputStyle": "Explanatory"
}
```

출력 스타일은 세션 시작 시 시스템 프롬프트에 설정되므로, 변경 사항은 새 세션을 시작할 때 적용됩니다. 이렇게 하면 시스템 프롬프트가 대화 전체에서 안정적으로 유지되어 프롬프트 캐싱이 지연 시간과 비용을 줄일 수 있습니다.

사용자 정의 출력 스타일 만들기

사용자 정의 출력 스타일은 frontmatter와 시스템 프롬프트에 추가될 텍스트가 포함된 Markdown 파일입니다:

```
---
name: My Custom Style
description:
  A brief description of what this style does, to be displayed to the user
---

## Custom Style Instructions

You are an interactive CLI tool that helps users with software engineering
tasks. [Your custom instructions here...]

### Specific Behaviors

[Define how the assistant should behave in this style...]
```

이러한 파일은 사용자 수준(`~/.claude/output-styles`) 또는 프로젝트 수준(`.claude/output-styles`)에 저장할 수 있습니다.

Frontmatter

출력 스타일 파일은 메타데이터 지정을 위한 frontmatter를 지원합니다:

Frontmatter	목적	기본값
`name`	출력 스타일의 이름(파일 이름이 아닌 경우)	파일 이름에서 상속

Frontmatter	목적	기본값
`description`	`/config` 선택기에 표시되는 출력 스타일의 설명	없음
`keep-coding-instructions`	Claude Code의 시스템 프롬프트의 코딩 관련 부분을 유지할지 여부	false

관련 기능과의 비교

출력 스타일 vs. CLAUDE.md vs. –append-system-prompt

출력 스타일은 소프트웨어 엔지니어링에 특정한 Claude Code의 기본 시스템 프롬프트 부분을 완전히 "끕니다". CLAUDE.md와 `--append-system-prompt` 는 Claude Code의 기본 시스템 프롬프트를 편집하지 않습니다. CLAUDE.md는 내용을 Claude Code의 기본 시스템 프롬프트 다음에 사용자 메시지로 추가합니다. `--append-system-prompt` 는 내용을 시스템 프롬프트에 추가합니다.

출력 스타일 vs. Agents

출력 스타일은 주 에이전트 루프에 직접 영향을 미치며 시스템 프롬프트에만 영향을 줍니다. 에이전트는 특정 작업을 처리하기 위해 호출되며 사용할 모델, 사용 가능한 도구, 에이전트 사용 시기에 대한 일부 컨텍스트와 같은 추가 설정을 포함할 수 있습니다.

출력 스타일 vs. Skills

출력 스타일은 Claude가 응답하는 방식(형식, 톤, 구조)을 수정하며 선택되면 항상 활성화됩니다. Skills는 `/skill-name` 으로 호출하거나 관련성이 있을 때 Claude가 자동으로 로드하는 작업별 프롬프트입니다. 일관된 형식 지정 기본 설정에는 출력 스타일을 사용하고, 재사용 가능한 워크플로우 및 작업에는 skills를 사용합니다.

빠른 모드로 응답 속도 향상

> Claude Code에서 빠른 모드를 전환하여 더 빠른 Opus 4.6 응답을 받습니다.

Note:

빠른 모드는 연구 미리보기 상태입니다. 피드백에 따라 기능, 가격 책정 및 가용성이 변경될 수 있습니다.

빠른 모드는 Claude Opus 4.6을 위한 고속 구성으로, 토큰당 더 높은 비용으로 모델을 2.5배 빠르게 만듭니다. 빠른 반복이나 라이브 디버깅과 같은 대화형 작업에서 속도가 필요할 때 `/fast`로 켜고, 비용이 지연 시간보다 중요할 때 끕니다.

빠른 모드는 다른 모델이 아닙니다. 비용 효율성보다 속도를 우선시하는 다른 API 구성을 사용하는 동일한 Opus 4.6을 사용합니다. 동일한 품질과 기능을 얻으며, 응답만 더 빠릅니다.

Note:

빠른 모드는 Claude Code v2.1.36 이상이 필요합니다. `claude --version`으로 버전을 확인합니다.

알아야 할 사항:

- Claude Code CLI에서 `/fast`를 사용하여 빠른 모드를 전환합니다. Claude Code VS Code 확장 프로그램에서도 `/fast`를 통해 사용할 수 있습니다.
- Opus 4.6의 빠른 모드 가격은 $30/150 MTok부터 시작합니다. 빠른 모드는 2월 16일 태평양 표준시 오후 11:59까지 모든 요금제에 대해 50% 할인으로 제공됩니다.
- 구독 요금제(Pro/Max/Team/Enterprise)의 모든 Claude Code 사용자 및 Claude Console에서 사용할 수 있습니다.
- 구독 요금제(Pro/Max/Team/Enterprise)의 Claude Code 사용자의 경우, 빠른 모드는 추가 사용을 통해서만 사용 가능하며 구독 요금제 사용량 제한에 포함되지 않습니다.

이 페이지에서는 빠른 모드 전환, 비용 트레이드오프, 빠른 모드 사용 시기, 요구사항, 세션별 옵트인 및 속도 제한 처리에 대해 설명합니다.

빠른 모드 전환

다음 중 한 가지 방법으로 빠른 모드를 전환합니다:

- `/fast`를 입력하고 Tab을 눌러 켜거나 끕니다
- 사용자 설정 파일에서 `"fastMode": true`를 설정합니다

기본적으로 빠른 모드는 세션 간에 유지됩니다. 관리자는 빠른 모드를 각 세션마다 재설정하도록 구성할 수 있습니다. 자세한 내용은 세션별 옵트인 필요를 참조합니다.

최상의 비용 효율성을 위해 대화 중간에 전환하기보다는 세션 시작 시 빠른 모드를 활성화합니다. 자세한 내용은 비용 트레이드오프 이해를 참조합니다.

빠른 모드를 활성화하면:

- 다른 모델을 사용 중인 경우 Claude Code가 자동으로 Opus 4.6으로 전환됩니다
- 확인 메시지가 표시됩니다: “Fast mode ON”
- 빠른 모드가 활성화된 동안 프롬프트 옆에 작은 `↯` 아이콘이 나타납니다
- 언제든지 `/fast`를 다시 실행하여 빠른 모드가 켜져 있는지 꺼져 있는지 확인합니다

`/fast`를 다시 실행하여 빠른 모드를 비활성화하면 Opus 4.6에 유지됩니다. 모델이 이전 모델로 되돌아가지 않습니다. 다른 모델로 전환하려면 `/model`을 사용합니다.

비용 트레이드오프 이해

빠른 모드는 표준 Opus 4.6보다 토큰당 가격이 높습니다:

모드	입력 (MTok)	출력 (MTok)
Opus 4.6의 빠른 모드 (<200K)	$30	$150
Opus 4.6의 빠른 모드 (>200K)	$60	$225

빠른 모드는 1M 토큰 확장 컨텍스트 윈도우와 호환됩니다.

대화 중간에 빠른 모드로 전환하면 전체 대화 컨텍스트에 대해 전체 빠른 모드 캐시되지 않은 입력 토큰 가격을 지불합니다. 이는 처음부터 빠른 모드를 활성화했을 경우보다 더 많은 비용이 듭니다.

빠른 모드 사용 시기 결정

빠른 모드는 응답 지연 시간이 비용보다 중요한 대화형 작업에 가장 적합합니다:

- 코드 변경에 대한 빠른 반복
- 라이브 디버깅 세션
- 긴급 마감이 있는 시간에 민감한 작업

표준 모드는 다음에 더 적합합니다:

- 속도가 덜 중요한 장기 자동 작업
- 배치 처리 또는 CI/CD 파이프라인
- 비용에 민감한 워크로드

빠른 모드 대 노력 수준

빠른 모드와 노력 수준 모두 응답 속도에 영향을 미치지만 방식이 다릅니다:

설정	효과
빠른 모드	동일한 모델 품질, 낮은 지연 시간, 높은 비용
낮은 노력 수준	더 적은 생각 시간, 더 빠른 응답, 복잡한 작업에서 잠재적으로 낮은 품질

둘 다 결합할 수 있습니다: 간단한 작업에서 최대 속도를 위해 낮은 노력 수준과 함께 빠른 모드를 사용합니다.

요구사항

빠른 모드는 다음 모두를 필요로 합니다:

- **타사 클라우드 제공자에서 사용 불가**: 빠른 모드는 Amazon Bedrock, Google Vertex AI 또는 Microsoft Azure Foundry에서 사용할 수 없습니다. 빠른 모드는 Anthropic Console API 및 추가 사용을 사용하는 Claude 구독 요금제를 통해 사용할 수 있습니다.
- **추가 사용 활성화**: 계정에 추가 사용이 활성화되어 있어야 하며, 이를 통해 요금제의 포함된 사용량을 초과하여 청구할 수 있습니다. 개인 계정의 경우 Console 청구 설정에서 활성화합니다. Teams 및 Enterprise의 경우 관리자가 조직에 대해 추가 사용을 활성화해야 합니다.

> **Note:**
>
> 빠른 모드 사용량은 요금제에 남은 사용량이 있더라도 추가 사용으로 직접 청구됩니다. 이는 빠른 모드 토큰이 요금제의 포함된 사용량에 포함되지 않으며 첫 번째 토큰부터 빠른 모드 요금으로 청구됨을 의미합니다.

- **Teams 및 Enterprise의 관리자 활성화**: 빠른 모드는 Teams 및 Enterprise 조직에 대해 기본적으로 비활성화됩니다. 사용자가 액세스할 수 있으려면 관리자가 명시적으로 빠른 모드를 활성화해야 합니다.

> **Note:**

> 관리자가 조직에 대해 빠른 모드를 활성화하지 않은 경우 `/fast` 명령은 “Fast mode has been disabled by your organization.” 을 표시합니다.

조직에 대해 빠른 모드 활성화

관리자는 다음에서 빠른 모드를 활성화할 수 있습니다:

- **Console** (API 고객): Claude Code 기본 설정
- **Claude AI** (Teams 및 Enterprise): 관리자 설정 > Claude Code

빠른 모드를 완전히 비활성화하는 또 다른 옵션은 `CLAUDE_CODE_DISABLE_FAST_MODE=1` 을 설정하는 것입니다. 환경 변수를 참조합니다.

세션별 옵트인 필요

기본적으로 빠른 모드는 세션 간에 유지됩니다: 사용자가 빠른 모드를 활성화하면 향후 세션에서도 켜져 있습니다. Teams 또는 Enterprise 요금제의 관리자는 관리되는 설정 또는 서버 관리 설정에서 `fastModePerSessionOptIn` 을 `true` 로 설정하여 이를 방지할 수 있습니다. 이로 인해 각 세션이 빠른 모드가 꺼진 상태로 시작되며, 사용자가 `/fast` 로 명시적으로 활성화해야 합니다.

```
{
  "fastModePerSessionOptIn": true
}
```

이는 사용자가 여러 동시 세션을 실행하는 조직에서 비용을 제어하는 데 유용합니다. 사용자는 속도가 필요할 때 `/fast` 로 빠른 모드를 활성화할 수 있지만 각 새 세션의 시작 시 재설정됩니다. 사용자의 빠른 모드 기본 설정은 여전히 저장되므로 이 설정을 제거하면 기본 지속 동작이 복원됩니다.

속도 제한 처리

빠른 모드는 표준 Opus 4.6과 별도의 속도 제한을 가집니다. 빠른 모드 속도 제한에 도달하거나 추가 사용 크레딧이 부족할 때:

1. 빠른 모드가 자동으로 표준 Opus 4.6으로 폴백됩니다
2. `ϟ` 아이콘이 회색으로 변하여 쿨다운을 나타냅니다
3. 표준 속도 및 가격으로 계속 작업합니다
4. 쿨다운이 만료되면 빠른 모드가 자동으로 다시 활성화됩니다

쿨다운을 기다리지 않고 빠른 모드를 수동으로 비활성화하려면 `/fast` 를 다시 실행합니다.

연구 미리보기

빠른 모드는 연구 미리보기 기능입니다. 이는 다음을 의미합니다:

- 기능은 피드백에 따라 변경될 수 있습니다
- 가용성 및 가격 책정은 변경될 수 있습니다
- 기본 API 구성이 진화할 수 있습니다

일반적인 Anthropic 지원 채널을 통해 문제 또는 피드백을 보고합니다.

참고 항목

- 모델 구성: 모델 전환 및 노력 수준 조정
- 비용 효과적으로 관리: 토큰 사용량 추적 및 비용 감소
- 상태 줄 구성: 모델 및 컨텍스트 정보 표시

비용을 효과적으로 관리하기

> 토큰 사용량을 추적하고, 팀 지출 한도를 설정하며, 컨텍스트 관리, 모델 선택, 확장 사고 설정 및 전처리 hooks를 통해 Claude Code 비용을 절감합니다.

Claude Code는 각 상호작용마다 토큰을 소비합니다. 비용은 코드베이스 크기, 쿼리 복잡도, 대화 길이에 따라 달라집니다. 평균 비용은 개발자당 하루에 $6이며, 90%의 사용자는 일일 비용이 $12 이하로 유지됩니다.

팀 사용의 경우, Claude Code는 API 토큰 소비량으로 청구됩니다. 평균적으로 Claude Code는 Sonnet 4.6으로 개발자당 월 약 $100-200의 비용이 발생하지만, 사용자가 실행 중인 인스턴스 수와 자동화에서 사용 여부에 따라 큰 편차가 있습니다.

이 페이지에서는 비용 추적 방법, 팀 비용 관리, 토큰 사용량 감소 방법을 다룹니다.

비용 추적

`/cost` 명령 사용

> **Note:**
>
> `/cost` 명령은 API 토큰 사용량을 표시하며 API 사용자를 위한 것입니다. Claude Max 및 Pro 구독자는 구독에 사용량이 포함되어 있으므로 `/cost` 데이터는 청구 목적으로 관련이 없습니다. 구독자는 `/stats`를 사용하여 사용 패턴을 볼 수 있습니다.

`/cost` 명령은 현재 세션에 대한 자세한 토큰 사용량 통계를 제공합니다:

```
Total cost:            $0.55
Total duration (API):  6m 19.7s
Total duration (wall): 6h 33m 10.2s
Total code changes:    0 lines added, 0 lines removed
```

팀 비용 관리

Claude API를 사용할 때, 워크스페이스 지출 한도를 설정하여 전체 Claude Code 워크스페이스 지출을 제어할 수 있습니다. 관리자는 Console에서 비용 및 사용량 보고서를 볼 수 있습니다.

> **Note:**
>
> Claude Code를 Claude Console 계정으로 처음 인증할 때, "Claude Code"라는 워크스페이스가 자동으로 생성됩니다. 이 워크스페이스는 조직의 모든 Claude Code 사용에 대한 중앙 집중식 비용 추적 및 관리를 제공합니다. 이 워크스페이스에 대해 API 키를 생성할 수 없습니다. 이는 Claude Code 인증 및 사용 전용입니다.

Bedrock, Vertex 및 Foundry에서 Claude Code는 클라우드에서 메트릭을 전송하지 않습니다. 비용 메트릭을 얻으려면 여러 대규모 엔터프라이즈에서 LiteLLM을 사용한다고 보고했으며, 이는 회사가 키별 지출을 추적하는 데 도움이 되는 오픈소스 도구입니다. 이 프로젝트는 Anthropic과 무관하며 보안 감사를 받지 않았습니다.

속도 제한 권장사항

팀을 위해 Claude Code를 설정할 때, 조직 규모에 따른 다음 분당 토큰(TPM) 및 분당 요청(RPM) 사용자당 권장사항을 고려하십시오:

팀 규모	사용자당 TPM	사용자당 RPM
1-5 사용자	200k-300k	5-7
5-20 사용자	100k-150k	2.5-3.5
20-50 사용자	50k-75k	1.25-1.75
50-100 사용자	25k-35k	0.62-0.87
100-500 사용자	15k-20k	0.37-0.47
500+ 사용자	10k-15k	0.25-0.35

예를 들어, 200명의 사용자가 있는 경우, 각 사용자에 대해 20k TPM을 요청하거나 총 400만 TPM(200*20,000 = 400만)을 요청할 수 있습니다.

팀 규모가 커질수록 사용자당 TPM이 감소하는 이유는 더 큰 조직에서 더 적은 수의 사용자가 Claude Code를 동시에 사용하는 경향이 있기 때문입니다. 이러한 속도 제한은 개별 사용자별이 아닌 조직 수준에서 적용되므로, 다른 사용자가 적극적으로 서비스를 사용하지 않을 때 개별 사용자는 일시적으로 계산된 할당량보다 더 많이 소비할 수 있습니다.

> **Note:**
>
> 대규모 그룹과의 라이브 교육 세션과 같이 비정상적으로 높은 동시 사용 시나리오를 예상하는 경우, 사용자당 더 높은 TPM 할당이 필요할 수 있습니다.

에이전트 팀 토큰 비용

에이전트 팀은 각각 자체 컨텍스트 윈도우를 가진 여러 Claude Code 인스턴스를 생성합니다. 토큰 사용량은 활성 팀원의 수와 각 팀원이 실행되는 시간에 따라 확장됩니다.

에이전트 팀 비용을 관리 가능하게 유지하려면:

- 팀원에게 Sonnet을 사용하십시오. 조정 작업을 위해 기능과 비용의 균형을 맞춥니다.
- 팀을 작게 유지하십시오. 각 팀원은 자체 컨텍스트 윈도우를 실행하므로 토큰 사용량은 대략 팀 규모에 비례합니다.
- spawn 프롬프트를 집중적으로 유지하십시오. 팀원은 CLAUDE.md, MCP servers 및 skills를 자동으로 로드하지만, spawn 프롬프트의 모든 것이 처음부터 컨텍스트에 추가됩니다.
- 작업이 완료되면 팀을 정리하십시오. 활성 팀원은 유휴 상태에서도 계속 토큰을 소비합니다.
- 에이전트 팀은 기본적으로 비활성화되어 있습니다. settings.json에서 `CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS=1` 을 설정하거나 환경에서 설정하여 활성화하십시오. 에이전트 팀 활성화를 참조하십시오.

토큰 사용량 감소

토큰 비용은 컨텍스트 크기에 따라 확장됩니다. Claude가 처리하는 컨텍스트가 많을수록 더 많은 토큰을 사용합니다. Claude Code는 prompt caching(시스템 프롬프트와 같은 반복되는 콘텐츠의 비용을 줄임)과 auto-compaction(컨텍스트 한도에 접근할 때 대화 기록을 요약함)을 통해 비용을 자동으로 최적화합니다.

다음 전략은 컨텍스트를 작게 유지하고 메시지당 비용을 줄이는 데 도움이 됩니다.

컨텍스트를 사전에 관리하기

`/cost` 를 사용하여 현재 토큰 사용량을 확인하거나, 상태 줄을 구성하여 지속적으로 표시하십시오.

- **작업 간 지우기**: 관련 없는 작업으로 전환할 때 `/clear` 를 사용하여 새로 시작하십시오. 오래된 컨텍스트는 이후의 모든 메시지에서 토큰을 낭비합니다. 지우기 전에 `/rename` 을 사용하여 나중에 세션을 쉽게 찾을 수 있도록 한 다음, `/resume` 을 사용하여 돌아가십시오.
- **사용자 정의 compaction 지침 추가**: `/compact Focus on code samples and API usage` 는 Claude에게 요약 중에 보존할 내용을 알려줍니다.

CLAUDE.md에서 compaction 동작을 사용자 정의할 수도 있습니다:

```
## Compact instructions

When you are using compact, please focus on test output and code changes
```

올바른 모델 선택

Sonnet은 대부분의 코딩 작업을 잘 처리하며 Opus보다 비용이 적습니다. 복잡한 아키텍처 결정이나 다단계 추론을 위해 Opus를 예약하십시오. `/model`을 사용하여 세션 중간에 모델을 전환하거나, `/config`에서 기본값을 설정하십시오. 간단한 subagent 작업의 경우, subagent 구성에서 `model: haiku`를 지정하십시오.

MCP server 오버헤드 감소

각 MCP server는 유휴 상태에서도 도구 정의를 컨텍스트에 추가합니다. `/context`를 실행하여 공간을 소비하는 것을 확인하십시오.

- **사용 가능한 경우 CLI 도구 선호**: `gh`, `aws`, `gcloud`, `sentry-cli`와 같은 도구는 지속적인 도구 정의를 추가하지 않기 때문에 MCP server보다 컨텍스트 효율적입니다. Claude는 오버헤드 없이 CLI 명령을 직접 실행할 수 있습니다.
- **사용하지 않는 server 비활성화**: `/mcp`를 실행하여 구성된 server를 확인하고 적극적으로 사용하지 않는 것을 비활성화하십시오.
- **도구 검색은 자동입니다**: MCP 도구 설명이 컨텍스트 윈도우의 10%를 초과할 때, Claude Code는 자동으로 이를 연기하고 도구 검색을 통해 필요에 따라 도구를 로드합니다. 연기된 도구는 실제로 사용될 때만 컨텍스트에 들어가므로, 더 낮은 임계값은 공간을 소비하는 유휴 도구 정의가 더 적다는 의미입니다. `ENABLE_TOOL_SEARCH=auto:<N>`으로 더 낮은 임계값을 설정하십시오(예: `auto:5`는 도구가 컨텍스트 윈도우의 5%를 초과할 때 트리거됨).

타입 언어를 위한 코드 인텔리전스 플러그인 설치

코드 인텔리전스 플러그인은 Claude에게 텍스트 기반 검색 대신 정확한 기호 탐색을 제공하여 낯선 코드를 탐색할 때 불필요한 파일 읽기를 줄입니다. 단일 "정의로 이동" 호출은 grep 다음에 여러 후보 파일을 읽는 것을 대체합니다. 설치된 언어 서버는 편집 후 자동으로 타입 오류를 보고하므로 Claude는 컴파일러를 실행하지 않고도 실수를 포착합니다.

hooks 및 skills로 처리 오프로드

사용자 정의 hooks는 Claude가 보기 전에 데이터를 전처리할 수 있습니다. Claude가 10,000줄 로그 파일을 읽어 오류를 찾는 대신, hook은 `ERROR`를 grep하고 일치하는 줄만 반환하여 컨텍스트를 수만 개의 토큰에서 수백 개로 줄일 수 있습니다.

skill은 Claude에게 도메인 지식을 제공하여 탐색할 필요가 없도록 할 수 있습니다. 예를 들어, “codebase-overview” skill은 프로젝트의 아키텍처, 주요 디렉토리 및 명명 규칙을 설명할 수 있습니다. Claude가 skill을 호출하면, 구조를 이해하기 위해 여러 파일을 읽는 데 토큰을 소비하는 대신 즉시 이 컨텍스트를 얻습니다.

예를 들어, 이 PreToolUse hook은 테스트 출력을 필터링하여 실패만 표시합니다:

settings.json

이를 settings.json에 추가하여 모든 Bash 명령 전에 hook을 실행하십시오:

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": "~/.claude/hooks/filter-test-output.sh"
          }
        ]
      }
    ]
  }
}
```

filter-test-output.sh

hook은 이 스크립트를 호출하며, 이는 명령이 테스트 러너인지 확인하고 실패만 표시하도록 수정합니다:

```
#!/bin/bash
input=$(cat)
cmd=$(echo "$input" | jq -r '.tool_input.command')

## If running tests, filter to show only failures
if [[ "$cmd" =~ ^(npm test|pytest|go test) ]]; then
  filtered_cmd="$cmd 2>&1 | grep -A 5 -E '(FAIL|ERROR|error:)' | head -100"
  echo "{\"hookSpecificOutput\":{\"hookEventName\":\"PreToolUse\",\"permissionDeci
sion\":\"allow\",\"updatedInput\":{\"command\":\"$filtered_cmd\"}}}"
else
  echo "{}"
fi
```

CLAUDE.md에서 skills로 지침 이동

CLAUDE.md 파일은 세션 시작 시 컨텍스트에 로드됩니다. PR 검토 또는 데이터베이스 마이그레이션과 같은 특정 워크플로우에 대한 자세한 지침이 포함되어 있으면, 관련 없는 작업을 수행할 때도 해당 토큰이 존재합니다. Skills는 호출될 때만 필요에 따라 로드되므로, 특화된 지침을 skills로 이동하면 기본 컨텍스트를 더 작게 유지합니다. CLAUDE.md를 필수 항목만 포함하여 약 500줄 이하로 유지하십시오.

확장 사고 조정

확장 사고는 기본적으로 31,999 토큰의 예산으로 활성화되어 있습니다. 복잡한 계획 및 추론 작업의 성능을 크게 향상시키기 때문입니다. 그러나 사고 토큰은 출력 토큰으로 청구되므로, 깊은 추론이 필요하지 않은 더 간단한 작업의 경우, `/model`에서 Opus 4.6의 노력 수준을 낮추거나, `/config`에서 사고를 비활성화하거나, 예산을 낮춤으로써(예: `MAX_THINKING_TOKENS=8000`) 비용을 줄일 수 있습니다.

자세한 작업을 subagents에 위임

테스트 실행, 문서 가져오기 또는 로그 파일 처리는 상당한 컨텍스트를 소비할 수 있습니다. 이를 subagents에 위임하여 자세한 출력이 subagent의 컨텍스트에 유지되는 동안 요약만 주 대화로 반환되도록 하십시오.

에이전트 팀 비용 관리

에이전트 팀은 팀원이 plan mode에서 실행될 때 표준 세션보다 약 7배 더 많은 토큰을 사용합니다. 각 팀원은 자체 컨텍스트 윈도우를 유지하고 별도의 Claude 인스턴스로 실행되기 때문입니다. 팀 작업을 작고 자체 포함되도록 유지하여 팀원당 토큰 사용량을 제한하십시오. 자세한 내용은 에이전트 팀을 참조하십시오.

구체적인 프롬프트 작성

"이 코드베이스 개선"과 같은 모호한 요청은 광범위한 스캔을 트리거합니다. "auth.ts의 로그인 함수에 입력 검증 추가"와 같은 구체적인 요청은 Claude가 최소한의 파일 읽기로 효율적으로 작업하도록 합니다.

복잡한 작업을 효율적으로 수행

더 길거나 복잡한 작업의 경우, 이러한 습관은 잘못된 경로로 인한 낭비된 토큰을 피하는 데 도움이 됩니다:

- **복잡한 작업에 plan mode 사용**: Shift+Tab을 눌러 구현 전에 plan mode에 들어가십시오. Claude는 코드베이스를 탐색하고 승인을 위한 접근 방식을 제안하여, 초기 방향이 잘못되었을 때 비용이 많이 드는 재작업을 방지합니다.
- **조기에 방향 수정**: Claude가 잘못된 방향으로 가기 시작하면, Escape를 눌러 즉시 중지하십시오. `/rewind`를 사용하거나 Escape를 두 번 눌러 대화 및 코드를 이전 checkpoint로 복원하십시오.
- **검증 대상 제공**: 테스트 케이스를 포함하고, 스크린샷을 붙여넣거나, 프롬프트에서 예상 출력을 정의하십시오. Claude가 자신의 작업을 검증할 수 있으면, 수정을 요청해야 하기 전에 문제를 포착합니다.
- **증분적으로 테스트**: 한 파일을 작성하고, 테스트한 다음, 계속하십시오. 이는 문제가 저렴하게 수정될 수 있을 때 조기에 포착합니다.

백그라운드 토큰 사용량

Claude Code는 유휴 상태에서도 일부 백그라운드 기능에 토큰을 사용합니다:

- **대화 요약**: `claude --resume` 기능을 위해 이전 대화를 요약하는 백그라운드 작업
- **명령 처리**: `/cost`와 같은 일부 명령은 상태를 확인하기 위해 요청을 생성할 수 있습니다

이러한 백그라운드 프로세스는 활성 상호작용 없이도 세션당 적은 양의 토큰(일반적으로 $0.04 미만)을 소비합니다.

Claude Code 동작 변경 이해

Claude Code는 비용 보고를 포함한 기능 작동 방식을 변경할 수 있는 정기적인 업데이트를 받습니다. `claude --version`을 실행하여 현재 버전을 확인하십시오. 특정 청구 질문의 경우, Console 계정을 통해 Anthropic 지원에 문의하십시오. 팀 배포의 경우, 더 광범위한 롤아웃 전에 사용 패턴을 설정하기 위해 작은 파일럿 그룹으로 시작하십시오.

Part 5: Extensibility

Hooks 참조

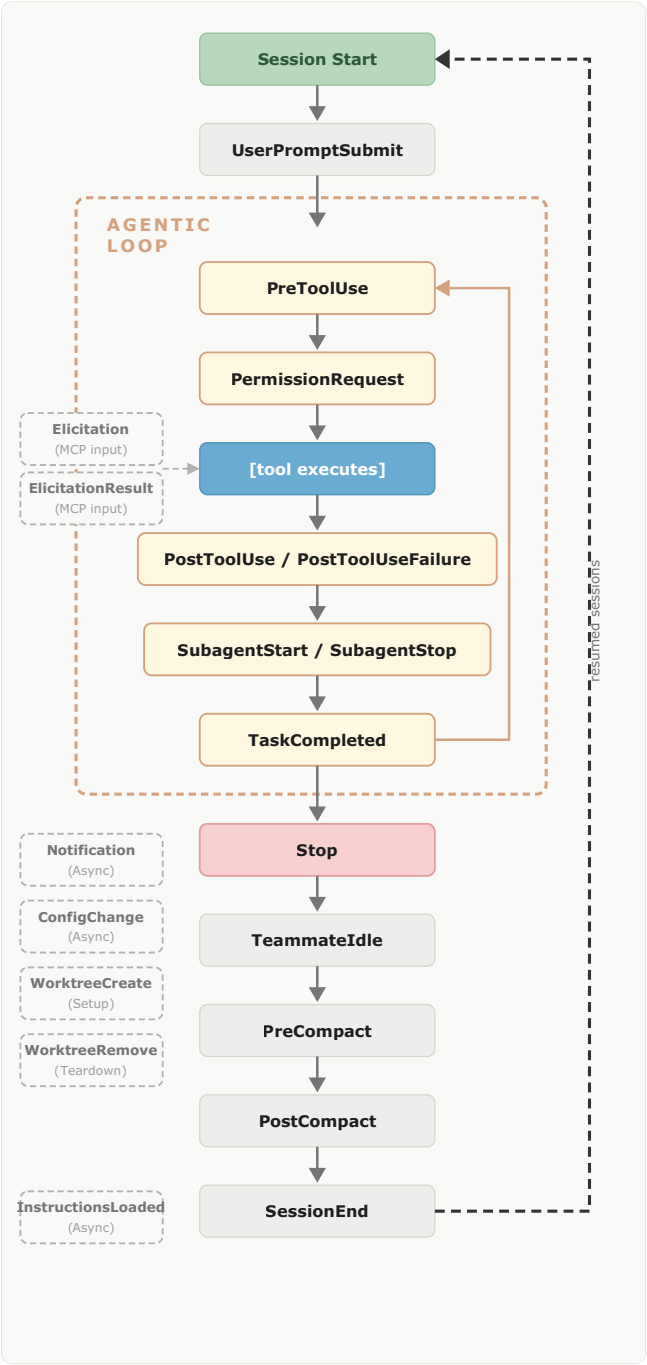
> Claude Code hook 이벤트, 구성 스키마, JSON 입출력 형식, 종료 코드, 비동기 hook, HTTP hook, 프롬프트 hook, MCP 도구 hook에 대한 참조입니다.

> **Tip:**
>
> 예제가 포함된 빠른 시작 가이드는 hook으로 워크플로우 자동화를 참조하세요.

Hook은 Claude Code의 수명 주기에서 특정 지점에 자동으로 실행되는 사용자 정의 셸 명령, HTTP 엔드포인트 또는 LLM 프롬프트입니다. 이 참조를 사용하여 이벤트 스키마, 구성 옵션, JSON 입출력 형식, 비동기 hook, HTTP hook, MCP 도구 hook과 같은 고급 기능을 조회할 수 있습니다. 처음으로 hook을 설정하는 경우 가이드부터 시작하세요.

Hook 수명 주기

Hook은 Claude Code 세션 중 특정 지점에서 실행됩니다. 이벤트가 발생하고 matcher가 일치하면 Claude Code는 이벤트에 대한 JSON 컨텍스트를 hook 핸들러에 전달합니다. 명령 hook의 경우 입력이 stdin에 도착합니다. HTTP hook의 경우 POST 요청 본문으로 도착합니다. 그러면 핸들러는 입력을 검사하고 조치를 취한 후 선택적으로 결정을 반환할 수 있습니다. 일부 이벤트는 세션당 한 번 발생하고 다른 이벤트는 에이전트 루프 내에서 반복적으로 발생합니다.


Session Start
UserPromptSubmit
AGENTIC LOOP
PreToolUse
PermissionRequest
Elicitation
(MCP input)
[tool executes]
ElicitationResult
(MCP input)
PostToolUse / PostToolUseFailure
SubagentStart / SubagentStop
TaskCompleted
Notification
(Async)
Stop
ConfigChange
(Async)
TeammateIdle
WorktreeCreate
(Setup)
PreCompact
WorktreeRemove
(Teardown)
PostCompact
InstructionsLoaded
(Async)
SessionEnd
resumed sessions

SessionStart에서 에이전트 루프를 거쳐 SessionEnd까지의 hook 시퀀스를 보여주는 hook 수명 주기 다이어그램, WorktreeCreate, WorktreeRemove, InstructionsLoaded는 독립적인 비동기 이벤트

아래 표는 각 이벤트가 언제 발생하는지 요약합니다. Hook 이벤트 섹션에서는 각 이벤트의 전체 입력 스키마와 결정 제어 옵션을 문서화합니다.

Event	When it fires
`SessionStart`	When a session begins or resumes
`UserPromptSubmit`	When you submit a prompt, before Claude processes it
`PreToolUse`	Before a tool call executes. Can block it
`PermissionRequest`	When a permission dialog appears
`PostToolUse`	After a tool call succeeds
`PostToolUseFailure`	After a tool call fails
`Notification`	When Claude Code sends a notification
`SubagentStart`	When a subagent is spawned
`SubagentStop`	When a subagent finishes
`Stop`	When Claude finishes responding
`TeammateIdle`	When an agent team teammate is about to go idle
`TaskCompleted`	When a task is being marked as completed
`InstructionsLoaded`	When a CLAUDE.md or `.claude/rules/*.md` file is loaded into context. Fires at session start and when files are lazily loaded during a session
`ConfigChange`	When a configuration file changes during a session
`WorktreeCreate`	When a worktree is being created via `--worktree` or `isolation: "worktree"`. Replaces default git behavior
`WorktreeRemove`	When a worktree is being removed, either at session exit or when a subagent finishes
`PreCompact`	Before context compaction
`PostCompact`	After context compaction completes

Event	When it fires
`Elicitation`	When an MCP server requests user input during a tool call
`ElicitationResult`	After a user responds to an MCP elicitation, before the response is sent back to the server
`SessionEnd`	When a session terminates

Hook이 어떻게 해결되는지

이러한 부분들이 어떻게 함께 작동하는지 보려면 파괴적인 셸 명령을 차단하는 이 `PreToolUse` hook을 고려하세요. hook은 모든 Bash 도구 호출 전에 `block-rm.sh`를 실행합니다.

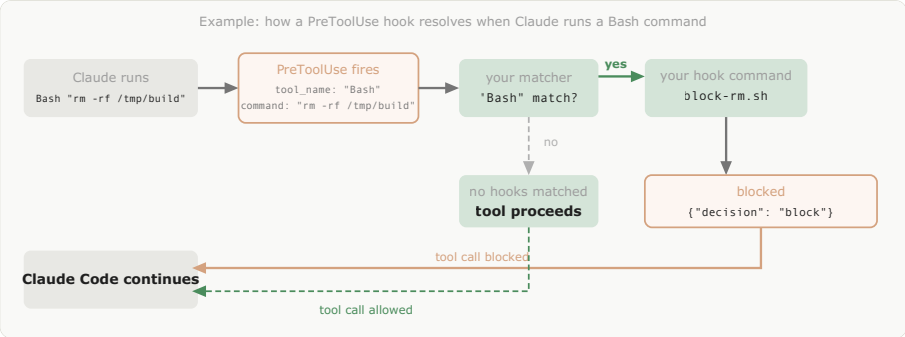
```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": ".claude/hooks/block-rm.sh"
          }
        ]
      }
    ]
  }
}
```

스크립트는 stdin에서 JSON 입력을 읽고 명령을 추출한 후 `rm -rf`를 포함하면 `permissionDecision`을 `"deny"`로 반환합니다.

```
#!/bin/bash
## .claude/hooks/block-rm.sh
COMMAND=$(jq -r '.tool_input.command')

if echo "$COMMAND" | grep -q 'rm -rf'; then
  jq -n '{
    hookSpecificOutput: {
      hookEventName: "PreToolUse",
      permissionDecision: "deny",
      permissionDecisionReason: "Destructive command blocked by hook"
    }
  }'
else
  exit 0  # allow the command
fi
```

이제 Claude Code가 `Bash "rm -rf /tmp/build"` 를 실행하기로 결정했다고 가정합니다. 다음은 발생하는 일입니다.



Hook 해결 흐름: PreToolUse 이벤트 발생, matcher가 Bash 일치 확인, hook 핸들러 실행, 결과가 Claude Code로 반환

Step 1: 이벤트 발생

`PreToolUse` 이벤트가 발생합니다. Claude Code는 도구 입력을 JSON으로 stdin에 hook으로 전송합니다.

```
{ "tool_name": "Bash", "tool_input": { "command": "rm -rf /tmp/build" }, ... }
```

Step 2: Matcher 확인

matcher `"Bash"`가 도구 이름과 일치하므로 `block-rm.sh`가 실행됩니다. matcher를 생략하거나 `"*"`를 사용하면 이벤트의 모든 발생에서 hook이 실행됩니다. matcher가 정의되어 있고 일치하지 않을 때만 hook이 건너뜁니다.

Step 3: Hook 핸들러 실행

스크립트는 입력에서 `"rm -rf /tmp/build"`를 추출하고 `rm -rf`를 찾으므로 stdout에 결정을 인쇄합니다.

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "deny",
    "permissionDecisionReason": "Destructive command blocked by hook"
  }
}
```

명령이 안전했다면 (예: `npm test`), 스크립트는 대신 `exit 0`을 실행하여 Claude Code에 도구 호출을 허용하고 추가 조치 없이 진행하도록 알립니다.

Step 4: Claude Code가 결과에 따라 행동

Claude Code는 JSON 결정을 읽고 도구 호출을 차단하며 Claude에 이유를 표시합니다.

아래 구성 섹션에서는 전체 스키마를 문서화하고, 각 hook 이벤트 섹션에서는 명령이 받는 입력과 반환할 수 있는 출력을 문서화합니다.

구성

Hook은 JSON 설정 파일에서 정의됩니다. 구성에는 세 가지 수준의 중첩이 있습니다.

1. 응답할 hook 이벤트를 선택합니다 (예: `PreToolUse` 또는 `Stop`).
2. 발생 시기를 필터링하는 matcher 그룹을 추가합니다 (예: “Bash 도구에만”).
3. 일치할 때 실행할 하나 이상의 hook 핸들러를 정의합니다.

위의 Hook이 어떻게 해결되는지에서 주석이 달린 예제를 포함한 완전한 설명을 참조하세요.

> **Note:**
>
> 이 페이지는 각 수준에 대해 특정 용어를 사용합니다. **hook 이벤트**는 수명 주기 지점, **matcher 그룹**은 필터, **hook 핸들러**는 실행되는 셸 명령, HTTP 엔드포인트, 프롬프트 또는 에이전트입니다. “Hook”은 일반적인 기능을 나타냅니다.

Hook 위치

hook을 정의하는 위치는 범위를 결정합니다.

위치	범위	공유 가능
`~/.claude/settings.json`	모든 프로젝트	아니오, 컴퓨터에 로컬
`.claude/settings.json`	단일 프로젝트	예, 리포지토리에 커밋 가능
`.claude/settings.local.json`	단일 프로젝트	아니오, gitignored
관리형 정책 설정	조직 전체	예, 관리자 제어
Plugin `hooks/hooks.json`	plugin이 활성화되었을 때	예, plugin과 함께 번들됨
Skill 또는 agent frontmatter	구성 요소가 활성화되어 있는 동안	예, 구성 요소 파일에서 정의됨

설정 파일 해결에 대한 자세한 내용은 설정을 참조하세요. 엔터프라이즈 관리자는 `allowManagedHooksOnly` 를 사용하여 사용자, 프로젝트 및 plugin hook을 차단할 수 있습니다. Hook 구성을 참조하세요.

Matcher 패턴

`matcher` 필드는 hook이 발생할 때를 필터링하는 정규식 문자열입니다. `"*"`, `""` 또는 `matcher` 를 완전히 생략하여 모든 발생과 일치시킵니다. 각 이벤트 유형은 다른 필드에서 일치합니다.

이벤트	Matcher가 필터링하는 항목	예제 matcher 값
`PreToolUse`, `PostToolUse`, `PostToolUseFailure`, `PermissionRequest`	도구 이름	`Bash`, `Edit|Write`, `mcp__.*`
`SessionStart`	세션이 시작된 방식	`startup`, `resume`, `clear`, `compact`

이벤트	Matcher가 필터링하는 항목	예제 matcher 값
`SessionEnd`	세션이 종료된 이유	`clear`, `logout`, `prompt_input_exit`, `bypass_permissions_disabled`, `other`
`Notification`	알림 유형	`permission_prompt`, `idle_prompt`, `auth_success`, `elicitation_dialog`
`SubagentStart`	에이전트 유형	`Bash`, `Explore`, `Plan` 또는 사용자 정의 에이전트 이름
`PreCompact`	압축을 트리거한 항목	`manual`, `auto`
`SubagentStop`	에이전트 유형	`SubagentStart`와 동일한 값
`ConfigChange`	구성 소스	`user_settings`, `project_settings`, `local_settings`, `policy_settings`, `skills`
`UserPromptSubmit`, `Stop`, `TeammateIdle`, `TaskCompleted`, `WorktreeCreate`, `WorktreeRemove`, `InstructionsLoaded`	matcher 지원 없음	모든 발생에서 항상 발생

matcher는 정규식이므로 `Edit|Write`는 두 도구와 일치하고 `Notebook.*`은 Notebook으로 시작하는 모든 도구와 일치합니다. matcher는 Claude Code가 stdin의 hook에 전송하는 JSON 입력의 필드에 대해 실행됩니다. 도구 이벤트의 경우 해당 필드는 `tool_name`입니다. 각 hook 이벤트 섹션에서는 matcher 값의 전체 집합과 해당 이벤트의 입력 스키마를 나열합니다.

이 예제는 Claude가 파일을 쓰거나 편집할 때만 linting 스크립트를 실행합니다.

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "/path/to/lint-check.sh"
          }
        ]
      }
    ]
  }
}
```

`UserPromptSubmit`, `Stop`, `TeammateIdle`, `TaskCompleted`, `WorktreeCreate`, `WorktreeRemove`, `InstructionsLoaded`는 matcher를 지원하지 않으며 모든 발생에서 항상 발생합니다. 이러한 이벤트에 `matcher` 필드를 추가하면 자동으로 무시됩니다.

MCP 도구 일치

MCP 서버 도구는 도구 이벤트 (`PreToolUse`, `PostToolUse`, `PostToolUseFailure`, `PermissionRequest`)에서 일반 도구로 나타나므로 다른 도구 이름과 동일한 방식으로 일치시킬 수 있습니다.

MCP 도구는 `mcp__<server>__<tool>` 패턴을 따릅니다. 예를 들어:

- `mcp__memory__create_entities` : Memory 서버의 create entities 도구
- `mcp__filesystem__read_file` : Filesystem 서버의 read file 도구
- `mcp__github__search_repositories` : GitHub 서버의 search 도구

정규식 패턴을 사용하여 특정 MCP 도구 또는 도구 그룹을 대상으로 합니다.

- `mcp__memory__.*`는 `memory` 서버의 모든 도구와 일치합니다.
- `mcp__.*__write.*`는 모든 서버의 “write”를 포함하는 모든 도구와 일치합니다.

이 예제는 모든 memory 서버 작업을 기록하고 모든 MCP 서버의 쓰기 작업을 검증합니다.

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "mcp__memory__.*",
        "hooks": [
          {
            "type": "command",
            "command": "echo 'Memory operation initiated' >> ~/mcp-operations.log"
          }
        ]
      },
      {
        "matcher": "mcp__.*__write.*",
        "hooks": [
          {
            "type": "command",
            "command": "/home/user/scripts/validate-mcp-write.py"
          }
        ]
      }
    ]
  }
}
```

Hook 핸들러 필드

내부 `hooks` 배열의 각 객체는 hook 핸들러입니다. matcher가 일치할 때 실행되는 셸 명령, HTTP 엔드포인트, LLM 프롬프트 또는 에이전트입니다. 네 가지 유형이 있습니다.

- **명령 hook** (`type: "command"`): 셸 명령을 실행합니다. 스크립트는 이벤트의 JSON 입력을 stdin에서 받고 종료 코드와 stdout을 통해 결과를 다시 전달합니다.
- **HTTP hook** (`type: "http"`): 이벤트의 JSON 입력을 HTTP POST 요청으로 URL에 전송합니다. 엔드포인트는 명령 hook과 동일한 JSON 출력 형식을 사용하여 응답 본문을 통해 결과를 다시 전달합니다.
- **프롬프트 hook** (`type: "prompt"`): Claude 모델에 단일 턴 평가를 위한 프롬프트를 전송합니다. 모델은 yes/no 결정을 JSON으로 반환합니다. 프롬프트 기반 hook을 참조하세요.

- **에이전트 hook** (`type: "agent"`): Read, Grep, Glob과 같은 도구를 사용하여 결정을 반환하기 전에 조건을 확인할 수 있는 subagent를 생성합니다. 에이전트 기반 hook을 참조하세요.

공통 필드

이러한 필드는 모든 hook 유형에 적용됩니다.

필드	필수	설명
`type`	예	`"command"`, `"http"`, `"prompt"` 또는 `"agent"`
`timeout`	아니오	취소하기 전 초 단위. 기본값: 명령의 경우 600, 프롬프트의 경우 30, 에이전트의 경우 60
`statusMessage`	아니오	hook이 실행되는 동안 표시되는 사용자 정의 스피너 메시지
`once`	아니오	`true` 인 경우 세션당 한 번만 실행한 후 제거됩니다. Skill만 해당, 에이전트 아님. Skill 및 에이전트의 Hook 참조

명령 hook 필드

공통 필드 외에도 명령 hook은 이러한 필드를 허용합니다.

필드	필수	설명
`command`	예	실행할 셸 명령
`async`	아니오	`true` 인 경우 차단하지 않고 백그라운드에서 실행됩니다. 백그라운드에서 hook 실행 참조

HTTP hook 필드

공통 필드 외에도 HTTP hook은 이러한 필드를 허용합니다.

필드	필수	설명
`url`	예	POST 요청을 전송할 URL
`headers`	아니오	키-값 쌍으로 된 추가 HTTP 헤더. 값은 `$VAR_NAME` 또는 `${VAR_NAME}` 구문을 사용한 환경 변수 보간을 지원합니다. `allowedEnvVars` 에 나열된 변수만 해결됩니다.
`allowedEnvVars`	아니오	헤더 값에 보간될 수 있는 환경 변수 이름 목록. 나열되지 않은 변수에 대한 참조는 빈 문자열로 바뀝니다. 환경 변수 보간이 작동하려면 필수입니다.

Claude Code는 hook의 JSON 입력을 `Content-Type: application/json` 과 함께 POST 요청 본문으로 전송합니다. 응답 본문은 명령 hook과 동일한 JSON 출력 형식을 사용합니다.

오류 처리는 명령 hook과 다릅니다. 2xx가 아닌 응답, 연결 실패, 시간 초과는 모두 실행을 계속하도록 하는 차단하지 않는 오류를 생성합니다. 도구 호출을 차단하거나 권한을 거부하려면 `decision: "block"` 또는 `permissionDecision: "deny"` 를 포함하는 JSON 본문이 있는 2xx 응답을 반환합니다.

이 예제는 `PreToolUse` 이벤트를 로컬 검증 서비스로 전송하고 `MY_TOKEN` 환경 변수의 토큰으로 인증합니다.

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "http",
            "url": "http://localhost:8080/hooks/pre-tool-use",
            "timeout": 30,
            "headers": {
              "Authorization": "Bearer $MY_TOKEN"
            },
            "allowedEnvVars": ["MY_TOKEN"]
          }
        ]
      }
    ]
  }
}
```

> **Note:**
>
> HTTP hook은 설정 JSON을 직접 편집하여 구성해야 합니다. `/hooks` 대화형 메뉴는 명령 hook 추가만 지원합니다.

프롬프트 및 에이전트 hook 필드

공통 필드 외에도 프롬프트 및 에이전트 hook은 이러한 필드를 허용합니다.

필드	필수	설명
`prompt`	예	모델에 전송할 프롬프트 텍스트. hook 입력 JSON에 대한 자리 표시자로 `$ARGUMENTS` 사용
`model`	아니오	평가에 사용할 모델. 기본값은 빠른 모델

모든 일치하는 hook은 병렬로 실행되며 동일한 핸들러는 자동으로 중복 제거됩니다. 명령 hook은 명령 문자열로 중복 제거되고 HTTP hook은 URL로 중복 제거됩니다. 핸들러는 현재 디렉토리에서 Claude Code의 환경으로 실행됩니다. `$CLAUDE_CODE_REMOTE` 환경 변수는 원격 웹 환경에서 `"true"` 로 설정되고 로컬 CLI에서는 설정되지 않습니다.

경로별로 스크립트 참조

환경 변수를 사용하여 hook이 실행될 때의 작업 디렉토리와 관계없이 프로젝트 또는 plugin 루트를 기준으로 hook 스크립트를 참조합니다.

- `$CLAUDE_PROJECT_DIR` : 프로젝트 루트. 공백이 있는 경로를 처리하려면 따옴표로 감싸세요.
- `${CLAUDE_PLUGIN_ROOT}` : plugin의 루트 디렉토리, plugin과 함께 번들된 스크립트의 경우.

프로젝트 스크립트

이 예제는 `$CLAUDE_PROJECT_DIR` 을 사용하여 `Write` 또는 `Edit` 도구 호출 후 프로젝트의 `.claude/hooks/` 디렉토리에서 스타일 검사기를 실행합니다.

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "\"$CLAUDE_PROJECT_DIR\"/.claude/hooks/check-style.sh"
          }
        ]
      }
    ]
  }
}
```

Plugin 스크립트

`hooks/hooks.json` 에서 plugin hook을 정의하고 선택적 최상위 `description` 필드를 포함합니다. plugin이 활성화되면 해당 hook이 사용자 및 프로젝트 hook과 병합됩니다.

이 예제는 plugin과 함께 번들된 형식 지정 스크립트를 실행합니다.

```
{
  "description": "Automatic code formatting",
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PLUGIN_ROOT}/scripts/format.sh",
            "timeout": 30
          }
        ]
      }
    ]
  }
}
```

plugin hook 생성에 대한 자세한 내용은 plugin 구성 요소 참조를 참조하세요.

Skill 및 에이전트의 Hook

설정 파일 및 plugin 외에도 hook은 frontmatter를 사용하여 skill 및 subagent에서 직접 정의할 수 있습니다. 이러한 hook은 구성 요소의 수명 주기로 범위가 지정되며 해당 구성 요소가 활성화되어 있을 때만 실행됩니다.

모든 hook 이벤트가 지원됩니다. subagent의 경우 `Stop` hook은 subagent가 완료될 때 발생하는 이벤트이므로 자동으로 `SubagentStop` 으로 변환됩니다.

Hook은 설정 기반 hook과 동일한 구성 형식을 사용하지만 구성 요소의 수명으로 범위가 지정되고 완료될 때 정리됩니다.

이 skill은 각 `Bash` 명령 전에 보안 검증 스크립트를 실행하는 `PreToolUse` hook을 정의합니다.

```
---
name: secure-operations
description: Perform operations with security checks
hooks:
  PreToolUse:
    - matcher: "Bash"
      hooks:
        - type: command
          command: "./scripts/security-check.sh"
---
```

에이전트는 YAML frontmatter에서 동일한 형식을 사용합니다.

`/hooks` 메뉴

Claude Code에서 `/hooks`를 입력하여 대화형 hook 관리자를 열고 설정 파일을 직접 편집하지 않고 hook을 보고, 추가하고, 삭제할 수 있습니다. 단계별 설명은 가이드의 첫 번째 hook 설정을 참조하세요.

메뉴의 각 hook은 소스를 나타내는 괄호 접두사로 표시됩니다.

- `[User]`: `~/.claude/settings.json`에서
- `[Project]`: `.claude/settings.json`에서
- `[Local]`: `.claude/settings.local.json`에서
- `[Plugin]`: plugin의 `hooks/hooks.json`에서, 읽기 전용

Hook 비활성화 또는 제거

hook을 제거하려면 설정 JSON 파일에서 해당 항목을 삭제하거나 `/hooks` 메뉴를 사용하여 hook을 선택하고 삭제합니다.

모든 hook을 제거하지 않고 일시적으로 비활성화하려면 설정 파일에서 `"disableAllHooks": true`를 설정하거나 `/hooks` 메뉴의 토글을 사용합니다. 구성에 유지하면서 개별 hook을 비활성화할 수 있는 방법은 없습니다.

`disableAllHooks` 설정은 관리형 설정 계층을 준수합니다. 관리자가 관리형 정책 설정을 통해 hook을 구성한 경우 사용자, 프로젝트 또는 로컬 설정에서 설정된 `disableAllHooks`는 해당 관리형 hook을 비활성화할 수 없습니다. 관리형 설정 수준에서 설정된 `disableAllHooks`만 관리형 hook을 비활성화할 수 있습니다.

설정 파일의 hook에 대한 직접 편집은 즉시 적용되지 않습니다. Claude Code는 시작 시 hook의 스냅샷을 캡처하고 세션 전체에서 사용합니다. 이는 악의적이거나 실수로 인한 hook 수정이 검토 없이 세션 중간에 적용되는 것을 방지합니다. hook이 외부에서 수정되면 Claude Code는 경고하고 변경 사항이 적용되기 전에 `/hooks` 메뉴에서 검토를 요구합니다.

Hook 입출력

명령 hook은 stdin을 통해 JSON 데이터를 받고 종료 코드, stdout, stderr를 통해 결과를 전달합니다. HTTP hook은 POST 요청 본문으로 동일한 JSON을 받고 HTTP 응답 본문을 통해 결과를 전달합니다. 이 섹션에서는 모든 이벤트에 공통적인 필드와 동작을 다룹니다. Hook 이벤트 아래의 각 이벤트 섹션에는 특정 입력 스키마와 결정 제어 옵션이 포함됩니다.

공통 입력 필드

모든 hook 이벤트는 각 hook 이벤트 섹션에서 문서화된 이벤트 특정 필드 외에 이러한 필드를 JSON으로 받습니다. 명령 hook의 경우 이 JSON은 stdin을 통해 도착합니다. HTTP hook의 경우 POST 요청 본문으로 도착합니다.

필드	설명
`session_id`	현재 세션 식별자
`transcript_path`	대화 JSON 경로
`cwd`	hook이 호출될 때의 현재 작업 디렉토리
`permission_mode`	현재 권한 모드: `"default"`, `"plan"`, `"acceptEdits"`, `"dontAsk"` 또는 `"bypassPermissions"`
`hook_event_name`	발생한 이벤트의 이름

`--agent` 로 실행하거나 subagent 내부에서 실행할 때 두 가지 추가 필드가 포함됩니다.

필드	설명
`agent_id`	subagent의 고유 식별자. hook이 subagent 호출 내부에서 발생할 때만 존재합니다. 이를 사용하여 subagent hook 호출을 메인 스레드 호출과 구별합니다.
`agent_type`	에이전트 이름 (예: `"Explore"` 또는 `"security-reviewer"`). 세션이 `--agent` 를 사용하거나 hook이 subagent 내부에서 발생할 때 존재합니다. subagent의 경우 subagent의 유형이 세션의 `--agent` 값보다 우선합니다.

예를 들어 Bash 명령에 대한 `PreToolUse` hook은 stdin에서 다음을 받습니다.

```
{
  "session_id": "abc123",
  "transcript_path": "/home/user/.claude/projects/.../transcript.jsonl",
  "cwd": "/home/user/my-project",
  "permission_mode": "default",
  "hook_event_name": "PreToolUse",
  "tool_name": "Bash",
  "tool_input": {
    "command": "npm test"
  }
}
```

`tool_name` 및 `tool_input` 필드는 이벤트 특정입니다. 각 hook 이벤트 섹션에서는 해당 이벤트의 추가 필드를 문서화합니다.

종료 코드 출력

hook 명령의 종료 코드는 Claude Code에 작업을 진행할지, 차단할지 또는 무시할지를 알려줍니다.

종료 0은 성공을 의미합니다. Claude Code는 JSON 출력 필드에 대해 stdout을 구문 분석합니다. JSON 출력은 종료 0에서만 처리됩니다. 대부분의 이벤트에서 stdout은 자세한 모드(`Ctrl+O`)에서만 표시됩니다. 예외는 `UserPromptSubmit` 및 `SessionStart`이며, 여기서 stdout은 Claude가 보고 작용할 수 있는 컨텍스트로 추가됩니다.

종료 2는 차단 오류를 의미합니다. Claude Code는 stdout과 그 안의 JSON을 무시합니다. 대신 stderr 텍스트가 Claude에 오류 메시지로 피드백됩니다. 효과는 이벤트에 따라 다릅니다. `PreToolUse`는 도구 호출을 차단하고 `UserPromptSubmit`은 프롬프트를 거부합니다. 전체 목록은 이벤트별 종료 코드 2 동작을 참조하세요.

다른 종료 코드는 차단하지 않는 오류입니다. stderr는 자세한 모드(`Ctrl+O`)에서 표시되고 실행이 계속됩니다.

예를 들어 위험한 Bash 명령을 차단하는 hook 명령 스크립트:

```
#!/bin/bash
## stdin에서 JSON 입력을 읽고 명령을 확인합니다
command=$(jq -r '.tool_input.command' < /dev/stdin)

if [[ "$command" == rm* ]]; then
  echo "Blocked: rm commands are not allowed" >&2
  exit 2  # 차단 오류: 도구 호출이 방지됨
fi

exit 0  # 성공: 도구 호출이 진행됨
```

이벤트별 종료 코드 2 동작

종료 코드 2는 hook이 "멈춰, 이것을 하지 마"를 신호하는 방법입니다. 효과는 이벤트에 따라 다릅니다. 일부 이벤트는 차단할 수 있는 작업 (아직 발생하지 않은 도구 호출 등)을 나타내고 다른 이벤트는 이미 발생했거나 방지할 수 없는 것을 나타냅니다.

Hook 이벤트	차단 가능?	종료 코드 2에서 발생하는 일
`PreToolUse`	예	도구 호출을 차단합니다
`PermissionRequest`	예	권한을 거부합니다
`UserPromptSubmit`	예	프롬프트 처리를 차단하고 프롬프트를 지웁니다
`Stop`	예	Claude가 중지되는 것을 방지하고 대화를 계속합니다
`SubagentStop`	예	subagent가 중지되는 것을 방지합니다
`TeammateIdle`	예	팀원이 유휴 상태가 되는 것을 방지합니다 (팀원이 계속 작동)
`TaskCompleted`	예	작업이 완료로 표시되는 것을 방지합니다
`ConfigChange`	예	구성 변경이 적용되는 것을 차단합니다 (`policy_settings` 제외)

Hook 이벤트	차단 가능?	종료 코드 2에서 발생하는 일
`PostToolUse`	아니오	Claude에 stderr을 표시합니다 (도구가 이미 실행됨)
`PostToolUseFailure`	아니오	Claude에 stderr을 표시합니다 (도구가 이미 실패함)
`Notification`	아니오	사용자에게만 stderr을 표시합니다
`SubagentStart`	아니오	사용자에게만 stderr을 표시합니다
`SessionStart`	아니오	사용자에게만 stderr을 표시합니다
`SessionEnd`	아니오	사용자에게만 stderr을 표시합니다
`PreCompact`	아니오	사용자에게만 stderr을 표시합니다
`WorktreeCreate`	예	0이 아닌 종료 코드로 인해 worktree 생성이 실패합니다
`WorktreeRemove`	아니오	실패는 디버그 모드에서만 기록됩니다
`InstructionsLoaded`	아니오	종료 코드는 무시됩니다

HTTP 응답 처리

HTTP hook은 종료 코드와 stdout 대신 HTTP 상태 코드와 응답 본문을 사용합니다.

- **2xx 빈 본문**: 성공, 출력 없이 종료 코드 0과 동등
- **2xx 일반 텍스트 본문**: 성공, 텍스트가 컨텍스트로 추가됨
- **2xx JSON 본문**: 성공, 명령 hook과 동일한 JSON 출력 스키마를 사용하여 구문 분석됨
- **2xx가 아닌 상태**: 차단하지 않는 오류, 실행이 계속됨
- **연결 실패 또는 시간 초과**: 차단하지 않는 오류, 실행이 계속됨

명령 hook과 달리 HTTP hook은 상태 코드만으로 차단 오류를 신호할 수 없습니다. 도구 호출을 차단하거나 권한을 거부하려면 적절한 결정 필드를 포함하는 JSON 본문이 있는 2xx 응답을 반환합니다.

JSON 출력

종료 코드를 사용하면 허용 또는 차단할 수 있지만 JSON 출력은 더 세밀한 제어를 제공합니다. 종료 코드 2로 차단하는 대신 종료 0으로 JSON 객체를 stdout에 인쇄합니다. Claude Code는 해당 JSON에서 특정 필드를 읽어 결정 제어를 포함한 동작을 제어합니다.

> **Note:**
>
> hook당 하나의 접근 방식을 선택해야 합니다. 둘 다 선택하지 마세요. 종료 코드만 사용하여 신호하거나 종료 0으로 JSON을 인쇄하여 구조화된 제어를 합니다. Claude Code는 종료 0에서만 JSON을 처리합니다. 종료 2로 종료하면 JSON은 무시됩니다.

hook의 stdout은 JSON 객체만 포함해야 합니다. 셸 프로필이 시작 시 텍스트를 인쇄하면 JSON 구문 분석을 방해할 수 있습니다. 문제 해결 가이드의 JSON 검증 실패를 참조하세요.

JSON 객체는 세 가지 종류의 필드를 지원합니다.

- **`continue` 와 같은 범용 필드**는 모든 이벤트에서 작동합니다. 이들은 아래 표에 나열되어 있습니다.
- **최상위 `decision` 및 `reason` **은 일부 이벤트에서 차단하거나 피드백을 제공하는 데 사용됩니다.
- ** `hookSpecificOutput` **은 더 풍부한 제어가 필요한 이벤트를 위한 중첩 객체입니다. 이벤트 이름으로 설정된 `hookEventName` 필드가 필요합니다.

필드	기본값	설명
`continue`	`true`	`false` 인 경우 hook이 실행된 후 Claude가 완전히 중지됩니다. 모든 이벤트 특정 결정 필드보다 우선합니다
`stopReason`	없음	`continue` 가 `false` 일 때 사용자에게 표시되는 메시지. Claude에는 표시되지 않음
`suppressOutput`	`false`	`true` 인 경우 자세한 모드 출력에서 stdout을 숨깁니다
`systemMessage`	없음	사용자에게 표시되는 경고 메시지

이벤트 유형과 관계없이 Claude를 완전히 중지하려면:

```
{ "continue": false, "stopReason": "Build failed, fix errors before continuing" }
```

결정 제어

모든 이벤트가 JSON을 통해 차단하거나 동작을 제어하는 것을 지원하지는 않습니다. 지원하는 이벤트는 각각 다른 필드 집합을 사용하여 해당 결정을 표현합니다. hook을 작성하기 전에 이 표를 빠른 참조로 사용하세요.

이벤트	결정 패턴	주요 필드
UserPromptSubmit, PostToolUse, PostToolUseFailure, Stop, SubagentStop, ConfigChange	최상위 `decision`	`decision: "block"`, `reason`
TeammateIdle, TaskCompleted	종료 코드 또는 `continue: false`	종료 코드 2는 stderr 피드백으로 작업을 차단합니다. JSON `{"continue": false, "stopReason": "..."}` 또한 팀원을 완전히 중지하여 `Stop` hook 동작과 일치합니다
PreToolUse	`hookSpecificOutput`	`permissionDecision` (allow/deny/ask), `permissionDecisionReason`
PermissionRequest	`hookSpecificOutput`	`decision.behavior` (allow/deny)
WorktreeCreate	stdout 경로	Hook은 생성된 worktree의 절대 경로를 인쇄합니다. 0이 아닌 종료는 생성을 실패합니다
WorktreeRemove, Notification, SessionEnd, PreCompact, InstructionsLoaded	없음	결정 제어 없음. 로깅 또는 정리와 같은 부작용에 사용됨

다음은 각 패턴의 실제 예입니다.

최상위 결정

`UserPromptSubmit`, `PostToolUse`, `PostToolUseFailure`, `Stop`, `SubagentStop`, `ConfigChange`에서 사용됩니다. 유일한 값은 `"block"`입니다. 작업을 진행하도록 허용하려면 JSON에서 `decision`을 생략하거나 JSON 없이 종료 0으로 종료합니다.

```
{
  "decision": "block",
  "reason": "Test suite must pass before proceeding"
}
```

PreToolUse

더 풍부한 제어를 위해 `hookSpecificOutput`을 사용합니다. 허용, 거부 또는 사용자에게 에스컬레이션할 수 있습니다. 도구 입력을 실행 전에 수정하거나 Claude를 위한 추가 컨텍스트를 주입할 수도 있습니다. 전체 옵션 집합은 PreToolUse 결정 제어를 참조하세요.

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "deny",
    "permissionDecisionReason": "Database writes are not allowed"
  }
}
```

PermissionRequest

`hookSpecificOutput`을 사용하여 사용자를 대신하여 권한 요청을 허용하거나 거부합니다. 허용할 때 도구의 입력을 수정하거나 권한 규칙을 적용하여 사용자가 다시 프롬프트되지 않도록 할 수 있습니다. 전체 옵션 집합은 PermissionRequest 결정 제어를 참조하세요.

```
{
  "hookSpecificOutput": {
    "hookEventName": "PermissionRequest",
    "decision": {
      "behavior": "allow",
      "updatedInput": {
        "command": "npm run lint"
      }
    }
  }
}
```

Bash 명령 검증, 프롬프트 필터링, 자동 승인 스크립트를 포함한 확장 예제는 가이드의 자동화할 수 있는 항목과 Bash 명령 검증기 참조 구현을 참조하세요.

Hook 이벤트

각 이벤트는 hook이 실행될 수 있는 Claude Code의 수명 주기의 지점에 해당합니다. 아래 섹션은 수명 주기와 일치하도록 정렬됩니다. 세션 설정에서 에이전트 루프를 거쳐 세션 종료까지입니다. 각 섹션에서는 이벤트가 언제 발생하는지, 지원하는 matcher, 받는 JSON 입력, 출력을 통해 동작을 제어하는 방법을 설명합니다.

SessionStart

Claude Code가 새 세션을 시작하거나 기존 세션을 재개할 때 실행됩니다. 기존 문제나 코드베이스의 최근 변경 사항과 같은 개발 컨텍스트를 로드하거나 환경 변수를 설정하는 데 유용합니다. 스크립트가 필요하지 않은 정적 컨텍스트의 경우 CLAUDE.md를 대신 사용하세요.

SessionStart는 모든 세션에서 실행되므로 이러한 hook을 빠르게 유지하세요. `type: "command"` hook만 지원됩니다.

matcher 값은 세션이 시작된 방식에 해당합니다.

Matcher	언제 발생하는지
`startup`	새 세션
`resume`	`--resume`, `--continue` 또는 `/resume`
`clear`	`/clear`
`compact`	자동 또는 수동 압축

SessionStart 입력

공통 입력 필드 외에도 SessionStart hook은 `source`, `model`, 선택적으로 `agent_type`을 받습니다. `source` 필드는 세션이 시작된 방식을 나타냅니다. 새 세션의 경우 `"startup"`, 재개된 세션의 경우 `"resume"`, `/clear` 후 `"clear"`, 압축 후 `"compact"`. `model` 필드는 모델 식별자를 포함합니다. `claude --agent <name>`으로 Claude Code를 시작하면 `agent_type` 필드에 에이전트 이름이 포함됩니다.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "SessionStart",
  "source": "startup",
  "model": "claude-sonnet-4-6"
}
```

SessionStart 결정 제어

hook 스크립트가 stdout에 인쇄하는 모든 텍스트는 Claude의 컨텍스트로 추가됩니다. 모든 hook에서 사용 가능한 JSON 출력 필드 외에도 이러한 이벤트 특정 필드를 반환할 수 있습니다.

필드	설명
`additionalContext`	Claude의 컨텍스트에 추가되는 문자열. 여러 hook의 값이 연결됨

```
{
  "hookSpecificOutput": {
    "hookEventName": "SessionStart",
    "additionalContext": "My additional context here"
  }
}
```

환경 변수 유지

SessionStart hook은 `CLAUDE_ENV_FILE` 환경 변수에 액세스할 수 있으며, 이는 후속 Bash 명령에 대한 환경 변수를 유지할 수 있는 파일 경로를 제공합니다.

개별 환경 변수를 설정하려면 `CLAUDE_ENV_FILE`에 `export` 문을 작성합니다. 다른 hook에서 설정한 변수를 유지하려면 추가 (`>>`)를 사용합니다.

```
#!/bin/bash

if [ -n "$CLAUDE_ENV_FILE" ]; then
  echo 'export NODE_ENV=production' >> "$CLAUDE_ENV_FILE"
  echo 'export DEBUG_LOG=true' >> "$CLAUDE_ENV_FILE"
  echo 'export PATH="$PATH:./node_modules/.bin"' >> "$CLAUDE_ENV_FILE"
fi

exit 0
```

설정 명령의 모든 환경 변경을 캡처하려면 내보낸 변수를 이전과 이후로 비교합니다.

```
#!/bin/bash

ENV_BEFORE=$(export -p | sort)

## 환경을 수정하는 설정 명령을 실행합니다
source ~/.nvm/nvm.sh
nvm use 20

if [ -n "$CLAUDE_ENV_FILE" ]; then
  ENV_AFTER=$(export -p | sort)
  comm -13 <(echo "$ENV_BEFORE") <(echo "$ENV_AFTER") >> "$CLAUDE_ENV_FILE"
fi

exit 0
```

이 파일에 작성된 모든 변수는 세션 중에 Claude Code가 실행하는 모든 후속 Bash 명령에서 사용 가능합니다.

> **Note:**
>
> `CLAUDE_ENV_FILE`은 SessionStart hook에서 사용 가능합니다. 다른 hook 유형은 이 변수에 액세스할 수 없습니다.

InstructionsLoaded

`CLAUDE.md` 또는 `.claude/rules/*.md` 파일이 컨텍스트에 로드될 때 발생합니다. 이 이벤트는 세션 시작 시 즉시 로드된 파일에 대해 발생하고 나중에 파일이 지연 로드될 때 다시 발생합니다. 예를 들어 Claude가 중첩된 `CLAUDE.md`를 포함하는 하위 디렉토리에 액세스할 때 또는 `paths:` frontmatter가 있는 조건부 규칙이 일치할 때입니다. hook은 차단 또는 결정 제어를 지원하지 않습니다. 관찰 가능성 목적으로 비동기적으로 실행됩니다.

InstructionsLoaded는 matcher를 지원하지 않으며 모든 로드 발생에서 발생합니다.

InstructionsLoaded 입력

공통 입력 필드 외에도 InstructionsLoaded hook은 이러한 필드를 받습니다.

필드	설명
`file_path`	로드된 명령 파일의 절대 경로
`memory_type`	파일의 범위: `"User"`, `"Project"`, `"Local"` 또는 `"Managed"`
`load_reason`	파일이 로드된 이유: `"session_start"`, `"nested_traversal"`, `"path_glob_match"` 또는 `"include"`
`globs`	파일의 `paths:` frontmatter의 경로 glob 패턴 (있는 경우). `path_glob_match` 로드에만 존재
`trigger_file_path`	지연 로드의 경우 이 로드를 트리거한 파일의 경로
`parent_file_path`	`include` 로드의 경우 이를 포함한 부모 명령 파일의 경로

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../transcript.jsonl",
  "cwd": "/Users/my-project",
  "permission_mode": "default",
  "hook_event_name": "InstructionsLoaded",
  "file_path": "/Users/my-project/CLAUDE.md",
  "memory_type": "Project",
  "load_reason": "session_start"
}
```

InstructionsLoaded 결정 제어

InstructionsLoaded hook은 결정 제어가 없습니다. 명령 로드를 차단하거나 수정할 수 없습니다. 감사 로깅, 규정 준수 추적 또는 관찰 가능성을 위해 이 이벤트를 사용합니다.

UserPromptSubmit

사용자가 프롬프트를 제출할 때 Claude가 처리하기 전에 실행됩니다. 이를 통해 프롬프트/대화를 기반으로 추가 컨텍스트를 추가하거나 프롬프트를 검증하거나 특정 유형의 프롬프트를 차단할 수 있습니다.

UserPromptSubmit 입력

공통 입력 필드 외에도 UserPromptSubmit hook은 사용자가 제출한 텍스트를 포함하는 `prompt` 필드를 받습니다.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "UserPromptSubmit",
  "prompt": "Write a function to calculate the factorial of a number"
}
```

UserPromptSubmit 결정 제어

`UserPromptSubmit` hook은 사용자 프롬프트 처리 여부를 제어하고 컨텍스트를 추가할 수 있습니다. 모든 JSON 출력 필드를 사용할 수 있습니다.

종료 코드 0에서 대화에 컨텍스트를 추가하는 두 가지 방법이 있습니다.

- **일반 텍스트 stdout**: stdout에 작성된 JSON이 아닌 텍스트는 컨텍스트로 추가됩니다.
- **`additionalContext` 가 있는 JSON**: 더 많은 제어를 위해 아래 JSON 형식을 사용합니다. `additionalContext` 필드는 컨텍스트로 추가됩니다.

일반 stdout은 대화에서 hook 출력으로 표시됩니다. `additionalContext` 필드는 더 신중하게 추가됩니다.

프롬프트를 차단하려면 `decision` 을 `"block"` 으로 설정한 JSON 객체를 반환합니다.

필드	설명
`decision`	`"block"` 은 프롬프트가 처리되는 것을 방지하고 컨텍스트에서 지웁니다. 프롬프트를 진행하도록 허용하려면 생략합니다
`reason`	`decision` 이 `"block"` 일 때 사용자에게 표시됩니다. 컨텍스트에 추가되지 않음
`additionalContext`	Claude의 컨텍스트에 추가되는 문자열

```
{
  "decision": "block",
  "reason": "Explanation for decision",
  "hookSpecificOutput": {
    "hookEventName": "UserPromptSubmit",
    "additionalContext": "My additional context here"
  }
}
```

> **Note:**
>
> JSON 형식은 간단한 사용 사례에는 필요하지 않습니다. 컨텍스트를 추가하려면 종료 코드 0으로 stdout에 일반 텍스트를 인쇄할 수 있습니다. 프롬프트를 차단하거나 더 구조화된 제어가 필요할 때 JSON을 사용합니다.

PreToolUse

Claude가 도구 매개변수를 생성한 후 도구 호출을 처리하기 전에 실행됩니다. 도구 이름에서 일치합니다. `Bash`, `Edit`, `Write`, `Read`, `Glob`, `Grep`, `Agent`, `WebFetch`, `WebSearch` 및 모든 MCP 도구 이름.

PreToolUse 결정 제어를 사용하여 도구 사용을 허용, 거부 또는 권한을 요청합니다.

PreToolUse 입력

공통 입력 필드 외에도 PreToolUse hook은 `tool_name`, `tool_input`, `tool_use_id` 를 받습니다. `tool_input` 필드는 도구에 따라 다릅니다.

Bash

셸 명령을 실행합니다.

필드	유형	예제	설명
`command`	문자열	`"npm test"`	실행할 셸 명령
`description`	문자열	`"Run test suite"`	명령이 수행하는 작업의 선택적 설명
`timeout`	숫자	`120000`	선택적 시간 초과 (밀리초)
`run_in_background`	부울	`false`	명령을 백그라운드에서 실행할지 여부

Write

파일을 생성하거나 덮어씁니다.

필드	유형	예제	설명
`file_path`	문자열	`"/path/to/file.txt"`	쓸 파일의 절대 경로
`content`	문자열	`"file content"`	파일에 쓸 내용

Edit

파일의 문자열을 바꿉니다.

필드	유형	예제	설명
`file_path`	문자열	`"/path/to/file.txt"`	편집할 파일의 절대 경로
`old_string`	문자열	`"original text"`	찾아서 바꿀 텍스트
`new_string`	문자열	`"replacement text"`	대체 텍스트
`replace_all`	부울	`false`	모든 발생을 바꿀지 여부

Read

파일 내용을 읽습니다.

필드	유형	예제	설명
`file_path`	문자열	`"/path/to/file.txt"`	읽을 파일의 절대 경로
`offset`	숫자	`10`	읽기를 시작할 선택적 줄 번호
`limit`	숫자	`50`	읽을 선택적 줄 수

Glob

glob 패턴과 일치하는 파일을 찾습니다.

필드	유형	예제	설명
`pattern`	문자열	`"**/*.ts"`	파일과 일치시킬 glob 패턴
`path`	문자열	`"/path/to/dir"`	검색할 선택적 디렉토리. 기본값은 현재 작업 디렉토리

Grep

정규식으로 파일 내용을 검색합니다.

필드	유형	예제	설명
`pattern`	문자열	`"TODO.*fix"`	검색할 정규식 패턴
`path`	문자열	`"/path/to/dir"`	검색할 선택적 파일 또는 디렉토리
`glob`	문자열	`"*.ts"`	파일을 필터링할 선택적 glob 패턴
`output_mode`	문자열	`"content"`	`"content"`, `"files_with_matches"` 또는 `"count"`. 기본값은 `"files_with_matches"`
`-i`	부울	`true`	대소문자를 구분하지 않는 검색

필드	유형	예제	설명
`multiline`	부울	`false`	다중 줄 일치 활성화

WebFetch

웹 콘텐츠를 가져오고 처리합니다.

필드	유형	예제	설명
`url`	문자열	`"https://example.com/api"`	콘텐츠를 가져올 URL
`prompt`	문자열	`"Extract the API endpoints"`	가져온 콘텐츠에서 실행할 프롬프트

WebSearch

웹을 검색합니다.

필드	유형	예제	설명
`query`	문자열	`"react hooks best practices"`	검색 쿼리
`allowed_domains`	배열	`["docs.example.com"]`	선택적: 이러한 도메인의 결과만 포함
`blocked_domains`	배열	`["spam.example.com"]`	선택적: 이러한 도메인의 결과 제외

Agent

subagent를 생성합니다.

필드	유형	예제	설명
`prompt`	문자열	`"Find all API endpoints"`	에이전트가 수행할 작업
`description`	문자열	`"Find API endpoints"`	작업의 짧은 설명

필드	유형	예제	설명
`subagent_type`	문자열	`"Explore"`	사용할 특화된 에이전트의 유형
`model`	문자열	`"sonnet"`	기본값을 재정의할 선택적 모델 별칭

PreToolUse 결정 제어

`PreToolUse` hook은 도구 호출 진행 여부를 제어할 수 있습니다. 최상위 `decision` 필드를 사용하는 다른 hook과 달리 PreToolUse는 `hookSpecificOutput` 객체 내에 결정을 반환합니다. 이는 더 풍부한 제어를 제공합니다. 세 가지 결과 (허용, 거부 또는 요청) 및 실행 전에 도구 입력을 수정하는 기능입니다.

필드	설명
`permissionDecision`	`"allow"` 는 권한 시스템을 우회하고, `"deny"` 는 도구 호출을 방지하고, `"ask"` 는 사용자에게 확인을 요청합니다
`permissionDecisionReason`	`"allow"` 및 `"ask"` 의 경우 사용자에게 표시되지만 Claude에는 표시되지 않습니다. `"deny"` 의 경우 Claude에 표시됩니다
`updatedInput`	실행 전에 도구의 입력 매개변수를 수정합니다. `"allow"` 와 결합하여 자동 승인하거나 `"ask"` 와 결합하여 수정된 입력을 사용자에게 표시합니다
`additionalContext`	도구가 실행되기 전에 Claude의 컨텍스트에 추가되는 문자열

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "allow",
    "permissionDecisionReason": "My reason here",
    "updatedInput": {
      "field_to_modify": "new value"
    },
    "additionalContext": "Current environment: production. Proceed with caution."
  }
}
```

> **Note:**
>
> PreToolUse는 이전에 최상위 `decision` 및 `reason` 필드를 사용했지만 이 이벤트에는 더 이상 사용되지 않습니다. 대신 `hookSpecificOutput.permissionDecision` 및 `hookSpecificOutput.permissionDecisionReason`을 사용합니다. 더 이상 사용되지 않는 값 `"approve"` 및 `"block"`은 각각 `"allow"` 및 `"deny"`로 매핑됩니다. PostToolUse 및 Stop과 같은 다른 이벤트는 계속 최상위 `decision` 및 `reason`을 현재 형식으로 사용합니다.

PermissionRequest

사용자에게 권한 대화 상자가 표시될 때 실행됩니다. PermissionRequest 결정 제어를 사용하여 사용자를 대신하여 허용하거나 거부합니다.

도구 이름에서 일치합니다. PreToolUse와 동일한 값입니다.

PermissionRequest 입력

PermissionRequest hook은 PreToolUse hook과 같은 `tool_name` 및 `tool_input` 필드를 받지만 `tool_use_id`는 없습니다. 선택적 `permission_suggestions` 배열에는 사용자가 권한 대화 상자에서 일반적으로 볼 수 있는 "항상 허용" 옵션이 포함됩니다. 차이점은 hook이 발생할 때입니다. PermissionRequest hook은 권한 대화 상자가 사용자에게 표시되려고 할 때 실행되고, PreToolUse hook은 권한 상태와 관계없이 도구 실행 전에 실행됩니다.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "PermissionRequest",
  "tool_name": "Bash",
  "tool_input": {
    "command": "rm -rf node_modules",
    "description": "Remove node_modules directory"
  },
  "permission_suggestions": [
    { "type": "toolAlwaysAllow", "tool": "Bash" }
  ]
}
```

PermissionRequest 결정 제어

`PermissionRequest` hook은 권한 요청을 허용하거나 거부할 수 있습니다. 모든 hook에서 사용 가능한 JSON 출력 필드 외에도 hook 스크립트는 이러한 이벤트 특정 필드가 있는 `decision` 객체를 반환할 수 있습니다.

필드	설명
`behavior`	`"allow"` 는 권한을 부여하고, `"deny"` 는 거부합니다
`updatedInput`	`"allow"` 만 해당: 실행 전에 도구의 입력 매개변수를 수정합니다
`updatedPermissions`	`"allow"` 만 해당: 권한 규칙 업데이트를 적용하며, 사용자가 “항상 허용” 옵션을 선택하는 것과 동등합니다
`message`	`"deny"` 만 해당: 권한이 거부된 이유를 Claude에 알립니다
`interrupt`	`"deny"` 만 해당: `true` 인 경우 Claude를 중지합니다

```
{
  "hookSpecificOutput": {
    "hookEventName": "PermissionRequest",
    "decision": {
      "behavior": "allow",
      "updatedInput": {
        "command": "npm run lint"
      }
    }
  }
}
```

PostToolUse

도구가 성공적으로 완료된 직후 실행됩니다.

도구 이름에서 일치합니다. PreToolUse와 동일한 값입니다.

PostToolUse 입력

`PostToolUse` hook은 도구가 이미 성공적으로 실행된 후에 발생합니다. 입력에는 도구에 전송된 인수인 `tool_input` 과 반환된 결과인 `tool_response` 가 모두 포함됩니다. 둘 다의 정확한 스키마는 도구에 따라 다릅니다.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "PostToolUse",
  "tool_name": "Write",
  "tool_input": {
    "file_path": "/path/to/file.txt",
    "content": "file content"
  },
  "tool_response": {
    "filePath": "/path/to/file.txt",
    "success": true
  },
  "tool_use_id": "toolu_01ABC123..."
}
```

PostToolUse 결정 제어

`PostToolUse` hook은 도구 실행 후 Claude에 피드백을 제공할 수 있습니다. 모든 hook에서 사용 가능한 JSON 출력 필드 외에도 hook 스크립트는 이러한 이벤트 특정 필드를 반환할 수 있습니다.

필드	설명
`decision`	`"block"` 은 Claude에 `reason` 을 표시합니다. 작업을 진행하도록 허용하려면 생략합니다
`reason`	`decision` 이 `"block"` 일 때 Claude에 표시되는 설명
`additionalContext`	Claude가 고려할 추가 컨텍스트
`updatedMCPToolOutput`	MCP 도구만 해당: 도구의 출력을 제공된 값으로 바꿉니다

```
{
  "decision": "block",
  "reason": "Explanation for decision",
  "hookSpecificOutput": {
    "hookEventName": "PostToolUse",
    "additionalContext": "Additional information for Claude"
  }
}
```

PostToolUseFailure

도구 실행이 실패할 때 실행됩니다. 이 이벤트는 오류를 throw하거나 실패 결과를 반환하는 도구 호출에 대해 발생합니다. 이를 사용하여 실패를 기록하고, 경고를 보내거나, Claude에 수정 피드백을 제공합니다.

도구 이름에서 일치합니다. PreToolUse와 동일한 값입니다.

PostToolUseFailure 입력

PostToolUseFailure hook은 PostToolUse와 동일한 `tool_name` 및 `tool_input` 필드를 받으며, 오류 정보는 최상위 필드로 받습니다.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "PostToolUseFailure",
  "tool_name": "Bash",
  "tool_input": {
    "command": "npm test",
    "description": "Run test suite"
  },
  "tool_use_id": "toolu_01ABC123...",
  "error": "Command exited with non-zero status code 1",
  "is_interrupt": false
}
```

필드	설명
`error`	무엇이 잘못되었는지 설명하는 문자열
`is_interrupt`	실패가 사용자 중단으로 인한 것인지 나타내는 선택적 부울

PostToolUseFailure 결정 제어

`PostToolUseFailure` hook은 도구 실패 후 Claude에 컨텍스트를 제공할 수 있습니다. 모든 hook에서 사용 가능한 JSON 출력 필드 외에도 hook 스크립트는 이러한 이벤트 특정 필드를 반환할 수 있습니다.

필드	설명
`additionalContext`	Claude가 오류와 함께 고려할 추가 컨텍스트

```
{
  "hookSpecificOutput": {
    "hookEventName": "PostToolUseFailure",
    "additionalContext": "Additional information about the failure for Claude"
  }
}
```

Notification

Claude Code가 알림을 보낼 때 실행됩니다. 알림 유형에서 일치합니다. `permission_prompt`, `idle_prompt`, `auth_success`, `elicitation_dialog`. matcher를 생략하여 모든 알림 유형에 대해 hook을 실행합니다.

별도의 matcher를 사용하여 알림 유형에 따라 다른 핸들러를 실행합니다. 이 구성은 Claude가 권한 승인이 필요할 때 권한 특정 경고 스크립트를 트리거하고 Claude가 유휴 상태일 때 다른 알림을 트리거합니다.

```
{
  "hooks": {
    "Notification": [
      {
        "matcher": "permission_prompt",
        "hooks": [
          {
            "type": "command",
            "command": "/path/to/permission-alert.sh"
          }
        ]
      },
      {
        "matcher": "idle_prompt",
        "hooks": [
          {
            "type": "command",
            "command": "/path/to/idle-notification.sh"
          }
        ]
      }
    ]
  }
}
```

Notification 입력

공통 입력 필드 외에도 Notification hook은 알림 텍스트가 있는 `message`, 선택적 `title`, 어떤 유형이 발생했는지 나타내는 `notification_type`을 받습니다.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "Notification",
  "message": "Claude needs your permission to use Bash",
  "title": "Permission needed",
  "notification_type": "permission_prompt"
}
```

Notification hook은 알림을 차단하거나 수정할 수 없습니다. 모든 hook에서 사용 가능한 JSON 출력 필드 외에도 `additionalContext`를 반환하여 대화에 컨텍스트를 추가할 수 있습니다.

필드	설명
`additionalContext`	Claude의 컨텍스트에 추가되는 문자열

SubagentStart

Agent 도구를 통해 Claude Code subagent가 생성될 때 실행됩니다. 에이전트 유형 이름으로 필터링하는 matcher를 지원합니다 (Bash, Explore, Plan과 같은 기본 제공 에이전트 또는 `.claude/agents/`의 사용자 정의 에이전트 이름).

SubagentStart 입력

공통 입력 필드 외에도 SubagentStart hook은 subagent의 고유 식별자가 있는 `agent_id`와 에이전트 이름이 있는 `agent_type` (Bash, Explore, Plan과 같은 기본 제공 에이전트 또는 사용자 정의 에이전트 이름)을 받습니다.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "SubagentStart",
  "agent_id": "agent-abc123",
  "agent_type": "Explore"
}
```

SubagentStart hook은 subagent 생성을 차단할 수 없지만 subagent에 컨텍스트를 주입할 수 있습니다. 모든 hook에서 사용 가능한 JSON 출력 필드 외에도 다음을 반환할 수 있습니다.

필드	설명
`additionalContext`	subagent의 컨텍스트에 추가되는 문자열

```
{
  "hookSpecificOutput": {
    "hookEventName": "SubagentStart",
    "additionalContext": "Follow security guidelines for this task"
  }
}
```

SubagentStop

Claude Code subagent가 응답을 마쳤을 때 실행됩니다. 에이전트 유형에서 일치합니다. SubagentStart와 동일한 값입니다.

SubagentStop 입력

공통 입력 필드 외에도 SubagentStop hook은 `stop_hook_active`, `agent_id`, `agent_type`, `agent_transcript_path`, `last_assistant_message`를 받습니다. `agent_type` 필드는 matcher 필터링에 사용되는 값입니다. `transcript_path`는 메인 세션의 대화이고 `agent_transcript_path`는 중첩된 `subagents/` 폴더에 저장된 subagent의 자체 대화입니다. `last_assistant_message` 필드는 subagent의 최종 응답의 텍스트 내용을 포함하므로 hook은 대화 파일을 구문 분석하지 않고도 액세스할 수 있습니다.

```
{
  "session_id": "abc123",
  "transcript_path": "~/.claude/projects/.../abc123.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "SubagentStop",
  "stop_hook_active": false,
  "agent_id": "def456",
  "agent_type": "Explore",
  "agent_transcript_path": "~/.claude/projects/.../abc123/subagents/agent-
def456.jsonl",
  "last_assistant_message": "Analysis complete. Found 3 potential issues..."
}
```

SubagentStop hook은 Stop hook과 동일한 결정 제어 형식을 사용합니다.

Stop

메인 Claude Code 에이전트가 응답을 마쳤을 때 실행됩니다. 중지가 사용자 중단으로 인해 발생한 경우 실행되지 않습니다.

Stop 입력

공통 입력 필드 외에도 Stop hook은 `stop_hook_active` 및 `last_assistant_message`를 받습니다. `stop_hook_active` 필드는 Claude Code가 이미 stop hook의 결과로 계속되고 있을 때 `true`입니다. 이 값을 확인하거나 대화를 처리하여 Claude Code가 무한정 실행되지 않도록 합니다. `last_assistant_message` 필드는 Claude의 최종 응답의 텍스트 내용을 포함하므로 hook은 대화 파일을 구문 분석하지 않고도 액세스할 수 있습니다.

```
{
  "session_id": "abc123",
  "transcript_path": "~/.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "Stop",
  "stop_hook_active": true,
  "last_assistant_message": "I've completed the refactoring. Here's a summary..."
}
```

Stop 결정 제어

`Stop` 및 `SubagentStop` hook은 Claude가 계속할지 여부를 제어할 수 있습니다. 모든 hook에서 사용 가능한 JSON 출력 필드 외에도 hook 스크립트는 이러한 이벤트 특정 필드를 반환할 수 있습니다.

필드	설명
`decision`	`"block"` 은 Claude가 중지되는 것을 방지합니다. Claude가 중지하도록 허용하려면 생략합니다
`reason`	`decision` 이 `"block"` 일 때 필수입니다. Claude가 계속해야 하는 이유를 알립니다

```
{
  "decision": "block",
  "reason": "Must be provided when Claude is blocked from stopping"
}
```

TeammateIdle

에이전트 팀 팀원이 자신의 턴을 마친 후 유휴 상태가 되려고 할 때 실행됩니다. 이를 사용하여 lint 검사 통과 또는 출력 파일 존재 확인과 같은 팀원이 작업을 중지하기 전에 품질 게이트를 적용합니다.

`TeammateIdle` hook이 코드 2로 종료되면 팀원은 stderr 메시지를 피드백으로 받고 유휴 상태가 되는 대신 계속 작동합니다. 팀원을 다시 실행하는 대신 완전히 중지하려면 `{"continue": false, "stopReason": "..."}` 이 있는 JSON을 반환합니다. TeammateIdle hook은 matcher를 지원하지 않으며 모든 발생에서 발생합니다.

TeammateIdle 입력

공통 입력 필드 외에도 TeammateIdle hook은 `teammate_name` 및 `team_name`을 받습니다.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "TeammateIdle",
  "teammate_name": "researcher",
  "team_name": "my-project"
}
```

필드	설명
`teammate_name`	유휴 상태가 되려고 하는 팀원의 이름
`team_name`	팀의 이름

TeammateIdle 결정 제어

TeammateIdle hook은 팀원 동작을 제어하는 두 가지 방법을 지원합니다.

- **종료 코드 2**: 팀원은 stderr 메시지를 피드백으로 받고 유휴 상태가 되는 대신 계속 작동합니다.
- **JSON** `{"continue": false, "stopReason": "..."}`: 팀원을 완전히 중지하여 `Stop` hook 동작과 일치합니다. `stopReason`은 사용자에게 표시됩니다.

이 예제는 팀원이 유휴 상태가 되도록 허용하기 전에 빌드 아티팩트가 존재하는지 확인합니다.

```
#!/bin/bash

if [ ! -f "./dist/output.js" ]; then
  echo "Build artifact missing. Run the build before stopping." >&2
  exit 2
fi

exit 0
```

TaskCompleted

작업이 완료로 표시될 때 실행됩니다. 이는 두 가지 상황에서 발생합니다. 모든 에이전트가 TaskUpdate 도구를 통해 작업을 명시적으로 완료로 표시할 때 또는 에이전트 팀 팀원이 진행 중인 작업으로 자신의 턴을 마칠 때입니다. 이를 사용하여 테스트 통과 또는 lint 검사와 같은 완료 기준을 적용하기 전에 작업을 닫을 수 있습니다.

`TaskCompleted` hook이 코드 2로 종료되면 작업이 완료로 표시되지 않고 stderr 메시지가 모델에 피드백으로 피드백됩니다. 팀원을 다시 실행하는 대신 완전히 중지하려면 `{"continue": false, "stopReason": "..."}` 이 있는 JSON을 반환합니다. TaskCompleted hook은 matcher를 지원하지 않으며 모든 발생에서 발생합니다.

TaskCompleted 입력

공통 입력 필드 외에도 TaskCompleted hook은 `task_id`, `task_subject`, 선택적으로 `task_description`, `teammate_name`, `team_name`을 받습니다.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "TaskCompleted",
  "task_id": "task-001",
  "task_subject": "Implement user authentication",
  "task_description": "Add login and signup endpoints",
  "teammate_name": "implementer",
  "team_name": "my-project"
}
```

필드	설명
`task_id`	완료되는 작업의 식별자
`task_subject`	작업의 제목
`task_description`	작업의 자세한 설명. 없을 수 있음
`teammate_name`	작업을 완료하는 팀원의 이름. 없을 수 있음
`team_name`	팀의 이름. 없을 수 있음

TaskCompleted 결정 제어

TaskCompleted hook은 작업 완료를 제어하는 두 가지 방법을 지원합니다.

- **종료 코드 2**: 작업이 완료로 표시되지 않고 stderr 메시지가 모델에 피드백으로 피드백됩니다.
- **JSON** `{"continue": false, "stopReason": "..."}`: 팀원을 완전히 중지하여 `Stop` hook 동작과 일치합니다. `stopReason`은 사용자에게 표시됩니다.

이 예제는 테스트를 실행하고 실패하면 작업 완료를 차단합니다.

```
#!/bin/bash
INPUT=$(cat)
TASK_SUBJECT=$(echo "$INPUT" | jq -r '.task_subject')

## 테스트 스위트를 실행합니다
if ! npm test 2>&1; then
  echo "Tests not passing. Fix failing tests before completing: $TASK_SUBJECT" >&2
  exit 2
fi

exit 0
```

ConfigChange

세션 중에 구성 파일이 변경될 때 실행됩니다. 이를 사용하여 설정 변경을 감사하고, 보안 정책을 적용하거나, 구성 파일에 대한 무단 수정을 차단합니다.

ConfigChange hook은 설정 파일, 관리형 정책 설정, skill 파일의 변경에 대해 발생합니다. 입력의 `source` 필드는 어떤 유형의 구성이 변경되었는지 알려주고, 선택적 `file_path` 필드는 변경된 파일의 경로를 제공합니다.

matcher는 구성 소스에서 필터링합니다.

Matcher	언제 발생하는지
`user_settings`	`~/.claude/settings.json` 변경
`project_settings`	`.claude/settings.json` 변경
`local_settings`	`.claude/settings.local.json` 변경
`policy_settings`	관리형 정책 설정 변경
`skills`	`.claude/skills/` 의 skill 파일 변경

이 예제는 보안 감사를 위해 모든 구성 변경을 기록합니다.

```
{
  "hooks": {
    "ConfigChange": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "\"$CLAUDE_PROJECT_DIR\"/.claude/hooks/audit-config-
change.sh"
          }
        ]
      }
    ]
  }
}
```

ConfigChange 입력

공통 입력 필드 외에도 ConfigChange hook은 `source` 및 선택적으로 `file_path` 를 받습니다. `source` 필드는 어떤 구성 유형이 변경되었는지 나타내고 `file_path` 는 수정된 특정 파일의 경로를 제공합니다.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "ConfigChange",
  "source": "project_settings",
  "file_path": "/Users/.../my-project/.claude/settings.json"
}
```

ConfigChange 결정 제어

ConfigChange hook은 구성 변경이 적용되는 것을 차단할 수 있습니다. 종료 코드 2 또는 JSON `decision`을 사용하여 변경을 방지합니다. 차단되면 새 설정이 실행 중인 세션에 적용되지 않습니다.

필드	설명
`decision`	`"block"`은 구성 변경이 적용되는 것을 방지합니다. 변경을 허용하려면 생략합니다
`reason`	`decision`이 `"block"`일 때 사용자에게 표시되는 설명

```
{
  "decision": "block",
  "reason": "Configuration changes to project settings require admin approval"
}
```

`policy_settings` 변경은 차단할 수 없습니다. Hook은 여전히 `policy_settings` 소스에 대해 발생하므로 감사 로깅에 사용할 수 있지만 모든 차단 결정은 무시됩니다. 이는 엔터프라이즈 관리 설정이 항상 적용되도록 합니다.

WorktreeCreate

`claude --worktree`를 실행하거나 subagent가 `isolation: "worktree"`를 사용할 때 Claude Code는 `git worktree`를 사용하여 격리된 작업 복사본을 생성합니다. WorktreeCreate hook을 구성하면 기본 git 동작을 대체하여 SVN, Perforce 또는 Mercurial과 같은 다른 버전 제어 시스템을 사용할 수 있습니다.

hook은 생성된 worktree 디렉토리의 절대 경로를 stdout에 인쇄해야 합니다. Claude Code는 이 경로를 격리된 세션의 작업 디렉토리로 사용합니다.

이 예제는 SVN 작업 복사본을 생성하고 Claude Code가 사용할 경로를 인쇄합니다. 리포지토리 URL을 자신의 것으로 바꾸세요.

```
{
  "hooks": {
    "WorktreeCreate": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "bash -c 'NAME=$(jq -r .name); DIR=\"$HOME/.claude/
worktrees/$NAME\"; svn checkout https://svn.example.com/repo/trunk
\"$DIR\" >&2 && echo \"$DIR\"'"
          }
        ]
      }
    ]
  }
}
```

hook은 stdin의 JSON 입력에서 worktree `name`을 읽고 새 디렉토리로 신선한 복사본을 체크아웃한 후 디렉토리 경로를 인쇄합니다. 마지막 줄의 `echo`는 Claude Code가 worktree 경로로 읽는 것입니다. 경로를 방해하지 않도록 다른 모든 출력을 stderr로 리디렉션합니다.

WorktreeCreate 입력

공통 입력 필드 외에도 WorktreeCreate hook은 `name` 필드를 받습니다. 이는 새 worktree의 slug 식별자이며, 사용자가 지정하거나 자동 생성됩니다 (예: `bold-oak-a3f2`).

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "hook_event_name": "WorktreeCreate",
  "name": "feature-auth"
}
```

WorktreeCreate 출력

hook은 생성된 worktree 디렉토리의 절대 경로를 stdout에 인쇄해야 합니다. hook이 실패하거나 출력을 생성하지 않으면 worktree 생성이 오류로 실패합니다.

WorktreeCreate hook은 표준 허용/차단 결정 모델을 사용하지 않습니다. 대신 hook의 성공 또는 실패가 결과를 결정합니다. `type: "command"` hook만 지원됩니다.

WorktreeRemove

WorktreeCreate의 정리 대응입니다. 이 hook은 worktree가 제거될 때 발생합니다. `--worktree` 세션을 종료하고 제거하도록 선택할 때 또는 `isolation: "worktree"`를 가진 subagent가 완료될 때입니다. git 기반 worktree의 경우 Claude는 `git worktree remove`로 정리를 자동으로 처리합니다. git이 아닌 버전 제어 시스템에 대해 WorktreeCreate hook을 구성한 경우 정리를 처리하기 위해 WorktreeRemove hook과 쌍을 이룹니다. 없으면 worktree 디렉토리가 디스크에 남아 있습니다.

Claude Code는 WorktreeCreate가 stdout에 인쇄한 경로를 hook 입력에서 `worktree_path`로 전달합니다. 이 예제는 해당 경로를 읽고 디렉토리를 제거합니다.

```
{
  "hooks": {
    "WorktreeRemove": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "bash -c 'jq -r .worktree_path | xargs rm -rf'"
          }
        ]
      }
    ]
  }
}
```

WorktreeRemove 입력

공통 입력 필드 외에도 WorktreeRemove hook은 제거되는 worktree의 절대 경로인 `worktree_path` 필드를 받습니다.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "hook_event_name": "WorktreeRemove",
  "worktree_path": "/Users/.../my-project/.claude/worktrees/feature-auth"
}
```

WorktreeRemove hook은 결정 제어가 없습니다. worktree 제거를 차단할 수 없지만 버전 제어 상태 제거 또는 변경 아카이빙과 같은 정리 작업을 수행할 수 있습니다. hook 실패는 디버그 모드에서만 기록됩니다. `type: "command"` hook만 지원됩니다.

PreCompact

Claude Code가 압축 작업을 실행하려고 하기 전에 실행됩니다.

matcher 값은 압축이 수동으로 또는 자동으로 트리거되었는지 나타냅니다.

Matcher	언제 발생하는지
`manual`	`/compact`
`auto`	컨텍스트 윈도우가 가득 찼을 때 자동 압축

PreCompact 입력

공통 입력 필드 외에도 PreCompact hook은 `trigger` 및 `custom_instructions`를 받습니다. `manual`의 경우 `custom_instructions`는 사용자가 `/compact`에 전달하는 것을 포함합니다. `auto`의 경우 `custom_instructions`는 비어 있습니다.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "PreCompact",
  "trigger": "manual",
  "custom_instructions": ""
}
```

SessionEnd

Claude Code 세션이 종료될 때 실행됩니다. 정리 작업, 세션 통계 로깅 또는 세션 상태 저장에 유용합니다. 종료 이유로 필터링하는 matcher를 지원합니다.

hook 입력의 `reason` 필드는 세션이 종료된 이유를 나타냅니다.

이유	설명
`clear`	`/clear` 명령으로 세션 지워짐
`logout`	사용자 로그아웃
`prompt_input_exit`	프롬프트 입력이 표시되는 동안 사용자 종료
`bypass_permissions_disabled`	권한 우회 모드 비활성화
`other`	기타 종료 이유

SessionEnd 입력

공통 입력 필드 외에도 SessionEnd hook은 세션이 종료된 이유를 나타내는 `reason` 필드를 받습니다. 모든 값은 위의 이유 표를 참조하세요.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "SessionEnd",
  "reason": "other"
}
```

SessionEnd hook은 결정 제어가 없습니다. 세션 종료를 차단할 수 없지만 정리 작업을 수행할 수 있습니다.

프롬프트 기반 hook

명령 및 HTTP hook 외에도 Claude Code는 LLM을 사용하여 작업을 허용할지 차단할지 평가하는 프롬프트 기반 hook (`type: "prompt"`)과 Read, Grep, Glob과 같은 도구를 사용할 수 있는 subagent를 생성하는 에이전트 hook (`type: "agent"`)을 지원합니다. 모든 이벤트가 모든 hook 유형을 지원하지는 않습니다.

네 가지 hook 유형 (`command` , `http` , `prompt` , `agent`)을 모두 지원하는 이벤트:

- `PermissionRequest`
- `PostToolUse`
- `PostToolUseFailure`
- `PreToolUse`
- `Stop`
- `SubagentStop`
- `TaskCompleted`
- `UserPromptSubmit`

`type: "command"` hook만 지원하는 이벤트:

- `ConfigChange`
- `InstructionsLoaded`
- `Notification`
- `PreCompact`
- `SessionEnd`

- `SessionStart`
- `SubagentStart`
- `TeammateIdle`
- `WorktreeCreate`
- `WorktreeRemove`

프롬프트 기반 hook이 어떻게 작동하는지

프롬프트 기반 hook은 Bash 명령을 실행하는 대신:

1. hook 입력과 프롬프트를 Claude 모델 (기본값 Haiku)로 전송합니다.
2. LLM은 결정을 포함하는 구조화된 JSON으로 응답합니다.
3. Claude Code는 결정을 자동으로 처리합니다.

프롬프트 hook 구성

`type`을 `"prompt"`로 설정하고 `command` 대신 `prompt` 문자열을 제공합니다. `$ARGUMENTS` 자리 표시자를 사용하여 hook의 JSON 입력 데이터를 프롬프트 텍스트에 주입합니다. Claude Code는 결합된 프롬프트와 입력을 빠른 Claude 모델로 전송하며, 이는 JSON 결정을 반환합니다.

이 `Stop` hook은 Claude가 완료되기 전에 모든 작업이 완료되었는지 평가하도록 LLM에 요청합니다.

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "prompt",
            "prompt": "Evaluate if Claude should stop: $ARGUMENTS. Check if all
tasks are complete."
          }
        ]
      }
    ]
  }
}
```

필드	필수	설명
`type`	예	`"prompt"` 여야 합니다
`prompt`	예	LLM으로 전송할 프롬프트 텍스트. hook 입력 JSON에 대한 자리 표시자로 `$ARGUMENTS` 사용. `$ARGUMENTS` 가 없으면 입력 JSON이 프롬프트에 추가됩니다
`model`	아니오	평가에 사용할 모델. 기본값은 빠른 모델
`timeout`	아니오	초 단위 시간 초과. 기본값: 30

응답 스키마

LLM은 다음을 포함하는 JSON으로 응답해야 합니다.

```
{
  "ok": true | false,
  "reason": "Explanation for the decision"
}
```

필드	설명
`ok`	`true` 는 작업을 허용하고 `false` 는 방지합니다
`reason`	`ok` 가 `false` 일 때 필수입니다. Claude에 표시되는 설명

예제: 다중 기준 Stop hook

이 `Stop` hook은 Claude가 중지하도록 허용하기 전에 세 가지 조건을 확인하는 자세한 프롬프트를 사용합니다. `"ok"` 가 `false` 이면 Claude는 제공된 이유를 다음 명령으로 받으며 계속 작동합니다. `SubagentStop` hook은 동일한 형식을 사용하여 subagent가 중지해야 하는지 평가합니다.

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "prompt",
            "prompt": "You are evaluating whether Claude should stop working.
Context: $ARGUMENTS\n\nAnalyze the conversation and determine if:\n1. All user-
requested tasks are complete\n2. Any errors need to be addressed\n3. Follow-up
work is needed\n\nRespond with JSON: {\"ok\": true} to allow stopping, or
{\"ok\": false, \"reason\": \"your explanation\"} to continue working.",
            "timeout": 30
          }
        ]
      }
    ]
  }
}
```

에이전트 기반 hook

에이전트 기반 hook (`type: "agent"`)은 프롬프트 기반 hook과 유사하지만 다중 턴 도구 액세스가 있습니다. 단일 LLM 호출 대신 에이전트 hook은 프롬프트와 hook의 JSON 입력으로 subagent를 생성합니다. 에이전트 hook은 프롬프트 기반 hook과 동일한 이벤트를 지원합니다.

에이전트 hook이 어떻게 작동하는지

에이전트 hook이 발생할 때:

1. Claude Code는 프롬프트와 hook의 JSON 입력으로 subagent를 생성합니다.
2. subagent는 Read, Grep, Glob과 같은 도구를 사용하여 조사할 수 있습니다.
3. 최대 50 턴 후 subagent는 구조화된 `{ "ok": true/false }` 결정을 반환합니다.
4. Claude Code는 프롬프트 hook과 동일한 방식으로 결정을 처리합니다.

에이전트 hook은 검증이 실제 파일을 검사하거나 테스트 출력을 확인해야 할 때 유용합니다. hook 입력 데이터만으로는 평가할 수 없습니다.

에이전트 hook 구성

`type`을 `"agent"`로 설정하고 `prompt` 문자열을 제공합니다. 구성 필드는 프롬프트 hook과 동일하지만 시간 초과가 더 깁니다.

필드	필수	설명
`type`	예	`"agent"`여야 합니다
`prompt`	예	확인할 항목을 설명하는 프롬프트. hook 입력 JSON에 대한 자리 표시자로 `$ARGUMENTS` 사용
`model`	아니오	사용할 모델. 기본값은 빠른 모델
`timeout`	아니오	초 단위 시간 초과. 기본값: 60

응답 스키마는 프롬프트 hook과 동일합니다. `{ "ok": true }`를 허용하거나 `{ "ok": false, "reason": "..." }`를 차단합니다.

이 `Stop` hook은 Claude가 완료되기 전에 모든 단위 테스트가 통과하는지 확인합니다.

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "agent",
            "prompt": "Verify that all unit tests pass. Run the test suite and
check the results. $ARGUMENTS",
            "timeout": 120
          }
        ]
      }
    ]
  }
}
```

백그라운드에서 hook 실행

기본적으로 hook은 완료될 때까지 Claude의 실행을 차단합니다. 배포, 테스트 스위트 또는 외부 API 호출과 같은 장기 실행 작업의 경우 `"async": true`를 설정하여 Claude가 계속 작동하는 동안 백그라운드에서 hook을 실행합니다. 비동기 hook은 차단하거나 Claude의 동작을 제어할 수 없습니다. `decision`, `permissionDecision`, `continue`와 같은 응답 필드는 효과가 없습니다. 제어했을 작업이 이미 완료되었기 때문입니다.

비동기 hook 구성

hook의 구성에 `"async": true`를 추가하여 Claude를 차단하지 않고 백그라운드에서 실행합니다. 이 필드는 `type: "command"` hook에서만 사용 가능합니다.

이 hook은 모든 `Write` 도구 호출 후 테스트 스크립트를 실행합니다. Claude는 `run-tests.sh`가 최대 120초 동안 실행되는 동안 즉시 계속 작동합니다. 스크립트가 완료되면 해당 출력이 다음 대화 턴에 전달됩니다.

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write",
        "hooks": [
          {
            "type": "command",
            "command": "/path/to/run-tests.sh",
            "async": true,
            "timeout": 120
          }
        ]
      }
    ]
  }
}
```

`timeout` 필드는 백그라운드 프로세스의 최대 시간을 초 단위로 설정합니다. 지정하지 않으면 비동기 hook은 동기 hook과 동일한 10분 기본값을 사용합니다.

비동기 hook이 어떻게 실행되는지

비동기 hook이 발생하면 Claude Code는 hook 프로세스를 시작하고 완료될 때까지 기다리지 않고 즉시 계속합니다. hook은 동기 hook과 동일한 JSON 입력을 stdin을 통해 받습니다.

백그라운드 프로세스가 종료된 후 hook이 `systemMessage` 또는 `additionalContext` 필드가 있는 JSON 응답을 생성한 경우 해당 콘텐츠는 다음 대화 턴에서 Claude에 컨텍스트로 전달됩니다.

예제: 파일 변경 후 테스트 실행

이 hook은 Claude가 파일을 쓸 때마다 백그라운드에서 테스트 스위트를 시작한 후 테스트가 완료되면 결과를 Claude에 보고합니다. 이 스크립트를 프로젝트의 `.claude/hooks/run-tests-async.sh`에 저장하고 `chmod +x`로 실행 가능하게 만듭니다.

```
#!/bin/bash
## run-tests-async.sh

## stdin에서 hook 입력을 읽습니다
INPUT=$(cat)
FILE_PATH=$(echo "$INPUT" | jq -r '.tool_input.file_path // empty')

## 소스 파일에 대해서만 테스트를 실행합니다
if [[ "$FILE_PATH" != *.ts && "$FILE_PATH" != *.js ]]; then
  exit 0
fi

## 테스트를 실행하고 systemMessage를 통해 결과를 보고합니다
RESULT=$(npm test 2>&1)
EXIT_CODE=$?

if [ $EXIT_CODE -eq 0 ]; then
  echo "{\"systemMessage\": \"Tests passed after editing $FILE_PATH\"}"
else
  echo "{\"systemMessage\": \"Tests failed after editing $FILE_PATH: $RESULT\"}"
fi
```

그런 다음 프로젝트 루트의 `.claude/settings.json`에 이 구성을 추가합니다. `async: true` 플래그를 사용하면 Claude가 테스트 실행 중에 계속 작동할 수 있습니다.

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "\"$CLAUDE_PROJECT_DIR\"/.claude/hooks/run-tests-async.sh",
            "async": true,
            "timeout": 300
          }
        ]
      }
    ]
  }
}
```

제한 사항

비동기 hook은 동기 hook과 비교하여 여러 제약이 있습니다.

- `async`를 지원하는 것은 `type: "command"` hook뿐입니다. 프롬프트 기반 hook은 비동기적으로 실행될 수 없습니다.
- 비동기 hook은 도구 호출을 차단하거나 결정을 반환할 수 없습니다. hook이 완료될 때까지 트리거 작업이 이미 진행되었습니다.
- Hook 출력은 다음 대화 턴에 전달됩니다. 세션이 유휴 상태이면 응답은 다음 사용자 상호 작용까지 기다립니다.
- 각 실행은 별도의 백그라운드 프로세스를 생성합니다. 동일한 비동기 hook의 여러 발생에 걸쳐 중복 제거가 없습니다.

보안 고려 사항

면책 조항

명령 hook은 시스템 사용자의 전체 권한으로 실행됩니다.

> **Warning:**

> 명령 hook은 사용자 계정이 액세스할 수 있는 모든 파일을 수정, 삭제 또는 액세스할 수 있는 전체 사용자 권한으로 셸 명령을 실행합니다. 구성에 추가하기 전에 모든 hook 명령을 검토하고 테스트합니다.

보안 모범 사례

hook을 작성할 때 이러한 관행을 염두에 두세요.

- **입력 검증 및 살균**: 입력 데이터를 맹목적으로 신뢰하지 마세요.
- **항상 셸 변수를 따옴표로 감싸세요**: `$VAR` 대신 `"$VAR"` 사용
- **경로 순회 차단**: 파일 경로에서 `..` 확인
- **절대 경로 사용**: `"$CLAUDE_PROJECT_DIR"` 을 사용하여 프로젝트 루트에 대한 전체 경로를 지정합니다.
- **민감한 파일 건너뛰기**: `.env` , `.git/` , 키 등을 피합니다.

Hook 디버그

`claude --debug` 를 실행하여 hook 실행 세부 정보를 확인합니다. 일치한 hook, 종료 코드, 출력을 포함합니다. `Ctrl+O` 로 자세한 모드를 전환하여 대화에서 hook 진행 상황을 확인합니다.

```
[DEBUG] Executing hooks for PostToolUse:Write
[DEBUG] Getting matching hook commands for PostToolUse with query: Write
[DEBUG] Found 1 hook matchers in settings
[DEBUG] Matched 1 hooks for query "Write"
[DEBUG] Found 1 hook commands to execute
[DEBUG] Executing hook command: <Your command> with timeout 600000ms
[DEBUG] Hook command completed with status 0: <Your stdout>
```

hook이 발생하지 않음, 무한 Stop hook 루프 또는 구성 오류와 같은 일반적인 문제 해결은 가이드의 제한 사항 및 문제 해결을 참조하세요.

hooks를 사용하여 워크플로우 자동화

> Claude Code가 파일을 편집하거나 작업을 완료하거나 입력이 필요할 때 자동으로 셸 명령을 실행합니다. 코드 형식 지정, 알림 전송, 명령 검증 및 프로젝트 규칙 적용.

Hooks는 Claude Code의 라이프사이클의 특정 지점에서 실행되는 사용자 정의 셸 명령입니다. 이들은 Claude Code의 동작에 대한 결정론적 제어를 제공하여 LLM이 실행하도록 선택하는 것에 의존하기보다는 특정 작업이 항상 발생하도록 보장합니다. Hooks를 사용하여 프로젝트 규칙을 적용하고, 반복적인 작업을 자동화하며, Claude Code를 기존 도구와 통합합니다.

판단이 필요한 결정의 경우 결정론적 규칙이 아닌 경우, 프롬프트 기반 hooks 또는 에이전트 기반 hooks를 사용할 수도 있습니다. 이들은 Claude 모델을 사용하여 조건을 평가합니다.

Claude Code를 확장하는 다른 방법은 skills를 참조하여 Claude에 추가 지침과 실행 가능한 명령을 제공하고, subagents를 사용하여 격리된 컨텍스트에서 작업을 실행하며, plugins를 사용하여 프로젝트 전체에서 공유할 확장을 패키징합니다.

> **Tip:**
>
> 이 가이드는 일반적인 사용 사례와 시작 방법을 다룹니다. 전체 이벤트 스키마, JSON 입출력 형식 및 비동기 hooks 및 MCP tool hooks와 같은 고급 기능은 Hooks 참조를 참조하세요.

첫 번째 hook 설정

hook을 만드는 가장 빠른 방법은 Claude Code의 `/hooks` 대화형 메뉴를 통하는 것입니다. 이 연습은 데스크톱 알림 hook을 만들므로 Claude가 터미널을 보는 대신 입력을 기다릴 때마다 알림을 받습니다.

Step 1: hooks 메뉴 열기

Claude Code CLI에서 `/hooks` 를 입력합니다. 사용 가능한 모든 hook 이벤트 목록과 모든 hooks를 비활성화하는 옵션이 표시됩니다. 각 이벤트는 Claude의 라이프사이클에서 사용자 정의 코드를 실행할 수 있는 지점에 해당합니다. Claude가 주의가 필요할 때 발생하는 hook을 만들려면 `Notification` 을 선택합니다.

Step 2: matcher 구성

메뉴는 hook이 발생할 때를 필터링하는 matchers 목록을 표시합니다. matcher를 `*` 로 설정하여 모든 알림 유형에서 발생하도록 합니다. 나중에 matcher를 `permission_prompt` 또는 `idle_prompt` 와 같은 특정 값으로 변경하여 좁힐 수 있습니다.

Step 3: 명령 추가

`+ Add new hook…` 을 선택합니다. 메뉴는 이벤트가 발생할 때 실행할 셸 명령을 입력하라는 메시지를 표시합니다. Hooks는 제공하는 모든 셸 명령을 실행하므로 플랫폼의 기본 제공 알림 도구를 사용할 수 있습니다. OS에 대한 명령을 복사합니다:

macOS

AppleScript를 통해 기본 macOS 알림을 트리거하기 위해 `osascript` 를 사용합니다:

```
osascript -e 'display notification "Claude Code needs your attention" with title
"Claude Code"'
```

Linux

대부분의 Linux 데스크톱에 알림 데몬과 함께 사전 설치된 `notify-send` 를 사용합니다:

```
notify-send 'Claude Code' 'Claude Code needs your attention'
```

Windows (PowerShell)

PowerShell을 사용하여 .NET의 Windows Forms를 통해 기본 메시지 상자를 표시합니다:

```
powershell.exe -Command "[System.Reflection.Assembly]::LoadWithPartialName('System
.Windows.Forms'); [System.Windows.Forms.MessageBox]::Show('Claude Code needs your
attention', 'Claude Code')"
```

Step 4: 저장 위치 선택

메뉴는 hook 구성을 저장할 위치를 묻습니다. `User settings` 를 선택하여 `~/.claude/settings.json` 에 저장하면 모든 프로젝트에 hook이 적용됩니다. 현재 프로젝트로 범위를 지정하려면 `Project settings` 를 선택할 수도 있습니다. 사용 가능한 모든 범위는 Configure hook location을 참조하세요.

Step 5: hook 테스트

`Esc` 를 눌러 CLI로 돌아갑니다. Claude에게 권한이 필요한 작업을 수행하도록 요청한 다음 터미널에서 전환합니다. 데스크톱 알림을 받아야 합니다.

자동화할 수 있는 것

Hooks를 사용하면 Claude Code의 라이프사이클의 주요 지점에서 코드를 실행할 수 있습니다: 편집 후 파일 형식 지정, 실행 전 명령 차단, Claude가 입력이 필요할 때 알림 전송, 세션 시작 시 컨텍스트 주입 등. 전체 hook 이벤트 목록은 Hooks 참조를 참조하세요.

각 예제에는 설정 파일에 추가하는 즉시 사용 가능한 구성 블록이 포함되어 있습니다. 가장 일반적인 패턴:

- Claude가 입력이 필요할 때 알림 받기
- 편집 후 코드 자동 형식 지정
- 보호된 파일에 대한 편집 차단
- 압축 후 컨텍스트 다시 주입
- 구성 변경 감사

Claude가 입력이 필요할 때 알림 받기

Claude가 작업을 완료하고 입력이 필요할 때마다 데스크톱 알림을 받으므로 터미널을 확인하지 않고 다른 작업으로 전환할 수 있습니다.

이 hook은 Claude가 입력 또는 권한을 기다릴 때 발생하는 `Notification` 이벤트를 사용합니다. 각 탭은 플랫폼의 기본 알림 명령을 사용합니다. `~/.claude/settings.json`에 추가하거나 위의 대화형 연습을 사용하여 `/hooks`로 구성합니다:

macOS

```
{
  "hooks": {
    "Notification": [
      {
        "matcher": "",
        "hooks": [
          {
            "type": "command",
            "command": "osascript -e 'display notification \"Claude Code needs
your attention\" with title \"Claude Code\"'"
          }
        ]
      }
    ]
  }
}
```

Linux

```
{
  "hooks": {
    "Notification": [
      {
        "matcher": "",
        "hooks": [
          {
            "type": "command",
            "command": "notify-send 'Claude Code' 'Claude Code needs your
attention'"
          }
        ]
      }
    ]
  }
}
```

Windows (PowerShell)

```
{
  "hooks": {
    "Notification": [
      {
        "matcher": "",
        "hooks": [
          {
            "type": "command",
            "command": "powershell.exe -Command \"[System.Reflection.Assembly]::LoadWithPartialName('System.Windows.Forms');
[System.Windows.Forms.MessageBox]::Show('Claude Code needs your attention', 'Claude Code')\""
          }
        ]
      }
    ]
  }
}
```

편집 후 코드 자동 형식 지정

Claude가 편집하는 모든 파일에서 Prettier를 자동으로 실행하여 수동 개입 없이 형식이 일관되게 유지되도록 합니다.

이 hook은 `PostToolUse` 이벤트를 `Edit|Write` matcher와 함께 사용하므로 파일 편집 도구 후에만 실행됩니다. 명령은 `jq` 를 사용하여 편집된 파일 경로를 추출하고 Prettier에 전달합니다. 프로젝트 루트의 `.claude/settings.json` 에 추가합니다:

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "jq -r '.tool_input.file_path' | xargs npx prettier --
write"
          }
        ]
      }
    ]
  }
}
```

> **Note:**
>
> 이 페이지의 Bash 예제는 JSON 구문 분석을 위해 `jq` 를 사용합니다. `brew install jq` (macOS), `apt-get install jq` (Debian/Ubuntu)로 설치하거나 `jq` 다운로드를 참조하세요.

보호된 파일에 대한 편집 차단

Claude가 `.env` , `package-lock.json` 또는 `.git/` 의 모든 항목과 같은 민감한 파일을 수정하지 못하도록 방지합니다. Claude는 편집이 차단된 이유를 설명하는 피드백을 받으므로 접근 방식을 조정할 수 있습니다.

이 예제는 hook이 호출하는 별도의 스크립트 파일을 사용합니다. 스크립트는 대상 파일 경로를 보호된 패턴 목록과 비교하고 종료 코드 2로 종료하여 편집을 차단합니다.

Step 1: hook 스크립트 만들기

이를 `.claude/hooks/protect-files.sh` 에 저장합니다:

```
#!/bin/bash
## protect-files.sh

INPUT=$(cat)
FILE_PATH=$(echo "$INPUT" | jq -r '.tool_input.file_path // empty')

PROTECTED_PATTERNS=(".env" "package-lock.json" ".git/")

for pattern in "${PROTECTED_PATTERNS[@]}"; do
  if [[ "$FILE_PATH" == *"$pattern"* ]]; then
    echo "Blocked: $FILE_PATH matches protected pattern '$pattern'" >&2
    exit 2
  fi
done

exit 0
```

Step 2: 스크립트를 실행 가능하게 만들기 (macOS/Linux)

Claude Code가 hook 스크립트를 실행하려면 실행 가능해야 합니다:

```
chmod +x .claude/hooks/protect-files.sh
```

Step 3: hook 등록

모든 `Edit` 또는 `Write` tool 호출 전에 스크립트를 실행하는 `PreToolUse` hook을 `.claude/settings.json`에 추가합니다:

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "\"$CLAUDE_PROJECT_DIR\"/.claude/hooks/protect-files.sh"
          }
        ]
      }
    ]
  }
}
```

압축 후 컨텍스트 다시 주입

Claude의 컨텍스트 윈도우가 가득 차면 압축은 대화를 요약하여 공간을 확보합니다. 이는 중요한 세부 정보를 잃을 수 있습니다. `compact` matcher와 함께 `SessionStart` hook을 사용하여 모든 압축 후 중요한 컨텍스트를 다시 주입합니다.

명령이 stdout에 쓰는 모든 텍스트는 Claude의 컨텍스트에 추가됩니다. 이 예제는 Claude에게 프로젝트 규칙과 최근 작업을 상기시킵니다. 프로젝트 루트의 `.claude/settings.json`에 추가합니다:

```
{
  "hooks": {
    "SessionStart": [
      {
        "matcher": "compact",
        "hooks": [
          {
            "type": "command",
            "command": "echo 'Reminder: use Bun, not npm. Run bun test before
committing. Current sprint: auth refactor.'"
          }
        ]
      }
    ]
  }
}
```

`echo`를 `git log --oneline -5`와 같이 동적 출력을 생성하는 모든 명령으로 바꿀 수 있습니다. 모든 세션 시작 시 컨텍스트를 주입하려면 CLAUDE.md 사용을 고려하세요. 환경 변수는 참조의 `CLAUDE_ENV_FILE`을 참조하세요.

구성 변경 감사

세션 중에 설정 또는 skills 파일이 변경될 때를 추적합니다. `ConfigChange` 이벤트는 외부 프로세스 또는 편집기가 구성 파일을 수정할 때 발생하므로 규정 준수를 위해 변경 사항을 기록하거나 무단 수정을 차단할 수 있습니다.

이 예제는 각 변경을 감사 로그에 추가합니다. `~/.claude/settings.json`에 추가합니다:

```
{
  "hooks": {
    "ConfigChange": [
      {
        "matcher": "",
        "hooks": [
          {
            "type": "command",
            "command": "jq -c '{timestamp: now | todate, source: .source,
file: .file_path}' >> ~/claude-config-audit.log"
          }
        ]
      }
    ]
  }
}
```

Matcher는 구성 유형으로 필터링합니다: `user_settings`, `project_settings`, `local_settings`, `policy_settings` 또는 `skills`. 변경이 적용되지 않도록 차단하려면 종료 코드 2로 종료하거나 `{"decision": "block"}`을 반환합니다. 전체 입력 스키마는 ConfigChange 참조를 참조하세요.

Hooks 작동 방식

Hook 이벤트는 Claude Code의 라이프사이클의 특정 지점에서 발생합니다. 이벤트가 발생하면 일치하는 모든 hooks가 병렬로 실행되고 동일한 hook 명령은 자동으로 중복 제거됩니다. 아래 표는 각 이벤트와 발생 시기를 보여줍니다:

Event	When it fires
`SessionStart`	When a session begins or resumes
`UserPromptSubmit`	When you submit a prompt, before Claude processes it
`PreToolUse`	Before a tool call executes. Can block it
`PermissionRequest`	When a permission dialog appears
`PostToolUse`	After a tool call succeeds
`PostToolUseFailure`	After a tool call fails

Event	When it fires
`Notification`	When Claude Code sends a notification
`SubagentStart`	When a subagent is spawned
`SubagentStop`	When a subagent finishes
`Stop`	When Claude finishes responding
`TeammateIdle`	When an agent team teammate is about to go idle
`TaskCompleted`	When a task is being marked as completed
`InstructionsLoaded`	When a CLAUDE.md or `.claude/rules/*.md` file is loaded into context. Fires at session start and when files are lazily loaded during a session
`ConfigChange`	When a configuration file changes during a session
`WorktreeCreate`	When a worktree is being created via `--worktree` or `isolation: "worktree"`. Replaces default git behavior
`WorktreeRemove`	When a worktree is being removed, either at session exit or when a subagent finishes
`PreCompact`	Before context compaction
`PostCompact`	After context compaction completes
`Elicitation`	When an MCP server requests user input during a tool call
`ElicitationResult`	After a user responds to an MCP elicitation, before the response is sent back to the server
`SessionEnd`	When a session terminates

각 hook에는 실행 방식을 결정하는 `type`이 있습니다. 대부분의 hooks는 `"type": "command"`를 사용하여 셸 명령을 실행합니다. 세 가지 다른 유형을 사용할 수 있습니다:

- `"type": "http"`: 이벤트 데이터를 URL에 POST합니다. HTTP hooks를 참조하세요.
- `"type": "prompt"`: 단일 턴 LLM 평가. 프롬프트 기반 hooks를 참조하세요.
- `"type": "agent"`: 도구 접근 권한이 있는 다중 턴 검증. 에이전트 기반 hooks를 참조하세요.

입력 읽기 및 출력 반환

Hooks는 stdin, stdout, stderr 및 종료 코드를 통해 Claude Code와 통신합니다. 이벤트가 발생하면 Claude Code는 이벤트별 데이터를 JSON으로 스크립트의 stdin에 전달합니다. 스크립트는 해당 데이터를 읽고 작업을 수행한 다음 종료 코드를 통해 Claude Code에 다음 작업을 알립니다.

Hook 입력

모든 이벤트에는 `session_id` 및 `cwd`와 같은 공통 필드가 포함되지만 각 이벤트 유형은 다른 데이터를 추가합니다. 예를 들어 Claude가 Bash 명령을 실행할 때 `PreToolUse` hook은 stdin에서 다음과 같은 것을 받습니다:

```
{
  "session_id": "abc123",           // 이 세션의 고유 ID
  "cwd": "/Users/sarah/myproject", // 이벤트가 발생했을 때의 작업 디렉토리
  "hook_event_name": "PreToolUse", // 이 hook을 트리거한 이벤트
  "tool_name": "Bash",              // Claude가 사용하려는 도구
  "tool_input": {                   // Claude가 도구에 전달한 인수
    "command": "npm test"           // Bash의 경우 이것이 셸 명령입니다
  }
}
```

스크립트는 해당 JSON을 구문 분석하고 해당 필드에 대해 작동할 수 있습니다.
`UserPromptSubmit` hooks는 `prompt` 텍스트를 대신 받고, `SessionStart` hooks는 `source` (startup, resume, clear, compact)를 받으며, 등등입니다. 공유 필드는 참조의 공통 입력 필드를 참조하고 각 이벤트별 섹션에서 이벤트별 스키마를 참조하세요.

Hook 출력

스크립트는 stdout 또는 stderr에 쓰고 특정 코드로 종료하여 Claude Code에 다음 작업을 알립니다. 예를 들어 명령을 차단하려는 `PreToolUse` hook:

```
#!/bin/bash
INPUT=$(cat)
COMMAND=$(echo "$INPUT" | jq -r '.tool_input.command')

if echo "$COMMAND" | grep -q "drop table"; then
  echo "Blocked: dropping tables is not allowed" >&2  # stderr는 Claude의 피드백이 됩니다
  exit 2 # exit 2 = 작업 차단
fi

exit 0  # exit 0 = 진행 허용
```

종료 코드는 다음에 일어날 일을 결정합니다:

- **Exit 0**: 작업이 진행됩니다. `UserPromptSubmit` 및 `SessionStart` hooks의 경우 stdout에 쓰는 모든 것이 Claude의 컨텍스트에 추가됩니다.
- **Exit 2**: 작업이 차단됩니다. stderr에 이유를 쓰면 Claude가 피드백으로 받아 조정할 수 있습니다.
- **다른 종료 코드**: 작업이 진행됩니다. Stderr는 기록되지만 Claude에 표시되지 않습니다. `Ctrl+O` 로 자세한 모드를 전환하여 트랜스크립트에서 이 메시지를 확인합니다.

구조화된 JSON 출력

종료 코드는 두 가지 옵션을 제공합니다: 허용 또는 차단. 더 많은 제어를 위해 exit 0을 하고 stdout에 JSON 객체를 인쇄합니다.

> **Note:**
>
> Exit 2를 사용하여 stderr 메시지로 차단하거나 exit 0을 사용하여 구조화된 제어를 위해 JSON을 사용합니다. 혼합하지 마세요: Claude Code는 exit 2일 때 JSON을 무시합니다.

예를 들어 `PreToolUse` hook은 도구 호출을 거부하고 이유를 알리거나 사용자 승인을 위해 에스컬레이션할 수 있습니다:

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "deny",
    "permissionDecisionReason": "Use rg instead of grep for better performance"
  }
}
```

Claude Code는 `permissionDecision`을 읽고 도구 호출을 취소한 다음 `permissionDecisionReason`을 Claude에게 피드백으로 전달합니다. 이 세 가지 옵션은 `PreToolUse`에만 해당합니다:

- `"allow"`: 권한 프롬프트를 표시하지 않고 진행
- `"deny"`: 도구 호출을 취소하고 이유를 Claude에 전송
- `"ask"`: 일반적으로 사용자에게 권한 프롬프트 표시

다른 이벤트는 다른 결정 패턴을 사용합니다. 예를 들어 `PostToolUse` 및 `Stop` hooks는 최상위 `decision: "block"` 필드를 사용하고 `PermissionRequest`는 `hookSpecificOutput.decision.behavior`를 사용합니다. 이벤트별 전체 분석은 참조의 요약 표를 참조하세요.

`UserPromptSubmit` hooks의 경우 `additionalContext`를 대신 사용하여 Claude의 컨텍스트에 텍스트를 주입합니다. 프롬프트 기반 hooks (`type: "prompt"`)는 출력을 다르게 처리합니다: 프롬프트 기반 hooks를 참조하세요.

Matchers로 hooks 필터링

Matcher가 없으면 hook은 이벤트의 모든 발생에서 발생합니다. Matchers를 사용하면 범위를 좁힐 수 있습니다. 예를 들어 모든 도구 호출 후가 아닌 파일 편집 후에만 포매터를 실행하려면 `PostToolUse` hook에 matcher를 추가합니다:

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          { "type": "command", "command": "prettier --write ..." }
        ]
      }
    ]
  }
}
```

`"Edit|Write"` matcher는 도구 이름과 일치하는 정규식 패턴입니다. Hook은 Claude가 `Bash`, `Read` 또는 다른 도구를 사용할 때가 아닌 `Edit` 또는 `Write` 도구를 사용할 때만 발생합니다.

각 이벤트 유형은 특정 필드에서 일치합니다. Matchers는 정확한 문자열과 정규식 패턴을 지원합니다:

이벤트	Matcher가 필터링하는 것	예제 matcher 값
`PreToolUse`, `PostToolUse`, `PostToolUseFailure`, `PermissionRequest`	도구 이름	`Bash`, `Edit|Write`, `mcp__.*`
`SessionStart`	세션이 시작된 방식	`startup`, `resume`, `clear`, `compact`
`SessionEnd`	세션이 종료된 이유	`clear`, `logout`, `prompt_input_exit`, `bypass_permissions_disabled`, `other`
`Notification`	알림 유형	`permission_prompt`, `idle_prompt`, `auth_success`, `elicitation_dialog`

이벤트	Matcher가 필터링하는 것	예제 matcher 값
`SubagentStart`	에이전트 유형	`Bash`, `Explore`, `Plan` 또는 사용자 정의 에이전트 이름
`PreCompact`	압축을 트리거한 것	`manual`, `auto`
`SubagentStop`	에이전트 유형	`SubagentStart`와 동일한 값
`ConfigChange`	구성 소스	`user_settings`, `project_settings`, `local_settings`, `policy_settings`, `skills`
`UserPromptSubmit`, `Stop`, `TeammateIdle`, `TaskCompleted`, `WorktreeCreate`, `WorktreeRemove`	matcher 지원 없음	모든 발생에서 항상 발생

다양한 이벤트 유형에서 matchers를 보여주는 몇 가지 추가 예제:

모든 Bash 명령 기록

`Bash` 도구 호출만 일치시키고 각 명령을 파일에 기록합니다. `PostToolUse` 이벤트는 명령이 완료된 후 발생하므로 `tool_input.command`는 실행된 내용을 포함합니다. Hook은 stdin에서 이벤트 데이터를 JSON으로 받고 `jq -r '.tool_input.command'`는 명령 문자열만 추출하며 `>>`는 로그 파일에 추가합니다:

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": "jq -r '.tool_input.command' >> ~/.claude/command-log.txt"
          }
        ]
      }
    ]
  }
}
```

MCP 도구 일치

MCP 도구는 기본 제공 도구와 다른 명명 규칙을 사용합니다: `mcp__<server>__<tool>`. 여기서 `<server>`는 MCP 서버 이름이고 `<tool>`은 제공하는 도구입니다. 예를 들어 `mcp__github__search_repositories` 또는 `mcp__filesystem__read_file`. 정규식 matcher를 사용하여 특정 서버의 모든 도구를 대상으로 하거나 `mcp__.*__write.*`와 같은 패턴으로 서버 전체에서 일치시킵니다. 전체 예제 목록은 참조의 MCP 도구 일치를 참조하세요.

아래 명령은 hook의 JSON 입력에서 `jq`를 사용하여 도구 이름을 추출하고 자세한 모드(`Ctrl+O`)에 표시되는 stderr에 씁니다:

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "mcp__github__.*",
        "hooks": [
          {
            "type": "command",
            "command": "echo \"GitHub tool called: $(jq -r '.tool_name')\" >&2"
          }
        ]
      }
    ]
  }
}
```

세션 종료 시 정리

`SessionEnd` 이벤트는 세션이 종료된 이유에 대한 matchers를 지원합니다. 이 hook은 일반 종료가 아닌 `/clear` 를 실행할 때만 발생합니다:

```
{
  "hooks": {
    "SessionEnd": [
      {
        "matcher": "clear",
        "hooks": [
          {
            "type": "command",
            "command": "rm -f /tmp/claude-scratch-*.txt"
          }
        ]
      }
    ]
  }
}
```

전체 matcher 구문은 Hooks 참조를 참조하세요.

Hook 위치 구성

Hook을 추가하는 위치는 범위를 결정합니다:

위치	범위	공유 가능
`~/.claude/settings.json`	모든 프로젝트	아니요, 컴퓨터에 로컬
`.claude/settings.json`	단일 프로젝트	예, 리포지토리에 커밋 가능
`.claude/settings.local.json`	단일 프로젝트	아니요, gitignored
관리형 정책 설정	조직 전체	예, 관리자 제어
Plugin `hooks/hooks.json`	플러그인이 활성화되었을 때	예, 플러그인과 함께 번들됨
Skill 또는 agent frontmatter	Skill 또는 에이전트가 활성화되어 있는 동안	예, 컴포넌트 파일에 정의됨

또한 Claude Code의 `/hooks` 메뉴를 사용하여 대화형으로 hooks를 추가, 삭제 및 보기할 수 있습니다. 모든 hooks를 한 번에 비활성화하려면 `/hooks` 메뉴 하단의 토글을 사용하거나 설정 파일에서 `"disableAllHooks": true`를 설정합니다.

`/hooks` 메뉴를 통해 추가된 Hooks는 즉시 적용됩니다. Claude Code가 실행 중인 동안 설정 파일을 직접 편집하면 `/hooks` 메뉴에서 검토하거나 세션을 다시 시작할 때까지 변경 사항이 적용되지 않습니다.

프롬프트 기반 hooks

판단이 필요한 결정의 경우 결정론적 규칙이 아닌 경우 `type: "prompt"` hooks를 사용합니다. 셸 명령을 실행하는 대신 Claude Code는 프롬프트와 hook의 입력 데이터를 Claude 모델 (기본적으로 Haiku)에 전송하여 결정을 내립니다. 더 많은 기능이 필요한 경우 `model` 필드로 다른 모델을 지정할 수 있습니다.

모델의 유일한 작업은 yes/no 결정을 JSON으로 반환하는 것입니다:

- `"ok": true`: 작업이 진행됩니다
- `"ok": false`: 작업이 차단됩니다. 모델의 `"reason"`은 Claude가 조정할 수 있도록 피드백으로 전달됩니다.

이 예제는 `Stop` hook을 사용하여 모든 요청된 작업이 완료되었는지 모델에 묻습니다. 모델이 `"ok": false`를 반환하면 Claude는 계속 작업하고 `reason`을 다음 지침으로 사용합니다:

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "prompt",
            "prompt": "Check if all tasks are complete. If not, respond with
{\"ok\": false, \"reason\": \"what remains to be done\"}."
          }
        ]
      }
    ]
  }
}
```

전체 구성 옵션은 참조의 프롬프트 기반 hooks를 참조하세요.

에이전트 기반 hooks

검증에 파일 검사 또는 명령 실행이 필요한 경우 `type: "agent"` hooks를 사용합니다. 단일 LLM 호출을 수행하는 프롬프트 hooks와 달리 에이전트 hooks는 결정을 반환하기 전에 파일을 읽고 코드를 검색하며 다른 도구를 사용할 수 있는 subagent를 생성합니다.

에이전트 hooks는 프롬프트 hooks와 동일한 `"ok"` / `"reason"` 응답 형식을 사용하지만 기본 타임아웃이 60초이고 최대 50개의 도구 사용 턴입니다.

이 예제는 Claude가 중지되기 전에 테스트가 통과하는지 확인합니다:

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "agent",
            "prompt": "Verify that all unit tests pass. Run the test suite and
check the results. $ARGUMENTS",
            "timeout": 120
          }
        ]
      }
    ]
  }
}
```

Hook 입력 데이터만으로 결정을 내릴 수 있을 때 프롬프트 hooks를 사용합니다. 코드베이스의 실제 상태에 대해 무언가를 확인해야 할 때 에이전트 hooks를 사용합니다.

전체 구성 옵션은 참조의 에이전트 기반 hooks를 참조하세요.

HTTP hooks

`type: "http"` hooks를 사용하여 셸 명령을 실행하는 대신 이벤트 데이터를 HTTP 엔드포인트에 POST합니다. 엔드포인트는 명령 hook이 stdin에서 받을 것과 동일한 JSON을 받고 동일한 JSON 형식을 사용하여 HTTP 응답 본문을 통해 결과를 반환합니다.

HTTP hooks는 웹 서버, 클라우드 함수 또는 외부 서비스가 hook 로직을 처리하기를 원할 때 유용합니다: 예를 들어 팀 전체에서 도구 사용 이벤트를 기록하는 공유 감사 서비스입니다.

이 예제는 모든 도구 사용을 로컬 로깅 서비스에 게시합니다:

```
{
  "hooks": {
    "PostToolUse": [
      {
        "hooks": [
          {
            "type": "http",
            "url": "http://localhost:8080/hooks/tool-use",
            "headers": {
              "Authorization": "Bearer $MY_TOKEN"
            },
            "allowedEnvVars": ["MY_TOKEN"]
          }
        ]
      }
    ]
  }
}
```

엔드포인트는 명령 hooks와 동일한 출력 형식을 사용하여 JSON 응답 본문을 반환해야 합니다. 도구 호출을 차단하려면 적절한 `hookSpecificOutput` 필드와 함께 2xx 응답을 반환합니다. HTTP 상태 코드만으로는 작업을 차단할 수 없습니다.

헤더 값은 `$VAR_NAME` 또는 `${VAR_NAME}` 구문을 사용한 환경 변수 보간을 지원합니다. `allowedEnvVars` 배열에 나열된 변수만 해결됩니다. 다른 모든 `$VAR` 참조는 비어 있습니다.

> **Note:**
>
> HTTP hooks는 설정 JSON을 직접 편집하여 구성해야 합니다. `/hooks` 대화형 메뉴는 명령 hooks 추가만 지원합니다.

전체 구성 옵션 및 응답 처리는 참조의 HTTP hooks를 참조하세요.

제한 사항 및 문제 해결

제한 사항

- 명령 hooks는 stdout, stderr 및 종료 코드를 통해서만 통신합니다. 명령 또는 도구 호출을 직접 트리거할 수 없습니다. HTTP hooks는 응답 본문을 통해 통신합니다.

- Hook 타임아웃은 기본적으로 10분이며 `timeout` 필드 (초 단위)로 hook당 구성 가능합니다.
- `PostToolUse` hooks는 도구가 이미 실행되었으므로 작업을 취소할 수 없습니다.
- `PermissionRequest` hooks는 비대화형 모드 (`-p`)에서 발생하지 않습니다. 자동화된 권한 결정을 위해 `PreToolUse` hooks를 사용합니다.
- `Stop` hooks는 작업 완료 시에만이 아니라 Claude가 응답을 완료할 때마다 발생합니다. 사용자 중단 시 발생하지 않습니다.

Hook이 발생하지 않음

Hook이 구성되었지만 실행되지 않습니다.

- `/hooks` 를 실행하고 hook이 올바른 이벤트 아래에 나타나는지 확인합니다
- Matcher 패턴이 도구 이름과 정확히 일치하는지 확인합니다 (matchers는 대소문자 구분)
- 올바른 이벤트 유형을 트리거하고 있는지 확인합니다 (예: `PreToolUse` 는 도구 실행 전에 발생, `PostToolUse` 는 후에 발생)
- 비대화형 모드 (`-p`)에서 `PermissionRequest` hooks를 사용하는 경우 대신 `PreToolUse` 로 전환합니다

출력에 Hook 오류

트랜스크립트에 “PreToolUse hook error: …”와 같은 메시지가 표시됩니다.

- 스크립트가 예기치 않게 0이 아닌 코드로 종료되었습니다. 샘플 JSON을 파이핑하여 수동으로 테스트합니다:

```
echo '{"tool_name":"Bash","tool_input":{"command":"ls"}}' | ./my-hook.sh
echo $?  # 종료 코드 확인
```

- “command not found”가 표시되면 절대 경로 또는 `$CLAUDE_PROJECT_DIR` 을 사용하여 스크립트를 참조합니다
- “jq: command not found”가 표시되면 `jq` 를 설치하거나 JSON 구문 분석을 위해 Python/Node.js를 사용합니다
- 스크립트가 실행되지 않으면 실행 가능하게 만듭니다: `chmod +x ./my-hook.sh`

`/hooks` 에 구성된 hooks가 없음

설정 파일을 편집했지만 hooks가 메뉴에 나타나지 않습니다.

- 세션을 다시 시작하거나 `/hooks` 를 열어 다시 로드합니다. `/hooks` 메뉴를 통해 추가된 Hooks는 즉시 적용되지만 수동 파일 편집은 다시 로드가 필요합니다.

- JSON이 유효한지 확인합니다 (후행 쉼표 및 주석은 허용되지 않음)
- 설정 파일이 올바른 위치에 있는지 확인합니다: 프로젝트 hooks의 경우 `.claude/settings.json`, 전역 hooks의 경우 `~/.claude/settings.json`

Stop hook이 무한 실행

Claude가 무한 루프에서 계속 작업하는 대신 중지합니다.

Stop hook 스크립트는 이미 트리거되었는지 확인해야 합니다. JSON 입력에서 `stop_hook_active` 필드를 구문 분석하고 `true`인 경우 조기에 종료합니다:

```
#!/bin/bash
INPUT=$(cat)
if [ "$(echo "$INPUT" | jq -r '.stop_hook_active')" = "true" ]; then
  exit 0  # Claude가 중지되도록 허용
fi
## ... hook 로직의 나머지
```

JSON 검증 실패

Claude Code가 hook 스크립트가 유효한 JSON을 출력하더라도 JSON 구문 분석 오류를 표시합니다.

Claude Code가 hook을 실행할 때 프로필 (`~/.zshrc` 또는 `~/.bashrc`)을 소싱하는 셸을 생성합니다. 프로필에 무조건적인 `echo` 문이 포함되어 있으면 해당 출력이 hook의 JSON에 앞에 붙습니다:

```
Shell ready on arm64
{"decision": "block", "reason": "Not allowed"}
```

Claude Code는 이를 JSON으로 구문 분석하려고 시도하고 실패합니다. 이를 수정하려면 셸 프로필의 echo 문을 래핑하여 대화형 셸에서만 실행되도록 합니다:

```
## ~/.zshrc 또는 ~/.bashrc에서
if [[ $- == *i* ]]; then
  echo "Shell ready"
fi
```

`$-` 변수는 셸 플래그를 포함하고 `i`는 대화형을 의미합니다. Hooks는 비대화형 셸에서 실행되므로 echo는 건너뜁니다.

디버그 기법

`Ctrl+O` 로 자세한 모드를 전환하여 트랜스크립트에서 hook 출력을 보거나 `claude --debug` 를 실행하여 일치한 hooks 및 종료 코드를 포함한 전체 실행 세부 정보를 확인합니다.

자세히 알아보기

- Hooks 참조: 전체 이벤트 스키마, JSON 출력 형식, 비동기 hooks 및 MCP tool hooks
- 보안 고려 사항: 공유 또는 프로덕션 환경에서 hooks를 배포하기 전에 검토합니다
- Bash 명령 검증기 예제: 완전한 참조 구현

MCP를 통해 Claude Code를 도구에 연결하기

> Model Context Protocol을 사용하여 Claude Code를 도구에 연결하는 방법을 알아봅니다.

Claude Code는 AI 도구 통합을 위한 오픈 소스 표준인 Model Context Protocol (MCP)를 통해 수백 개의 외부 도구 및 데이터 소스에 연결할 수 있습니다. MCP 서버는 Claude Code에 도구, 데이터베이스 및 API에 대한 액세스를 제공합니다.

MCP로 할 수 있는 것

MCP 서버가 연결되면 Claude Code에 다음을 요청할 수 있습니다:

- **이슈 추적기에서 기능 구현**: "JIRA 이슈 ENG-4521에 설명된 기능을 추가하고 GitHub에서 PR을 생성하세요."
- **모니터링 데이터 분석**: "Sentry와 Statsig을 확인하여 ENG-4521에 설명된 기능의 사용량을 확인하세요."
- **데이터베이스 쿼리**: "PostgreSQL 데이터베이스를 기반으로 기능 ENG-4521을 사용한 무작위 사용자 10명의 이메일을 찾으세요."
- **디자인 통합**: "Slack에 게시된 새로운 Figma 디자인을 기반으로 표준 이메일 템플릿을 업데이트하세요."
- **워크플로우 자동화**: "이 10명의 사용자를 새로운 기능에 대한 피드백 세션에 초대하는 Gmail 초안을 생성하세요."

인기 있는 MCP 서버

Claude Code에 연결할 수 있는 일반적으로 사용되는 MCP 서버는 다음과 같습니다:

> **Warning:**
>
> 타사 MCP 서버를 사용할 때는 자신의 책임하에 사용하십시오 - Anthropic은 이러한 모든 서버의 정확성이나 보안을 검증하지 않았습니다. 설치하는 MCP 서버를 신뢰하는지 확인하세요. 신뢰할 수 없는 콘텐츠를 가져올 수 있는 MCP 서버를 사용할 때는 특히 주의하세요. 이러한 서버는 프롬프트 주입 위험에 노출될 수 있습니다.

Server	Description	Command
Notion	Connect your Notion workspace to search, update, and power workflows across tools	`claude mcp add --transport http notion https://mcp.notion.com/mcp`
Canva	Search, create, autofill, and export Canva designs	`claude mcp add --transport http canva https://mcp.canva.com/mcp`
Figma	Generate diagrams and better code from Figma context	`claude mcp add --transport http figma-remote-mcp https://mcp.figma.com/mcp`
Atlassian	Access Jira & Confluence from Claude	`claude mcp add --transport http atlassian https://mcp.atlassian.com/v1/mcp`
Linear	Manage issues, projects & team workflows in Linear	`claude mcp add --transport http linear https://mcp.linear.app/mcp`
monday.com	Manage projects, boards, and workflows in monday.com	`claude mcp add --transport http monday https://mcp.monday.com/mcp`
Intercom	Access to Intercom data for better customer insights	`claude mcp add --transport http intercom https://mcp.intercom.com/mcp`
Vercel	Analyze, debug, and manage projects and deployments	`claude mcp add --transport http vercel https://mcp.vercel.com`

Server	Description	Command
Granola	The AI notepad for meetings	`claude mcp add --transport http granola https://mcp.granola.ai/mcp`
Asana	Connect to Asana to coordinate tasks, projects, and goals	`claude mcp add --transport streamable-http asana https://mcp.asana.com/v2/mcp`
Miro	Access and create new content on Miro boards	`claude mcp add --transport http miro https://mcp.miro.com/`
Sentry	Search, query, and debug errors intelligently	`claude mcp add --transport http sentry https://mcp.sentry.dev/mcp`
Supabase	Manage databases, authentication, and storage	`claude mcp add --transport http supabase https://mcp.supabase.com/mcp`
Hugging Face	Access the Hugging Face Hub and thousands of Gradio Apps	`claude mcp add --transport http hugging-face https://huggingface.co/mcp`
Context7	Up-to-date docs for LLMs and AI code editors	`claude mcp add --transport http context7 https://mcp.context7.com/mcp`
Stripe	Payment processing and financial infrastructure tools	`claude mcp add --transport http stripe https://mcp.stripe.com`

Server	Description	Command
Microsoft Learn	Search trusted Microsoft docs to power your development	`claude mcp add --transport http microsoft-learn https://learn.microsoft.com/api/mcp`
Clay	Find prospects. Research accounts. Personalize outreach	`claude mcp add --transport http clay https://api.clay.com/v3/mcp`
Webflow	Manage Webflow CMS, pages, assets and sites	`claude mcp add --transport http webflow https://mcp.webflow.com/mcp`
Cloudflare	Build applications with compute, storage, and AI	`claude mcp add --transport http cloudflare https://bindings.mcp.cloudflare.com/mcp`
Ramp	Search, access, and analyze your Ramp financial data	`claude mcp add --transport http ramp https://ramp-mcp-remote.ramp.com/mcp`
ZoomInfo	Enrich contacts & accounts with GTM intelligence	`claude mcp add --transport http zoominfo https://mcp.zoominfo.com/mcp`
Netlify	Create, deploy, manage, and secure websites on Netlify	`claude mcp add --transport http netlify https://netlify-mcp.netlify.app/mcp`
Make	Run Make scenarios and manage your Make account	`claude mcp add --transport http make https://mcp.make.com`

Server	Description	Command
GoDaddy	Search domains and check availability	`claude mcp add --transport http godaddy https://api.godaddy.com/v1/domains/mcp`
Google Cloud BigQuery	BigQuery: Advanced analytical insights for agents	`claude mcp add --transport http bigquery https://bigquery.googleapis.com/mcp`
PayPal	Access PayPal payments platform	`claude mcp add --transport http paypal https://mcp.paypal.com/mcp`
PostHog	Query, analyze, and manage your PostHog insights	`claude mcp add --transport http posthog https://mcp.posthog.com/mcp`
Similarweb	Real time web, mobile app, and market data	`claude mcp add --transport http similarweb https://mcp.similarweb.com`
Crypto.com	Real time prices, orders, charts, and more for crypto	`claude mcp add --transport http crypto.com https://mcp.crypto.com/market-data/mcp`
Attio	Search, manage, and update your Attio CRM from Claude	`claude mcp add --transport http attio https://mcp.attio.com/mcp`
Trivago	Find your ideal hotel at the best price	`claude mcp add --transport http trivago https://mcp.trivago.com/mcp`

Server	Description	Command
Jam	Record screen and collect automatic context for issues	`claude mcp add --transport http jam https://mcp.jam.dev/mcp`
Consensus	Explore scientific research	`claude mcp add --transport http consensus https://mcp.consensus.app/mcp`
Clockwise	Advanced scheduling and time management for work	`claude mcp add --transport http clockwise https://mcp.getclockwise.com/mcp`
Square	Search and manage transaction, merchant, and payment data	`claude mcp add --transport sse square https://mcp.squareup.com/sse`
Egnyte	Securely access and analyze Egnyte content	`claude mcp add --transport http egnyte https://mcp-server.egnyte.com/mcp`
Pylon	Search and manage Pylon support issues	`claude mcp add --transport http pylon https://mcp.usepylon.com/`
Honeycomb	Query and explore observability data and SLOs	`claude mcp add --transport http honeycomb https://mcp.honeycomb.io/mcp`

Note:

특정 통합이 필요하신가요? GitHub에서 수백 개 이상의 MCP 서버를 찾거나, MCP SDK를 사용하여 자신만의 서버를 구축하세요.

MCP 서버 설치

MCP 서버는 필요에 따라 세 가지 방식으로 구성할 수 있습니다:

옵션 1: 원격 HTTP 서버 추가

HTTP 서버는 원격 MCP 서버에 연결하기 위한 권장 옵션입니다. 이는 클라우드 기반 서비스에 가장 널리 지원되는 전송 방식입니다.

```
## 기본 구문
claude mcp add --transport http <name> <url>

## 실제 예: Notion에 연결
claude mcp add --transport http notion https://mcp.notion.com/mcp

## Bearer 토큰을 사용한 예
claude mcp add --transport http secure-api https://api.example.com/mcp \
  --header "Authorization: Bearer your-token"
```

옵션 2: 원격 SSE 서버 추가

Warning:

SSE (Server-Sent Events) 전송은 더 이상 사용되지 않습니다. 가능한 경우 HTTP 서버를 사용하세요.

```
## 기본 구문
claude mcp add --transport sse <name> <url>

## 실제 예: Asana에 연결
claude mcp add --transport sse asana https://mcp.asana.com/sse

## 인증 헤더를 사용한 예
claude mcp add --transport sse private-api https://api.company.com/sse \
  --header "X-API-Key: your-key-here"
```

옵션 3: 로컬 stdio 서버 추가

Stdio 서버는 컴퓨터에서 로컬 프로세스로 실행됩니다. 시스템에 직접 액세스하거나 사용자 정의 스크립트가 필요한 도구에 이상적입니다.

```
## 기본 구문
claude mcp add [options] <name> -- <command> [args...]

## 실제 예: Airtable 서버 추가
claude mcp add --transport stdio --env AIRTABLE_API_KEY=YOUR_KEY airtable \
  -- npx -y airtable-mcp-server
```

> **Note:**
>
> **중요: 옵션 순서**
>
> 모든 옵션(`--transport` , `--env` , `--scope` , `--header`)은 서버 이름 **앞에** 와야 합니다. `--` (이중 대시)는 서버 이름과 MCP 서버에 전달되는 명령 및 인수를 구분합니다.
>
> 예를 들어:
>
> - `claude mcp add --transport stdio myserver -- npx server` → `npx server` 실행
> - `claude mcp add --transport stdio --env KEY=value myserver -- python server.py --port 8080` → `KEY=value` 를 환경에서 `python server.py --port 8080` 실행
>
> 이는 Claude의 플래그와 서버의 플래그 간의 충돌을 방지합니다.

서버 관리

구성한 후에는 다음 명령으로 MCP 서버를 관리할 수 있습니다:

```
## 구성된 모든 서버 나열
claude mcp list

## 특정 서버의 세부 정보 가져오기
claude mcp get github

## 서버 제거
claude mcp remove github

## (Claude Code 내에서) 서버 상태 확인
/mcp
```

동적 도구 업데이트

Claude Code는 MCP `list_changed` 알림을 지원하므로 MCP 서버가 연결을 끊었다가 다시 연결할 필요 없이 사용 가능한 도구, 프롬프트 및 리소스를 동적으로 업데이트할 수 있습니다. MCP 서버가 `list_changed` 알림을 보내면 Claude Code는 해당 서버에서 사용 가능한 기능을 자동으로 새로 고칩니다.

Tip:

팁:

- `--scope` 플래그를 사용하여 구성이 저장되는 위치를 지정하세요:
- `local` (기본값): 현재 프로젝트에서만 사용자에게만 사용 가능 (이전 버전에서는 `project` 라고 불렸음)
- `project`: `.mcp.json` 파일을 통해 프로젝트의 모든 사람과 공유
- `user`: 모든 프로젝트에서 사용자에게 사용 가능 (이전 버전에서는 `global` 이라고 불렸음)
- `--env` 플래그로 환경 변수를 설정하세요 (예: `--env KEY=value`)
- `MCP_TIMEOUT` 환경 변수를 사용하여 MCP 서버 시작 시간 초과를 구성하세요 (예: `MCP_TIMEOUT=10000 claude` 는 10초 시간 초과를 설정)
- Claude Code는 MCP 도구 출력이 10,000 토큰을 초과할 때 경고를 표시합니다. 이 제한을 늘리려면 `MAX_MCP_OUTPUT_TOKENS` 환경 변수를 설정하세요 (예: `MAX_MCP_OUTPUT_TOKENS=50000`)
- OAuth 2.0 인증이 필요한 원격 서버로 인증하려면 `/mcp` 를 사용하세요

Warning:

Windows 사용자: 기본 Windows (WSL 아님)에서 `npx` 를 사용하는 로컬 MCP 서버는 올바른 실행을 보장하기 위해 `cmd /c` 래퍼가 필요합니다.

```
## 이는 Windows가 실행할 수 있는 command="cmd"를 생성합니다
claude mcp add --transport stdio my-server -- cmd /c npx -y @some/package
```

`cmd /c` 래퍼가 없으면 Windows가 `npx`를 직접 실행할 수 없기 때문에 "Connection closed" 오류가 발생합니다. (위의 참고 사항에서 `--` 매개변수에 대한 설명을 참조하세요.)

플러그인 제공 MCP 서버

플러그인은 MCP 서버를 번들로 제공할 수 있으며, 플러그인이 활성화되면 도구 및 통합을 자동으로 제공합니다. 플러그인 MCP 서버는 사용자 구성 서버와 동일하게 작동합니다.

플러그인 MCP 서버의 작동 방식:

- 플러그인은 플러그인 루트의 `.mcp.json` 또는 `plugin.json`에 인라인으로 MCP 서버를 정의합니다
- 플러그인이 활성화되면 MCP 서버가 자동으로 시작됩니다
- 플러그인 MCP 도구는 수동으로 구성된 MCP 도구와 함께 나타납니다
- 플러그인 서버는 플러그인 설치를 통해 관리됩니다 (`/mcp` 명령이 아님)

플러그인 MCP 구성 예:

플러그인 루트의 `.mcp.json`:

```
{
  "database-tools": {
    "command": "${CLAUDE_PLUGIN_ROOT}/servers/db-server",
    "args": ["--config", "${CLAUDE_PLUGIN_ROOT}/config.json"],
    "env": {
      "DB_URL": "${DB_URL}"
    }
  }
}
```

또는 `plugin.json`에 인라인:

```
{
  "name": "my-plugin",
  "mcpServers": {
    "plugin-api": {
      "command": "${CLAUDE_PLUGIN_ROOT}/servers/api-server",
      "args": ["--port", "8080"]
    }
  }
}
```

플러그인 MCP 기능:

- **자동 라이프사이클**: 플러그인이 활성화되면 서버가 시작되지만 MCP 서버 변경 사항을 적용하려면 Claude Code를 다시 시작해야 합니다 (활성화 또는 비활성화)
- **환경 변수**: 플러그인 상대 경로에 `${CLAUDE_PLUGIN_ROOT}` 사용
- **사용자 환경 액세스**: 수동으로 구성된 서버와 동일한 환경 변수에 액세스
- **여러 전송 유형**: stdio, SSE 및 HTTP 전송 지원 (전송 지원은 서버에 따라 다를 수 있음)

플러그인 MCP 서버 보기:

```
## Claude Code 내에서 플러그인 서버를 포함한 모든 MCP 서버 보기
/mcp
```

플러그인 서버는 플러그인에서 온 것을 나타내는 표시기와 함께 목록에 나타납니다.

플러그인 MCP 서버의 이점:

- **번들 배포**: 도구 및 서버가 함께 패키징됨
- **자동 설정**: 수동 MCP 구성이 필요 없음
- **팀 일관성**: 플러그인이 설치되면 모든 사람이 동일한 도구를 얻음

플러그인과 함께 MCP 서버를 번들로 제공하는 방법에 대한 자세한 내용은 플러그인 구성 요소 참조를 참조하세요.

MCP 설치 범위

MCP 서버는 서버 접근성 및 공유를 관리하기 위해 세 가지 다른 범위 수준에서 구성할 수 있습니다. 이러한 범위를 이해하면 특정 요구 사항에 맞게 서버를 구성하는 최선의 방법을 결정하는 데 도움이 됩니다.

로컬 범위

로컬 범위 서버는 기본 구성 수준을 나타내며 프로젝트 경로 아래 `~/.claude.json`에 저장됩니다. 이러한 서버는 사용자에게만 비공개이며 현재 프로젝트 디렉토리 내에서 작업할 때만 액세스할 수 있습니다. 이 범위는 개인 개발 서버, 실험적 구성 또는 공유하면 안 되는 민감한 자격 증명을 포함하는 서버에 이상적입니다.

> **Note:**
>
> MCP 서버의 "로컬 범위"라는 용어는 일반 로컬 설정과 다릅니다. MCP 로컬 범위 서버는 `~/.claude.json` (홈 디렉토리)에 저장되고, 일반 로컬 설정은 `.claude/settings.local.json` (프로젝트 디렉토리)을 사용합니다. 설정 파일 위치에 대한 자세한 내용은 설정을 참조하세요.

```
## 로컬 범위 서버 추가 (기본값)
claude mcp add --transport http stripe https://mcp.stripe.com

## 명시적으로 로컬 범위 지정
claude mcp add --transport http stripe --scope local https://mcp.stripe.com
```

프로젝트 범위

프로젝트 범위 서버는 프로젝트 루트 디렉토리의 `.mcp.json` 파일에 구성을 저장하여 팀 협업을 가능하게 합니다. 이 파일은 버전 제어에 체크인되도록 설계되어 모든 팀 멤버가 동일한 MCP 도구 및 서비스에 액세스할 수 있도록 합니다. 프로젝트 범위 서버를 추가하면 Claude Code는 자동으로 이 파일을 생성하거나 적절한 구성 구조로 업데이트합니다.

```
## 프로젝트 범위 서버 추가
claude mcp add --transport http paypal --scope project https://mcp.paypal.com/mcp
```

결과 `.mcp.json` 파일은 표준화된 형식을 따릅니다:

```
{
  "mcpServers": {
    "shared-server": {
      "command": "/path/to/server",
      "args": [],
      "env": {}
    }
  }
}
```

보안상의 이유로 Claude Code는 `.mcp.json` 파일의 프로젝트 범위 서버를 사용하기 전에 승인을 요청합니다. 이러한 승인 선택을 재설정해야 하는 경우 `claude mcp reset-project-choices` 명령을 사용하세요.

사용자 범위

사용자 범위 서버는 `~/.claude.json`에 저장되며 교차 프로젝트 접근성을 제공하므로 컴퓨터의 모든 프로젝트에서 사용할 수 있으면서 사용자 계정에만 비공개입니다. 이 범위는 개인 유틸리티 서버, 개발 도구 또는 다양한 프로젝트에서 자주 사용하는 서비스에 적합합니다.

```
## 사용자 서버 추가
claude mcp add --transport http hubspot --scope user https://mcp.hubspot.com/
anthropic
```

올바른 범위 선택

다음을 기반으로 범위를 선택하세요:

- **로컬 범위**: 개인 서버, 실험적 구성 또는 한 프로젝트에만 해당하는 민감한 자격 증명
- **프로젝트 범위**: 팀 공유 서버, 프로젝트 특정 도구 또는 협업에 필요한 서비스
- **사용자 범위**: 여러 프로젝트에서 필요한 개인 유틸리티, 개발 도구 또는 자주 사용하는 서비스

> **Note:**
>
> **MCP 서버는 어디에 저장되나요?**
>
> - **사용자 및 로컬 범위**: `~/.claude.json` (`mcpServers` 필드 또는 프로젝트 경로 아래)
> - **프로젝트 범위**: 프로젝트 루트의 `.mcp.json` (소스 제어에 체크인됨)
> - **관리됨**: 시스템 디렉토리의 `managed-mcp.json` (관리되는 MCP 구성 참조)

범위 계층 및 우선순위

MCP 서버 구성은 명확한 우선순위 계층을 따릅니다. 동일한 이름의 서버가 여러 범위에 존재할 때 시스템은 로컬 범위 서버를 먼저 우선시하고, 그 다음 프로젝트 범위 서버, 마지막으로 사용자 범위 서버를 우선시하여 충돌을 해결합니다. 이 설계는 필요할 때 개인 구성이 공유 구성을 재정의할 수 있도록 합니다.

`.mcp.json` 의 환경 변수 확장

Claude Code는 `.mcp.json` 파일의 환경 변수 확장을 지원하므로 팀이 구성을 공유하면서 머신 특정 경로 및 API 키와 같은 민감한 값에 대한 유연성을 유지할 수 있습니다.

지원되는 구문:

- `${VAR}` - 환경 변수 `VAR` 의 값으로 확장
- `${VAR:-default}` - `VAR` 이 설정되면 확장, 그렇지 않으면 `default` 사용

확장 위치: 환경 변수는 다음에서 확장할 수 있습니다:

- `command` - 서버 실행 파일 경로
- `args` - 명령줄 인수
- `env` - 서버에 전달되는 환경 변수
- `url` - HTTP 서버 유형의 경우
- `headers` - HTTP 서버 인증의 경우

변수 확장을 사용한 예:

```
{
  "mcpServers": {
    "api-server": {
      "type": "http",
      "url": "${API_BASE_URL:-https://api.example.com}/mcp",
      "headers": {
        "Authorization": "Bearer ${API_KEY}"
      }
    }
  }
}
```

필수 환경 변수가 설정되지 않았고 기본값이 없으면 Claude Code는 구성을 구문 분석하지 못합니다.

실제 예

{/* ### 예: Playwright로 브라우저 테스트 자동화

```
claude mcp add --transport stdio playwright -- npx -y @playwright/mcp@latest
```

그런 다음 브라우저 테스트를 작성하고 실행합니다:

```
test@example.com으로 로그인 흐름이 작동하는지 테스트
```

```
모바일에서 체크아웃 페이지의 스크린샷 촬영
```

```
 검색 기능이 결과를 반환하는지 확인
  ``` */}

예: Sentry로 오류 모니터링

```bash  theme={null}
claude mcp add --transport http sentry https://mcp.sentry.dev/mcp
```

Sentry 계정으로 인증합니다:

```
/mcp
```

그런 다음 프로덕션 문제를 디버깅합니다:

```
지난 24시간 동안 가장 일반적인 오류는 무엇입니까?
```

```
오류 ID abc123의 스택 추적을 보여주세요
```

```
어떤 배포가 이러한 새로운 오류를 도입했습니까?
```

예: 코드 검토를 위해 GitHub에 연결

```
claude mcp add --transport http github https://api.githubcopilot.com/mcp/
```

필요한 경우 GitHub에 대해 “인증”을 선택하여 인증합니다:

```
/mcp
```

그런 다음 GitHub로 작업합니다:

```
PR #456을 검토하고 개선 사항을 제안하세요
```

```
방금 발견한 버그에 대한 새 이슈를 생성하세요
```

```
나에게 할당된 모든 열린 PR을 보여주세요
```

예: PostgreSQL 데이터베이스 쿼리

```
claude mcp add --transport stdio db -- npx -y @bytebase/dbhub \
  --dsn "postgresql://readonly:pass@prod.db.com:5432/analytics"
```

그런 다음 자연스럽게 데이터베이스를 쿼리합니다:

```
이번 달 총 수익은 얼마입니까?
```

```
주문 테이블의 스키마를 보여주세요
```

```
지난 90일 동안 구매하지 않은 고객을 찾으세요
```

원격 MCP 서버로 인증

많은 클라우드 기반 MCP 서버는 인증이 필요합니다. Claude Code는 보안 연결을 위해 OAuth 2.0을 지원합니다.

Step 1: 인증이 필요한 서버 추가

예를 들어:

```
claude mcp add --transport http sentry https://mcp.sentry.dev/mcp
```

Step 2: Claude Code 내에서 /mcp 명령 사용

Claude Code에서 다음 명령을 사용합니다:

```
/mcp
```

그런 다음 브라우저에서 로그인 단계를 따릅니다.

> **Tip:**
>
> 팁:
>
> - 인증 토큰은 안전하게 저장되고 자동으로 새로 고쳐집니다
> - `/mcp` 메뉴에서 “Clear authentication”을 사용하여 액세스를 취소합니다
> - 브라우저가 자동으로 열리지 않으면 제공된 URL을 복사하여 수동으로 엽니다
> - 인증 후 브라우저 리디렉션이 연결 오류로 실패하면 브라우저의 주소 표시줄에서 전체 콜백 URL을 복사하여 Claude Code에 나타나는 URL 프롬프트에 붙여넣습니다
> - OAuth 인증은 HTTP 서버에서 작동합니다

고정 OAuth 콜백 포트 사용

일부 MCP 서버는 미리 등록된 특정 리디렉션 URI가 필요합니다. 기본적으로 Claude Code는 OAuth 콜백을 위해 무작위로 사용 가능한 포트를 선택합니다. `--callback-port` 를 사용하여 포트를 고정하여 `http://localhost:PORT/callback` 형식의 사전 등록된 리디렉션 URI와 일치하도록 합니다.

`--callback-port` 를 단독으로 사용할 수 있습니다 (동적 클라이언트 등록 포함) 또는 `--client-id` 와 함께 사용할 수 있습니다 (사전 구성된 자격 증명 포함).

```
## 동적 클라이언트 등록을 사용한 고정 콜백 포트
claude mcp add --transport http \
  --callback-port 8080 \
  my-server https://mcp.example.com/mcp
```

사전 구성된 OAuth 자격 증명 사용

일부 MCP 서버는 자동 OAuth 설정을 지원하지 않습니다. “Incompatible auth server: does not support dynamic client registration” 과 같은 오류가 표시되면 서버에 사전 구성된 자격 증명이 필요합니다. 먼저 서버의 개발자 포털을 통해 OAuth 앱을 등록한 다음 서버를 추가할 때 자격 증명을 제공합니다.

Step 1: 서버로 OAuth 앱 등록

서버의 개발자 포털을 통해 앱을 생성하고 클라이언트 ID와 클라이언트 시크릿을 기록합니다.

많은 서버는 리디렉션 URI도 필요합니다. 그렇다면 포트를 선택하고 `http://localhost:PORT/callback` 형식으로 리디렉션 URI를 등록합니다. 다음 단계에서 `--callback-port` 와 함께 동일한 포트를 사용합니다.

Step 2: 자격 증명으로 서버 추가

다음 방법 중 하나를 선택합니다. `--callback-port` 에 사용되는 포트는 사용 가능한 모든 포트일 수 있습니다. 이전 단계에서 등록한 리디렉션 URI와 일치하기만 하면 됩니다.

claude mcp add

`--client-id` 를 사용하여 앱의 클라이언트 ID를 전달합니다. `--client-secret` 플래그는 마스킹된 입력으로 시크릿을 요청합니다:

```
claude mcp add --transport http \
  --client-id your-client-id --client-secret --callback-port 8080 \
  my-server https://mcp.example.com/mcp
```

claude mcp add-json

JSON 구성에 `oauth` 객체를 포함하고 `--client-secret` 을 별도의 플래그로 전달합니다:

```
claude mcp add-json my-server \
  '{"type":"http","url":"https://mcp.example.com/mcp","oauth":{"clientId":"your-client-id","callbackPort":8080}}' \
  --client-secret
```

claude mcp add-json (콜백 포트만)

동적 클라이언트 등록을 사용하면서 포트를 고정하려면 클라이언트 ID 없이 `--callback-port`를 사용합니다:

```
claude mcp add-json my-server \
  '{"type":"http","url":"https://mcp.example.com/mcp","oauth":{"callbackPort":8080}}'
```

CI / 환경 변수

환경 변수를 통해 시크릿을 설정하여 대화형 프롬프트를 건너뜁니다:

```
MCP_CLIENT_SECRET=your-secret claude mcp add --transport http \
  --client-id your-client-id --client-secret --callback-port 8080 \
  my-server https://mcp.example.com/mcp
```

Step 3: Claude Code에서 인증

Claude Code에서 `/mcp`를 실행하고 브라우저 로그인 흐름을 따릅니다.

> **Tip:**
>
> 팁:
>
> - 클라이언트 시크릿은 구성에 저장되지 않고 시스템 키체인 (macOS) 또는 자격 증명 파일에 안전하게 저장됩니다
> - 서버가 시크릿이 없는 공개 OAuth 클라이언트를 사용하는 경우 `--client-secret` 없이 `--client-id`만 사용합니다
> - `--callback-port`는 `--client-id`와 함께 또는 없이 사용할 수 있습니다
> - 이러한 플래그는 HTTP 및 SSE 전송에만 적용됩니다. stdio 서버에는 영향을 주지 않습니다
> - `claude mcp get <name>`을 사용하여 OAuth 자격 증명이 서버에 대해 구성되었는지 확인합니다

OAuth 메타데이터 검색 재정의

MCP 서버가 표준 OAuth 메타데이터 엔드포인트 (`/.well-known/oauth-authorization-server`)에서 오류를 반환하지만 작동하는 OIDC 엔드포인트를 노출하는 경우 Claude Code에 표준 검색 체인을 우회하여 지정한 URL에서 직접 OAuth 메타데이터를 가져오도록 지시할 수 있습니다.

`.mcp.json` 의 서버 구성의 `oauth` 객체에 `authServerMetadataUrl` 을 설정합니다:

```
{
  "mcpServers": {
    "my-server": {
      "type": "http",
      "url": "https://mcp.example.com/mcp",
      "oauth": {
        "authServerMetadataUrl": "https://auth.example.com/.well-known/openid-
configuration"
      }
    }
  }
}
```

URL은 `https://` 를 사용해야 합니다. 이 옵션은 Claude Code v2.1.64 이상이 필요합니다.

JSON 구성에서 MCP 서버 추가

MCP 서버에 대한 JSON 구성이 있는 경우 직접 추가할 수 있습니다:

Step 1: JSON에서 MCP 서버 추가

```
## 기본 구문
claude mcp add-json <name> '<json>'

## 예: JSON 구성으로 HTTP 서버 추가
claude mcp add-json weather-api '{"type":"http","url":"https://api.weather.com/
mcp","headers":{"Authorization":"Bearer token"}}'

## 예: JSON 구성으로 stdio 서버 추가
claude mcp add-json local-weather '{"type":"stdio","command":"/path/to/weather-
cli","args":["--api-key","abc123"],"env":{"CACHE_DIR":"/tmp"}}'

## 예: 사전 구성된 OAuth 자격 증명으로 HTTP 서버 추가
claude mcp add-json my-server '{"type":"http","url":"https://mcp.example.com/
mcp","oauth":{"clientId":"your-client-id","callbackPort":8080}}' --client-secret
```

Step 2: 서버가 추가되었는지 확인

```
claude mcp get weather-api
```

> **Tip:**
>
> 팁:
>
> - JSON이 셸에서 올바르게 이스케이프되었는지 확인합니다
> - JSON은 MCP 서버 구성 스키마를 준수해야 합니다
> - `--scope user`를 사용하여 프로젝트 특정 구성 대신 사용자 구성에 서버를 추가할 수 있습니다

Claude Desktop에서 MCP 서버 가져오기

Claude Desktop에서 MCP 서버를 이미 구성한 경우 가져올 수 있습니다:

Step 1: Claude Desktop에서 서버 가져오기

```
## 기본 구문
claude mcp add-from-claude-desktop
```

Step 2: 가져올 서버 선택

명령을 실행한 후 가져올 서버를 선택할 수 있는 대화형 대화 상자가 표시됩니다.

Step 3: 서버가 가져와졌는지 확인

```
claude mcp list
```

> **Tip:**
>
> 팁:
>
> - 이 기능은 macOS 및 Windows Subsystem for Linux (WSL)에서만 작동합니다
> - 이러한 플랫폼의 표준 위치에서 Claude Desktop 구성 파일을 읽습니다
> - `--scope user` 플래그를 사용하여 사용자 구성에 서버를 추가합니다
> - 가져온 서버는 Claude Desktop과 동일한 이름을 갖습니다
> - 동일한 이름의 서버가 이미 존재하면 숫자 접미사가 붙습니다 (예: `server_1`)

Claude.ai에서 MCP 서버 사용

Claude.ai 계정으로 Claude Code에 로그인한 경우 Claude.ai에서 추가한 MCP 서버는 Claude Code에서 자동으로 사용 가능합니다:

Step 1: Claude.ai에서 MCP 서버 구성

claude.ai/settings/connectors에서 서버를 추가합니다. Team 및 Enterprise 플랜에서는 관리자만 서버를 추가할 수 있습니다.

Step 2: MCP 서버 인증

Claude.ai에서 필요한 인증 단계를 완료합니다.

Step 3: Claude Code에서 서버 보기 및 관리

Claude Code에서 다음 명령을 사용합니다:

```
/mcp
```

Claude.ai 서버는 Claude.ai에서 온 것을 나타내는 표시기와 함께 목록에 나타납니다.

Claude Code에서 claude.ai MCP 서버를 비활성화하려면 `ENABLE_CLAUDEAI_MCP_SERVERS` 환경 변수를 `false`로 설정합니다:

```
ENABLE_CLAUDEAI_MCP_SERVERS=false claude
```

Claude Code를 MCP 서버로 사용

Claude Code 자체를 다른 애플리케이션이 연결할 수 있는 MCP 서버로 사용할 수 있습니다:

```
## Claude를 stdio MCP 서버로 시작
claude mcp serve
```

claude_desktop_config.json에 이 구성을 추가하여 Claude Desktop에서 사용할 수 있습니다:

```
{
  "mcpServers": {
    "claude-code": {
      "type": "stdio",
      "command": "claude",
      "args": ["mcp", "serve"],
      "env": {}
    }
  }
}
```

Warning:

실행 파일 경로 구성: `command` 필드는 Claude Code 실행 파일을 참조해야 합니다. `claude` 명령이 시스템의 PATH에 없으면 실행 파일의 전체 경로를 지정해야 합니다.

전체 경로를 찾으려면:

```
which claude
```

그런 다음 구성에서 전체 경로를 사용합니다:

```
{
  "mcpServers": {
    "claude-code": {
      "type": "stdio",
      "command": "/full/path/to/claude",
      "args": ["mcp", "serve"],
      "env": {}
    }
  }
}
```

올바른 실행 파일 경로가 없으면 `spawn claude ENOENT` 와 같은 오류가 발생합니다.

Tip:

팁:

- 서버는 View, Edit, LS 등과 같은 Claude의 도구에 대한 액세스를 제공합니다
- Claude Desktop에서 Claude에게 디렉토리의 파일을 읽고, 편집하는 등을 요청해 보세요
- 이 MCP 서버는 Claude Code의 도구만 MCP 클라이언트에 노출하므로 클라이언트는 개별 도구 호출에 대한 사용자 확인을 구현할 책임이 있습니다.

MCP 출력 제한 및 경고

MCP 도구가 큰 출력을 생성할 때 Claude Code는 토큰 사용량을 관리하여 대화 컨텍스트가 압도되지 않도록 합니다:

- **출력 경고 임계값**: Claude Code는 MCP 도구 출력이 10,000 토큰을 초과할 때 경고를 표시합니다
- **구성 가능한 제한**: `MAX_MCP_OUTPUT_TOKENS` 환경 변수를 사용하여 최대 허용 MCP 출력 토큰을 조정할 수 있습니다
- **기본 제한**: 기본 최대값은 25,000 토큰입니다

큰 출력을 생성하는 도구의 제한을 늘리려면:

```
## MCP 도구 출력의 제한을 높게 설정
export MAX_MCP_OUTPUT_TOKENS=50000
claude
```

이는 다음을 수행하는 MCP 서버로 작업할 때 특히 유용합니다:

- 대규모 데이터 세트 또는 데이터베이스 쿼리
- 상세한 보고서 또는 문서 생성
- 광범위한 로그 파일 또는 디버깅 정보 처리

> **Warning:**
>
> 특정 MCP 서버에서 자주 출력 경고가 발생하면 제한을 늘리거나 서버를 구성하여 응답을 페이지 매김하거나 필터링하는 것을 고려하세요.

MCP 리소스 사용

MCP 서버는 파일을 참조하는 방식과 유사하게 @ 멘션을 사용하여 참조할 수 있는 리소스를 노출할 수 있습니다.

MCP 리소스 참조

Step 1: 사용 가능한 리소스 나열

프롬프트에 `@`를 입력하여 연결된 모든 MCP 서버의 사용 가능한 리소스를 확인합니다. 리소스는 자동 완성 메뉴의 파일과 함께 나타납니다.

Step 2: 특정 리소스 참조

`@server:protocol://resource/path` 형식을 사용하여 리소스를 참조합니다:

```
@github:issue://123을 분석하고 수정 사항을 제안할 수 있나요?
```

```
@docs:file://api/authentication의 API 문서를 검토해 주세요
```

Step 3: 여러 리소스 참조

단일 프롬프트에서 여러 리소스를 참조할 수 있습니다:

```
@postgres:schema://users와 @docs:file://database/user-model을 비교하세요
```

> **Tip:**
>
> 팁:
>
> - 리소스는 참조될 때 자동으로 가져와지고 첨부 파일로 포함됩니다

- 리소스 경로는 @ 멘션 자동 완성에서 퍼지 검색 가능합니다
- Claude Code는 서버가 지원할 때 MCP 리소스를 나열하고 읽을 수 있는 도구를 자동으로 제공합니다
- 리소스는 MCP 서버가 제공하는 모든 유형의 콘텐츠를 포함할 수 있습니다 (텍스트, JSON, 구조화된 데이터 등)

MCP Tool Search로 확장

많은 MCP 서버를 구성한 경우 도구 정의가 컨텍스트 윈도우의 상당 부분을 차지할 수 있습니다. MCP Tool Search는 모든 도구를 미리 로드하는 대신 필요에 따라 동적으로 도구를 로드하여 이 문제를 해결합니다.

작동 방식

Claude Code는 MCP 도구 설명이 컨텍스트 윈도우의 10% 이상을 차지할 때 자동으로 Tool Search를 활성화합니다. 이 임계값을 조정하거나 Tool Search를 완전히 비활성화할 수 있습니다. 트리거될 때:

1. MCP 도구는 미리 컨텍스트에 로드되지 않고 연기됩니다
2. Claude는 검색 도구를 사용하여 필요할 때 관련 MCP 도구를 검색합니다
3. Claude가 실제로 필요한 도구만 컨텍스트에 로드됩니다
4. MCP 도구는 관점에서 이전과 정확히 동일하게 계속 작동합니다

MCP 서버 작성자용

MCP 서버를 구축하는 경우 Tool Search가 활성화되면 서버 지침 필드가 더 유용해집니다. 서버 지침은 Claude가 skills의 작동 방식과 유사하게 도구를 검색할 시기를 이해하는 데 도움이 됩니다.

다음을 설명하는 명확하고 설명적인 서버 지침을 추가합니다:

- 도구가 처리하는 작업의 범주
- Claude가 도구를 검색해야 할 때
- 서버가 제공하는 주요 기능

Tool Search 구성

Tool Search는 기본적으로 자동 모드에서 실행되므로 MCP 도구 정의가 컨텍스트 임계값을 초과할 때만 활성화됩니다. 도구가 적으면 Tool Search 없이 정상적으로 로드됩니다. 이 기능은 `tool_reference` 블록을 지원하는 모델이 필요합니다: Sonnet 4 이상 또는 Opus 4 이상. Haiku 모델은 Tool Search를 지원하지 않습니다.

`ENABLE_TOOL_SEARCH` 환경 변수로 Tool Search 동작을 제어합니다:

값	동작
`auto`	MCP 도구가 컨텍스트의 10%를 초과할 때 활성화 (기본값)
`auto:<N>`	사용자 정의 임계값에서 활성화, `<N>` 은 백분율 (예: `auto:5` 는 5%)
`true`	항상 활성화
`false`	비활성화, 모든 MCP 도구 미리 로드

```
## 사용자 정의 5% 임계값 사용
ENABLE_TOOL_SEARCH=auto:5 claude

## Tool Search 완전히 비활성화
ENABLE_TOOL_SEARCH=false claude
```

또는 settings.json `env` 필드에서 값을 설정합니다.

`disallowedTools` 설정을 사용하여 MCPSearch 도구를 특별히 비활성화할 수도 있습니다:

```
{
  "permissions": {
    "deny": ["MCPSearch"]
  }
}
```

MCP 프롬프트를 명령으로 사용

MCP 서버는 Claude Code에서 명령으로 사용 가능하게 되는 프롬프트를 노출할 수 있습니다.

MCP 프롬프트 실행

Step 1: 사용 가능한 프롬프트 검색

`/` 를 입력하여 MCP 서버의 프롬프트를 포함한 모든 사용 가능한 명령을 확인합니다. MCP 프롬프트는 `/mcp__servername__promptname` 형식으로 나타납니다.

Step 2: 인수 없이 프롬프트 실행

```
/mcp__github__list_prs
```

Step 3: 인수를 사용하여 프롬프트 실행

많은 프롬프트는 인수를 허용합니다. 명령 뒤에 공백으로 구분하여 전달합니다:

```
/mcp__github__pr_review 456
```

```
/mcp__jira__create_issue "로그인 흐름의 버그" high
```

> **Tip:**
>
> 팁:
>
> - MCP 프롬프트는 연결된 서버에서 동적으로 검색됩니다
> - 인수는 프롬프트의 정의된 매개변수를 기반으로 구문 분석됩니다
> - 프롬프트 결과는 대화에 직접 주입됩니다
> - 서버 및 프롬프트 이름은 정규화됩니다 (공백은 밑줄이 됨)

관리되는 MCP 구성

MCP 서버에 대한 중앙 집중식 제어가 필요한 조직의 경우 Claude Code는 두 가지 구성 옵션을 지원합니다:

1. `managed-mcp.json` **을 사용한 독점 제어**: 사용자가 수정하거나 확장할 수 없는 고정된 MCP 서버 세트 배포
2. **허용 목록/거부 목록을 사용한 정책 기반 제어**: 사용자가 자신의 서버를 추가할 수 있지만 허용되는 서버를 제한

이러한 옵션을 통해 IT 관리자는 다음을 수행할 수 있습니다:

- **직원이 액세스할 수 있는 MCP 서버 제어**: 조직 전체에 표준화된 승인된 MCP 서버 세트 배포
- **승인되지 않은 MCP 서버 방지**: 사용자가 승인되지 않은 MCP 서버를 추가하지 못하도록 제한
- **MCP 완전히 비활성화**: 필요한 경우 MCP 기능을 완전히 제거

옵션 1: managed-mcp.json을 사용한 독점 제어

`managed-mcp.json` 파일을 배포하면 모든 MCP 서버에 대한 **독점 제어**를 갖습니다. 사용자는 이 파일에 정의된 서버 이외의 MCP 서버를 추가, 수정 또는 사용할 수 없습니다. 이는 완전한 제어를 원하는 조직에 가장 간단한 방법입니다.

시스템 관리자는 구성 파일을 시스템 전체 디렉토리에 배포합니다:

- macOS: `/Library/Application Support/ClaudeCode/managed-mcp.json`

- Linux 및 WSL: `/etc/claude-code/managed-mcp.json`
- Windows: `C:\Program Files\ClaudeCode\managed-mcp.json`

> **Note:**
>
> 이는 시스템 전체 경로입니다 (`~/Library/...` 와 같은 사용자 홈 디렉토리가 아님). IT 관리자가 배포하기 위해 관리자 권한이 필요합니다.

`managed-mcp.json` 파일은 표준 `.mcp.json` 파일과 동일한 형식을 사용합니다:

```
{
  "mcpServers": {
    "github": {
      "type": "http",
      "url": "https://api.githubcopilot.com/mcp/"
    },
    "sentry": {
      "type": "http",
      "url": "https://mcp.sentry.dev/mcp"
    },
    "company-internal": {
      "type": "stdio",
      "command": "/usr/local/bin/company-mcp-server",
      "args": ["--config", "/etc/company/mcp-config.json"],
      "env": {
        "COMPANY_API_URL": "https://internal.company.com"
      }
    }
  }
}
```

옵션 2: 허용 목록 및 거부 목록을 사용한 정책 기반 제어

독점 제어를 하는 대신 관리자는 사용자가 자신의 MCP 서버를 구성할 수 있도록 허용하면서 허용되는 서버에 제한을 적용할 수 있습니다. 이 방법은 관리되는 설정 파일의 `allowedMcpServers` 및 `deniedMcpServers` 를 사용합니다.

> **Note:**

> **옵션 선택**: 사용자 사용자 정의 없이 고정된 서버 세트를 배포하려면 옵션 1 (`managed-mcp.json`)을 사용합니다. 사용자가 정책 제약 내에서 자신의 서버를 추가할 수 있도록 하려면 옵션 2 (허용 목록/거부 목록)를 사용합니다.

제한 옵션

허용 목록 또는 거부 목록의 각 항목은 세 가지 방식으로 서버를 제한할 수 있습니다:

1. **서버 이름으로** (`serverName`): 서버의 구성된 이름과 일치
2. **명령으로** (`serverCommand`): stdio 서버를 시작하는 데 사용되는 정확한 명령 및 인수와 일치
3. **URL 패턴으로** (`serverUrl`): 와일드카드 지원을 사용하여 원격 서버 URL과 일치

중요: 각 항목은 `serverName` , `serverCommand` 또는 `serverUrl` 중 정확히 하나를 가져야 합니다.

구성 예

```
{
  "allowedMcpServers": [
    // 서버 이름으로 허용
    { "serverName": "github" },
    { "serverName": "sentry" },

    // 정확한 명령으로 허용 (stdio 서버의 경우)
    { "serverCommand": ["npx", "-y", "@modelcontextprotocol/server-filesystem"] },
    { "serverCommand": ["python", "/usr/local/bin/approved-server.py"] },

    // URL 패턴으로 허용 (원격 서버의 경우)
    { "serverUrl": "https://mcp.company.com/*" },
    { "serverUrl": "https://*.internal.corp/*" }
  ],
  "deniedMcpServers": [
    // 서버 이름으로 차단
    { "serverName": "dangerous-server" },

    // 정확한 명령으로 차단 (stdio 서버의 경우)
    { "serverCommand": ["npx", "-y", "unapproved-package"] },

    // URL 패턴으로 차단 (원격 서버의 경우)
    { "serverUrl": "https://*.untrusted.com/*" }
  ]
}
```

명령 기반 제한의 작동 방식

정확한 일치:

- 명령 배열은 **정확히** 일치해야 합니다 - 명령과 올바른 순서의 모든 인수
- 예: `["npx", "-y", "server"]` 는 `["npx", "server"]` 또는 `["npx", "-y", "server", "--flag"]` 와 일치하지 않습니다

Stdio 서버 동작:

- 허용 목록에 **모든** `serverCommand` 항목이 포함되면 stdio 서버는 해당 명령 중 하나와 일치해야 합니다

- Stdio 서버는 명령 제한이 있을 때 이름만으로는 통과할 수 없습니다
- 이는 관리자가 실행할 수 있는 명령을 적용할 수 있도록 합니다

비 stdio 서버 동작:

- 원격 서버 (HTTP, SSE, WebSocket)는 허용 목록에 `serverUrl` 항목이 있을 때 URL 기반 일치를 사용합니다
- URL 항목이 없으면 원격 서버는 이름 기반 일치로 돌아갑니다
- 명령 제한은 원격 서버에 적용되지 않습니다

URL 기반 제한의 작동 방식

URL 패턴은 `*`를 사용하여 와일드카드를 지원하여 모든 문자 시퀀스와 일치합니다. 이는 전체 도메인 또는 하위 도메인을 허용하는 데 유용합니다.

와일드카드 예:

- `https://mcp.company.com/*` - 특정 도메인의 모든 경로 허용
- `https://*.example.com/*` - example.com의 모든 하위 도메인 허용
- `http://localhost:*/*` - localhost의 모든 포트 허용

원격 서버 동작:

- 허용 목록에 **모든** `serverUrl` 항목이 포함되면 원격 서버는 해당 URL 패턴 중 하나와 일치해야 합니다
- 원격 서버는 URL 제한이 있을 때 이름만으로는 통과할 수 없습니다
- 이는 관리자가 허용되는 원격 엔드포인트를 적용할 수 있도록 합니다

```
{
  "allowedMcpServers": [
    { "serverUrl": "https://mcp.company.com/*" },
    { "serverUrl": "https://*.internal.corp/*" }
  ]
}
```

결과:

- `https://mcp.company.com/api`의 HTTP 서버: ✅ 허용됨 (URL 패턴과 일치)
- `https://api.internal.corp/mcp`의 HTTP 서버: ✅ 허용됨 (와일드카드 하위 도메인과 일치)
- `https://external.com/mcp`의 HTTP 서버: ❌ 차단됨 (URL 패턴과 일치하지 않음)

- 모든 명령의 Stdio 서버: ❌ 차단됨 (일치할 이름 또는 명령 항목 없음)

```
{
  "allowedMcpServers": [
    { "serverCommand": ["npx", "-y", "approved-package"] }
  ]
}
```

결과:

- `["npx", "-y", "approved-package"]` 를 사용한 Stdio 서버: ✅ 허용됨 (명령과 일치)
- `["node", "server.js"]` 를 사용한 Stdio 서버: ❌ 차단됨 (명령과 일치하지 않음)
- “my-api” 라는 이름의 HTTP 서버: ❌ 차단됨 (일치할 이름 항목 없음)

```
{
  "allowedMcpServers": [
    { "serverName": "github" },
    { "serverCommand": ["npx", "-y", "approved-package"] }
  ]
}
```

결과:

- `["npx", "-y", "approved-package"]` 를 사용한 “local-tool”이라는 Stdio 서버: ✅ 허용됨 (명령과 일치)
- `["node", "server.js"]` 를 사용한 “local-tool”이라는 Stdio 서버: ❌ 차단됨 (명령 항목이 있지만 일치하지 않음)
- `["node", "server.js"]` 를 사용한 “github”라는 Stdio 서버: ❌ 차단됨 (명령 항목이 있을 때 stdio 서버는 명령과 일치해야 함)
- “github” 라는 이름의 HTTP 서버: ✅ 허용됨 (이름과 일치)
- “other-api” 라는 이름의 HTTP 서버: ❌ 차단됨 (이름과 일치하지 않음)

```
{
  "allowedMcpServers": [
    { "serverName": "github" },
    { "serverName": "internal-tool" }
  ]
}
```

결과:

- 모든 명령을 사용한 “github”라는 Stdio 서버: ✅ 허용됨 (명령 제한 없음)
- 모든 명령을 사용한 “internal-tool”이라는 Stdio 서버: ✅ 허용됨 (명령 제한 없음)
- “github” 라는 이름의 HTTP 서버: ✅ 허용됨 (이름과 일치)
- “other” 라는 이름의 모든 서버: ❌ 차단됨 (이름과 일치하지 않음)

허용 목록 동작 (`allowedMcpServers`)

- `undefined` (기본값): 제한 없음 - 사용자는 모든 MCP 서버를 구성할 수 있습니다
- 빈 배열 `[]` : 완전한 잠금 - 사용자는 MCP 서버를 구성할 수 없습니다
- 항목 목록: 사용자는 이름, 명령 또는 URL 패턴과 일치하는 서버만 구성할 수 있습니다

거부 목록 동작 (`deniedMcpServers`)

- `undefined` (기본값): 차단된 서버 없음
- 빈 배열 `[]` : 차단된 서버 없음
- 항목 목록: 지정된 서버는 모든 범위에서 명시적으로 차단됩니다

중요한 참고 사항

- **옵션 1과 옵션 2를 결합할 수 있습니다**: `managed-mcp.json` 이 존재하면 독점 제어를 가지며 사용자는 서버를 추가할 수 없습니다. 허용 목록/거부 목록은 여전히 관리되는 서버 자체에 적용됩니다.
- **거부 목록이 절대 우선순위를 갖습니다**: 서버가 거부 목록 항목과 일치하면 (이름, 명령 또는 URL로) 허용 목록에 있어도 차단됩니다
- 이름 기반, 명령 기반 및 URL 기반 제한이 함께 작동합니다: 서버는 이름 항목, 명령 항목 또는 URL 패턴과 일치하면 통과합니다 (거부 목록으로 차단되지 않는 한)

> **Note:**

> **`managed-mcp.json` 사용 시**: 사용자는 `claude mcp add` 또는 구성 파일을 통해 MCP 서버를 추가할 수 없습니다. `allowedMcpServers` 및 `deniedMcpServers` 설정은 여전히 실제로 로드되는 관리되는 서버를 필터링하기 위해 적용됩니다.

Claude를 skills로 확장하기

> Claude Code에서 skills를 생성, 관리, 공유하여 Claude의 기능을 확장합니다. 사용자 정의 명령어와 번들 skills를 포함합니다.

Skills는 Claude가 할 수 있는 작업을 확장합니다. `SKILL.md` 파일을 지침과 함께 생성하면 Claude가 이를 자신의 도구 모음에 추가합니다. Claude는 관련이 있을 때 skills를 사용하거나 `/skill-name` 으로 직접 호출할 수 있습니다.

> **Note:**
>
> `/help` 및 `/compact` 와 같은 기본 제공 명령어는 대화형 모드를 참조하세요.
>
> **사용자 정의 명령어가 skills로 병합되었습니다.** `.claude/commands/deploy.md` 의 파일과 `.claude/skills/deploy/SKILL.md` 의 skill은 모두 `/deploy` 를 생성하고 동일하게 작동합니다. 기존 `.claude/commands/` 파일은 계속 작동합니다. Skills는 선택적 기능을 추가합니다: 지원 파일을 위한 디렉토리, skill 호출을 제어하기 위한 frontmatter, 그리고 Claude가 관련이 있을 때 자동으로 로드할 수 있는 기능입니다.

Claude Code skills는 여러 AI 도구에서 작동하는 Agent Skills 개방형 표준을 따릅니다. Claude Code는 호출 제어, subagent 실행, 동적 컨텍스트 주입과 같은 추가 기능으로 표준을 확장합니다.

번들 skills

번들 skills는 Claude Code와 함께 제공되며 모든 세션에서 사용 가능합니다. 고정 로직을 직접 실행하는 기본 제공 명령어와 달리 번들 skills는 프롬프트 기반입니다: Claude에 상세한 플레이북을 제공하고 도구를 사용하여 작업을 조율하도록 합니다. 이는 번들 skills가 병렬 에이전트를 생성하고, 파일을 읽고, 코드베이스에 적응할 수 있음을 의미합니다.

번들 skills는 다른 skill과 동일한 방식으로 호출합니다: `/` 다음에 skill 이름을 입력합니다.

- **`/simplify`**: 최근에 변경된 파일을 코드 재사용, 품질, 효율성 문제에 대해 검토한 다음 수정합니다. 기능이나 버그 수정을 구현한 후 실행하여 작업을 정리합니다. 3개의 검토 에이전트를 병렬로 생성하고(코드 재사용, 코드 품질, 효율성), 결과를 집계하고 수정을 적용합니다. 특정 관심사에 집중하도록 선택적 텍스트를 전달합니다: `/simplify focus on memory efficiency`.
- **`/batch <instruction>`**: 코드베이스 전체에서 대규모 변경을 병렬로 조율합니다. 변경 설명을 제공하면 `/batch` 는 코드베이스를 조사하고, 작업을 5~30개의 독립적인 단위로 분해하고,

승인을 위한 계획을 제시합니다. 승인되면 각 단위당 하나의 백그라운드 에이전트를 생성하고, 각각은 격리된 git worktree에 있습니다. 각 에이전트는 자신의 단위를 구현하고, 테스트를 실행하고, pull request를 엽니다. Git 저장소가 필요합니다. 예: `/batch migrate src/ from Solid to React`.

- `/debug [description]`: 세션 디버그 로그를 읽어 현재 Claude Code 세션을 문제 해결합니다. 선택적으로 분석에 집중하도록 문제를 설명합니다.
- `/loop [interval] <prompt>`: 세션이 열려 있는 동안 프롬프트를 간격에 따라 반복적으로 실행합니다. Claude는 간격을 구문 분석하고, 반복 cron 작업을 예약하고, 주기를 확인합니다. 배포를 폴링하거나, PR을 감시하거나, 다른 skill을 주기적으로 다시 실행하는 데 유용합니다. 예: `/loop 5m check if the deploy finished`. 일정에 따라 프롬프트 실행을 참조하세요.
- `/claude-api`: 프로젝트의 언어(Python, TypeScript, Java, Go, Ruby, C#, PHP 또는 cURL)에 대한 Claude API 참조 자료와 Python 및 TypeScript에 대한 Agent SDK 참조를 로드합니다. 도구 사용, 스트리밍, 배치, 구조화된 출력 및 일반적인 함정을 다룹니다. 또한 코드가 `anthropic`, `@anthropic-ai/sdk` 또는 `claude_agent_sdk`를 가져올 때 자동으로 활성화됩니다.

시작하기

첫 번째 skill 생성

이 예제는 Claude에게 시각적 다이어그램과 유추를 사용하여 코드를 설명하도록 가르치는 skill을 생성합니다. 기본 frontmatter를 사용하므로 Claude는 코드가 어떻게 작동하는지 물어볼 때 자동으로 로드하거나 `/explain-code`로 직접 호출할 수 있습니다.

Step 1: skill 디렉토리 생성

개인 skills 폴더에 skill을 위한 디렉토리를 생성합니다. 개인 skills는 모든 프로젝트에서 사용 가능합니다.

```
mkdir -p ~/.claude/skills/explain-code
```

Step 2: SKILL.md 작성

모든 skill에는 두 부분이 있는 `SKILL.md` 파일이 필요합니다: Claude에게 skill을 사용할 시기를 알려주는 YAML frontmatter(`---` 마커 사이)와 skill이 호출될 때 Claude가 따르는 지침이 있는 markdown 콘텐츠입니다. `name` 필드는 `/slash-command`가 되고, `description`은 Claude가 자동으로 로드할 시기를 결정하는 데 도움이 됩니다.

`~/.claude/skills/explain-code/SKILL.md` 생성:

```
---
name: explain-code
description: Explains code with visual diagrams and analogies. Use when explaining
how code works, teaching about a codebase, or when the user asks "how does this
work?"
---

When explaining code, always include:

1. **Start with an analogy**: Compare the code to something from everyday life
2. **Draw a diagram**: Use ASCII art to show the flow, structure, or relationships
3. **Walk through the code**: Explain step-by-step what happens
4. **Highlight a gotcha**: What's a common mistake or misconception?

Keep explanations conversational. For complex concepts, use multiple analogies.
```

Step 3: skill 테스트

두 가지 방법으로 테스트할 수 있습니다:

Claude가 자동으로 호출하도록 하기 - 설명과 일치하는 항목을 물어봅니다:

```
How does this code work?
```

또는 skill 이름으로 직접 호출:

```
/explain-code src/auth/login.ts
```

어느 쪽이든 Claude는 설명에 유추와 ASCII 다이어그램을 포함해야 합니다.

Skills가 있는 위치

skill을 저장하는 위치는 누가 사용할 수 있는지를 결정합니다:

위치	경로	적용 대상
Enterprise	관리 설정 참조	조직의 모든 사용자
Personal	`~/.claude/skills/<skill-name>/SKILL.md`	모든 프로젝트

위치	경로	적용 대상
Project	`.claude/skills/<skill-name>/SKILL.md`	이 프로젝트만
Plugin	`<plugin>/skills/<skill-name>/SKILL.md`	플러그인이 활성화된 위치

Skills가 여러 수준에서 같은 이름을 공유할 때 더 높은 우선순위 위치가 우선합니다: enterprise > personal > project. Plugin skills는 `plugin-name:skill-name` 네임스페이스를 사용하므로 다른 수준과 충돌할 수 없습니다. `.claude/commands/` 에 파일이 있으면 동일하게 작동하지만, skill과 명령어가 같은 이름을 공유하면 skill이 우선합니다.

중첩된 디렉토리에서 자동 검색

하위 디렉토리의 파일로 작업할 때 Claude Code는 중첩된 `.claude/skills/` 디렉토리에서 skills를 자동으로 검색합니다. 예를 들어 `packages/frontend/` 의 파일을 편집하는 경우 Claude Code는 `packages/frontend/.claude/skills/` 에서도 skills를 찾습니다. 이는 패키지가 자신의 skills를 가진 monorepo 설정을 지원합니다.

각 skill은 `SKILL.md` 를 진입점으로 하는 디렉토리입니다:

```
my-skill/
├── SKILL.md           # Main instructions (required)
├── template.md        # Template for Claude to fill in
├── examples/
│   └── sample.md      # Example output showing expected format
└── scripts/
    └── validate.sh    # Script Claude can execute
```

`SKILL.md` 는 주요 지침을 포함하며 필수입니다. 다른 파일은 선택사항이며 더 강력한 skills를 구축할 수 있습니다: Claude가 채울 템플릿, 예상 형식을 보여주는 예제 출력, Claude가 실행할 수 있는 스크립트 또는 상세한 참조 문서입니다. `SKILL.md` 에서 지원 파일을 참조하여 Claude가 각 파일의 내용과 로드할 시기를 알 수 있도록 합니다. 자세한 내용은 지원 파일 추가를 참조하세요.

> **Note:**
>
> `.claude/commands/` 의 파일은 계속 작동하며 동일한 frontmatter를 지원합니다. Skills는 지원 파일과 같은 추가 기능을 지원하므로 권장됩니다.

추가 디렉토리의 Skills

`--add-dir`을 통해 추가된 디렉토리 내의 `.claude/skills/`에 정의된 Skills는 자동으로 로드되고 라이브 변경 감지에 의해 선택되므로 세션을 다시 시작하지 않고도 세션 중에 편집할 수 있습니다.

> **Note:**
>
> `--add-dir` 디렉토리의 CLAUDE.md 파일은 기본적으로 로드되지 않습니다. 로드하려면 `CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD=1`을 설정하세요. 추가 디렉토리에서 로드를 참조하세요.

Skills 구성

Skills는 `SKILL.md` 상단의 YAML frontmatter와 그 뒤의 markdown 콘텐츠를 통해 구성됩니다.

Skill 콘텐츠 유형

Skill 파일은 모든 지침을 포함할 수 있지만, 호출 방식을 생각하면 포함할 내용을 안내하는 데 도움이 됩니다:

참조 콘텐츠는 Claude가 현재 작업에 적용하는 지식을 추가합니다. 규칙, 패턴, 스타일 가이드, 도메인 지식입니다. 이 콘텐츠는 인라인으로 실행되므로 Claude가 대화 컨텍스트와 함께 사용할 수 있습니다.

```
---
name: api-conventions
description: API design patterns for this codebase
---

When writing API endpoints:
- Use RESTful naming conventions
- Return consistent error formats
- Include request validation
```

작업 콘텐츠는 배포, 커밋 또는 코드 생성과 같은 특정 작업에 대한 단계별 지침을 제공합니다. 이는 Claude가 자동으로 실행하도록 하기보다는 `/skill-name`으로 직접 호출하려는 작업입니다. Claude가 자동으로 트리거하는 것을 방지하려면 `disable-model-invocation: true`를 추가합니다.

```
---
name: deploy
description: Deploy the application to production
context: fork
disable-model-invocation: true
---

Deploy the application:
1. Run the test suite
2. Build the application
3. Push to the deployment target
```

`SKILL.md` 는 모든 것을 포함할 수 있지만, skill을 호출하는 방식(사용자, Claude 또는 둘 다)과 실행 위치(인라인 또는 subagent)를 생각하면 포함할 내용을 안내하는 데 도움이 됩니다. 복잡한 skills의 경우 지원 파일을 추가하여 주요 skill을 집중적으로 유지할 수도 있습니다.

Frontmatter 참조

markdown 콘텐츠 외에도 `SKILL.md` 파일 상단의 `---` 마커 사이의 YAML frontmatter 필드를 사용하여 skill 동작을 구성할 수 있습니다:

```
---
name: my-skill
description: What this skill does
disable-model-invocation: true
allowed-tools: Read, Grep
---

Your skill instructions here...
```

모든 필드는 선택사항입니다. Claude가 skill을 사용할 시기를 알 수 있도록 `description` 만 권장됩니다.

필드	필수	설명
`name`	아니오	Skill의 표시 이름입니다. 생략하면 디렉토리 이름을 사용합니다. 소문자, 숫자 및 하이픈만 사용 가능(최대 64자).

필드	필수	설명
`description`	권장	Skill이 수행하는 작업과 사용 시기입니다. Claude는 이를 사용하여 skill을 적용할 시기를 결정합니다. 생략하면 markdown 콘텐츠의 첫 번째 단락을 사용합니다.
`argument-hint`	아니오	예상 인수를 나타내기 위해 자동 완성 중에 표시되는 힌트입니다. 예: `[issue-number]` 또는 `[filename] [format]`.
`disable-model-invocation`	아니오	Claude가 이 skill을 자동으로 로드하는 것을 방지하려면 `true`로 설정합니다. `/name`으로 수동으로 트리거하려는 워크플로우에 사용합니다. 기본값: `false`.
`user-invocable`	아니오	`/` 메뉴에서 숨기려면 `false`로 설정합니다. 사용자가 직접 호출하지 않아야 하는 배경 지식에 사용합니다. 기본값: `true`.
`allowed-tools`	아니오	이 skill이 활성화되었을 때 Claude가 권한을 요청하지 않고 사용할 수 있는 도구입니다.
`model`	아니오	이 skill이 활성화되었을 때 사용할 모델입니다.
`context`	아니오	포크된 subagent 컨텍스트에서 실행하려면 `fork`로 설정합니다.
`agent`	아니오	`context: fork`가 설정되었을 때 사용할 subagent 유형입니다.

필드	필수	설명
`hooks`	아니오	이 skill의 라이프사이클에 범위가 지정된 hooks입니다. 구성 형식은 Skills 및 agents의 Hooks를 참조하세요.

사용 가능한 문자열 치환

Skills는 skill 콘텐츠의 동적 값에 대한 문자열 치환을 지원합니다:

변수	설명
`$ARGUMENTS`	skill을 호출할 때 전달된 모든 인수입니다. `$ARGUMENTS` 가 콘텐츠에 없으면 인수가 `ARGUMENTS: <value>` 로 추가됩니다.
`$ARGUMENTS[N]`	0 기반 인덱스로 특정 인수에 액세스합니다(예: `$ARGUMENTS[0]` 은 첫 번째 인수).
`$N`	`$ARGUMENTS[N]` 의 약자입니다(예: `$0` 은 첫 번째 인수, `$1` 은 두 번째 인수).
`${CLAUDE_SESSION_ID}`	현재 세션 ID입니다. 로깅, 세션별 파일 생성 또는 skill 출력을 세션과 연관시키는 데 유용합니다.
`${CLAUDE_SKILL_DIR}`	Skill의 `SKILL.md` 파일을 포함하는 디렉토리입니다. Plugin skills의 경우 plugin 루트가 아닌 plugin 내의 skill 하위 디렉토리입니다. 현재 작업 디렉토리에 관계없이 skill과 함께 번들된 스크립트나 파일을 참조하려면 bash 주입 명령어에서 이를 사용합니다.

치환을 사용한 예제:

```
---
name: session-logger
description: Log activity for this session
---

Log the following to logs/${CLAUDE_SESSION_ID}.log:

$ARGUMENTS
```

지원 파일 추가

Skills는 디렉토리에 여러 파일을 포함할 수 있습니다. 이는 `SKILL.md`를 필수 항목에 집중하게 하면서 Claude가 필요할 때만 상세한 참조 자료에 액세스할 수 있게 합니다. 큰 참조 문서, API 사양 또는 예제 컬렉션은 skill이 실행될 때마다 컨텍스트에 로드될 필요가 없습니다.

```
my-skill/
├── SKILL.md (required - overview and navigation)
├── reference.md (detailed API docs - loaded when needed)
├── examples.md (usage examples - loaded when needed)
└── scripts/
    └── helper.py (utility script - executed, not loaded)
```

`SKILL.md`에서 지원 파일을 참조하여 Claude가 각 파일의 내용과 로드할 시기를 알 수 있도록 합니다:

```
### Additional resources

- For complete API details, see [reference.md](reference.md)
- For usage examples, see [examples.md](examples.md)
```

> **Tip:**
>
> `SKILL.md`를 500줄 이하로 유지합니다. 상세한 참조 자료를 별도 파일로 이동합니다.

Skill을 호출하는 사람 제어

기본적으로 사용자와 Claude 모두 모든 skill을 호출할 수 있습니다. `/skill-name`을 입력하여 직접 호출할 수 있고, Claude는 대화와 관련이 있을 때 자동으로 로드할 수 있습니다. 두 frontmatter 필드를 사용하면 이를 제한할 수 있습니다:

- **`disable-model-invocation: true`**: 사용자만 skill을 호출할 수 있습니다. `/commit`, `/deploy` 또는 `/send-slack-message`와 같이 부작용이 있거나 타이밍을 제어하려는 워크플로우에 사용합니다. Claude가 코드가 준비된 것처럼 보인다고 해서 배포하기로 결정하지 않기를 원합니다.
- **`user-invocable: false`**: Claude만 skill을 호출할 수 있습니다. 명령어로 실행할 수 없는 배경 지식에 사용합니다. `legacy-system-context` skill은 오래된 시스템이 어떻게 작동하는지 설명합니다. Claude는 관련이 있을 때 이를 알아야 하지만 `/legacy-system-context`는 사용자가 취할 의미 있는 작업이 아닙니다.

이 예제는 사용자만 트리거할 수 있는 배포 skill을 생성합니다. `disable-model-invocation: true` 필드는 Claude가 자동으로 실행하는 것을 방지합니다:

```
---
name: deploy
description: Deploy the application to production
disable-model-invocation: true
---

Deploy $ARGUMENTS to production:

1. Run the test suite
2. Build the application
3. Push to the deployment target
4. Verify the deployment succeeded
```

두 필드가 호출 및 컨텍스트 로딩에 미치는 영향은 다음과 같습니다:

Frontmatter	사용자가 호출 가능	Claude가 호출 가능	컨텍스트에 로드되는 시기
(기본값)	예	예	설명은 항상 컨텍스트에 있고, 호출될 때 전체 skill이 로드됨
`disable-model-invocation: true`	예	아니오	설명은 컨텍스트에 없고, 사용자가 호출할 때 전체 skill이 로드됨
`user-invocable: false`	아니오	예	설명은 항상 컨텍스트에 있고, 호출될 때 전체 skill이 로드됨

> **Note:**
>
> 일반 세션에서 skill 설명은 Claude가 사용 가능한 항목을 알 수 있도록 컨텍스트에 로드되지만 전체 skill 콘텐츠는 호출될 때만 로드됩니다. 사전 로드된 skills가 있는 Subagents는 다르게 작동합니다: 전체 skill 콘텐츠는 시작 시 주입됩니다.

도구 액세스 제한

`allowed-tools` 필드를 사용하여 skill이 활성화되었을 때 Claude가 사용할 수 있는 도구를 제한합니다. 이 skill은 Claude가 파일을 탐색할 수 있지만 수정할 수 없는 읽기 전용 모드를 생성합니다:

```
---
name: safe-reader
description: Read files without making changes
allowed-tools: Read, Grep, Glob
---
```

Skills에 인수 전달

사용자와 Claude 모두 skill을 호출할 때 인수를 전달할 수 있습니다. 인수는 `$ARGUMENTS` 자리 표시자를 통해 사용 가능합니다.

이 skill은 번호로 GitHub 문제를 수정합니다. `$ARGUMENTS` 자리 표시자는 skill 이름 뒤에 오는 모든 것으로 대체됩니다:

```
---
name: fix-issue
description: Fix a GitHub issue
disable-model-invocation: true
---

Fix GitHub issue $ARGUMENTS following our coding standards.

1. Read the issue description
2. Understand the requirements
3. Implement the fix
4. Write tests
5. Create a commit
```

`/fix-issue 123` 을 실행하면 Claude는 “Fix GitHub issue 123 following our coding standards…” 를 받습니다.

인수를 사용하여 skill을 호출하지만 skill에 `$ARGUMENTS` 가 포함되지 않으면 Claude Code는 `ARGUMENTS: <your input>` 을 skill 콘텐츠의 끝에 추가하므로 Claude는 여전히 입력한 내용을 봅니다.

위치별로 개별 인수에 액세스하려면 `$ARGUMENTS[N]` 또는 더 짧은 `$N`을 사용합니다:

```
---
name: migrate-component
description: Migrate a component from one framework to another
---

Migrate the $ARGUMENTS[0] component from $ARGUMENTS[1] to $ARGUMENTS[2].
Preserve all existing behavior and tests.
```

`/migrate-component SearchBar React Vue`를 실행하면 `$ARGUMENTS[0]`을 `SearchBar`로, `$ARGUMENTS[1]`을 `React`로, `$ARGUMENTS[2]`를 `Vue`로 대체합니다. `$N` 약자를 사용하는 동일한 skill:

```
---
name: migrate-component
description: Migrate a component from one framework to another
---

Migrate the $0 component from $1 to $2.
Preserve all existing behavior and tests.
```

고급 패턴

동적 컨텍스트 주입

`` !`command` `` 구문은 skill 콘텐츠가 Claude로 전송되기 전에 shell 명령어를 실행합니다. 명령어 출력이 자리 표시자를 대체하므로 Claude는 명령어 자체가 아닌 실제 데이터를 받습니다.

이 skill은 GitHub CLI를 사용하여 라이브 PR 데이터를 가져와 pull request를 요약합니다. `` !`gh pr diff` ``와 다른 명령어가 먼저 실행되고 출력이 프롬프트에 삽입됩니다:

```
---
name: pr-summary
description: Summarize changes in a pull request
context: fork
agent: Explore
allowed-tools: Bash(gh *)
---

### Pull request context
- PR diff: !`gh pr diff`
- PR comments: !`gh pr view --comments`
- Changed files: !`gh pr diff --name-only`

### Your task
Summarize this pull request...
```

이 skill이 실행될 때:

1. 각 `` !`command` ``가 즉시 실행됩니다(Claude가 보기 전에)
2. 출력이 skill 콘텐츠의 자리 표시자를 대체합니다
3. Claude는 실제 PR 데이터가 있는 완전히 렌더링된 프롬프트를 받습니다

이는 전처리이며 Claude가 실행하는 것이 아닙니다. Claude는 최종 결과만 봅니다.

> **Tip:**
> Skill에서 확장 사고를 활성화하려면 skill 콘텐츠의 어디든 "ultrathink"라는 단어를 포함합니다.

Subagent에서 Skills 실행

skill을 격리 상태에서 실행하려면 frontmatter에 `context: fork`를 추가합니다. Skill 콘텐츠는 subagent를 구동하는 프롬프트가 됩니다. 대화 기록에 액세스할 수 없습니다.

> **Warning:**
> `context: fork`는 명시적 지침이 있는 skills에만 의미가 있습니다. Skill에 작업 없이 "이 API 규칙을 사용하세요"와 같은 지침이 포함되어 있으면 subagent는 지침을 받지만 실행 가능한 프롬프트가 없으므로 의미 있는 출력 없이 반환됩니다.

Skills와 subagents는 두 방향으로 함께 작동합니다:

접근 방식	시스템 프롬프트	작업	또한 로드
`context: fork`가 있는Skill	에이전트 유형(`Explore`, `Plan` 등)에서	SKILL.md 콘텐츠	CLAUDE.md
`skills` 필드가 있는 Subagent	Subagent의 markdown 본문	Claude의 위임 메시지	사전 로드된 skills + CLAUDE.md

`context: fork`를 사용하면 skill에 작업을 작성하고 실행할 에이전트 유형을 선택합니다. 역방향(skills를 참조 자료로 사용하는 사용자 정의 subagent 정의)의 경우 Subagents를 참조하세요.

예제: Explore 에이전트를 사용하는 Research Skill

이 skill은 포크된 Explore 에이전트에서 연구를 실행합니다. Skill 콘텐츠는 작업이 되고 에이전트는 코드베이스 탐색에 최적화된 읽기 전용 도구를 제공합니다:

```
---
name: deep-research
description: Research a topic thoroughly
context: fork
agent: Explore
---

Research $ARGUMENTS thoroughly:

1. Find relevant files using Glob and Grep
2. Read and analyze the code
3. Summarize findings with specific file references
```

이 skill이 실행될 때:

1. 새로운 격리된 컨텍스트가 생성됩니다
2. Subagent는 skill 콘텐츠를 프롬프트로 받습니다(“Research $ARGUMENTS thoroughly…”)
3. `agent` 필드는 실행 환경을 결정합니다(모델, 도구 및 권한)
4. 결과가 요약되어 주 대화로 반환됩니다

`agent` 필드는 사용할 subagent 구성을 지정합니다. 옵션에는 기본 제공 에이전트(`Explore`, `Plan`, `general-purpose`) 또는 `.claude/agents/`의 모든 사용자 정의 subagent가 포함됩니다. 생략하면 `general-purpose`를 사용합니다.

Claude의 Skill 액세스 제한

기본적으로 Claude는 `disable-model-invocation: true`가 설정되지 않은 모든 skill을 호출할 수 있습니다. `allowed-tools`를 정의하는 Skills는 skill이 활성화되었을 때 사용자별 승인 없이 Claude에게 해당 도구에 대한 액세스 권한을 부여합니다. 권한 설정은 여전히 다른 모든 도구에 대한 기본 승인 동작을 관리합니다. `/compact` 및 `/init`과 같은 기본 제공 명령어는 Skill 도구를 통해 사용할 수 없습니다.

Claude가 호출할 수 있는 skills를 제어하는 세 가지 방법:

`/permissions`에서 Skill 도구를 거부하여 모든 skills 비활성화:

```
## Add to deny rules:
Skill
```

권한 규칙을 사용하여 특정 skills 허용 또는 거부:

```
## Allow only specific skills
Skill(commit)
Skill(review-pr *)

## Deny specific skills
Skill(deploy *)
```

권한 구문: 정확한 일치는 `Skill(name)`, 인수가 있는 접두사 일치는 `Skill(name *)`.

Frontmatter에 `disable-model-invocation: true`를 추가하여 개별 skills 숨기기. 이는 Claude의 컨텍스트에서 skill을 완전히 제거합니다.

> **Note:**
>
> `user-invocable` 필드는 메뉴 가시성만 제어하고 Skill 도구 액세스는 제어하지 않습니다. 프로그래밍 방식 호출을 차단하려면 `disable-model-invocation: true`를 사용합니다.

Skills 공유

Skills는 대상에 따라 다양한 범위에서 배포할 수 있습니다:

- **Project skills**: `.claude/skills/` 를 버전 제어에 커밋
- **Plugins**: plugin에서 `skills/` 디렉토리 생성
- **Managed**: 관리 설정을 통해 조직 전체에 배포

시각적 출력 생성

Skills는 모든 언어의 스크립트를 번들하고 실행할 수 있으므로 Claude에게 단일 프롬프트로 가능한 것 이상의 기능을 제공합니다. 강력한 패턴 중 하나는 시각적 출력을 생성하는 것입니다: 데이터 탐색, 디버깅 또는 보고서 생성을 위해 브라우저에서 열 수 있는 대화형 HTML 파일입니다.

이 예제는 코드베이스 탐색기를 생성합니다: 디렉토리를 확장 및 축소할 수 있는 대화형 트리 보기, 한눈에 파일 크기를 볼 수 있고, 색상으로 파일 유형을 식별할 수 있습니다.

Skill 디렉토리 생성:

```
mkdir -p ~/.claude/skills/codebase-visualizer/scripts
```

`~/.claude/skills/codebase-visualizer/SKILL.md` 생성. 설명은 Claude에게 이 Skill을 활성화할 시기를 알려주고, 지침은 Claude에게 번들 스크립트를 실행하도록 알려줍니다:

````
---
name: codebase-visualizer
description: Generate an interactive collapsible tree visualization of your
codebase. Use when exploring a new repo, understanding project structure, or
identifying large files.
allowed-tools: Bash(python *)
---

## Codebase Visualizer

Generate an interactive HTML tree view that shows your project's file structure
with collapsible directories.

### Usage

Run the visualization script from your project root:

```bash
python ~/.claude/skills/codebase-visualizer/scripts/visualize.py .
```

This creates `codebase-map.html` in the current directory and opens it in your
default browser.

### What the visualization shows

- **Collapsible directories**: Click folders to expand/collapse
- **File sizes**: Displayed next to each file
- **Colors**: Different colors for different file types
- **Directory totals**: Shows aggregate size of each folder
````

`~/.claude/skills/codebase-visualizer/scripts/visualize.py` 생성. 이 스크립트는 디렉토리 트리를 스캔하고 다음을 포함하는 자체 포함 HTML 파일을 생성합니다:

- **요약 사이드바** - 파일 수, 디렉토리 수, 총 크기 및 파일 유형 수 표시
- **막대 차트** - 파일 유형별로 코드베이스 분석(크기 기준 상위 8개)
- **축소 가능한 트리** - 디렉토리를 확장 및 축소할 수 있으며, 색상으로 구분된 파일 유형 표시기

스크립트는 Python이 필요하지만 기본 제공 라이브러리만 사용하므로 설치할 패키지가 없습니다:

```
#!/usr/bin/env python3
"""Generate an interactive collapsible tree visualization of a codebase."""

from pathlib import Path
from collections import Counter

IGNORE = {'.git', 'node_modules', '__pycache__', '.venv', 'venv', 'dist', 'build'}

def scan(path: Path, stats: dict) -> dict:
    result = {"name": path.name, "children": [], "size": 0}
    try:
        for item in sorted(path.iterdir()):
            if item.name in IGNORE or item.name.startswith('.'):
                continue
            if item.is_file():
                size = item.stat().st_size
                ext = item.suffix.lower() or '(no ext)'
                result["children"].append({"name": item.name, "size": size,
"ext": ext})
                result["size"] += size
                stats["files"] += 1
                stats["extensions"][ext] += 1
                stats["ext_sizes"][ext] += size
            elif item.is_dir():
                stats["dirs"] += 1
                child = scan(item, stats)
                if child["children"]:
                    result["children"].append(child)
                    result["size"] += child["size"]
    except PermissionError:
        pass
    return result

def generate_html(data: dict, stats: dict, output: Path) -> None:
    ext_sizes = stats["ext_sizes"]
    total_size = sum(ext_sizes.values()) or 1
    sorted_exts = sorted(ext_sizes.items(), key=lambda x: -x[1])[:8]
    colors = {
```

```
        '.js': '#f7df1e', '.ts': '#3178c6', '.py': '#3776ab', '.go': '#00add8',
        '.rs': '#dea584', '.rb': '#cc342d', '.css': '#264de4', '.html': '#e34c26',
        '.json': '#6b7280', '.md': '#083fa1', '.yaml': '#cb171e', '.yml': '#cb171e
',
        '.mdx': '#083fa1', '.tsx': '#3178c6', '.jsx': '#61dafb', '.sh': '#4eaa25',
    }
    lang_bars = "".join(
        f'<div class="bar-row"><span class="bar-label">{ext}</span>'
        f'<div class="bar" style="width:{(size/total_size)*100}%;background:{color
s.get(ext,"#6b7280")}"></div>'
        f'<span class="bar-pct">{(size/total_size)*100:.1f}%</span></div>'
        for ext, size in sorted_exts
    )
    def fmt(b):
        if b < 1024: return f"{b} B"
        if b < 1048576: return f"{b/1024:.1f} KB"
        return f"{b/1048576:.1f} MB"

    html = f'''<!DOCTYPE html>
<html><head>
  <meta charset="utf-8"><title>Codebase Explorer</title>
  <style>
    body {{ font: 14px/1.5 system-ui, sans-serif; margin: 0; background: #1a1a2e;
color: #eee; }}
    .container {{ display: flex; height: 100vh; }}
    .sidebar {{ width: 280px; background: #252542; padding: 20px; border-right:
1px solid #3d3d5c; overflow-y: auto; flex-shrink: 0; }}
    .main {{ flex: 1; padding: 20px; overflow-y: auto; }}
    h1 {{ margin: 0 0 10px 0; font-size: 18px; }}
    h2 {{ margin: 20px 0 10px 0; font-size: 14px; color: #888; text-transform:
uppercase; }}
    .stat {{ display: flex; justify-content: space-between; padding: 8px 0;
border-bottom: 1px solid #3d3d5c; }}
    .stat-value {{ font-weight: bold; }}
    .bar-row {{ display: flex; align-items: center; margin: 6px 0; }}
    .bar-label {{ width: 55px; font-size: 12px; color: #aaa; }}
    .bar {{ height: 18px; border-radius: 3px; }}
    .bar-pct {{ margin-left: 8px; font-size: 12px; color: #666; }}
    .tree {{ list-style: none; padding-left: 20px; }}
```

```
    details {{ cursor: pointer; }}
    summary {{ padding: 4px 8px; border-radius: 4px; }}
    summary:hover {{ background: #2d2d44; }}
    .folder {{ color: #ffd700; }}
    .file {{ display: flex; align-items: center; padding: 4px 8px; border-radius:
4px; }}
    .file:hover {{ background: #2d2d44; }}
    .size {{ color: #888; margin-left: auto; font-size: 12px; }}
    .dot {{ width: 8px; height: 8px; border-radius: 50%; margin-right: 8px; }}
  </style>
</head><body>
  <div class="container">
    <div class="sidebar">
      <h1>📊 Summary</h1>
      <div class="stat"><span>Files</span><span class="stat-
value">{stats["files"]:,}</span></div>
      <div class="stat"><span>Directories</span><span class="stat-value">{stats["d
irs"]:,}</span></div>
      <div class="stat"><span>Total size</span><span class="stat-
value">{fmt(data["size"])}</span></div>
      <div class="stat"><span>File types</span><span class="stat-value">{len(stats
["extensions"])}</span></div>
      <h2>By file type</h2>
      {lang_bars}
    </div>
    <div class="main">
      <h1>📁 {data["name"]}</h1>
      <ul class="tree" id="root"></ul>
    </div>
  </div>
  <script>
    const data = {json.dumps(data)};
    const colors = {json.dumps(colors)};
    function fmt(b) {{ if (b < 1024) return b + ' B'; if (b < 1048576) return (b/
1024).toFixed(1) + ' KB'; return (b/1048576).toFixed(1) + ' MB'; }}
    function render(node, parent) {{
      if (node.children) {{
        const det = document.createElement('details');
        det.open = parent === document.getElementById('root');
```

```
        det.innerHTML = `<summary><span class="folder">📁 ${{node.name}}</
span><span class="size">${{fmt(node.size)}}</span></summary>`;
        const ul = document.createElement('ul'); ul.className = 'tree';
        node.children.sort((a,b) => (b.children?1:0)-(a.children?1:0) ||
a.name.localeCompare(b.name));
        node.children.forEach(c => render(c, ul));
        det.appendChild(ul);
        const li = document.createElement('li'); li.appendChild(det);
parent.appendChild(li);
      }} else {{
        const li = document.createElement('li'); li.className = 'file';
        li.innerHTML = `<span class="dot" style="background:${{colors[node.ext]||'
#6b7280'}}"></span>${{node.name}}<span class="size">${{fmt(node.size)}}</span>`;
        parent.appendChild(li);
      }}
    }}
    data.children.forEach(c => render(c, document.getElementById('root')));
  </script>
</body></html>'''
    output.write_text(html)

if __name__ == '__main__':
    target = Path(sys.argv[1] if len(sys.argv) > 1 else '.').resolve()
    stats = {"files": 0, "dirs": 0, "extensions": Counter(), "ext_sizes":
Counter()}
    data = scan(target, stats)
    out = Path('codebase-map.html')
    generate_html(data, stats, out)
    print(f'Generated {out.absolute()}')
    webbrowser.open(f'file://{out.absolute()}')
```

테스트하려면 모든 프로젝트에서 Claude Code를 열고 “Visualize this codebase”를 요청합니다. Claude는 스크립트를 실행하고, `codebase-map.html` 을 생성하고, 브라우저에서 엽니다.

이 패턴은 모든 시각적 출력에 작동합니다: 종속성 그래프, 테스트 커버리지 보고서, API 문서 또는 데이터베이스 스키마 시각화입니다. 번들 스크립트가 무거운 작업을 수행하는 동안 Claude는 조율을 처리합니다.

문제 해결

Skill이 트리거되지 않음

Claude가 예상대로 skill을 사용하지 않는 경우:

1. 설명에 사용자가 자연스럽게 말할 키워드가 포함되어 있는지 확인합니다
2. Skill이 `What skills are available?`에 나타나는지 확인합니다
3. 설명과 더 가깝게 일치하도록 요청을 다시 표현해 봅니다
4. Skill이 사용자 호출 가능하면 `/skill-name`으로 직접 호출합니다

Skill이 너무 자주 트리거됨

Claude가 원하지 않을 때 skill을 사용하는 경우:

1. 설명을 더 구체적으로 만듭니다
2. 수동 호출만 원하면 `disable-model-invocation: true`를 추가합니다

Claude가 모든 skills를 보지 못함

Skill 설명은 Claude가 사용 가능한 항목을 알 수 있도록 컨텍스트에 로드됩니다. 많은 skills가 있으면 문자 예산을 초과할 수 있습니다. 예산은 컨텍스트 윈도우의 2%에서 동적으로 확장되며, 16,000자의 폴백이 있습니다. `/context`를 실행하여 제외된 skills에 대한 경고를 확인합니다.

제한을 재정의하려면 `SLASH_COMMAND_TOOL_CHAR_BUDGET` 환경 변수를 설정합니다.

관련 리소스

- **Subagents**: 특화된 에이전트에 작업 위임
- **Plugins**: 다른 확장과 함께 skills 패키징 및 배포
- **Hooks**: 도구 이벤트 주변 워크플로우 자동화
- **Memory**: 지속적인 컨텍스트를 위한 CLAUDE.md 파일 관리
- **Interactive mode**: 기본 제공 명령어 및 바로가기
- **Permissions**: 도구 및 skill 액세스 제어

플러그인 만들기

> skills, agents, hooks, MCP servers를 사용하여 Claude Code를 확장하는 사용자 정의 플러그인을 만듭니다.

플러그인을 사용하면 프로젝트와 팀 전체에서 공유할 수 있는 사용자 정의 기능으로 Claude Code를 확장할 수 있습니다. 이 가이드에서는 skills, agents, hooks, MCP servers를 사용하여 자신의 플러그인을 만드는 방법을 다룹니다.

기존 플러그인을 설치하려고 하시나요? 플러그인 발견 및 설치를 참조하세요. 완전한 기술 사양은 플러그인 참조를 참조하세요.

플러그인 대 독립 실행형 구성 사용 시기

Claude Code는 사용자 정의 skills, agents, hooks를 추가하는 두 가지 방법을 지원합니다:

접근 방식	Skill 이름	최적 용도
독립 실행형 (`.claude/` 디렉토리)	`/hello`	개인 워크플로우, 프로젝트별 사용자 정의, 빠른 실험
플러그인 (`.claude-plugin/plugin.json`이 있는 디렉토리)	`/plugin-name:hello`	팀원과 공유, 커뮤니티에 배포, 버전 관리 릴리스, 프로젝트 간 재사용

다음의 경우 독립 실행형 구성을 사용하세요:

- 단일 프로젝트에 대해 Claude Code를 사용자 정의하는 경우
- 구성이 개인적이며 공유할 필요가 없는 경우
- skills 또는 hooks를 패키징하기 전에 실험하는 경우
- `/hello` 또는 `/deploy`와 같은 짧은 skill 이름을 원하는 경우

다음의 경우 플러그인을 사용하세요:

- 팀 또는 커뮤니티와 기능을 공유하려는 경우
- 여러 프로젝트에서 동일한 skills/agents가 필요한 경우
- 확장 기능에 대한 버전 제어 및 쉬운 업데이트를 원하는 경우
- 마켓플레이스를 통해 배포하는 경우
- `/my-plugin:hello`와 같은 네임스페이스 skills를 사용해도 괜찮은 경우 (네임스페이싱은 플러그인 간 충돌을 방지합니다)

> **Tip:**
>
> 빠른 반복을 위해 `.claude/` 의 독립 실행형 구성으로 시작한 다음, 공유할 준비가 되면 기존 구성을 플러그인으로 변환하세요.

빠른 시작

이 빠른 시작은 사용자 정의 skill을 사용하여 플러그인을 만드는 과정을 안내합니다. 매니페스트(플러그인을 정의하는 구성 파일)를 만들고, skill을 추가하고, `--plugin-dir` 플래그를 사용하여 로컬에서 테스트합니다.

필수 조건

- Claude Code 설치 및 인증
- Claude Code 버전 1.0.33 이상 (`claude --version` 을 실행하여 확인)

> **Note:**
>
> `/plugin` 명령이 보이지 않으면 Claude Code를 최신 버전으로 업데이트하세요. 업그레이드 지침은 문제 해결을 참조하세요.

첫 번째 플러그인 만들기

Step 1: 플러그인 디렉토리 만들기

모든 플러그인은 매니페스트와 skills, agents 또는 hooks를 포함하는 자체 디렉토리에 있습니다. 지금 만들어보세요:

```
mkdir my-first-plugin
```

Step 2: 플러그인 매니페스트 만들기

`.claude-plugin/plugin.json` 의 매니페스트 파일은 플러그인의 정체성을 정의합니다: 이름, 설명, 버전. Claude Code는 이 메타데이터를 사용하여 플러그인 관리자에서 플러그인을 표시합니다.

플러그인 폴더 내에 `.claude-plugin` 디렉토리를 만듭니다:

```
mkdir my-first-plugin/.claude-plugin
```

그런 다음 다음 내용으로 `my-first-plugin/.claude-plugin/plugin.json` 을 만듭니다:

```
{
"name": "my-first-plugin",
"description": "A greeting plugin to learn the basics",
"version": "1.0.0",
"author": {
"name": "Your Name"
}
}
```

필드	목적
`name`	고유 식별자 및 skill 네임스페이스. Skills는 이것으로 접두사가 붙습니다 (예: `/my-first-plugin:hello`).
`description`	플러그인을 검색하거나 설치할 때 플러그인 관리자에 표시됩니다.
`version`	의미 있는 버전 관리를 사용하여 릴리스를 추적합니다.
`author`	선택 사항. 속성에 유용합니다.

`homepage` , `repository` , `license` 와 같은 추가 필드는 전체 매니페스트 스키마를 참조하세요.

Step 3: Skill 추가

Skills는 `skills/` 디렉토리에 있습니다. 각 skill은 `SKILL.md` 파일을 포함하는 폴더입니다. 폴더 이름은 skill 이름이 되며, 플러그인의 네임스페이스가 접두사로 붙습니다 (`my-first-plugin` 이라는 플러그인의 `hello/` 는 `/my-first-plugin:hello` 를 만듭니다).

플러그인 폴더에 skill 디렉토리를 만듭니다:

```
mkdir -p my-first-plugin/skills/hello
```

그런 다음 다음 내용으로 `my-first-plugin/skills/hello/SKILL.md` 를 만듭니다:

```
---
description: Greet the user with a friendly message
disable-model-invocation: true
---

Greet the user warmly and ask how you can help them today.
```

Step 4: 플러그인 테스트

`--plugin-dir` 플래그를 사용하여 Claude Code를 실행하여 플러그인을 로드합니다:

```
claude --plugin-dir ./my-first-plugin
```

Claude Code가 시작되면 새 skill을 시도해보세요:

```
/my-first-plugin:hello
```

Claude가 인사말로 응답하는 것을 볼 수 있습니다. `/help` 를 실행하여 플러그인 네임스페이스 아래에 나열된 skill을 확인하세요.

> **Note:**
>
> **네임스페이싱이 필요한 이유?** 플러그인 skills는 항상 네임스페이스가 지정됩니다 (예: `/greet:hello`). 여러 플러그인이 동일한 이름의 skills를 가질 때 충돌을 방지합니다.
>
> 네임스페이스 접두사를 변경하려면 `plugin.json` 의 `name` 필드를 업데이트하세요.

Step 5: Skill 인수 추가

사용자 입력을 수락하여 skill을 동적으로 만듭니다. `$ARGUMENTS` 자리 표시자는 사용자가 skill 이름 뒤에 제공하는 모든 텍스트를 캡처합니다.

`SKILL.md` 파일을 업데이트합니다:

```
---
description: Greet the user with a personalized message
---

## Hello Skill

Greet the user named "$ARGUMENTS" warmly and ask how you can help them today. Make
the greeting personal and encouraging.
```

`/reload-plugins` 를 실행하여 변경 사항을 적용한 다음 이름으로 skill을 시도해보세요:

```
/my-first-plugin:hello Alex
```

Claude가 이름으로 인사할 것입니다. skills에 인수를 전달하는 방법에 대한 자세한 내용은 Skills 를 참조하세요.

다음과 같은 주요 구성 요소로 플러그인을 성공적으로 만들고 테스트했습니다:

- **플러그인 매니페스트** (`.claude-plugin/plugin.json`): 플러그인의 메타데이터를 설명합니다
- **Skills 디렉토리** (`skills/`): 사용자 정의 skills를 포함합니다
- **Skill 인수** (`$ARGUMENTS`): 동적 동작을 위해 사용자 입력을 캡처합니다

> **Tip:**
>
> `--plugin-dir` 플래그는 개발 및 테스트에 유용합니다. 플러그인을 다른 사람과 공유할 준비가 되면 플러그인 마켓플레이스 만들기 및 배포를 참조하세요.

플러그인 구조 개요

skills를 사용하여 플러그인을 만들었지만, 플러그인에는 훨씬 더 많은 것이 포함될 수 있습니다: 사용자 정의 agents, hooks, MCP servers, LSP servers.

> **Warning:**
>
> **일반적인 실수**: `commands/` , `agents/` , `skills/` , `hooks/` 를 `.claude-plugin/` 디렉토리 내에 넣지 마세요. `.claude-plugin/` 내에는 `plugin.json` 만 들어갑니다. 다른 모든 디렉토리는 플러그인 루트 수준에 있어야 합니다.

디렉토리	위치	목적
`.claude-plugin/`	플러그인 루트	`plugin.json` 매니페스트를 포함합니다 (구성 요소가 기본 위치를 사용하는 경우 선택 사항)
`commands/`	플러그인 루트	Markdown 파일로서의 Skills
`agents/`	플러그인 루트	사용자 정의 agent 정의
`skills/`	플러그인 루트	`SKILL.md` 파일이 있는 Agent Skills
`hooks/`	플러그인 루트	`hooks.json`의 이벤트 핸들러
`.mcp.json`	플러그인 루트	MCP server 구성
`.lsp.json`	플러그인 루트	코드 인텔리전스를 위한 LSP server 구성
`settings.json`	플러그인 루트	플러그인이 활성화될 때 적용되는 기본 설정

Note:

다음 단계: 더 많은 기능을 추가할 준비가 되셨나요? 더 복잡한 플러그인 개발로 이동하여 agents, hooks, MCP servers, LSP servers를 추가하세요. 모든 플러그인 구성 요소의 완전한 기술 사양은 플러그인 참조를 참조하세요.

더 복잡한 플러그인 개발

기본 플러그인에 익숙해지면 더 정교한 확장 기능을 만들 수 있습니다.

플러그인에 Skills 추가

플러그인은 Claude의 기능을 확장하기 위해 Agent Skills를 포함할 수 있습니다. Skills는 모델 호출입니다: Claude는 작업 컨텍스트에 따라 자동으로 사용합니다.

플러그인 루트에 `SKILL.md` 파일을 포함하는 Skill 폴더가 있는 `skills/` 디렉토리를 추가합니다:

```
my-plugin/
├── .claude-plugin/
│   └── plugin.json
└── skills/
    └── code-review/
        └── SKILL.md
```

각 `SKILL.md` 는 `name` 및 `description` 필드가 있는 프론트매터와 그 뒤에 지침이 필요합니다:

```
---
name: code-review
description: Reviews code for best practices and potential issues. Use when
reviewing code, checking PRs, or analyzing code quality.
---

When reviewing code, check for:
1. Code organization and structure
2. Error handling
3. Security concerns
4. Test coverage
```

플러그인을 설치한 후 `/reload-plugins` 를 실행하여 Skills를 로드합니다. 점진적 공개 및 도구 제한을 포함한 완전한 Skill 작성 지침은 Agent Skills를 참조하세요.

플러그인에 LSP servers 추가

> **Tip:**
>
> TypeScript, Python, Rust와 같은 일반적인 언어의 경우 공식 마켓플레이스에서 미리 빌드된 LSP 플러그인을 설치하세요. 이미 다루어진 언어가 아닌 언어에 대한 지원이 필요한 경우에만 사용자 정의 LSP 플러그인을 만드세요.

LSP (Language Server Protocol) 플러그인은 Claude에 실시간 코드 인텔리전스를 제공합니다. 아직 공식 LSP 플러그인이 없는 언어를 지원해야 하는 경우 플러그인에 `.lsp.json` 파일을 추가하여 자신의 플러그인을 만들 수 있습니다:

```
{
  "go": {
    "command": "gopls",
    "args": ["serve"],
    "extensionToLanguage": {
      ".go": "go"
    }
  }
}
```

플러그인을 설치하는 사용자는 자신의 머신에 언어 server 바이너리를 설치해야 합니다.

완전한 LSP 구성 옵션은 LSP servers를 참조하세요.

플러그인과 함께 기본 설정 제공

플러그인은 플러그인 루트에 `settings.json` 파일을 포함하여 플러그인이 활성화될 때 기본 구성을 적용할 수 있습니다. 현재 `agent` 키만 지원됩니다.

`agent` 를 설정하면 플러그인의 사용자 정의 agents 중 하나를 주 스레드로 활성화하여 시스템 프롬프트, 도구 제한, 모델을 적용합니다. 이를 통해 플러그인은 활성화될 때 Claude Code의 동작 방식을 기본적으로 변경할 수 있습니다.

```
{
  "agent": "security-reviewer"
}
```

이 예제는 플러그인의 `agents/` 디렉토리에 정의된 `security-reviewer` agent를 활성화합니다. `settings.json` 의 설정은 `plugin.json` 에 선언된 `settings` 보다 우선합니다. 알 수 없는 키는 자동으로 무시됩니다.

복잡한 플러그인 구성

많은 구성 요소가 있는 플러그인의 경우 기능별로 디렉토리 구조를 구성합니다. 완전한 디렉토리 레이아웃 및 구성 패턴은 플러그인 디렉토리 구조를 참조하세요.

플러그인을 로컬에서 테스트

개발 중에 플러그인을 테스트하려면 `--plugin-dir` 플래그를 사용합니다. 이는 설치를 요구하지 않고 플러그인을 직접 로드합니다.

```
claude --plugin-dir ./my-plugin
```

플러그인을 변경할 때 `/reload-plugins` 를 실행하여 다시 시작하지 않고 업데이트를 적용합니다. LSP server 구성 변경은 여전히 전체 다시 시작이 필요합니다. 플러그인 구성 요소를 테스트합니다:

- `/plugin-name:skill-name` 으로 skills를 시도해보세요
- agents가 `/agents` 에 나타나는지 확인하세요
- hooks가 예상대로 작동하는지 확인하세요

> **Tip:**
>
> 플래그를 여러 번 지정하여 한 번에 여러 플러그인을 로드할 수 있습니다:
>
> ```
> claude --plugin-dir ./plugin-one --plugin-dir ./plugin-two
> ```

플러그인 문제 디버깅

플러그인이 예상대로 작동하지 않는 경우:

1. **구조 확인**: 디렉토리가 `.claude-plugin/` 내부가 아닌 플러그인 루트에 있는지 확인하세요
2. **구성 요소를 개별적으로 테스트**: 각 명령, agent, hook을 별도로 확인하세요
3. **검증 및 디버깅 도구 사용**: CLI 명령 및 문제 해결 기법은 디버깅 및 개발 도구를 참조하세요

플러그인 공유

플러그인을 공유할 준비가 되면:

1. **문서 추가**: 설치 및 사용 지침이 포함된 `README.md` 를 포함하세요
2. **플러그인 버전 관리**: `plugin.json` 에서 의미 있는 버전 관리를 사용하세요
3. **마켓플레이스 만들기 또는 사용**: 플러그인 마켓플레이스를 통해 배포하여 설치하세요
4. **다른 사람과 테스트**: 더 광범위한 배포 전에 팀원이 플러그인을 테스트하도록 하세요

플러그인이 마켓플레이스에 있으면 다른 사람들이 플러그인 발견 및 설치의 지침을 사용하여 설치할 수 있습니다.

플러그인을 공식 마켓플레이스에 제출

플러그인을 공식 Anthropic 마켓플레이스에 제출하려면 다음 앱 내 제출 양식 중 하나를 사용하세요:

- **Claude.ai**: claude.ai/settings/plugins/submit
- **Console**: platform.claude.com/plugins/submit

> **Note:**
>
> 완전한 기술 사양, 디버깅 기법, 배포 전략은 플러그인 참조를 참조하세요.

기존 구성을 플러그인으로 변환

`.claude/` 디렉토리에 이미 skills 또는 hooks가 있는 경우 더 쉬운 공유 및 배포를 위해 플러그인으로 변환할 수 있습니다.

마이그레이션 단계

Step 1: 플러그인 구조 만들기

새 플러그인 디렉토리를 만듭니다:

```
mkdir -p my-plugin/.claude-plugin
```

`my-plugin/.claude-plugin/plugin.json`에 매니페스트 파일을 만듭니다:

```
{
  "name": "my-plugin",
  "description": "Migrated from standalone configuration",
  "version": "1.0.0"
}
```

Step 2: 기존 파일 복사

기존 구성을 플러그인 디렉토리에 복사합니다:

```
## Copy commands
cp -r .claude/commands my-plugin/

## Copy agents (if any)
cp -r .claude/agents my-plugin/

## Copy skills (if any)
cp -r .claude/skills my-plugin/
```

Step 3: Hooks 마이그레이션

설정에 hooks가 있는 경우 hooks 디렉토리를 만듭니다:

```
mkdir my-plugin/hooks
```

`my-plugin/hooks/hooks.json`을 hooks 구성으로 만듭니다. `.claude/settings.json` 또는 `settings.local.json`에서 `hooks` 객체를 복사합니다. 형식이 동일하기 때문입니다. 명령은 stdin에서 JSON으로 hook 입력을 받으므로 `jq`를 사용하여 파일 경로를 추출합니다:

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [{ "type": "command", "command":
"jq -r '.tool_input.file_path' | xargs npm run lint:fix" }]
      }
    ]
  }
}
```

Step 4: 마이그레이션된 플러그인 테스트

플러그인을 로드하여 모든 것이 작동하는지 확인합니다:

```
claude --plugin-dir ./my-plugin
```

각 구성 요소를 테스트합니다: 명령을 실행하고, agents가 `/agents`에 나타나는지 확인하고, hooks가 올바르게 트리거되는지 확인합니다.

마이그레이션 시 변경되는 사항

독립 실행형 (`.claude/`)	플러그인
한 프로젝트에서만 사용 가능	마켓플레이스를 통해 공유 가능
`.claude/commands/` 의 파일	`plugin-name/commands/` 의 파일
`settings.json` 의 Hooks	`hooks/hooks.json` 의 Hooks
공유하려면 수동으로 복사해야 함	`/plugin install` 로 설치

> **Note:**
>
> 마이그레이션 후 중복을 피하기 위해 `.claude/` 에서 원본 파일을 제거할 수 있습니다. 플러그인 버전이 로드될 때 우선합니다.

다음 단계

이제 Claude Code의 플러그인 시스템을 이해했으므로 다양한 목표에 대한 제안된 경로는 다음과 같습니다:

플러그인 사용자의 경우

- 플러그인 발견 및 설치: 마켓플레이스를 검색하고 플러그인을 설치합니다
- 팀 마켓플레이스 구성: 팀을 위한 저장소 수준 플러그인을 설정합니다

플러그인 개발자의 경우

- 마켓플레이스 만들기 및 배포: 플러그인을 패키징하고 공유합니다
- 플러그인 참조: 완전한 기술 사양
- 특정 플러그인 구성 요소에 대해 더 깊이 있게 살펴보세요:
  - Skills: skill 개발 세부 사항
  - Subagents: agent 구성 및 기능
  - Hooks: 이벤트 처리 및 자동화
  - MCP: 외부 도구 통합

플러그인 참조

> Claude Code 플러그인 시스템의 완전한 기술 참조, 스키마, CLI 명령어 및 컴포넌트 사양 포함.

Tip:

플러그인을 설치하려고 하시나요? 플러그인 발견 및 설치를 참조하세요. 플러그인 생성에 대해서는 플러그인을 참조하세요. 플러그인 배포에 대해서는 플러그인 마켓플레이스를 참조하세요.

이 참조는 Claude Code 플러그인 시스템의 완전한 기술 사양을 제공하며, 컴포넌트 스키마, CLI 명령어 및 개발 도구를 포함합니다.

플러그인은 Claude Code를 사용자 정의 기능으로 확장하는 자체 포함된 컴포넌트 디렉토리입니다. 플러그인 컴포넌트에는 skills, agents, hooks, MCP servers 및 LSP servers가 포함됩니다.

플러그인 컴포넌트 참조

Skills

플러그인은 Claude Code에 skills를 추가하여 사용자나 Claude가 호출할 수 있는 `/name` 바로가기를 생성합니다.

위치: 플러그인 루트의 `skills/` 또는 `commands/` 디렉토리

파일 형식: Skills는 `SKILL.md` 가 있는 디렉토리이고, commands는 간단한 마크다운 파일입니다.

Skill 구조:

```
skills/
├── pdf-processor/
│   ├── SKILL.md
│   ├── reference.md (선택사항)
│   └── scripts/ (선택사항)
└── code-reviewer/
    └── SKILL.md
```

통합 동작:

- Skills와 commands는 플러그인이 설치될 때 자동으로 발견됩니다.
- Claude는 작업 컨텍스트에 따라 자동으로 이들을 호출할 수 있습니다.
- Skills는 SKILL.md와 함께 지원 파일을 포함할 수 있습니다.

완전한 세부 정보는 Skills를 참조하세요.

Agents

플러그인은 Claude가 적절할 때 자동으로 호출할 수 있는 특정 작업을 위한 특화된 subagents를 제공할 수 있습니다.

위치: 플러그인 루트의 `agents/` 디렉토리

파일 형식: 에이전트 기능을 설명하는 마크다운 파일

Agent 구조:

```
---
name: agent-name
description: 이 에이전트가 전문으로 하는 분야와 Claude가 이를 호출해야 할 때
---

에이전트의 역할, 전문성 및 동작을 설명하는 상세한 시스템 프롬프트입니다.
```

통합 지점:

- Agents는 `/agents` 인터페이스에 나타납니다.
- Claude는 작업 컨텍스트에 따라 agents를 자동으로 호출할 수 있습니다.
- Agents는 사용자가 수동으로 호출할 수 있습니다.
- 플러그인 agents는 기본 제공 Claude agents와 함께 작동합니다.

완전한 세부 정보는 Subagents를 참조하세요.

Hooks

플러그인은 Claude Code 이벤트에 자동으로 응답하는 이벤트 핸들러를 제공할 수 있습니다.

위치: 플러그인 루트의 `hooks/hooks.json` 또는 plugin.json에 인라인

형식: 이벤트 매처 및 작업이 있는 JSON 구성

Hook 구성:

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PLUGIN_ROOT}/scripts/format-code.sh"
          }
        ]
      }
    ]
  }
}
```

사용 가능한 이벤트:

- `PreToolUse`: Claude가 도구를 사용하기 전
- `PostToolUse`: Claude가 도구를 성공적으로 사용한 후
- `PostToolUseFailure`: Claude 도구 실행이 실패한 후
- `PermissionRequest`: 권한 대화상자가 표시될 때
- `UserPromptSubmit`: 사용자가 프롬프트를 제출할 때
- `Notification`: Claude Code가 알림을 보낼 때
- `Stop`: Claude가 중지를 시도할 때
- `SubagentStart`: subagent가 시작될 때
- `SubagentStop`: subagent가 중지를 시도할 때
- `SessionStart`: 세션의 시작 시
- `SessionEnd`: 세션의 끝 시
- `TeammateIdle`: 에이전트 팀 팀원이 유휴 상태가 될 때
- `TaskCompleted`: 작업이 완료로 표시될 때
- `PreCompact`: 대화 기록이 압축되기 전

Hook 유형:

- `command`: 셸 명령어 또는 스크립트 실행
- `prompt`: LLM으로 프롬프트 평가 (컨텍스트에 대해 `$ARGUMENTS` 플레이스홀더 사용)

- `agent`: 복잡한 검증 작업을 위해 도구가 있는 에이전트 검증자 실행

MCP servers

플러그인은 Claude Code를 외부 도구 및 서비스와 연결하기 위해 Model Context Protocol (MCP) servers를 번들로 제공할 수 있습니다.

위치: 플러그인 루트의 `.mcp.json` 또는 plugin.json에 인라인

형식: 표준 MCP 서버 구성

MCP 서버 구성:

```
{
  "mcpServers": {
    "plugin-database": {
      "command": "${CLAUDE_PLUGIN_ROOT}/servers/db-server",
      "args": ["--config", "${CLAUDE_PLUGIN_ROOT}/config.json"],
      "env": {
        "DB_PATH": "${CLAUDE_PLUGIN_ROOT}/data"
      }
    },
    "plugin-api-client": {
      "command": "npx",
      "args": ["@company/mcp-server", "--plugin-mode"],
      "cwd": "${CLAUDE_PLUGIN_ROOT}"
    }
  }
}
```

통합 동작:

- 플러그인 MCP servers는 플러그인이 활성화될 때 자동으로 시작됩니다.
- Servers는 Claude의 도구 키트에서 표준 MCP 도구로 나타납니다.
- 서버 기능은 Claude의 기존 도구와 원활하게 통합됩니다.
- 플러그인 servers는 사용자 MCP servers와 독립적으로 구성할 수 있습니다.

LSP servers

> **Tip:**

> LSP 플러그인을 사용하려고 하시나요? 공식 마켓플레이스에서 설치하세요: `/plugin` Discover 탭에서 "lsp"를 검색하세요. 이 섹션은 공식 마켓플레이스에서 다루지 않는 언어에 대해 LSP 플러그인을 만드는 방법을 문서화합니다.

플러그인은 Language Server Protocol (LSP) servers를 제공하여 코드베이스에서 작업할 때 Claude에게 실시간 코드 인텔리전스를 제공할 수 있습니다.

LSP 통합은 다음을 제공합니다:

- **즉시 진단**: Claude는 각 편집 후 즉시 오류 및 경고를 봅니다.
- **코드 네비게이션**: 정의로 이동, 참조 찾기 및 호버 정보
- **언어 인식**: 코드 기호에 대한 타입 정보 및 문서

위치: 플러그인 루트의 `.lsp.json` 또는 `plugin.json`에 인라인

형식: 언어 서버 이름을 해당 구성에 매핑하는 JSON 구성

`.lsp.json` 파일 형식:

```
{
  "go": {
    "command": "gopls",
    "args": ["serve"],
    "extensionToLanguage": {
      ".go": "go"
    }
  }
}
```

`plugin.json`에 인라인:

```
{
  "name": "my-plugin",
  "lspServers": {
    "go": {
      "command": "gopls",
      "args": ["serve"],
      "extensionToLanguage": {
        ".go": "go"
      }
    }
  }
}
```

필수 필드:

필드	설명
`command`	실행할 LSP 바이너리 (PATH에 있어야 함)
`extensionToLanguage`	파일 확장자를 언어 식별자에 매핑

선택사항 필드:

필드	설명
`args`	LSP 서버의 명령줄 인수
`transport`	통신 전송: `stdio` (기본값) 또는 `socket`
`env`	서버 시작 시 설정할 환경 변수
`initializationOptions`	초기화 중에 서버에 전달되는 옵션
`settings`	`workspace/didChangeConfiguration`을 통해 전달되는 설정
`workspaceFolder`	서버의 작업 공간 폴더 경로
`startupTimeout`	서버 시작을 기다릴 최대 시간 (밀리초)
`shutdownTimeout`	정상 종료를 기다릴 최대 시간 (밀리초)
`restartOnCrash`	서버가 충돌하면 자동으로 다시 시작할지 여부

필드	설명
`maxRestarts`	포기하기 전 최대 재시작 시도 횟수

> **Warning:**
>
> **언어 서버 바이너리를 별도로 설치해야 합니다.** LSP 플러그인은 Claude Code가 언어 서버에 연결하는 방법을 구성하지만, 서버 자체는 포함하지 않습니다. `/plugin` Errors 탭에서 `Executable not found in $PATH` 를 보면 언어에 필요한 바이너리를 설치하세요.

사용 가능한 LSP 플러그인:

플러그인	언어 서버	설치 명령어
`pyright-lsp`	Pyright (Python)	`pip install pyright` 또는 `npm install -g pyright`
`typescript-lsp`	TypeScript Language Server	`npm install -g typescript-language-server typescript`
`rust-lsp`	rust-analyzer	rust-analyzer 설치 참조

먼저 언어 서버를 설치한 다음 마켓플레이스에서 플러그인을 설치하세요.

플러그인 설치 범위

플러그인을 설치할 때 플러그인이 사용 가능한 위치와 다른 사람이 사용할 수 있는지를 결정하는 **범위**를 선택합니다:

범위	설정 파일	사용 사례
`user`	`~/.claude/settings.json`	모든 프로젝트에서 사용 가능한 개인 플러그인 (기본값)
`project`	`.claude/settings.json`	버전 제어를 통해 공유되는 팀 플러그인
`local`	`.claude/settings.local.json`	프로젝트별 플러그인, gitignored

범위	설정 파일	사용 사례
`managed`	관리되는 설정	관리되는 플러그인 (읽기 전용, 업데이트만 가능)

플러그인은 다른 Claude Code 구성과 동일한 범위 시스템을 사용합니다. 설치 지침 및 범위 플래그는 플러그인 설치를 참조하세요. 범위에 대한 완전한 설명은 구성 범위를 참조하세요.

플러그인 매니페스트 스키마

`.claude-plugin/plugin.json` 파일은 플러그인의 메타데이터 및 구성을 정의합니다. 이 섹션은 지원되는 모든 필드 및 옵션을 문서화합니다.

매니페스트는 선택사항입니다. 생략하면 Claude Code는 기본 위치에서 컴포넌트를 자동으로 발견하고 디렉토리 이름에서 플러그인 이름을 파생합니다. 메타데이터를 제공하거나 사용자 정의 컴포넌트 경로가 필요할 때 매니페스트를 사용하세요.

완전한 스키마

```
{
  "name": "plugin-name",
  "version": "1.2.0",
  "description": "간단한 플러그인 설명",
  "author": {
    "name": "작성자 이름",
    "email": "author@example.com",
    "url": "https://github.com/author"
  },
  "homepage": "https://docs.example.com/plugin",
  "repository": "https://github.com/author/plugin",
  "license": "MIT",
  "keywords": ["keyword1", "keyword2"],
  "commands": ["./custom/commands/special.md"],
  "agents": "./custom/agents/",
  "skills": "./custom/skills/",
  "hooks": "./config/hooks.json",
  "mcpServers": "./mcp-config.json",
  "outputStyles": "./styles/",
  "lspServers": "./.lsp.json"
}
```

필수 필드

매니페스트를 포함하는 경우 `name`이 유일한 필수 필드입니다.

필드	타입	설명	예시
`name`	string	고유 식별자 (kebab-case, 공백 없음)	`"deployment-tools"`

이 이름은 컴포넌트 네임스페이싱에 사용됩니다. 예를 들어 UI에서 이름이 `plugin-dev`인 플러그인의 agent `agent-creator`는 `plugin-dev:agent-creator`로 나타납니다.

메타데이터 필드

필드	타입	설명	예시
`version`	string	의미 있는 버전. 마켓플레이스 항목에도 설정된 경우 `plugin.json` 이 우선합니다. 한 곳에만 설정하면 됩니다.	`"2.1.0"`
`description`	string	플러그인 목적에 대한 간단한 설명	`"배포 자동화 도구"`
`author`	object	작성자 정보	`{"name": "Dev Team", "email": "dev@company.com"}`
`homepage`	string	문서 URL	`"https://docs.example.com"`
`repository`	string	소스 코드 URL	`"https://github.com/user/plugin"`
`license`	string	라이선스 식별자	`"MIT"`, `"Apache-2.0"`
`keywords`	array	발견 태그	`["deployment", "ci-cd"]`

컴포넌트 경로 필드

필드	타입	설명	예시
`commands`	string|array	추가 명령어 파일/디렉토리	`"./custom/cmd.md"` 또는 `["./cmd1.md"]`
`agents`	string|array	추가 agent 파일	`"./custom/agents/reviewer.md"`

필드	타입	설명	예시
`skills`	string|array	추가 skill 디렉토리	`"./custom/skills/"`
`hooks`	string|array|object	Hook 구성 경로 또는 인라인 구성	`"./my-extra-hooks.json"`
`mcpServers`	string|array|object	MCP 구성 경로 또는 인라인 구성	`"./my-extra-mcp-config.json"`
`outputStyles`	string|array	추가 출력 스타일 파일/디렉토리	`"./styles/"`
`lspServers`	string|array|object	Language Server Protocol 코드 인텔리전스 구성 (정의로 이동, 참조 찾기 등)	`"./.lsp.json"`

경로 동작 규칙

중요: 사용자 정의 경로는 기본 디렉토리를 대체하지 않고 보완합니다.

- `commands/`가 존재하면 사용자 정의 명령어 경로와 함께 로드됩니다.
- 모든 경로는 플러그인 루트에 상대적이어야 하며 `./`로 시작해야 합니다.
- 사용자 정의 경로의 명령어는 동일한 명명 및 네임스페이싱 규칙을 사용합니다.
- 유연성을 위해 여러 경로를 배열로 지정할 수 있습니다.

경로 예시:

```
{
  "commands": [
    "./specialized/deploy.md",
    "./utilities/batch-process.md"
  ],
  "agents": [
    "./custom-agents/reviewer.md",
    "./custom-agents/tester.md"
  ]
}
```

환경 변수

`${CLAUDE_PLUGIN_ROOT}` : 플러그인 디렉토리의 절대 경로를 포함합니다. hooks, MCP servers 및 스크립트에서 이를 사용하여 설치 위치와 관계없이 올바른 경로를 보장하세요.

```
{
  "hooks": {
    "PostToolUse": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PLUGIN_ROOT}/scripts/process.sh"
          }
        ]
      }
    ]
  }
}
```

플러그인 캐싱 및 파일 해석

플러그인은 두 가지 방법 중 하나로 지정됩니다:

- `claude --plugin-dir` 을 통해, 세션 기간 동안.
- 마켓플레이스를 통해, 향후 세션을 위해 설치됨.

보안 및 검증 목적으로 Claude Code는 마켓플레이스 플러그인을 제자리에서 사용하는 대신 사용자의 로컬 **플러그인 캐시** (`~/.claude/plugins/cache`)에 복사합니다. 외부 파일을 참조하는 플러그인을 개발할 때 이 동작을 이해하는 것이 중요합니다.

경로 순회 제한

설치된 플러그인은 해당 디렉토리 외부의 파일을 참조할 수 없습니다. 플러그인 루트 외부를 순회하는 경로 (예: `../shared-utils`)는 설치 후 작동하지 않습니다. 왜냐하면 이러한 외부 파일이 캐시에 복사되지 않기 때문입니다.

외부 종속성 작업

플러그인이 디렉토리 외부의 파일에 액세스해야 하는 경우 플러그인 디렉토리 내에서 외부 파일에 대한 심볼릭 링크를 만들 수 있습니다. 심볼릭 링크는 복사 프로세스 중에 인정됩니다:

```
## 플러그인 디렉토리 내부
ln -s /path/to/shared-utils ./shared-utils
```

심볼릭 링크된 콘텐츠는 플러그인 캐시에 복사됩니다. 이는 캐싱 시스템의 보안 이점을 유지하면서 유연성을 제공합니다.

플러그인 디렉토리 구조

표준 플러그인 레이아웃

완전한 플러그인은 다음 구조를 따릅니다:

```
enterprise-plugin/
├── .claude-plugin/           # 메타데이터 디렉토리 (선택사항)
│   └── plugin.json             # 플러그인 매니페스트
├── commands/                 # 기본 명령어 위치
│   ├── status.md
│   └── logs.md
├── agents/                   # 기본 agent 위치
│   ├── security-reviewer.md
│   ├── performance-tester.md
│   └── compliance-checker.md
├── skills/                   # Agent Skills
│   ├── code-reviewer/
│   │   └── SKILL.md
│   └── pdf-processor/
│       ├── SKILL.md
│       └── scripts/
├── hooks/                    # Hook 구성
│   ├── hooks.json           # 주 hook 구성
│   └── security-hooks.json  # 추가 hooks
├── settings.json             # 플러그인의 기본 설정
├── .mcp.json                 # MCP 서버 정의
├── .lsp.json                 # LSP 서버 구성
├── scripts/                  # Hook 및 유틸리티 스크립트
│   ├── security-scan.sh
│   ├── format-code.py
│   └── deploy.js
├── LICENSE                   # 라이선스 파일
└── CHANGELOG.md              # 버전 기록
```

Warning:

`.claude-plugin/` 디렉토리는 `plugin.json` 파일을 포함합니다. 다른 모든 디렉토리(commands/, agents/, skills/, hooks/)는 `.claude-plugin/` 내부가 아닌 플러그인 루트에 있어야 합니다.

파일 위치 참조

컴포넌트	기본 위치	목적
매니페스트	`.claude-plugin/plugin.json`	플러그인 메타데이터 및 구성 (선택사항)
명령어	`commands/`	Skill 마크다운 파일 (레거시; 새 skills에는 `skills/` 사용)
Agents	`agents/`	Subagent 마크다운 파일
Skills	`skills/`	`<name>/SKILL.md` 구조의 Skills
Hooks	`hooks/hooks.json`	Hook 구성
MCP servers	`.mcp.json`	MCP 서버 정의
LSP servers	`.lsp.json`	언어 서버 구성
설정	`settings.json`	플러그인이 활성화될 때 적용되는 기본 구성. 현재 `agent` 설정만 지원됩니다.

CLI 명령어 참조

Claude Code는 스크립팅 및 자동화에 유용한 비대화형 플러그인 관리를 위한 CLI 명령어를 제공합니다.

plugin install

사용 가능한 마켓플레이스에서 플러그인을 설치합니다.

```
claude plugin install <plugin> [options]
```

인수:

- `<plugin>`: 플러그인 이름 또는 특정 마켓플레이스의 경우 `plugin-name@marketplace-name`

옵션:

옵션	설명	기본값
`-s, --scope <scope>`	설치 범위: `user`, `project` 또는 `local`	`user`
`-h, --help`	명령어 도움말 표시	

범위는 설치된 플러그인이 추가되는 설정 파일을 결정합니다. 예를 들어 –scope project는 `.claude/settings.json` 의 `enabledPlugins` 에 쓰므로 프로젝트 저장소를 복제하는 모든 사람이 플러그인을 사용할 수 있습니다.

예시:

```
## 사용자 범위에 설치 (기본값)
claude plugin install formatter@my-marketplace

## 프로젝트 범위에 설치 (팀과 공유)
claude plugin install formatter@my-marketplace --scope project

## 로컬 범위에 설치 (gitignored)
claude plugin install formatter@my-marketplace --scope local
```

plugin uninstall

설치된 플러그인을 제거합니다.

```
claude plugin uninstall <plugin> [options]
```

인수:

- `<plugin>`: 플러그인 이름 또는 `plugin-name@marketplace-name`

옵션:

옵션	설명	기본값
`-s, --scope <scope>`	범위에서 제거: `user`, `project` 또는 `local`	`user`
`-h, --help`	명령어 도움말 표시	

별칭: `remove`, `rm`

plugin enable

비활성화된 플러그인을 활성화합니다.

```
claude plugin enable <plugin> [options]
```

인수:

- `<plugin>`: 플러그인 이름 또는 `plugin-name@marketplace-name`

옵션:

옵션	설명	기본값
`-s, --scope <scope>`	활성화할 범위: `user`, `project` 또는 `local`	`user`
`-h, --help`	명령어 도움말 표시	

plugin disable

플러그인을 제거하지 않고 비활성화합니다.

```
claude plugin disable <plugin> [options]
```

인수:

- `<plugin>`: 플러그인 이름 또는 `plugin-name@marketplace-name`

옵션:

옵션	설명	기본값
`-s, --scope <scope>`	비활성화할 범위: `user`, `project` 또는 `local`	`user`
`-h, --help`	명령어 도움말 표시	

plugin update

플러그인을 최신 버전으로 업데이트합니다.

```
claude plugin update <plugin> [options]
```

인수:

- `<plugin>`: 플러그인 이름 또는 `plugin-name@marketplace-name`

옵션:

옵션	설명	기본값
`-s, --scope <scope>`	업데이트할 범위: `user`, `project`, `local` 또는 `managed`	`user`
`-h, --help`	명령어 도움말 표시	

디버깅 및 개발 도구

디버깅 명령어

`claude --debug` (또는 TUI 내 `/debug`)를 사용하여 플러그인 로딩 세부 정보를 확인하세요:

이는 다음을 표시합니다:

- 로드되는 플러그인
- 플러그인 매니페스트의 오류
- 명령어, agent 및 hook 등록
- MCP 서버 초기화

일반적인 문제

문제	원인	해결책
플러그인이 로드되지 않음	잘못된 `plugin.json`	`claude plugin validate` 또는 `/plugin validate`로 JSON 구문 검증
명령어가 나타나지 않음	잘못된 디렉토리 구조	`commands/`가 루트에 있는지 확인, `.claude-plugin/` 내부가 아님
Hooks가 실행되지 않음	스크립트가 실행 가능하지 않음	`chmod +x script.sh` 실행
MCP 서버 실패	`${CLAUDE_PLUGIN_ROOT}` 누락	모든 플러그인 경로에 변수 사용

문제	원인	해결책
경로 오류	절대 경로 사용됨	모든 경로는 상대적이어야 하며 `./` 로 시작해야 함
LSP `Executable not found in $PATH`	언어 서버가 설치되지 않음	바이너리 설치 (예: `npm install -g typescript-language-server typescript`)

예시 오류 메시지

매니페스트 검증 오류:

- `Invalid JSON syntax: Unexpected token } in JSON at position 142` : 누락된 쉼표, 추가 쉼표 또는 따옴표 없는 문자열 확인
- `Plugin has an invalid manifest file at .claude-plugin/plugin.json. Validation errors: name: Required` : 필수 필드가 누락됨
- `Plugin has a corrupt manifest file at .claude-plugin/plugin.json. JSON parse error: ...` : JSON 구문 오류

플러그인 로딩 오류:

- `Warning: No commands found in plugin my-plugin custom directory: ./cmds. Expected .md files or SKILL.md in subdirectories.` : 명령어 경로가 존재하지만 유효한 명령어 파일이 없음
- `Plugin directory not found at path: ./plugins/my-plugin. Check that the marketplace entry has the correct path.` : marketplace.json의 `source` 경로가 존재하지 않는 디렉토리를 가리킴
- `Plugin my-plugin has conflicting manifests: both plugin.json and marketplace entry specify components.` : 중복 컴포넌트 정의 제거 또는 marketplace 항목에서 `strict: false` 제거

Hook 문제 해결

Hook 스크립트가 실행되지 않음:

1. 스크립트가 실행 가능한지 확인: `chmod +x ./scripts/your-script.sh`
2. shebang 라인 확인: 첫 번째 줄은 `#!/bin/bash` 또는 `#!/usr/bin/env bash` 여야 함
3. 경로가 `${CLAUDE_PLUGIN_ROOT}` 사용하는지 확인: `"command": "${CLAUDE_PLUGIN_ROOT}/scripts/your-script.sh"`
4. 스크립트를 수동으로 테스트: `./scripts/your-script.sh`

Hook이 예상 이벤트에서 트리거되지 않음:

1. 이벤트 이름이 올바른지 확인 (대소문자 구분): `PostToolUse`, `postToolUse` 아님
2. 매처 패턴이 도구와 일치하는지 확인: 파일 작업의 경우 `"matcher": "Write|Edit"`
3. Hook 유형이 유효한지 확인: `command`, `prompt` 또는 `agent`

MCP 서버 문제 해결

서버가 시작되지 않음:

1. 명령어가 존재하고 실행 가능한지 확인
2. 모든 경로가 `${CLAUDE_PLUGIN_ROOT}` 변수를 사용하는지 확인
3. MCP 서버 로그 확인: `claude --debug`는 초기화 오류를 표시합니다.
4. Claude Code 외부에서 서버를 수동으로 테스트

서버 도구가 나타나지 않음:

1. 서버가 `.mcp.json` 또는 `plugin.json`에 올바르게 구성되었는지 확인
2. 서버가 MCP 프로토콜을 올바르게 구현하는지 확인
3. 디버그 출력에서 연결 시간 초과 확인

디렉토리 구조 실수

증상: 플러그인이 로드되지만 컴포넌트 (명령어, agents, hooks)가 누락됨.

올바른 구조: 컴포넌트는 플러그인 루트에 있어야 하며 `.claude-plugin/` 내부가 아닙니다. `plugin.json`만 `.claude-plugin/`에 속합니다.

```
my-plugin/
├── .claude-plugin/
│   └── plugin.json      ← 매니페스트만 여기
├── commands/            ← 루트 수준
├── agents/              ← 루트 수준
└── hooks/               ← 루트 수준
```

컴포넌트가 `.claude-plugin/` 내부에 있으면 플러그인 루트로 이동하세요.

디버그 체크리스트:

1. `claude --debug`를 실행하고 “loading plugin” 메시지를 찾으세요.
2. 각 컴포넌트 디렉토리가 디버그 출력에 나열되는지 확인
3. 파일 권한이 플러그인 파일 읽기를 허용하는지 확인

배포 및 버전 관리 참조

버전 관리

플러그인 릴리스에 대해 의미 있는 버전 관리를 따르세요:

```
{
  "name": "my-plugin",
  "version": "2.1.0"
}
```

버전 형식: `MAJOR.MINOR.PATCH`

- **MAJOR**: 주요 변경 사항 (호환되지 않는 API 변경)
- **MINOR**: 새로운 기능 (하위 호환 추가)
- **PATCH**: 버그 수정 (하위 호환 수정)

모범 사례:

- 첫 번째 안정 릴리스에서 `1.0.0`으로 시작
- 변경 사항을 배포하기 전에 `plugin.json`의 버전 업데이트
- `CHANGELOG.md` 파일에 변경 사항 문서화
- 테스트를 위해 `2.0.0-beta.1`과 같은 사전 릴리스 버전 사용

> **Warning:**
>
> Claude Code는 버전을 사용하여 플러그인을 업데이트할지 여부를 결정합니다. 플러그인의 코드를 변경했지만 `plugin.json`의 버전을 범프하지 않으면 캐싱으로 인해 플러그인의 기존 사용자가 변경 사항을 보지 못합니다.
>
> 플러그인이 마켓플레이스 디렉토리 내에 있으면 `marketplace.json`을 통해 버전을 관리할 수 있으며 `plugin.json`에서 `version` 필드를 생략할 수 있습니다.

참고 항목

- 플러그인 - 튜토리얼 및 실제 사용
- 플러그인 마켓플레이스 - 마켓플레이스 생성 및 관리
- Skills - Skill 개발 세부 정보
- Subagents - Agent 구성 및 기능
- Hooks - 이벤트 처리 및 자동화

- MCP - 외부 도구 통합
- 설정 - 플러그인의 구성 옵션

플러그인 마켓플레이스 생성 및 배포

> Claude Code 확장 프로그램을 팀과 커뮤니티에 배포하기 위한 플러그인 마켓플레이스를 구축하고 호스팅합니다.

플러그인 마켓플레이스는 다른 사용자에게 플러그인을 배포할 수 있는 카탈로그입니다. 마켓플레이스는 중앙 집중식 검색, 버전 추적, 자동 업데이트 및 여러 소스 유형(git 저장소, 로컬 경로 등)을 지원합니다. 이 가이드에서는 팀이나 커뮤니티와 플러그인을 공유하기 위해 자신의 마켓플레이스를 만드는 방법을 보여줍니다.

기존 마켓플레이스에서 플러그인을 설치하려고 하시나요? 미리 빌드된 플러그인 검색 및 설치를 참조하세요.

개요

마켓플레이스를 생성하고 배포하는 과정은 다음과 같습니다:

1. **플러그인 생성**: 명령어, 에이전트, hooks, MCP 서버 또는 LSP 서버를 사용하여 하나 이상의 플러그인을 빌드합니다. 이 가이드에서는 배포할 플러그인이 이미 있다고 가정합니다. 플러그인 생성 방법에 대한 자세한 내용은 플러그인 생성을 참조하세요.
2. **마켓플레이스 파일 생성**: 플러그인을 나열하고 플러그인을 찾을 위치를 정의하는 `marketplace.json` 을 정의합니다(마켓플레이스 파일 생성 참조).
3. **마켓플레이스 호스팅**: GitHub, GitLab 또는 다른 git 호스트에 푸시합니다(마켓플레이스 호스팅 및 배포 참조).
4. **사용자와 공유**: 사용자가 `/plugin marketplace add` 로 마켓플레이스를 추가하고 개별 플러그인을 설치합니다(플러그인 검색 및 설치 참조).

마켓플레이스가 라이브 상태가 되면 저장소에 변경 사항을 푸시하여 업데이트할 수 있습니다. 사용자는 `/plugin marketplace update` 로 로컬 복사본을 새로 고칩니다.

연습: 로컬 마켓플레이스 생성

이 예제에서는 하나의 플러그인으로 마켓플레이스를 생성합니다: 코드 리뷰를 위한 `/quality-review` skill입니다. 디렉터리 구조를 생성하고, skill을 추가하고, 플러그인 매니페스트와 마켓플레이스 카탈로그를 생성한 다음, 설치하고 테스트합니다.

Step 1: 디렉터리 구조 생성

```
mkdir -p my-marketplace/.claude-plugin
mkdir -p my-marketplace/plugins/quality-review-plugin/.claude-plugin
mkdir -p my-marketplace/plugins/quality-review-plugin/skills/quality-review
```

Step 2: skill 생성

`/quality-review` skill이 수행하는 작업을 정의하는 `SKILL.md` 파일을 생성합니다.

```
---
description: 버그, 보안 및 성능에 대한 코드 검토
disable-model-invocation: true
---

선택한 코드 또는 최근 변경 사항을 다음 항목에 대해 검토합니다:
- 잠재적 버그 또는 엣지 케이스
- 보안 문제
- 성능 문제
- 가독성 개선

간결하고 실행 가능한 내용을 제공합니다.
```

Step 3: 플러그인 매니페스트 생성

플러그인을 설명하는 `plugin.json` 파일을 생성합니다. 매니페스트는 `.claude-plugin/` 디렉터리에 위치합니다.

```
{
  "name": "quality-review-plugin",
  "description": "빠른 코드 리뷰를 위한 /quality-review skill 추가",
  "version": "1.0.0"
}
```

Step 4: 마켓플레이스 파일 생성

플러그인을 나열하는 마켓플레이스 카탈로그를 생성합니다.

```
{
  "name": "my-plugins",
  "owner": {
    "name": "Your Name"
  },
  "plugins": [
    {
      "name": "quality-review-plugin",
      "source": "./plugins/quality-review-plugin",
      "description": "빠른 코드 리뷰를 위한 /quality-review skill 추가"
    }
  ]
}
```

Step 5: 추가 및 설치

마켓플레이스를 추가하고 플러그인을 설치합니다.

```
/plugin marketplace add ./my-marketplace
/plugin install quality-review-plugin@my-plugins
```

Step 6: 시도해보기

편집기에서 일부 코드를 선택하고 새 명령어를 실행합니다.

```
/review
```

hooks, 에이전트, MCP 서버 및 LSP 서버를 포함하여 플러그인이 수행할 수 있는 작업에 대해 자세히 알아보려면 플러그인을 참조하세요.

> **Note:**
>
> **플러그인 설치 방법**: 사용자가 플러그인을 설치하면 Claude Code는 플러그인 디렉터리를 캐시 위치에 복사합니다. 이는 `../shared-utils` 와 같은 경로를 사용하여 플러그인 디렉터리 외부의 파일을 참조할 수 없다는 의미입니다. 왜냐하면 해당 파일이 복사되지 않기 때문입니다.
>
> 플러그인 간에 파일을 공유해야 하는 경우 symlink를 사용합니다(복사 중에 따릅니다). 자세한 내용은 플러그인 캐싱 및 파일 해석을 참조하세요.

마켓플레이스 파일 생성

저장소 루트에 `.claude-plugin/marketplace.json`을 생성합니다. 이 파일은 마켓플레이스의 이름, 소유자 정보 및 소스가 있는 플러그인 목록을 정의합니다.

각 플러그인 항목에는 최소한 `name`과 `source`(가져올 위치)가 필요합니다. 사용 가능한 모든 필드는 아래의 전체 스키마를 참조하세요.

```
{
  "name": "company-tools",
  "owner": {
    "name": "DevTools Team",
    "email": "devtools@example.com"
  },
  "plugins": [
    {
      "name": "code-formatter",
      "source": "./plugins/formatter",
      "description": "저장 시 자동 코드 포맷팅",
      "version": "2.1.0",
      "author": {
        "name": "DevTools Team"
      }
    },
    {
      "name": "deployment-tools",
      "source": {
        "source": "github",
        "repo": "company/deploy-plugin"
      },
      "description": "배포 자동화 도구"
    }
  ]
}
```

마켓플레이스 스키마

필수 필드

필드	유형	설명	예제
`name`	string	마켓플레이스 식별자 (kebab-case, 공백 없음). 이는 공개 대면입니다: 사용자는 플러그인을 설치할 때 이를 봅니다(예: `/plugin install my-tool@your-marketplace`).	`"acme-tools"`
`owner`	object	마켓플레이스 유지 관리자 정보(아래 필드 참조)	
`plugins`	array	사용 가능한 플러그인 목록	아래 참조

Note:

예약된 이름: 다음 마켓플레이스 이름은 공식 Anthropic 사용을 위해 예약되어 있으며 타사 마켓플레이스에서 사용할 수 없습니다: `claude-code-marketplace`, `claude-code-plugins`, `claude-plugins-official`, `anthropic-marketplace`, `anthropic-plugins`, `agent-skills`, `life-sciences`. 공식 마켓플레이스를 사칭하는 이름(예: `official-claude-plugins` 또는 `anthropic-tools-v2`)도 차단됩니다.

소유자 필드

필드	유형	필수	설명
`name`	string	예	유지 관리자 또는 팀의 이름
`email`	string	아니오	유지 관리자의 연락처 이메일

선택적 메타데이터

필드	유형	설명
`metadata.description`	string	간단한 마켓플레이스 설명
`metadata.version`	string	마켓플레이스 버전
`metadata.pluginRoot`	string	상대 플러그인 소스 경로에 앞에 붙는 기본 디렉터리(예: `"./plugins"` 를 사용하면 `"source": "./plugins/formatter"` 대신 `"source": "formatter"` 를 작성할 수 있습니다)

플러그인 항목

`plugins` 배열의 각 플러그인 항목은 플러그인과 플러그인을 찾을 위치를 설명합니다. 플러그인 매니페스트 스키마의 모든 필드(예: `description`, `version`, `author`, `commands`, `hooks` 등)와 이러한 마켓플레이스 특정 필드를 포함할 수 있습니다: `source`, `category`, `tags` 및 `strict`.

필수 필드

필드	유형	설명
`name`	string	플러그인 식별자(kebab-case, 공백 없음). 이는 공개 대면입니다: 사용자는 설치할 때 이를 봅니다(예: `/plugin install my-plugin@marketplace`).
`source`	string|object	플러그인을 가져올 위치(아래 플러그인 소스 참조)

선택적 플러그인 필드

표준 메타데이터 필드:

필드	유형	설명
`description`	string	간단한 플러그인 설명

필드	유형	설명
`version`	string	플러그인 버전
`author`	object	플러그인 작성자 정보 (`name` 필수, `email` 선택)
`homepage`	string	플러그인 홈페이지 또는 문서 URL
`repository`	string	소스 코드 저장소 URL
`license`	string	SPDX 라이선스 식별자(예: MIT, Apache-2.0)
`keywords`	array	플러그인 검색 및 분류를 위한 태그
`category`	string	조직을 위한 플러그인 카테고리
`tags`	array	검색 가능성을 위한 태그
`strict`	boolean	`plugin.json`이 구성 요소 정의의 권한인지 여부를 제어합니다(기본값: true). 아래의 Strict 모드를 참조하세요.

구성 요소 구성 필드:

필드	유형	설명
`commands`	string|array	명령어 파일 또는 디렉터리의 사용자 정의 경로
`agents`	string|array	에이전트 파일의 사용자 정의 경로
`hooks`	string|object	사용자 정의 hooks 구성 또는 hooks 파일 경로
`mcpServers`	string|object	MCP 서버 구성 또는 MCP 구성 경로
`lspServers`	string|object	LSP 서버 구성 또는 LSP 구성 경로

플러그인 소스

플러그인 소스는 Claude Code에 마켓플레이스에 나열된 각 개별 플러그인을 가져올 위치를 알려줍니다. 이는 `marketplace.json` 의 각 플러그인 항목의 `source` 필드에 설정됩니다.

플러그인이 로컬 머신에 복제되거나 복사되면 `~/.claude/plugins/cache` 의 로컬 버전 관리 플러그인 캐시에 복사됩니다.

소스	유형	필드	참고
상대 경로	`string` (예: `"./my-plugin"`)	—	마켓플레이스 저장소 내의 로컬 디렉터리. `./` 로 시작해야 합니다
`github`	object	`repo` , `ref?` , `sha?`	
`url`	object	`url` (.git로 끝나야 함), `ref?` , `sha?`	Git URL 소스
`git-subdir`	object	`url` , `path` , `ref?` , `sha?`	git 저장소 내의 하위 디렉터리. 모노레포의 대역폭을 최소화하기 위해 희소하게 복제합니다
`npm`	object	`package` , `version?` , `registry?`	`npm install` 을 통해 설치됨
`pip`	object	`package` , `version?` , `registry?`	pip을 통해 설치됨

> **Note:**
>
> **마켓플레이스 소스 vs 플러그인 소스**: 이는 다양한 것을 제어하는 다양한 개념입니다.
>
> - **마켓플레이스 소스** — `marketplace.json` 카탈로그 자체를 가져올 위치. 사용자가 `/plugin marketplace add` 를 실행하거나 `extraKnownMarketplaces` 설정에서 설정합니다. `ref` (분기/태그)를 지원하지만 `sha` 는 지원하지 않습니다.
> - **플러그인 소스** — 마켓플레이스에 나열된 개별 플러그인을 가져올 위치. `marketplace.json` 내의 각 플러그인 항목의 `source` 필드에 설정됩니다. `ref` (분기/태그)와 `sha` (정확한 커밋) 모두를 지원합니다.

예를 들어, `acme-corp/plugin-catalog` 에서 호스팅되는 마켓플레이스(마켓플레이스 소스)는 `acme-corp/code-formatter` 에서 가져온 플러그인을 나열할 수 있습니다(플러그인 소스). 마켓플레이스 소스와 플러그인 소스는 다양한 저장소를 가리키며 독립적으로 고정됩니다.

상대 경로

동일한 저장소의 플러그인의 경우:

```
{
  "name": "my-plugin",
  "source": "./plugins/my-plugin"
}
```

Note:

상대 경로는 사용자가 Git(GitHub, GitLab 또는 git URL)을 통해 마켓플레이스를 추가할 때만 작동합니다. 사용자가 `marketplace.json` 파일에 대한 직접 URL을 통해 마켓플레이스를 추가하면 상대 경로가 올바르게 해석되지 않습니다. URL 기반 배포의 경우 GitHub, npm 또는 git URL 소스를 대신 사용합니다. 자세한 내용은 문제 해결을 참조하세요.

GitHub 저장소

```
{
  "name": "github-plugin",
  "source": {
    "source": "github",
    "repo": "owner/plugin-repo"
  }
}
```

특정 분기, 태그 또는 커밋에 고정할 수 있습니다:

```
{
  "name": "github-plugin",
  "source": {
    "source": "github",
    "repo": "owner/plugin-repo",
    "ref": "v2.0.0",
    "sha": "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0"
  }
}
```

필드	유형	설명
`repo`	string	필수. `owner/repo` 형식의 GitHub 저장소
`ref`	string	선택. Git 분기 또는 태그(저장소 기본 분기로 기본값)
`sha`	string	선택. 정확한 버전에 고정하기 위한 전체 40자 git 커밋 SHA

Git 저장소

```
{
  "name": "git-plugin",
  "source": {
    "source": "url",
    "url": "https://gitlab.com/team/plugin.git"
  }
}
```

특정 분기, 태그 또는 커밋에 고정할 수 있습니다:

```
{
  "name": "git-plugin",
  "source": {
    "source": "url",
    "url": "https://gitlab.com/team/plugin.git",
    "ref": "main",
    "sha": "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0"
  }
}
```

필드	유형	설명
`url`	string	필수. 전체 git 저장소 URL(.git로 끝나야 함)
`ref`	string	선택. Git 분기 또는 태그(저장소 기본 분기로 기본값)
`sha`	string	선택. 정확한 버전에 고정하기 위한 전체 40자 git 커밋 SHA

Git 하위 디렉터리

`git-subdir` 을 사용하여 git 저장소의 하위 디렉터리 내에 있는 플러그인을 가리킵니다. Claude Code는 희소하고 부분적인 복제를 사용하여 하위 디렉터리만 가져오므로 대규모 모노레포의 대역폭을 최소화합니다.

```
{
  "name": "my-plugin",
  "source": {
    "source": "git-subdir",
    "url": "https://github.com/acme-corp/monorepo.git",
    "path": "tools/claude-plugin"
  }
}
```

특정 분기, 태그 또는 커밋에 고정할 수 있습니다:

```
{
  "name": "my-plugin",
  "source": {
    "source": "git-subdir",
    "url": "https://github.com/acme-corp/monorepo.git",
    "path": "tools/claude-plugin",
    "ref": "v2.0.0",
    "sha": "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0"
  }
}
```

`url` 필드는 GitHub 단축형(`owner/repo`) 또는 SSH URL(`git@github.com:owner/repo.git`)도 허용합니다.

필드	유형	설명
`url`	string	필수. Git 저장소 URL, GitHub `owner/repo` 단축형 또는 SSH URL
`path`	string	필수. 플러그인을 포함하는 저장소 내의 하위 디렉터리 경로(예: `"tools/claude-plugin"`)
`ref`	string	선택. Git 분기 또는 태그(저장소 기본 분기로 기본값)
`sha`	string	선택. 정확한 버전에 고정하기 위한 전체 40자 git 커밋 SHA

npm 패키지

npm 패키지로 배포되는 플러그인은 `npm install` 을 사용하여 설치됩니다. 이는 공개 npm 레지스트리 또는 팀이 호스팅하는 개인 레지스트리의 모든 패키지에서 작동합니다.

```
{
  "name": "my-npm-plugin",
  "source": {
    "source": "npm",
    "package": "@acme/claude-plugin"
  }
}
```

특정 버전에 고정하려면 `version` 필드를 추가합니다:

```
{
  "name": "my-npm-plugin",
  "source": {
    "source": "npm",
    "package": "@acme/claude-plugin",
    "version": "2.1.0"
  }
}
```

개인 또는 내부 레지스트리에서 설치하려면 `registry` 필드를 추가합니다:

```
{
  "name": "my-npm-plugin",
  "source": {
    "source": "npm",
    "package": "@acme/claude-plugin",
    "version": "^2.0.0",
    "registry": "https://npm.example.com"
  }
}
```

필드	유형	설명
`package`	string	필수. 패키지 이름 또는 범위 지정 패키지(예: `@org/plugin`)

필드	유형	설명
`version`	string	선택. 버전 또는 버전 범위 (예: `2.1.0`, `^2.0.0`, `~1.5.0`)
`registry`	string	선택. 사용자 정의 npm 레지스트리 URL. 시스템 npm 레지스트리(일반적으로 npmjs.org)로 기본값

고급 플러그인 항목

이 예제는 명령어, 에이전트, hooks 및 MCP 서버의 사용자 정의 경로를 포함하여 많은 선택적 필드를 사용하는 플러그인 항목을 보여줍니다:

```
{
  "name": "enterprise-tools",
  "source": {
    "source": "github",
    "repo": "company/enterprise-plugin"
  },
  "description": "엔터프라이즈 워크플로우 자동화 도구",
  "version": "2.1.0",
  "author": {
    "name": "Enterprise Team",
    "email": "enterprise@example.com"
  },
  "homepage": "https://docs.example.com/plugins/enterprise-tools",
  "repository": "https://github.com/company/enterprise-plugin",
  "license": "MIT",
  "keywords": ["enterprise", "workflow", "automation"],
  "category": "productivity",
  "commands": [
    "./commands/core/",
    "./commands/enterprise/",
    "./commands/experimental/preview.md"
  ],
  "agents": ["./agents/security-reviewer.md", "./agents/compliance-checker.md"],
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PLUGIN_ROOT}/scripts/validate.sh"
          }
        ]
      }
    ]
  },
  "mcpServers": {
    "enterprise-db": {
```

```
      "command": "${CLAUDE_PLUGIN_ROOT}/servers/db-server",
      "args": ["--config", "${CLAUDE_PLUGIN_ROOT}/config.json"]
    }
  },
  "strict": false
}
```

주목할 주요 사항:

- `commands` 및 `agents`: 여러 디렉터리 또는 개별 파일을 지정할 수 있습니다. 경로는 플러그인 루트에 상대적입니다.
- `${CLAUDE_PLUGIN_ROOT}`: hooks 및 MCP 서버 구성에서 이 변수를 사용하여 플러그인의 설치 디렉터리 내의 파일을 참조합니다. 플러그인이 설치될 때 캐시 위치에 복사되기 때문에 필요합니다.
- `strict: false`: 이것이 false로 설정되어 있으므로 플러그인은 자신의 `plugin.json` 이 필요하지 않습니다. 마켓플레이스 항목이 모든 것을 정의합니다. 아래의 Strict 모드를 참조하세요.

Strict 모드

`strict` 필드는 `plugin.json` 이 구성 요소 정의(명령어, 에이전트, hooks, skills, MCP 서버, 출력 스타일)의 권한인지 여부를 제어합니다.

값	동작
`true` (기본값)	`plugin.json` 이 권한입니다. 마켓플레이스 항목은 추가 구성 요소로 이를 보완할 수 있으며 두 소스가 병합됩니다.
`false`	마켓플레이스 항목이 전체 정의입니다. 플러그인에 구성 요소를 선언하는 `plugin.json` 도 있으면 충돌이 발생하고 플러그인이 로드되지 않습니다.

각 모드를 사용할 때:

- `strict: true`: 플러그인은 자신의 `plugin.json` 을 가지고 있으며 자신의 구성 요소를 관리합니다. 마켓플레이스 항목은 맨 위에 추가 명령어 또는 hooks를 추가할 수 있습니다. 이것이 기본값이며 대부분의 플러그인에서 작동합니다.
- `strict: false`: 마켓플레이스 운영자가 완전한 제어를 원합니다. 플러그인 저장소는 원본 파일을 제공하고 마켓플레이스 항목은 이러한 파일 중 어느 것이 명령어, 에이전트, hooks 등으로 노출되는지 정의합니다. 마켓플레이스가 플러그인 작성자의 의도와 다르게 플러그인의 구성 요소를 재구성하거나 큐레이션할 때 유용합니다.

마켓플레이스 호스팅 및 배포

GitHub에서 호스팅(권장)

GitHub는 가장 쉬운 배포 방법을 제공합니다:

1. **저장소 생성**: 마켓플레이스를 위한 새 저장소 설정
2. **마켓플레이스 파일 추가**: 플러그인 정의와 함께 `.claude-plugin/marketplace.json` 생성
3. **팀과 공유**: 사용자가 `/plugin marketplace add owner/repo`로 마켓플레이스를 추가합니다

이점: 기본 제공 버전 제어, 문제 추적 및 팀 협업 기능.

다른 git 서비스에서 호스팅

GitLab, Bitbucket 및 자체 호스팅 서버와 같은 모든 git 호스팅 서비스가 작동합니다. 사용자는 전체 저장소 URL로 추가합니다:

```
/plugin marketplace add https://gitlab.com/company/plugins.git
```

개인 저장소

Claude Code는 개인 저장소에서 플러그인 설치를 지원합니다. 수동 설치 및 업데이트의 경우 Claude Code는 기존 git 자격 증명 도우미를 사용합니다. 터미널에서 개인 저장소에 대해 `git clone`이 작동하면 Claude Code에서도 작동합니다. 일반적인 자격 증명 도우미에는 GitHub의 `gh auth login`, macOS Keychain 및 `git-credential-store`가 포함됩니다.

백그라운드 자동 업데이트는 대화형 프롬프트가 Claude Code 시작을 차단하므로 자격 증명 도우미 없이 시작 시 실행됩니다. 개인 마켓플레이스에 대한 자동 업데이트를 활성화하려면 환경에서 적절한 인증 토큰을 설정합니다:

공급자	환경 변수	참고
GitHub	`GITHUB_TOKEN` 또는 `GH_TOKEN`	개인 액세스 토큰 또는 GitHub App 토큰
GitLab	`GITLAB_TOKEN` 또는 `GL_TOKEN`	개인 액세스 토큰 또는 프로젝트 토큰
Bitbucket	`BITBUCKET_TOKEN`	앱 비밀번호 또는 저장소 액세스 토큰

셸 구성(예: `.bashrc`, `.zshrc`)에서 토큰을 설정하거나 Claude Code를 실행할 때 전달합니다:

```
export GITHUB_TOKEN=ghp_xxxxxxxxxxxxxxxxxxxx
```

> **Note:**
>
> CI/CD 환경의 경우 토큰을 비밀 환경 변수로 구성합니다. GitHub Actions는 동일한 조직의 저장소에 대해 자동으로 `GITHUB_TOKEN` 을 제공합니다.

배포 전에 로컬에서 테스트

공유하기 전에 마켓플레이스를 로컬에서 테스트합니다:

```
/plugin marketplace add ./my-local-marketplace
/plugin install test-plugin@my-local-marketplace
```

추가 명령어의 전체 범위(GitHub, Git URL, 로컬 경로, 원격 URL)는 마켓플레이스 추가를 참조하세요.

팀을 위한 마켓플레이스 필수

프로젝트 폴더를 신뢰할 때 팀 구성원이 자동으로 마켓플레이스를 설치하도록 저장소를 구성할 수 있습니다. 마켓플레이스를 `.claude/settings.json` 에 추가합니다:

```
{
  "extraKnownMarketplaces": {
    "company-tools": {
      "source": {
        "source": "github",
        "repo": "your-org/claude-plugins"
      }
    }
  }
}
```

기본적으로 활성화해야 하는 플러그인을 지정할 수도 있습니다:

```
{
  "enabledPlugins": {
    "code-formatter@company-tools": true,
    "deployment-tools@company-tools": true
  }
}
```

전체 구성 옵션은 플러그인 설정을 참조하세요.

관리되는 마켓플레이스 제한

플러그인 소스에 대한 엄격한 제어가 필요한 조직의 경우 관리자는 관리되는 설정에서 `strictKnownMarketplaces` 설정을 사용하여 사용자가 추가할 수 있는 플러그인 마켓플레이스를 제한할 수 있습니다.

`strictKnownMarketplaces`가 관리되는 설정에서 구성되면 제한 동작은 값에 따라 달라집니다:

값	동작
정의되지 않음(기본값)	제한 없음. 사용자는 모든 마켓플레이스를 추가할 수 있습니다
빈 배열 `[]`	완전한 잠금. 사용자는 새 마켓플레이스를 추가할 수 없습니다
소스 목록	사용자는 허용 목록과 정확히 일치하는 마켓플레이스만 추가할 수 있습니다

일반적인 구성

모든 마켓플레이스 추가 비활성화:

```
{
  "strictKnownMarketplaces": []
}
```

특정 마켓플레이스만 허용:

```
{
  "strictKnownMarketplaces": [
    {
      "source": "github",
      "repo": "acme-corp/approved-plugins"
    },
    {
      "source": "github",
      "repo": "acme-corp/security-tools",
      "ref": "v2.0"
    },
    {
      "source": "url",
      "url": "https://plugins.example.com/marketplace.json"
    }
  ]
}
```

호스트에 대한 정규식 패턴 일치를 사용하여 내부 git 서버의 모든 마켓플레이스 허용:

```
{
  "strictKnownMarketplaces": [
    {
      "source": "hostPattern",
      "hostPattern": "^github\\.example\\.com$"
    }
  ]
}
```

경로에 대한 정규식 패턴 일치를 사용하여 특정 디렉터리의 파일 시스템 기반 마켓플레이스 허용:

```
{
  "strictKnownMarketplaces": [
    {
      "source": "pathPattern",
      "pathPattern": "^/opt/approved/"
    }
  ]
}
```

`pathPattern` 으로 모든 파일 시스템 경로를 허용하면서 `hostPattern` 으로 네트워크 소스를 제어하려면 `".*"` 를 `pathPattern` 으로 사용합니다.

제한 작동 방식

제한은 플러그인 설치 프로세스 초기에 검증되며 네트워크 요청 또는 파일 시스템 작업이 발생하기 전입니다. 이는 무단 마켓플레이스 액세스 시도를 방지합니다.

허용 목록은 대부분의 소스 유형에 대해 정확한 일치를 사용합니다. 마켓플레이스가 허용되려면 지정된 모든 필드가 정확히 일치해야 합니다:

- GitHub 소스의 경우: `repo` 는 필수이며 허용 목록에 지정된 경우 `ref` 또는 `path` 도 일치해야 합니다
- URL 소스의 경우: 전체 URL이 정확히 일치해야 합니다
- `hostPattern` 소스의 경우: 마켓플레이스 호스트가 정규식 패턴과 일치합니다
- `pathPattern` 소스의 경우: 마켓플레이스의 파일 시스템 경로가 정규식 패턴과 일치합니다

`strictKnownMarketplaces` 는 관리되는 설정에서 설정되므로 개별 사용자 및 프로젝트 구성은 이러한 제한을 재정의할 수 없습니다.

전체 구성 세부 정보(지원되는 모든 소스 유형 및 `extraKnownMarketplaces` 와의 비교 포함)는 strictKnownMarketplaces 참조를 참조하세요.

버전 해석 및 릴리스 채널

플러그인 버전은 캐시 경로 및 업데이트 감지를 결정합니다. 플러그인 매니페스트(`plugin.json`) 또는 마켓플레이스 항목(`marketplace.json`)에서 버전을 지정할 수 있습니다.

> **Warning:**

> 가능하면 두 위치에서 버전을 설정하지 마세요. 플러그인 매니페스트가 항상 자동으로 우선합니다. 이는 마켓플레이스 버전이 무시될 수 있습니다. 상대 경로 플러그인의 경우 마켓플레이스 항목에서 버전을 설정합니다. 다른 모든 플러그인 소스의 경우 플러그인 매니페스트에서 설정합니다.

릴리스 채널 설정

플러그인에 대한 “stable” 및 “latest” 릴리스 채널을 지원하려면 동일한 저장소의 다양한 refs 또는 SHA를 가리키는 두 개의 마켓플레이스를 설정할 수 있습니다. 그런 다음 관리되는 설정을 통해 두 마켓플레이스를 다양한 사용자 그룹에 할당할 수 있습니다.

> **Warning:**
>
> 플러그인의 `plugin.json` 은 각 고정된 ref 또는 커밋에서 다양한 `version` 을 선언해야 합니다. 두 refs 또는 커밋이 동일한 매니페스트 버전을 가지면 Claude Code는 이들을 동일한 것으로 취급하고 업데이트를 건너뜁니다.

예제

```
{
  "name": "stable-tools",
  "plugins": [
    {
      "name": "code-formatter",
      "source": {
        "source": "github",
        "repo": "acme-corp/code-formatter",
        "ref": "stable"
      }
    }
  ]
}
```

```
{
  "name": "latest-tools",
  "plugins": [
    {
      "name": "code-formatter",
      "source": {
        "source": "github",
        "repo": "acme-corp/code-formatter",
        "ref": "latest"
      }
    }
  ]
}
```

사용자 그룹에 채널 할당

관리되는 설정을 통해 각 마켓플레이스를 적절한 사용자 그룹에 할당합니다. 예를 들어 stable 그룹은 다음을 받습니다:

```
{
  "extraKnownMarketplaces": {
    "stable-tools": {
      "source": {
        "source": "github",
        "repo": "acme-corp/stable-tools"
      }
    }
  }
}
```

early-access 그룹은 대신 `latest-tools`를 받습니다:

```
{
  "extraKnownMarketplaces": {
    "latest-tools": {
      "source": {
        "source": "github",
        "repo": "acme-corp/latest-tools"
      }
    }
  }
}
```

검증 및 테스트

공유하기 전에 마켓플레이스를 테스트합니다.

마켓플레이스 JSON 구문 검증:

```
claude plugin validate .
```

또는 Claude Code 내에서:

```
/plugin validate .
```

테스트를 위해 마켓플레이스 추가:

```
/plugin marketplace add ./path/to/marketplace
```

모든 것이 작동하는지 확인하기 위해 테스트 플러그인 설치:

```
/plugin install test-plugin@marketplace-name
```

전체 플러그인 테스트 워크플로우는 플러그인을 로컬에서 테스트를 참조하세요. 기술적 문제 해결은 플러그인 참조를 참조하세요.

문제 해결

마켓플레이스가 로드되지 않음

증상: 마켓플레이스를 추가할 수 없거나 플러그인을 볼 수 없습니다

해결책:

- 마켓플레이스 URL이 액세스 가능한지 확인합니다
- `.claude-plugin/marketplace.json`이 지정된 경로에 있는지 확인합니다
- `claude plugin validate` 또는 `/plugin validate`를 사용하여 JSON 구문이 유효한지 확인합니다
- 개인 저장소의 경우 액세스 권한이 있는지 확인합니다

마켓플레이스 검증 오류

마켓플레이스 디렉터리에서 `claude plugin validate .` 또는 `/plugin validate .`를 실행하여 문제를 확인합니다. 일반적인 오류:

오류	원인	해결책
`File not found: .claude-plugin/marketplace.json`	누락된 매니페스트	필수 필드를 사용하여 `.claude-plugin/marketplace.json` 생성
`Invalid JSON syntax: Unexpected token...`	JSON 구문 오류	누락된 쉼표, 추가 쉼표 또는 인용되지 않은 문자열 확인
`Duplicate plugin name "x" found in marketplace`	두 플러그인이 동일한 이름을 공유합니다	각 플러그인에 고유한 `name` 값 지정
`plugins[0].source: Path traversal not allowed`	소스 경로에 `..` 포함	마켓플레이스 루트에 상대적인 경로를 `..` 없이 사용

경고(차단하지 않음):

- `Marketplace has no plugins defined`: `plugins` 배열에 최소한 하나의 플러그인 추가
- `No marketplace description provided`: 사용자가 마켓플레이스를 이해하도록 돕기 위해 `metadata.description` 추가

플러그인 설치 실패

증상: 마켓플레이스가 나타나지만 플러그인 설치가 실패합니다

해결책:

- 플러그인 소스 URL이 액세스 가능한지 확인합니다
- 플러그인 디렉터리에 필수 파일이 포함되어 있는지 확인합니다
- GitHub 소스의 경우 저장소가 공개이거나 액세스 권한이 있는지 확인합니다
- 플러그인 소스를 수동으로 복제/다운로드하여 테스트합니다

개인 저장소 인증 실패

증상: 개인 저장소에서 플러그인을 설치할 때 인증 오류

해결책:

수동 설치 및 업데이트의 경우:

- git 공급자로 인증되었는지 확인합니다(예: GitHub의 경우 `gh auth status` 실행).
- 자격 증명 도우미가 올바르게 구성되었는지 확인합니다: `git config --global credential.helper`
- 저장소를 수동으로 복제하여 자격 증명이 작동하는지 확인합니다

백그라운드 자동 업데이트의 경우:

- 환경에서 적절한 토큰이 설정되었는지 확인합니다: `echo $GITHUB_TOKEN`
- 토큰에 필수 권한이 있는지 확인합니다(저장소에 대한 읽기 액세스)
- GitHub의 경우 토큰에 개인 저장소에 대한 `repo` 범위가 있는지 확인합니다
- GitLab의 경우 토큰에 최소한 `read_repository` 범위가 있는지 확인합니다
- 토큰이 만료되지 않았는지 확인합니다

Git 작업 시간 초과

증상: 플러그인 설치 또는 마켓플레이스 업데이트가 “Git clone timed out after 120s” 또는 “Git pull timed out after 120s” 와 같은 시간 초과 오류로 실패합니다.

원인: Claude Code는 플러그인 저장소 복제 및 마켓플레이스 업데이트 끌어오기를 포함한 모든 git 작업에 120초 시간 초과를 사용합니다. 대규모 저장소 또는 느린 네트워크 연결이 이 제한을 초과할 수 있습니다.

해결책: `CLAUDE_CODE_PLUGIN_GIT_TIMEOUT_MS` 환경 변수를 사용하여 시간 초과를 늘립니다. 값은 밀리초 단위입니다:

```
export CLAUDE_CODE_PLUGIN_GIT_TIMEOUT_MS=300000  # 5분
```

상대 경로가 있는 플러그인이 URL 기반 마켓플레이스에서 실패합니다

증상: URL을 통해 마켓플레이스를 추가했습니다(예: `https://example.com/marketplace.json`). 하지만 `"./plugins/my-plugin"`과 같은 상대 경로 소스가 있는 플러그인이 "path not found" 오류로 설치되지 않습니다.

원인: URL 기반 마켓플레이스는 `marketplace.json` 파일 자체만 다운로드합니다. 서버에서 플러그인 파일을 다운로드하지 않습니다. 마켓플레이스 항목의 상대 경로는 다운로드되지 않은 원격 서버의 파일을 참조합니다.

해결책:

- **외부 소스 사용**: 플러그인 항목을 상대 경로 대신 GitHub, npm 또는 git URL 소스를 사용하도록 변경합니다:

```
{ "name": "my-plugin", "source": { "source": "github", "repo": "owner/repo" } }
```

- **Git 기반 마켓플레이스 사용**: 마켓플레이스를 Git 저장소에서 호스팅하고 git URL로 추가합니다. Git 기반 마켓플레이스는 전체 저장소를 복제하므로 상대 경로가 올바르게 작동합니다.

설치 후 파일을 찾을 수 없음

증상: 플러그인이 설치되지만 파일 참조가 실패합니다. 특히 플러그인 디렉터리 외부의 파일

원인: 플러그인은 제자리에 사용되지 않고 캐시 디렉터리에 복사됩니다. 플러그인 디렉터리 외부의 파일을 참조하는 경로(예: `../shared-utils`)는 해당 파일이 복사되지 않기 때문에 작동하지 않습니다.

해결책: symlink 및 디렉터리 재구성을 포함한 해결 방법은 플러그인 캐싱 및 파일 해석을 참조하세요.

추가 디버깅 도구 및 일반적인 문제는 디버깅 및 개발 도구를 참조하세요.

참고 항목

- 미리 빌드된 플러그인 검색 및 설치 - 기존 마켓플레이스에서 플러그인 설치
- 플러그인 - 자신의 플러그인 생성
- 플러그인 참조 - 완전한 기술 사양 및 스키마
- 플러그인 설정 - 플러그인 구성 옵션
- strictKnownMarketplaces 참조 - 관리되는 마켓플레이스 제한

마켓플레이스를 통해 미리 빌드된 플러그인 발견 및 설치

> 마켓플레이스에서 플러그인을 찾아 설치하여 Claude Code를 새로운 명령어, 에이전트 및 기능으로 확장합니다.

플러그인은 Claude Code를 skills, agents, hooks 및 MCP servers로 확장합니다. 플러그인 마켓플레이스는 직접 빌드하지 않고도 이러한 확장 기능을 발견하고 설치할 수 있도록 도와주는 카탈로그입니다.

자신의 마켓플레이스를 만들고 배포하려고 하시나요? 플러그인 마켓플레이스 만들기 및 배포를 참조하세요.

마켓플레이스 작동 방식

마켓플레이스는 다른 사람이 만들어 공유한 플러그인의 카탈로그입니다. 마켓플레이스를 사용하는 것은 두 단계의 프로세스입니다:

Step 1: 마켓플레이스 추가

이는 카탈로그를 Claude Code에 등록하여 사용 가능한 항목을 검색할 수 있도록 합니다. 아직 플러그인이 설치되지 않습니다.

Step 2: 개별 플러그인 설치

카탈로그를 검색하고 원하는 플러그인을 설치합니다.

앱 스토어를 추가하는 것과 같다고 생각하면 됩니다. 스토어를 추가하면 해당 컬렉션을 검색할 수 있지만, 여전히 개별적으로 다운로드할 앱을 선택합니다.

공식 Anthropic 마켓플레이스

공식 Anthropic 마켓플레이스(`claude-plugins-official`)는 Claude Code를 시작할 때 자동으로 사용 가능합니다. `/plugin`을 실행하고 **Discover** 탭으로 이동하여 사용 가능한 항목을 검색합니다.

공식 마켓플레이스에서 플러그인을 설치하려면:

```
/plugin install plugin-name@claude-plugins-official
```

> **Note:**

> 공식 마켓플레이스는 Anthropic에서 유지 관리합니다. 공식 마켓플레이스에 플러그인을 제출하려면 다음 앱 내 제출 양식 중 하나를 사용하세요:
>
> - **Claude.ai**: claude.ai/settings/plugins/submit
> - **Console**: platform.claude.com/plugins/submit
>
> 플러그인을 독립적으로 배포하려면 자신의 마켓플레이스를 만들고 사용자와 공유하세요.

공식 마켓플레이스에는 여러 카테고리의 플러그인이 포함되어 있습니다:

코드 인텔리전스

코드 인텔리전스 플러그인은 Claude Code의 기본 제공 LSP 도구를 활성화하여 Claude가 정의로 이동하고, 참조를 찾으며, 편집 직후 타입 오류를 볼 수 있도록 합니다. 이러한 플러그인은 Language Server Protocol 연결을 구성하며, 이는 VS Code의 코드 인텔리전스를 지원하는 동일한 기술입니다.

이러한 플러그인은 언어 서버 바이너리가 시스템에 설치되어 있어야 합니다. 이미 언어 서버가 설치되어 있으면 프로젝트를 열 때 Claude가 해당 플러그인을 설치하도록 요청할 수 있습니다.

언어	플러그인	필요한 바이너리
C/C++	`clangd-lsp`	`clangd`
C#	`csharp-lsp`	`csharp-ls`
Go	`gopls-lsp`	`gopls`
Java	`jdtls-lsp`	`jdtls`
Kotlin	`kotlin-lsp`	`kotlin-language-server`
Lua	`lua-lsp`	`lua-language-server`
PHP	`php-lsp`	`intelephense`
Python	`pyright-lsp`	`pyright-langserver`
Rust	`rust-analyzer-lsp`	`rust-analyzer`
Swift	`swift-lsp`	`sourcekit-lsp`
TypeScript	`typescript-lsp`	`typescript-language-server`

다른 언어를 위한 자신의 LSP 플러그인을 만들 수도 있습니다.

> **Note:**
>
> 플러그인을 설치한 후 `/plugin` Errors 탭에서 `Executable not found in $PATH`를 보면 위 표에서 필요한 바이너리를 설치하세요.

코드 인텔리전스 플러그인이 Claude에 제공하는 것

코드 인텔리전스 플러그인이 설치되고 해당 언어 서버 바이너리를 사용할 수 있으면 Claude는 두 가지 기능을 얻습니다:

- **자동 진단**: Claude가 파일을 편집할 때마다 언어 서버는 변경 사항을 분석하고 오류 및 경고를 자동으로 보고합니다. Claude는 컴파일러나 린터를 실행할 필요 없이 타입 오류, 누락된 import 및 구문 문제를 봅니다. Claude가 오류를 도입하면 같은 턴에서 문제를 알아차리고 수정합니다. 이는 플러그인 설치 이상의 구성이 필요하지 않습니다. "진단 발견됨" 표시기가 나타날 때 **Ctrl+O**를 눌러 진단을 인라인으로 볼 수 있습니다.
- **코드 네비게이션**: Claude는 언어 서버를 사용하여 정의로 이동하고, 참조를 찾으며, 호버 시 타입 정보를 얻고, 기호를 나열하고, 구현을 찾으며, 호출 계층을 추적할 수 있습니다. 이러한 작업은 Claude에게 grep 기반 검색보다 더 정확한 네비게이션을 제공하지만, 가용성은 언어 및 환경에 따라 다를 수 있습니다.

문제가 발생하면 코드 인텔리전스 문제 해결을 참조하세요.

외부 통합

이러한 플러그인은 미리 구성된 MCP servers를 번들로 제공하므로 수동 설정 없이 Claude를 외부 서비스에 연결할 수 있습니다:

- **소스 제어**: `github`, `gitlab`
- **프로젝트 관리**: `atlassian` (Jira/Confluence), `asana`, `linear`, `notion`
- **디자인**: `figma`
- **인프라**: `vercel`, `firebase`, `supabase`
- **커뮤니케이션**: `slack`
- **모니터링**: `sentry`

개발 워크플로우

일반적인 개발 작업을 위한 명령어 및 에이전트를 추가하는 플러그인:

- **commit-commands**: commit, push 및 PR 생성을 포함한 Git commit 워크플로우
- **pr-review-toolkit**: pull request 검토를 위한 특화된 에이전트
- **agent-sdk-dev**: Claude Agent SDK로 빌드하기 위한 도구
- **plugin-dev**: 자신의 플러그인을 만들기 위한 도구 모음

출력 스타일

Claude가 응답하는 방식을 사용자 정의합니다:

- **explanatory-output-style**: 구현 선택에 대한 교육적 통찰력
- **learning-output-style**: 기술 습득을 위한 대화형 학습 모드

시도해보기: 데모 마켓플레이스 추가

Anthropic은 또한 플러그인 시스템으로 가능한 것을 보여주는 예제 플러그인이 있는 데모 플러그인 마켓플레이스(`claude-code-plugins`)를 유지 관리합니다. 공식 마켓플레이스와 달리 이 마켓플레이스는 수동으로 추가해야 합니다.

Step 1: 마켓플레이스 추가

Claude Code 내에서 `anthropics/claude-code` 마켓플레이스에 대해 `plugin marketplace add` 명령어를 실행합니다:

```
/plugin marketplace add anthropics/claude-code
```

이는 마켓플레이스 카탈로그를 다운로드하고 해당 플러그인을 사용 가능하게 합니다.

Step 2: 사용 가능한 플러그인 검색

`/plugin` 을 실행하여 플러그인 관리자를 엽니다. 이는 **Tab**(또는 뒤로 가려면 **Shift+Tab**)을 사용하여 순환할 수 있는 네 개의 탭이 있는 탭 인터페이스를 엽니다:

- **Discover**: 모든 마켓플레이스에서 사용 가능한 플러그인 검색
- **Installed**: 설치된 플러그인 보기 및 관리
- **Marketplaces**: 추가된 마켓플레이스 추가, 제거 또는 업데이트
- **Errors**: 플러그인 로딩 오류 보기

방금 추가한 마켓플레이스의 플러그인을 보려면 **Discover** 탭으로 이동합니다.

Step 3: 플러그인 설치

플러그인을 선택하여 세부 정보를 보고 설치 범위를 선택합니다:

- **User scope**: 모든 프로젝트에서 자신을 위해 설치
- **Project scope**: 이 저장소의 모든 협력자를 위해 설치
- **Local scope**: 이 저장소에서만 자신을 위해 설치

예를 들어 **commit-commands**(git 워크플로우 명령어를 추가하는 플러그인)를 선택하고 사용자 범위에 설치합니다.

명령줄에서 직접 설치할 수도 있습니다:

```
/plugin install commit-commands@anthropics-claude-code
```

범위에 대해 자세히 알아보려면 구성 범위를 참조하세요.

Step 4: 새 플러그인 사용

설치 후 플러그인의 명령어를 즉시 사용할 수 있습니다. 플러그인 명령어는 플러그인 이름으로 네임스페이스되므로 **commit-commands**는 `/commit-commands:commit` 과 같은 명령어를 제공합니다.

파일을 변경하고 다음을 실행하여 시도해보세요:

```
/commit-commands:commit
```

이는 변경 사항을 스테이징하고, commit 메시지를 생성하며, commit을 만듭니다.

각 플러그인은 다르게 작동합니다. **Discover** 탭의 플러그인 설명이나 해당 홈페이지를 확인하여 제공하는 명령어 및 기능을 알아보세요.

이 가이드의 나머지 부분에서는 마켓플레이스를 추가하고, 플러그인을 설치하며, 구성을 관리하는 모든 방법을 다룹니다.

마켓플레이스 추가

`/plugin marketplace add` 명령어를 사용하여 다양한 소스에서 마켓플레이스를 추가합니다.

> **Tip:**
>
> **바로가기**: `/plugin marketplace` 대신 `/plugin market` 을 사용할 수 있으며, `remove` 대신 `rm` 을 사용할 수 있습니다.

- **GitHub 저장소**: `owner/repo` 형식(예: `anthropics/claude-code`)
- **Git URL**: 모든 git 저장소 URL(GitLab, Bitbucket, 자체 호스팅)
- **로컬 경로**: 디렉토리 또는 `marketplace.json` 파일에 대한 직접 경로
- **원격 URL**: 호스팅된 `marketplace.json` 파일에 대한 직접 URL

GitHub에서 추가

`.claude-plugin/marketplace.json` 파일을 포함하는 GitHub 저장소를 `owner/repo` 형식을 사용하여 추가합니다. 여기서 `owner`는 GitHub 사용자 이름 또는 조직이고 `repo`는 저장소 이름입니다.

예를 들어 `anthropics/claude-code`는 `anthropics`가 소유한 `claude-code` 저장소를 나타냅니다:

```
/plugin marketplace add anthropics/claude-code
```

다른 Git 호스트에서 추가

전체 URL을 제공하여 모든 git 저장소를 추가합니다. 이는 GitLab, Bitbucket 및 자체 호스팅 서버를 포함한 모든 Git 호스트에서 작동합니다:

HTTPS 사용:

```
/plugin marketplace add https://gitlab.com/company/plugins.git
```

SSH 사용:

```
/plugin marketplace add git@gitlab.com:company/plugins.git
```

특정 브랜치 또는 태그를 추가하려면 `#` 뒤에 ref를 추가합니다:

```
/plugin marketplace add https://gitlab.com/company/plugins.git#v1.0.0
```

로컬 경로에서 추가

`.claude-plugin/marketplace.json` 파일을 포함하는 로컬 디렉토리를 추가합니다:

```
/plugin marketplace add ./my-marketplace
```

`marketplace.json` 파일에 대한 직접 경로를 추가할 수도 있습니다:

```
/plugin marketplace add ./path/to/marketplace.json
```

원격 URL에서 추가

URL을 통해 원격 `marketplace.json` 파일을 추가합니다:

```
/plugin marketplace add https://example.com/marketplace.json
```

> **Note:**
>
> URL 기반 마켓플레이스는 Git 기반 마켓플레이스에 비해 몇 가지 제한 사항이 있습니다. 플러그인 설치 시 "경로를 찾을 수 없음" 오류가 발생하면 문제 해결을 참조하세요.

플러그인 설치

마켓플레이스를 추가한 후 플러그인을 직접 설치할 수 있습니다(기본적으로 사용자 범위에 설치됨):

```
/plugin install plugin-name@marketplace-name
```

다른 설치 범위를 선택하려면 대화형 UI를 사용합니다: `/plugin` 을 실행하고 **Discover** 탭으로 이동한 후 플러그인에서 **Enter**를 누릅니다. 다음 옵션이 표시됩니다:

- **User scope**(기본값): 모든 프로젝트에서 자신을 위해 설치
- **Project scope**: 이 저장소의 모든 협력자를 위해 설치(`.claude/settings.json` 에 추가)
- **Local scope**: 이 저장소에서만 자신을 위해 설치(협력자와 공유되지 않음)

managed 범위의 플러그인도 볼 수 있습니다. 이는 관리자가 관리되는 설정을 통해 설치하며 수정할 수 없습니다.

`/plugin` 을 실행하고 **Installed** 탭으로 이동하여 범위별로 그룹화된 플러그인을 확인합니다.

> **Warning:**
>
> 플러그인을 설치하기 전에 신뢰할 수 있는지 확인하세요. Anthropic은 플러그인에 포함된 MCP servers, 파일 또는 기타 소프트웨어를 제어하지 않으며 의도한 대로 작동하는지 확인할 수 없습니다. 자세한 내용은 각 플러그인의 홈페이지를 확인하세요.

설치된 플러그인 관리

`/plugin` 을 실행하고 **Installed** 탭으로 이동하여 플러그인을 보고, 활성화하고, 비활성화하거나, 제거합니다. 플러그인 이름 또는 설명으로 목록을 필터링하려면 입력합니다.

직접 명령어로 플러그인을 관리할 수도 있습니다.

플러그인을 제거하지 않고 비활성화합니다:

```
/plugin disable plugin-name@marketplace-name
```

비활성화된 플러그인을 다시 활성화합니다:

```
/plugin enable plugin-name@marketplace-name
```

플러그인을 완전히 제거합니다:

```
/plugin uninstall plugin-name@marketplace-name
```

`--scope` 옵션을 사용하면 CLI 명령어로 특정 범위를 대상으로 할 수 있습니다:

```
claude plugin install formatter@your-org --scope project
claude plugin uninstall formatter@your-org --scope project
```

재시작 없이 플러그인 변경 사항 적용

세션 중에 플러그인을 설치, 활성화 또는 비활성화할 때 일부 변경 사항(새 명령어 및 hooks)은 즉시 적용됩니다. LSP 서버 업데이트를 포함한 다른 변경 사항은 재시작이 필요합니다.

재시작 없이 모든 보류 중인 플러그인 변경 사항을 활성화하려면 다음을 실행합니다:

```
/reload-plugins
```

Claude Code는 모든 활성 플러그인을 다시 로드하고 로드된 항목을 보고합니다. LSP 서버가 추가되거나 업데이트된 경우 적용되려면 재시작이 필요함을 알려줍니다.

마켓플레이스 관리

대화형 `/plugin` 인터페이스 또는 CLI 명령어를 통해 마켓플레이스를 관리할 수 있습니다.

대화형 인터페이스 사용

`/plugin`을 실행하고 **Marketplaces** 탭으로 이동하여:

- 소스 및 상태와 함께 추가된 모든 마켓플레이스 보기

- 새 마켓플레이스 추가
- 마켓플레이스 목록을 업데이트하여 최신 플러그인 가져오기
- 더 이상 필요하지 않은 마켓플레이스 제거

CLI 명령어 사용

직접 명령어로 마켓플레이스를 관리할 수도 있습니다.

구성된 모든 마켓플레이스 나열:

```
/plugin marketplace list
```

마켓플레이스에서 플러그인 목록 새로 고침:

```
/plugin marketplace update marketplace-name
```

마켓플레이스 제거:

```
/plugin marketplace remove marketplace-name
```

> **Warning:**
> 마켓플레이스를 제거하면 해당 마켓플레이스에서 설치한 모든 플러그인이 제거됩니다.

자동 업데이트 구성

Claude Code는 시작 시 마켓플레이스 및 설치된 플러그인을 자동으로 업데이트할 수 있습니다. 마켓플레이스에 대해 자동 업데이트가 활성화되면 Claude Code는 마켓플레이스 데이터를 새로 고치고 설치된 플러그인을 최신 버전으로 업데이트합니다. 플러그인이 업데이트된 경우 `/reload-plugins` 를 실행하도록 요청하는 알림이 표시됩니다.

UI를 통해 개별 마켓플레이스에 대한 자동 업데이트를 전환합니다:

1. `/plugin` 을 실행하여 플러그인 관리자 열기
2. **Marketplaces** 선택
3. 목록에서 마켓플레이스 선택
4. **자동 업데이트 활성화** 또는 **자동 업데이트 비활성화** 선택

공식 Anthropic 마켓플레이스는 기본적으로 자동 업데이트가 활성화되어 있습니다. 타사 및 로컬 개발 마켓플레이스는 기본적으로 자동 업데이트가 비활성화되어 있습니다.

Claude Code 및 모든 플러그인에 대해 모든 자동 업데이트를 완전히 비활성화하려면 `DISABLE_AUTOUPDATER` 환경 변수를 설정합니다. 자세한 내용은 자동 업데이트를 참조하세요.

Claude Code 자동 업데이트를 비활성화하면서 플러그인 자동 업데이트를 활성화된 상태로 유지하려면 `DISABLE_AUTOUPDATER` 와 함께 `FORCE_AUTOUPDATE_PLUGINS=true` 를 설정합니다:

```
export DISABLE_AUTOUPDATER=true
export FORCE_AUTOUPDATE_PLUGINS=true
```

Claude Code 업데이트를 수동으로 관리하지만 여전히 자동 플러그인 업데이트를 받으려는 경우에 유용합니다.

팀 마켓플레이스 구성

팀 관리자는 `.claude/settings.json` 에 마켓플레이스 구성을 추가하여 프로젝트에 대한 자동 마켓플레이스 설치를 설정할 수 있습니다. 팀 멤버가 저장소 폴더를 신뢰하면 Claude Code는 이러한 마켓플레이스 및 플러그인을 설치하도록 요청합니다.

프로젝트의 `.claude/settings.json` 에 `extraKnownMarketplaces` 를 추가합니다:

```
{
  "extraKnownMarketplaces": {
    "my-team-tools": {
      "source": {
        "source": "github",
        "repo": "your-org/claude-plugins"
      }
    }
  }
}
```

`extraKnownMarketplaces` 및 `enabledPlugins` 를 포함한 전체 구성 옵션은 플러그인 설정을 참조하세요.

보안

플러그인 및 마켓플레이스는 사용자 권한으로 머신에서 임의의 코드를 실행할 수 있는 매우 신뢰할 수 있는 구성 요소입니다. 신뢰할 수 있는 소스에서만 플러그인을 설치하고 마켓플레이스를 추가합니다. 조직은 관리되는 마켓플레이스 제한을 사용하여 사용자가 추가할 수 있는 마켓플레이스를 제한할 수 있습니다.

문제 해결

/plugin 명령어를 인식하지 못함

"알 수 없는 명령어" 또는 `/plugin` 명령어가 나타나지 않으면:

1. **버전 확인**: `claude --version` 을 실행합니다. 플러그인은 버전 1.0.33 이상이 필요합니다.
2. **Claude Code 업데이트**:
   - **Homebrew**: `brew upgrade claude-code`
   - **npm**: `npm update -g @anthropic-ai/claude-code`
   - **네이티브 설치 프로그램**: 설정에서 설치 명령어를 다시 실행합니다.
3. **Claude Code 재시작**: 업데이트 후 터미널을 재시작하고 `claude` 를 다시 실행합니다.

일반적인 문제

- **마켓플레이스가 로드되지 않음**: URL에 액세스할 수 있고 `.claude-plugin/marketplace.json` 이 경로에 있는지 확인합니다.
- **플러그인 설치 실패**: 플러그인 소스 URL에 액세스할 수 있고 저장소가 공개되어 있거나(또는 액세스 권한이 있는지) 확인합니다.
- **설치 후 파일을 찾을 수 없음**: 플러그인은 캐시에 복사되므로 플러그인 디렉토리 외부의 파일을 참조하는 경로는 작동하지 않습니다.
- **플러그인 skills가 나타나지 않음**: `rm -rf ~/.claude/plugins/cache` 로 캐시를 지우고, Claude Code를 재시작한 후 플러그인을 다시 설치합니다.

자세한 문제 해결 및 솔루션은 마켓플레이스 가이드의 문제 해결을 참조하세요. 디버깅 도구는 디버깅 및 개발 도구를 참조하세요.

코드 인텔리전스 문제

- **언어 서버가 시작되지 않음**: 바이너리가 설치되어 있고 `$PATH` 에서 사용 가능한지 확인합니다. `/plugin` Errors 탭에서 세부 정보를 확인합니다.
- **높은 메모리 사용량**: `rust-analyzer` 및 `pyright` 와 같은 언어 서버는 대규모 프로젝트에서 상당한 메모리를 소비할 수 있습니다. 메모리 문제가 발생하면 `/plugin disable <plugin-name>` 으로 플러그인을 비활성화하고 대신 Claude의 기본 제공 검색 도구를 사용합니다.
- **모노레포에서 거짓 양성 진단**: 작업 공간이 올바르게 구성되지 않으면 언어 서버가 내부 패키지에 대해 해결되지 않은 import 오류를 보고할 수 있습니다. 이는 Claude의 코드 편집 능력에 영향을 주지 않습니다.

다음 단계

- **자신의 플러그인 빌드**: 플러그인을 참조하여 skills, agents 및 hooks를 만듭니다.

- **마켓플레이스 만들기**: 플러그인 마켓플레이스 만들기를 참조하여 팀 또는 커뮤니티에 플러그인을 배포합니다.
- **기술 참조**: 플러그인 참조를 참조하여 완전한 사양을 확인합니다.

Part 6: Advanced & Automation

사용자 정의 subagent 만들기

> Claude Code에서 작업별 워크플로우 및 향상된 컨텍스트 관리를 위해 특화된 AI subagent를 만들고 사용합니다.

Subagent는 특정 유형의 작업을 처리하는 특화된 AI 어시스턴트입니다. 각 subagent는 자체 컨텍스트 윈도우에서 실행되며 사용자 정의 시스템 프롬프트, 특정 도구 액세스 및 독립적인 권한을 가집니다. Claude가 subagent의 설명과 일치하는 작업을 만나면 해당 subagent에 위임하고, subagent는 독립적으로 작동하여 결과를 반환합니다.

> **Note:**
>
> 여러 에이전트가 병렬로 작동하고 서로 통신해야 하는 경우 agent teams를 참조하십시오. Subagent는 단일 세션 내에서 작동하고, agent team은 별도의 세션 간에 조정합니다.

Subagent는 다음을 도와줍니다:

- **컨텍스트 보존** - 탐색 및 구현을 주 대화에서 분리하여 유지
- **제약 조건 적용** - subagent가 사용할 수 있는 도구 제한
- **구성 재사용** - 사용자 수준 subagent를 통해 프로젝트 간 구성 재사용
- **동작 특화** - 특정 도메인을 위한 집중된 시스템 프롬프트
- **비용 제어** - Haiku와 같은 더 빠르고 저렴한 모델로 작업 라우팅

Claude는 각 subagent의 설명을 사용하여 작업을 위임할 시기를 결정합니다. Subagent를 만들 때 Claude가 언제 사용할지 알 수 있도록 명확한 설명을 작성하십시오.

Claude Code에는 **Explore**, **Plan**, **general-purpose**와 같은 여러 내장 subagent가 포함되어 있습니다. 특정 작업을 처리하기 위해 사용자 정의 subagent를 만들 수도 있습니다. 이 페이지에서는 내장 subagent, 자신의 subagent를 만드는 방법, 전체 구성 옵션, subagent 작업 패턴, 예제 subagent를 다룹니다.

내장 subagent

Claude Code에는 Claude가 적절할 때 자동으로 사용하는 내장 subagent가 포함되어 있습니다. 각각은 부모 대화의 권한을 상속하며 추가 도구 제한이 있습니다.

Explore

코드베이스 검색 및 분석에 최적화된 빠른 읽기 전용 에이전트입니다.

- **모델**: Haiku (빠름, 낮은 지연시간)
- **도구**: 읽기 전용 도구 (Write 및 Edit 도구에 대한 액세스 거부)
- **목적**: 파일 검색, 코드 검색, 코드베이스 탐색

Claude는 변경 없이 코드베이스를 검색하거나 이해해야 할 때 Explore에 위임합니다. 이렇게 하면 탐색 결과가 주 대화 컨텍스트에서 분리됩니다.

Explore를 호출할 때 Claude는 철저함 수준을 지정합니다: 대상 조회의 경우 **quick**, 균형 잡힌 탐색의 경우 **medium**, 포괄적인 분석의 경우 **very thorough**.

Plan

계획을 제시하기 전에 컨텍스트를 수집하기 위해 plan mode 중에 사용되는 연구 에이전트입니다.

- **모델**: 주 대화에서 상속
- **도구**: 읽기 전용 도구 (Write 및 Edit 도구에 대한 액세스 거부)
- **목적**: 계획을 위한 코드베이스 연구

plan mode에 있고 Claude가 코드베이스를 이해해야 할 때 연구를 Plan subagent에 위임합니다. 이렇게 하면 무한 중첩을 방지하면서(subagent는 다른 subagent를 생성할 수 없음) 필요한 컨텍스트를 수집합니다.

General-purpose

탐색과 작업 모두를 필요로 하는 복잡한 다단계 작업을 위한 유능한 에이전트입니다.

- **모델**: 주 대화에서 상속
- **도구**: 모든 도구
- **목적**: 복잡한 연구, 다단계 작업, 코드 수정

Claude는 작업이 탐색과 수정 모두를 필요로 하거나, 결과를 해석하기 위한 복잡한 추론이 필요하거나, 여러 종속 단계가 필요할 때 general-purpose에 위임합니다.

Other

Claude Code에는 특정 작업을 위한 추가 도우미 에이전트가 포함되어 있습니다. 이들은 일반적으로 자동으로 호출되므로 직접 사용할 필요가 없습니다.

에이전트	모델	Claude가 사용하는 경우
Bash	상속	별도의 컨텍스트에서 터미널 명령 실행
statusline-setup	Sonnet	`/statusline` 을 실행하여 상태 표시줄을 구성할 때
Claude Code Guide	Haiku	Claude Code 기능에 대한 질문을 할 때

이러한 내장 subagent 외에도 사용자 정의 프롬프트, 도구 제한, 권한 모드, hooks, skills를 사용하여 자신의 subagent를 만들 수 있습니다. 다음 섹션에서는 시작하는 방법과 subagent를 사용자 정의하는 방법을 보여줍니다.

빠른 시작: 첫 번째 subagent 만들기

Subagent는 YAML frontmatter가 있는 Markdown 파일로 정의됩니다. 수동으로 만들거나 `/agents` 명령을 사용할 수 있습니다.

이 연습에서는 `/agent` 명령을 사용하여 사용자 수준 subagent를 만드는 과정을 안내합니다. Subagent는 코드를 검토하고 코드베이스에 대한 개선 사항을 제안합니다.

Step 1: subagent 인터페이스 열기

Claude Code에서 다음을 실행합니다:

```
/agents
```

Step 2: 새 사용자 수준 에이전트 만들기

Create new agent를 선택한 다음 **User-level**을 선택합니다. 이렇게 하면 subagent가 `~/.claude/agents/` 에 저장되어 모든 프로젝트에서 사용할 수 있습니다.

Step 3: Claude로 생성

Generate with Claude를 선택합니다. 메시지가 표시되면 subagent를 설명합니다:

```
A code improvement agent that scans files and suggests improvements
for readability, performance, and best practices. It should explain
each issue, show the current code, and provide an improved version.
```

Claude가 시스템 프롬프트와 구성을 생성합니다. 사용자 정의하려면 `e` 를 눌러 편집기에서 엽니다.

Step 4: 도구 선택

읽기 전용 검토자의 경우 **Read-only tools**를 제외한 모든 항목을 선택 해제합니다. 모든 도구를 선택한 상태로 유지하면 subagent는 주 대화에서 사용 가능한 모든 도구를 상속합니다.

Step 5: 모델 선택

subagent가 사용할 모델을 선택합니다. 이 예제 에이전트의 경우 코드 패턴 분석을 위해 기능과 속도의 균형을 맞추는 **Sonnet**을 선택합니다.

Step 6: 색상 선택

subagent의 배경색을 선택합니다. 이렇게 하면 UI에서 실행 중인 subagent를 식별하는 데 도움이 됩니다.

Step 7: 저장 및 시도

subagent를 저장합니다. 즉시 사용 가능합니다(재시작 필요 없음). 시도해봅니다:

```
Use the code-improver agent to suggest improvements in this project
```

Claude가 새 subagent에 위임하고, subagent는 코드베이스를 스캔하여 개선 제안을 반환합니다.

이제 머신의 모든 프로젝트에서 코드베이스를 분석하고 개선 사항을 제안하는 데 사용할 수 있는 subagent가 있습니다.

Markdown 파일로 수동으로 subagent를 만들거나, CLI 플래그를 통해 정의하거나, 플러그인을 통해 배포할 수도 있습니다. 다음 섹션에서는 모든 구성 옵션을 다룹니다.

Subagent 구성

/agents 명령 사용

`/agents` 명령은 subagent를 관리하기 위한 대화형 인터페이스를 제공합니다. `/agents` 를 실행하여:

- 사용 가능한 모든 subagent 보기 (내장, 사용자, 프로젝트, 플러그인)
- 안내된 설정 또는 Claude 생성으로 새 subagent 만들기
- 기존 subagent 구성 및 도구 액세스 편집
- 사용자 정의 subagent 삭제
- 중복이 있을 때 활성 subagent 확인

이것이 subagent를 만들고 관리하는 권장 방법입니다. 수동 생성 또는 자동화의 경우 subagent 파일을 직접 추가할 수도 있습니다.

대화형 세션을 시작하지 않고 명령줄에서 구성된 모든 subagent를 나열하려면 `claude agents`를 실행합니다. 이렇게 하면 에이전트가 소스별로 그룹화되고 더 높은 우선순위 정의로 재정의되는 에이전트가 표시됩니다.

Subagent 범위 선택

Subagent는 YAML frontmatter가 있는 Markdown 파일입니다. 범위에 따라 다른 위치에 저장합니다. 여러 subagent가 같은 이름을 공유할 때 더 높은 우선순위 위치가 우선합니다.

위치	범위	우선순위	만드는 방법
`--agents` CLI 플래그	현재 세션	1 (최고)	Claude Code 시작 시 JSON 전달
`.claude/agents/`	현재 프로젝트	2	대화형 또는 수동
`~/.claude/agents/`	모든 프로젝트	3	대화형 또는 수동
플러그인의 `agents/` 디렉토리	플러그인이 활성화된 위치	4 (최저)	플러그인과 함께 설치

프로젝트 subagent (`.claude/agents/`)는 코드베이스에 특정한 subagent에 이상적입니다. 버전 제어에 체크인하여 팀이 협력적으로 사용하고 개선할 수 있습니다.

사용자 subagent (`~/.claude/agents/`)는 모든 프로젝트에서 사용 가능한 개인 subagent입니다.

CLI 정의 subagent는 Claude Code를 시작할 때 JSON으로 전달됩니다. 해당 세션에만 존재하며 디스크에 저장되지 않으므로 빠른 테스트 또는 자동화 스크립트에 유용합니다:

```
claude --agents '{
  "code-reviewer": {
    "description": "Expert code reviewer. Use proactively after code changes.",
    "prompt": "You are a senior code reviewer. Focus on code quality, security,
and best practices.",
    "tools": ["Read", "Grep", "Glob", "Bash"],
    "model": "sonnet"
  }
}'
```

`--agents` 플래그는 파일 기반 subagent와 동일한 frontmatter 필드를 가진 JSON을 허용합니다: `description`, `prompt`, `tools`, `disallowedTools`, `model`, `permissionMode`, `mcpServers`, `hooks`, `maxTurns`, `skills`, `memory`. 시스템 프롬프트에는 `prompt` 를 사용하며, 이는 파일 기반 subagent의 markdown 본문과 동등합니다. 전체 JSON 형식은 CLI 참조를 참조하십시오.

플러그인 subagent는 설치한 플러그인에서 제공됩니다. 사용자 정의 subagent와 함께 `/agents` 에 나타납니다. 플러그인 subagent 만드는 방법에 대한 자세한 내용은 플러그인 컴포넌트 참조를 참조하십시오.

Subagent 파일 작성

Subagent 파일은 구성을 위한 YAML frontmatter를 사용하고 그 뒤에 Markdown의 시스템 프롬프트가 옵니다:

> **Note:**
>
> Subagent는 세션 시작 시 로드됩니다. 파일을 수동으로 추가하여 subagent를 만드는 경우 세션을 다시 시작하거나 `/agents` 를 사용하여 즉시 로드합니다.

```
---
name: code-reviewer
description: Reviews code for quality and best practices
tools: Read, Glob, Grep
model: sonnet
---

You are a code reviewer. When invoked, analyze the code and provide
specific, actionable feedback on quality, security, and best practices.
```

Frontmatter는 subagent의 메타데이터와 구성을 정의합니다. 본문은 subagent의 동작을 안내하는 시스템 프롬프트가 됩니다. Subagent는 이 시스템 프롬프트만 받습니다(작업 디렉토리와 같은 기본 환경 세부 정보 포함). 전체 Claude Code 시스템 프롬프트는 받지 않습니다.

지원되는 frontmatter 필드

다음 필드를 YAML frontmatter에서 사용할 수 있습니다. `name`과 `description`만 필수입니다.

필드	필수	설명
`name`	예	소문자 및 하이픈을 사용하는 고유 식별자
`description`	예	Claude가 이 subagent에 위임해야 할 때
`tools`	아니오	Subagent가 사용할 수 있는 도구. 생략하면 모든 도구 상속
`disallowedTools`	아니오	거부할 도구, 상속되거나 지정된 목록에서 제거됨
`model`	아니오	사용할 모델: `sonnet`, `opus`, `haiku`, 또는 `inherit`. 기본값은 `inherit`

필드	필수	설명
`permissionMode`	아니오	권한 모드: `default`, `acceptEdits`, `dontAsk`, `bypassPermissions`, 또는 `plan`
`maxTurns`	아니오	subagent가 중지되기 전의 최대 에이전트 턴 수
`skills`	아니오	시작 시 subagent의 컨텍스트에 로드할 skills. 전체 skill 콘텐츠가 주입되며, 호출을 위해 사용 가능하게 만들어지지 않습니다. Subagent는 부모 대화에서 skill을 상속하지 않습니다
`mcpServers`	아니오	이 subagent에서 사용 가능한 MCP servers. 각 항목은 이미 구성된 서버를 참조하는 서버 이름(예: `"slack"`) 또는 서버 이름을 키로 하고 전체 MCP server config를 값으로 하는 인라인 정의입니다
`hooks`	아니오	이 subagent로 범위가 지정된 라이프사이클 hooks
`memory`	아니오	지속적 메모리 범위: `user`, `project`, 또는 `local`. 교차 세션 학습 활성화
`background`	아니오	이 subagent를 항상 배경 작업으로 실행하려면 `true`로 설정합니다. 기본값: `false`
`isolation`	아니오	Subagent를 임시 git worktree에서 실행하려면 `worktree`로 설정하여 리포지토리의 격리된 복사본을 제공합니다. Subagent가 변경하지 않으면 worktree가 자동으로 정리됩니다

모델 선택

`model` 필드는 subagent가 사용하는 AI 모델을 제어합니다:

- **모델 별칭**: 사용 가능한 별칭 중 하나를 사용합니다: `sonnet`, `opus`, 또는 `haiku`
- **inherit**: 주 대화와 동일한 모델 사용
- **생략됨**: 지정하지 않으면 기본값은 `inherit` 입니다(주 대화와 동일한 모델 사용).

Subagent 기능 제어

도구 액세스, 권한 모드, 조건부 규칙을 통해 subagent가 할 수 있는 작업을 제어할 수 있습니다.

사용 가능한 도구

Subagent는 Claude Code의 내부 도구 중 하나를 사용할 수 있습니다. 기본적으로 subagent는 MCP 도구를 포함하여 주 대화에서 모든 도구를 상속합니다.

도구를 제한하려면 `tools` 필드(허용 목록) 또는 `disallowedTools` 필드(거부 목록)를 사용합니다:

```
---
name: safe-researcher
description: Research agent with restricted capabilities
tools: Read, Grep, Glob, Bash
disallowedTools: Write, Edit
---
```

생성할 수 있는 subagent 제한

에이전트가 `claude --agent` 를 사용하여 주 스레드로 실행될 때 Agent 도구를 사용하여 subagent를 생성할 수 있습니다. 생성할 수 있는 subagent 유형을 제한하려면 `tools` 필드에서 `Agent(agent_type)` 구문을 사용합니다.

> **Note:**
>
> 버전 2.1.63에서 Task 도구의 이름이 Agent로 변경되었습니다. 설정 및 에이전트 정의의 기존 `Task(...)` 참조는 별칭으로 계속 작동합니다.

```
---
name: coordinator
description: Coordinates work across specialized agents
tools: Agent(worker, researcher), Read, Bash
---
```

이것은 허용 목록입니다: `worker` 및 `researcher` subagent만 생성할 수 있습니다. 에이전트가 다른 유형을 생성하려고 하면 요청이 실패하고 에이전트는 프롬프트에서 허용된 유형만 봅니다. 다른 모든 에이전트를 허용하면서 특정 에이전트를 차단하려면 `permissions.deny` 를 대신 사용합니다.

제한 없이 모든 subagent를 생성할 수 있도록 허용하려면 괄호 없이 `Agent` 를 사용합니다:

```
tools: Agent, Read, Bash
```

`Agent` 가 `tools` 목록에서 완전히 생략되면 에이전트는 subagent를 생성할 수 없습니다. 이 제한은 `claude --agent` 를 사용하여 주 스레드로 실행되는 에이전트에만 적용됩니다.
Subagent는 다른 subagent를 생성할 수 없으므로 `Agent(agent_type)` 은 subagent 정의에서 효과가 없습니다.

권한 모드

`permissionMode` 필드는 subagent가 권한 프롬프트를 처리하는 방식을 제어합니다.
Subagent는 주 대화에서 권한 컨텍스트를 상속하지만 모드를 재정의할 수 있습니다.

모드	동작
`default`	프롬프트를 사용한 표준 권한 확인
`acceptEdits`	파일 편집 자동 수락
`dontAsk`	권한 프롬프트 자동 거부 (명시적으로 허용된 도구는 계속 작동)
`bypassPermissions`	모든 권한 확인 건너뛰기
`plan`	Plan mode (읽기 전용 탐색)

> **Warning:**
>
> `bypassPermissions` 를 주의해서 사용하십시오. 모든 권한 확인을 건너뛰어 subagent가 승인 없이 모든 작업을 실행할 수 있습니다.

부모가 `bypassPermissions`를 사용하면 이것이 우선하며 재정의할 수 없습니다.

Subagent에 skill 미리 로드

`skills` 필드를 사용하여 시작 시 subagent의 컨텍스트에 skill 콘텐츠를 주입합니다. 이렇게 하면 실행 중에 skill을 검색하고 로드할 필요 없이 subagent에 도메인 지식을 제공합니다.

```
---
name: api-developer
description: Implement API endpoints following team conventions
skills:
  - api-conventions
  - error-handling-patterns
---

Implement API endpoints. Follow the conventions and patterns from the preloaded
skills.
```

각 skill의 전체 콘텐츠가 subagent의 컨텍스트에 주입되며, 호출을 위해 사용 가능하게 만들어지지 않습니다. Subagent는 부모 대화에서 skill을 상속하지 않으므로 명시적으로 나열해야 합니다.

> **Note:**
>
> 이것은 subagent에서 skill 실행의 역입니다. Subagent에서 `skills`를 사용하면 subagent가 시스템 프롬프트를 제어하고 skill 콘텐츠를 로드합니다. Skill에서 `context: fork`를 사용하면 skill 콘텐츠가 지정한 에이전트에 주입됩니다. 둘 다 동일한 기본 시스템을 사용합니다.

지속적 메모리 활성화

`memory` 필드는 subagent에 대화 간에 유지되는 지속적 디렉토리를 제공합니다. Subagent는 이 디렉토리를 사용하여 코드베이스 패턴, 디버깅 통찰력, 아키텍처 결정과 같은 지식을 시간에 따라 구축합니다.

```
---
name: code-reviewer
description: Reviews code for quality and best practices
memory: user
---

You are a code reviewer. As you review code, update your agent memory with
patterns, conventions, and recurring issues you discover.
```

메모리가 얼마나 광범위하게 적용되어야 하는지에 따라 범위를 선택합니다:

범위	위치	사용 시기
`user`	`~/.claude/agent-memory/<name-of-agent>/`	subagent가 모든 프로젝트에서 학습을 기억해야 할 때
`project`	`.claude/agent-memory/<name-of-agent>/`	subagent의 지식이 프로젝트별이고 버전 제어를 통해 공유 가능할 때
`local`	`.claude/agent-memory-local/<name-of-agent>/`	subagent의 지식이 프로젝트별이지만 버전 제어에 체크인되지 않아야 할 때

메모리가 활성화되면:

- Subagent의 시스템 프롬프트에는 메모리 디렉토리 읽기 및 쓰기 지침이 포함됩니다.
- Subagent의 시스템 프롬프트에는 메모리 디렉토리의 `MEMORY.md` 의 처음 200줄이 포함되며, `MEMORY.md` 가 200줄을 초과하면 큐레이션하도록 지침이 포함됩니다.
- Read, Write, Edit 도구가 자동으로 활성화되어 subagent가 메모리 파일을 관리할 수 있습니다.

지속적 메모리 팁

- `user` 는 권장되는 기본 범위입니다. Subagent의 지식이 특정 코드베이스에만 관련이 있을 때 `project` 또는 `local` 을 사용합니다.
- Subagent에 작업을 시작하기 전에 메모리를 확인하도록 요청합니다: “Review this PR, and check your memory for patterns you’ ve seen before.”
- Subagent에 작업을 완료한 후 메모리를 업데이트하도록 요청합니다: “Now that you’ re done, save what you learned to your memory.” 시간이 지남에 따라 이렇게 하면 subagent를 더 효과적으로 만드는 지식 기반이 구축됩니다.

- Subagent가 자신의 지식 기반을 적극적으로 유지하도록 메모리 지침을 subagent의 markdown 파일에 직접 포함합니다:

```
Update your agent memory as you discover codepaths, patterns, library
locations, and key architectural decisions. This builds up institutional
knowledge across conversations. Write concise notes about what you found
and where.
```

Hook을 사용한 조건부 규칙

도구 사용을 더 동적으로 제어하려면 `PreToolUse` hook을 사용하여 실행 전에 작업을 검증합니다. 도구의 일부 작업은 허용하면서 다른 작업은 차단해야 할 때 유용합니다.

이 예제는 읽기 전용 데이터베이스 쿼리만 허용하는 subagent를 만듭니다. `PreToolUse` hook은 각 Bash 명령이 실행되기 전에 `command`에 지정된 스크립트를 실행합니다:

```
---
name: db-reader
description: Execute read-only database queries
tools: Bash
hooks:
  PreToolUse:
    - matcher: "Bash"
      hooks:
        - type: command
          command: "./scripts/validate-readonly-query.sh"
---
```

Claude Code는 hook 입력을 JSON으로 stdin을 통해 hook 명령에 전달합니다. 검증 스크립트는 이 JSON을 읽고 Bash 명령을 추출하며 쓰기 작업을 차단하기 위해 exit code 2로 종료합니다:

```
#!/bin/bash
## ./scripts/validate-readonly-query.sh

INPUT=$(cat)
COMMAND=$(echo "$INPUT" | jq -r '.tool_input.command // empty')

## Block SQL write operations (case-insensitive)
if echo "$COMMAND" | grep -iE '\b(INSERT|UPDATE|DELETE|DROP|CREATE|ALTER|TRUNCATE)
\b' > /dev/null; then
  echo "Blocked: Only SELECT queries are allowed" >&2
  exit 2
fi

exit 0
```

전체 입력 스키마는 Hook input을 참조하고 exit code가 동작에 미치는 영향은 exit codes를 참조하십시오.

특정 subagent 비활성화

설정의 `deny` 배열에 추가하여 Claude가 특정 subagent를 사용하지 못하도록 할 수 있습니다. `Agent(subagent-name)` 형식을 사용합니다. 여기서 `subagent-name` 은 subagent의 name 필드와 일치합니다.

```
{
  "permissions": {
    "deny": ["Agent(Explore)", "Agent(my-custom-agent)"]
  }
}
```

이것은 내장 및 사용자 정의 subagent 모두에 작동합니다. `--disallowedTools` CLI 플래그를 사용할 수도 있습니다:

```
claude --disallowedTools "Agent(Explore)"
```

권한 규칙에 대한 자세한 내용은 권한 설명서를 참조하십시오.

Subagent에 대한 hook 정의

Subagent는 subagent의 라이프사이클 중에 실행되는 hooks를 정의할 수 있습니다. Hook을 구성하는 두 가지 방법이 있습니다:

1. **Subagent의 frontmatter에서**: 해당 subagent가 활성화된 동안만 실행되는 hook 정의
2. **`settings.json`에서**: Subagent가 시작되거나 중지될 때 주 세션에서 실행되는 hook 정의

Subagent frontmatter의 hook

Subagent의 markdown 파일에 직접 hook을 정의합니다. 이러한 hook은 해당 특정 subagent가 활성화된 동안만 실행되고 완료되면 정리됩니다.

모든 hook 이벤트가 지원됩니다. Subagent에 가장 일반적인 이벤트는:

이벤트	Matcher 입력	실행 시기
`PreToolUse`	도구 이름	Subagent가 도구를 사용하기 전
`PostToolUse`	도구 이름	Subagent가 도구를 사용한 후
`Stop`	(없음)	Subagent가 완료될 때 (런타임에 `SubagentStop`으로 변환됨)

이 예제는 `PreToolUse` hook으로 Bash 명령을 검증하고 `PostToolUse`로 파일 편집 후 linter를 실행합니다:

```
---
name: code-reviewer
description: Review code changes with automatic linting
hooks:
  PreToolUse:
    - matcher: "Bash"
      hooks:
        - type: command
          command: "./scripts/validate-command.sh $TOOL_INPUT"
  PostToolUse:
    - matcher: "Edit|Write"
      hooks:
        - type: command
          command: "./scripts/run-linter.sh"
---
```

Frontmatter의 `Stop` hook은 자동으로 `SubagentStop` 이벤트로 변환됩니다.

Subagent 이벤트에 대한 프로젝트 수준 hook

주 세션에서 subagent 라이프사이클 이벤트에 응답하는 `settings.json`의 hook을 구성합니다.

이벤트	Matcher 입력	실행 시기
`SubagentStart`	에이전트 유형 이름	Subagent가 실행을 시작할 때
`SubagentStop`	에이전트 유형 이름	Subagent가 완료될 때

두 이벤트 모두 이름으로 특정 에이전트 유형을 대상으로 하는 matcher를 지원합니다. 이 예제는 `db-agent` subagent가 시작될 때만 설정 스크립트를 실행하고 모든 subagent가 중지될 때 정리 스크립트를 실행합니다:

```
{
  "hooks": {
    "SubagentStart": [
      {
        "matcher": "db-agent",
        "hooks": [
          { "type": "command", "command": "./scripts/setup-db-connection.sh" }
        ]
      }
    ],
    "SubagentStop": [
      {
        "hooks": [
          { "type": "command", "command": "./scripts/cleanup-db-connection.sh" }
        ]
      }
    ]
  }
}
```

전체 hook 구성 형식은 Hooks를 참조하십시오.

Subagent 작업

자동 위임 이해

Claude는 요청의 작업 설명, subagent 구성의 `description` 필드, 현재 컨텍스트를 기반으로 자동으로 작업을 위임합니다. 적극적인 위임을 장려하려면 subagent의 description 필드에 "use proactively"와 같은 구문을 포함합니다.

특정 subagent를 명시적으로 요청할 수도 있습니다:

```
Use the test-runner subagent to fix failing tests
Have the code-reviewer subagent look at my recent changes
```

Subagent를 포그라운드 또는 백그라운드에서 실행

Subagent는 포그라운드(차단) 또는 백그라운드(동시)에서 실행할 수 있습니다:

- **포그라운드 subagent**는 완료될 때까지 주 대화를 차단합니다. 권한 프롬프트 및 명확한 질문(예: `AskUserQuestion`)이 사용자에게 전달됩니다.
- **백그라운드 subagent**는 작업을 계속하는 동안 동시에 실행됩니다. 시작하기 전에 Claude Code는 subagent가 필요로 할 도구 권한을 요청하여 필요한 승인이 있는지 확인합니다. 실행되면 subagent는 이러한 권한을 상속하고 사전 승인되지 않은 모든 것을 자동으로 거부합니다. 백그라운드 subagent가 명확한 질문을 해야 하면 해당 도구 호출이 실패하지만 subagent는 계속됩니다.

백그라운드 subagent가 권한 부족으로 인해 실패하면 재개하여 포그라운드에서 대화형 프롬프트로 다시 시도할 수 있습니다.

Claude는 작업을 기반으로 subagent를 포그라운드 또는 백그라운드에서 실행할지 결정합니다. 다음을 할 수도 있습니다:

- Claude에 "run this in the background"를 요청
- **Ctrl+B**를 눌러 실행 중인 작업을 백그라운드로 이동

모든 백그라운드 작업 기능을 비활성화하려면 `CLAUDE_CODE_DISABLE_BACKGROUND_TASKS` 환경 변수를 `1`로 설정합니다. 환경 변수를 참조하십시오.

일반적인 패턴

대량 작업 격리

Subagent의 가장 효과적인 사용 중 하나는 많은 양의 출력을 생성하는 작업을 격리하는 것입니다. 테스트 실행, 문서 가져오기 또는 로그 파일 처리는 상당한 컨텍스트를 소비할 수 있습니다. 이를 subagent에 위임하면 자세한 출력이 subagent의 컨텍스트에 유지되고 관련 요약만 주 대화로 반환됩니다.

```
Use a subagent to run the test suite and report only the failing tests with their
error messages
```

병렬 연구 실행

독립적인 조사의 경우 여러 subagent를 동시에 작동하도록 생성합니다:

```
Research the authentication, database, and API modules in parallel using separate
subagents
```

각 subagent는 자신의 영역을 독립적으로 탐색한 다음 Claude가 결과를 종합합니다. 이것은 연구 경로가 서로 의존하지 않을 때 가장 잘 작동합니다.

> **Warning:**
>
> Subagent가 완료되면 결과가 주 대화로 반환됩니다. 각각 자세한 결과를 반환하는 많은 subagent를 실행하면 상당한 컨텍스트를 소비할 수 있습니다.

지속적인 병렬 처리가 필요하거나 컨텍스트 윈도우를 초과하는 작업의 경우 agent teams는 각 워커에 자신의 독립적인 컨텍스트를 제공합니다.

Subagent 체인

다단계 워크플로우의 경우 Claude에 subagent를 순차적으로 사용하도록 요청합니다. 각 subagent는 작업을 완료하고 결과를 Claude에 반환하고, Claude는 관련 컨텍스트를 다음 subagent에 전달합니다.

```
Use the code-reviewer subagent to find performance issues, then use the optimizer
subagent to fix them
```

Subagent와 주 대화 중 선택

다음의 경우 **주 대화**를 사용합니다:

- 작업이 빈번한 왕복 또는 반복적인 개선이 필요함
- 여러 단계가 상당한 컨텍스트를 공유함 (계획 → 구현 → 테스트)
- 빠르고 대상이 지정된 변경을 수행 중
- 지연시간이 중요함. Subagent는 새로 시작하고 컨텍스트를 수집하는 데 시간이 걸릴 수 있음

다음의 경우 **subagent**를 사용합니다:

- 작업이 주 컨텍스트에 필요하지 않은 자세한 출력을 생성함
- 특정 도구 제한 또는 권한을 적용하려고 함
- 작업이 자체 포함되어 있고 요약을 반환할 수 있음

격리된 subagent 컨텍스트가 아닌 주 대화 컨텍스트에서 실행되는 재사용 가능한 프롬프트 또는 워크플로우를 원할 때 Skills를 대신 고려합니다.

대화에 이미 있는 것에 대한 빠른 질문의 경우 subagent 대신 `/btw` 를 사용합니다. 전체 컨텍스트를 보지만 도구 액세스가 없으며 답변은 기록에 추가되지 않습니다.

> **Note:**

> Subagent는 다른 subagent를 생성할 수 없습니다. 워크플로우가 중첩된 위임이 필요한 경우 Skills 또는 주 대화에서 subagent 체인을 사용합니다.

Subagent 컨텍스트 관리

Subagent 재개

각 subagent 호출은 새로운 인스턴스를 만들고 새로운 컨텍스트를 생성합니다. 처음부터 시작하는 대신 기존 subagent의 작업을 계속하려면 Claude에 재개하도록 요청합니다.

재개된 subagent는 모든 이전 도구 호출, 결과, 추론을 포함한 전체 대화 기록을 유지합니다. Subagent는 새로 시작하는 대신 정확히 중단한 위치에서 계속됩니다.

Subagent가 완료되면 Claude는 에이전트 ID를 받습니다. Subagent를 재개하려면 Claude에 이전 작업을 계속하도록 요청합니다:

```
Use the code-reviewer subagent to review the authentication module
[Agent completes]

Continue that code review and now analyze the authorization logic
[Claude resumes the subagent with full context from previous conversation]
```

에이전트 ID를 명시적으로 참조하려면 Claude에 ID를 요청할 수도 있으며, `~/.claude/projects/{project}/{sessionId}/subagents/` 의 트랜스크립트 파일에서 ID를 찾을 수 있습니다. 각 트랜스크립트는 `agent-{agentId}.jsonl` 로 저장됩니다.

Subagent 트랜스크립트는 주 대화와 독립적으로 유지됩니다:

- **주 대화 압축**: 주 대화가 압축될 때 subagent 트랜스크립트는 영향을 받지 않습니다. 별도의 파일에 저장됩니다.
- **세션 지속성**: Subagent 트랜스크립트는 세션 내에서 유지됩니다. Claude Code를 다시 시작한 후 동일한 세션을 재개하여 subagent를 재개할 수 있습니다.
- **자동 정리**: 트랜스크립트는 `cleanupPeriodDays` 설정(기본값: 30일)을 기반으로 정리됩니다.

자동 압축

Subagent는 주 대화와 동일한 논리를 사용하여 자동 압축을 지원합니다. 기본적으로 자동 압축은 약 95% 용량에서 트리거됩니다. 압축을 더 일찍 트리거하려면 `CLAUDE_AUTOCOMPACT_PCT_OVERRIDE` 를 더 낮은 백분율(예: `50`)로 설정합니다. 자세한 내용은 환경 변수를 참조하십시오.

압축 이벤트는 subagent 트랜스크립트 파일에 기록됩니다:

```
{
  "type": "system",
  "subtype": "compact_boundary",
  "compactMetadata": {
    "trigger": "auto",
    "preTokens": 167189
  }
}
```

`preTokens` 값은 압축이 발생하기 전에 사용된 토큰 수를 보여줍니다.

예제 subagent

이러한 예제는 subagent를 구축하기 위한 효과적인 패턴을 보여줍니다. 시작점으로 사용하거나 Claude로 사용자 정의된 버전을 생성합니다.

> **Tip:**
>
> **모범 사례:**
>
> - **집중된 subagent 설계:** 각 subagent는 특정 작업에서 탁월해야 함
> - **자세한 설명 작성:** Claude는 설명을 사용하여 위임 시기를 결정함
> - **도구 액세스 제한:** 보안 및 집중을 위해 필요한 권한만 부여
> - **버전 제어에 체크인:** 팀과 프로젝트 subagent 공유

코드 검토자

코드를 수정하지 않고 검토하는 읽기 전용 subagent입니다. 이 예제는 제한된 도구 액세스(Edit 또는 Write 없음)와 정확히 무엇을 찾을지 그리고 출력을 형식화하는 방법을 지정하는 자세한 프롬프트를 사용하여 집중된 subagent를 설계하는 방법을 보여줍니다.

```
---
name: code-reviewer
description: Expert code review specialist. Proactively reviews code for quality,
security, and maintainability. Use immediately after writing or modifying code.
tools: Read, Grep, Glob, Bash
model: inherit
---

You are a senior code reviewer ensuring high standards of code quality and
security.

When invoked:
1. Run git diff to see recent changes
2. Focus on modified files
3. Begin review immediately

Review checklist:
- Code is clear and readable
- Functions and variables are well-named
- No duplicated code
- Proper error handling
- No exposed secrets or API keys
- Input validation implemented
- Good test coverage
- Performance considerations addressed

Provide feedback organized by priority:
- Critical issues (must fix)
- Warnings (should fix)
- Suggestions (consider improving)

Include specific examples of how to fix issues.
```

디버거

문제를 분석하고 수정할 수 있는 subagent입니다. 코드 검토자와 달리 이것은 버그 수정이 코드 수정을 필요로 하기 때문에 Edit을 포함합니다. 프롬프트는 진단에서 검증까지의 명확한 워크플로우를 제공합니다.

```
---
name: debugger
description: Debugging specialist for errors, test failures, and unexpected
behavior. Use proactively when encountering any issues.
tools: Read, Edit, Bash, Grep, Glob
---

You are an expert debugger specializing in root cause analysis.

When invoked:
1. Capture error message and stack trace
2. Identify reproduction steps
3. Isolate the failure location
4. Implement minimal fix
5. Verify solution works

Debugging process:
- Analyze error messages and logs
- Check recent code changes
- Form and test hypotheses
- Add strategic debug logging
- Inspect variable states

For each issue, provide:
- Root cause explanation
- Evidence supporting the diagnosis
- Specific code fix
- Testing approach
- Prevention recommendations

Focus on fixing the underlying issue, not the symptoms.
```

데이터 과학자

데이터 분석 작업을 위한 도메인별 subagent입니다. 이 예제는 일반적인 코딩 작업 외부의 특화된 워크플로우를 위해 subagent를 만드는 방법을 보여줍니다. 더 유능한 분석을 위해 명시적으로 `model: sonnet`을 설정합니다.

```
---
name: data-scientist
description: Data analysis expert for SQL queries, BigQuery operations, and data
insights. Use proactively for data analysis tasks and queries.
tools: Bash, Read, Write
model: sonnet
---

You are a data scientist specializing in SQL and BigQuery analysis.

When invoked:
1. Understand the data analysis requirement
2. Write efficient SQL queries
3. Use BigQuery command line tools (bq) when appropriate
4. Analyze and summarize results
5. Present findings clearly

Key practices:
- Write optimized SQL queries with proper filters
- Use appropriate aggregations and joins
- Include comments explaining complex logic
- Format results for readability
- Provide data-driven recommendations

For each analysis:
- Explain the query approach
- Document any assumptions
- Highlight key findings
- Suggest next steps based on data

Always ensure queries are efficient and cost-effective.
```

데이터베이스 쿼리 검증자

Bash 액세스를 허용하지만 읽기 전용 SQL 쿼리만 허용하도록 명령을 검증하는 subagent입니다. 이 예제는 `tools` 필드가 제공하는 것보다 더 세밀한 제어가 필요할 때 `PreToolUse` hook을 사용하는 방법을 보여줍니다.

```
---
name: db-reader
description: Execute read-only database queries. Use when analyzing data or
generating reports.
tools: Bash
hooks:
  PreToolUse:
    - matcher: "Bash"
      hooks:
        - type: command
          command: "./scripts/validate-readonly-query.sh"
---

You are a database analyst with read-only access. Execute SELECT queries to answer
questions about the data.

When asked to analyze data:
1. Identify which tables contain the relevant data
2. Write efficient SELECT queries with appropriate filters
3. Present results clearly with context

You cannot modify data. If asked to INSERT, UPDATE, DELETE, or modify schema,
explain that you only have read access.
```

Claude Code는 hook 입력을 JSON으로 stdin을 통해 hook 명령에 전달합니다. 검증 스크립트는 이 JSON을 읽고 실행 중인 명령을 추출하고 SQL 쓰기 작업 목록에 대해 확인합니다. 쓰기 작업이 감지되면 스크립트는 exit code 2로 종료하고 stderr를 통해 Claude에 오류 메시지를 반환합니다.

프로젝트의 어디든지 검증 스크립트를 만듭니다. 경로는 hook 구성의 `command` 필드와 일치해야 합니다:

```
#!/bin/bash
## Blocks SQL write operations, allows SELECT queries

## Read JSON input from stdin
INPUT=$(cat)

## Extract the command field from tool_input using jq
COMMAND=$(echo "$INPUT" | jq -r '.tool_input.command // empty')

if [ -z "$COMMAND" ]; then
  exit 0
fi

## Block write operations (case-insensitive)
if echo "$COMMAND" | grep -iE '\b(INSERT|UPDATE|DELETE|DROP|CREATE|ALTER|TRUNCATE|
REPLACE|MERGE)\b' > /dev/null; then
  echo "Blocked: Write operations not allowed. Use SELECT queries only." >&2
  exit 2
fi

exit 0
```

스크립트를 실행 가능하게 만듭니다:

```
chmod +x ./scripts/validate-readonly-query.sh
```

Hook은 stdin을 통해 JSON을 받으며 `tool_input.command`에 Bash 명령이 있습니다. Exit code 2는 작업을 차단하고 오류 메시지를 Claude에 다시 피드합니다. Exit code에 대한 자세한 내용은 Hooks를 참조하고 전체 입력 스키마는 Hook input을 참조하십시오.

다음 단계

이제 subagent를 이해했으므로 다음 관련 기능을 탐색합니다:

- 플러그인으로 subagent 배포 - 팀 또는 프로젝트 간에 subagent 공유
- Claude Code를 프로그래밍 방식으로 실행 - CI/CD 및 자동화를 위해 Agent SDK 사용
- MCP servers 사용 - subagent에 외부 도구 및 데이터에 대한 액세스 제공

Claude Code 세션 팀 조율하기

> 공유 작업, 에이전트 간 메시징, 중앙 집중식 관리를 통해 함께 작동하는 여러 Claude Code 인스턴스를 조율합니다.

Warning:

에이전트 팀은 실험적이며 기본적으로 비활성화되어 있습니다. settings.json이나 환경에 `CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS` 를 추가하여 활성화합니다. 에이전트 팀은 세션 재개, 작업 조율, 종료 동작 관련 알려진 제한 사항이 있습니다.

에이전트 팀을 사용하면 함께 작동하는 여러 Claude Code 인스턴스를 조율할 수 있습니다. 한 세션이 팀 리더 역할을 하여 작업을 조율하고, 작업을 할당하며, 결과를 종합합니다. 팀원들은 독립적으로 작동하며, 각각 자신의 컨텍스트 윈도우에서 작동하고, 서로 직접 통신합니다.

단일 세션 내에서 실행되고 메인 에이전트에게만 보고할 수 있는 subagents와 달리, 리더를 거치지 않고 개별 팀원과 직접 상호작용할 수도 있습니다.

Note:

에이전트 팀은 Claude Code v2.1.32 이상이 필요합니다. `claude --version` 으로 버전을 확인합니다.

이 페이지에서 다루는 내용:

- 에이전트 팀을 사용할 때, 최적의 사용 사례 및 subagents와의 비교 포함
- 팀 시작하기
- 팀원 제어하기, 표시 모드, 작업 할당, 위임 포함
- 병렬 작업 모범 사례

에이전트 팀을 사용할 때

에이전트 팀은 병렬 탐색이 실질적인 가치를 더하는 작업에 가장 효과적입니다. 전체 시나리오는 사용 사례 예시를 참조합니다. 가장 강력한 사용 사례는 다음과 같습니다:

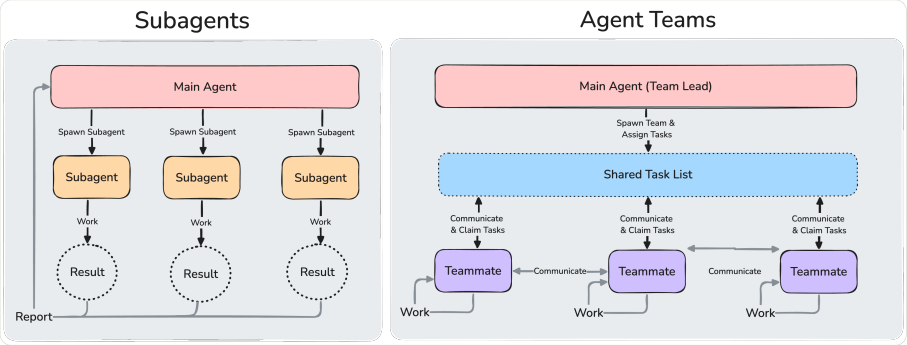
- **연구 및 검토**: 여러 팀원이 문제의 다양한 측면을 동시에 조사한 후 서로의 발견을 공유하고 도전할 수 있습니다

- **새로운 모듈 또는 기능**: 팀원들이 각각 별도의 부분을 소유하면서 서로 간섭하지 않을 수 있습니다
- **경쟁하는 가설로 디버깅하기**: 팀원들이 다양한 이론을 병렬로 테스트하고 더 빠르게 답에 수렴합니다
- **교차 계층 조율**: 프론트엔드, 백엔드, 테스트에 걸친 변경 사항으로, 각각 다른 팀원이 소유합니다

에이전트 팀은 조율 오버헤드를 추가하고 단일 세션보다 훨씬 더 많은 토큰을 사용합니다. 팀원들이 독립적으로 작동할 수 있을 때 가장 잘 작동합니다. 순차적 작업, 동일 파일 편집, 또는 많은 종속성이 있는 작업의 경우 단일 세션이나 subagents가 더 효과적입니다.

subagents와 비교

에이전트 팀과 subagents 모두 작업을 병렬화할 수 있지만, 다르게 작동합니다. 워커들이 서로 통신해야 하는지 여부에 따라 선택합니다:



Subagent와 에이전트 팀 아키텍처를 비교하는 다이어그램입니다. Subagents는 메인 에이전트에 의해 생성되고, 작업을 수행하며, 결과를 보고합니다. 에이전트 팀은 공유 작업 목록을 통해 조율되며, 팀원들이 서로 직접 통신합니다.

	Subagents	에이전트 팀
컨텍스트	자신의 컨텍스트 윈도우; 결과는 호출자에게 반환됨	자신의 컨텍스트 윈도우; 완전히 독립적
통신	메인 에이전트에게만 결과 보고	팀원들이 서로 직접 메시지 전송
조율	메인 에이전트가 모든 작업 관리	자체 조율을 통한 공유 작업 목록
최적 용도	결과만 중요한 집중된 작업	논의와 협업이 필요한 복잡한 작업

	Subagents	에이전트 팀
토큰 비용	낮음: 결과가 메인 컨텍스트로 요약됨	높음: 각 팀원이 별도의 Claude 인스턴스

결과를 보고하는 빠르고 집중된 워커가 필요할 때는 subagents를 사용합니다. 팀원들이 발견을 공유하고, 서로 도전하며, 자체적으로 조율해야 할 때는 에이전트 팀을 사용합니다.

에이전트 팀 활성화

에이전트 팀은 기본적으로 비활성화되어 있습니다. `CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS` 환경 변수를 `1`로 설정하여 활성화합니다. 셸 환경이나 settings.json을 통해 설정할 수 있습니다:

```
{
  "env": {
    "CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS": "1"
  }
}
```

첫 번째 에이전트 팀 시작하기

에이전트 팀을 활성화한 후, Claude에게 에이전트 팀을 만들도록 요청하고 자연어로 원하는 작업과 팀 구조를 설명합니다. Claude는 팀을 만들고, 팀원들을 생성하며, 프롬프트에 따라 작업을 조율합니다.

이 예시는 세 가지 역할이 독립적이고 서로를 기다리지 않고 문제를 탐색할 수 있기 때문에 잘 작동합니다:

```
I'm designing a CLI tool that helps developers track TODO comments across
their codebase. Create an agent team to explore this from different angles: one
teammate on UX, one on technical architecture, one playing devil's advocate.
```

그러면 Claude는 공유 작업 목록을 가진 팀을 만들고, 각 관점에 대한 팀원들을 생성하며, 문제를 탐색하고, 발견을 종합하며, 완료되었을 때 팀을 정리하려고 시도합니다.

리더의 터미널은 모든 팀원과 그들이 작업 중인 내용을 나열합니다. Shift+Down을 사용하여 팀원들을 순환하고 직접 메시지를 보냅니다. 마지막 팀원 이후, Shift+Down은 리더로 돌아갑니다.

각 팀원이 자신의 분할 창에 있기를 원하면 표시 모드 선택을 참조합니다.

에이전트 팀 제어하기

리더에게 자연어로 원하는 것을 말합니다. 지시에 따라 팀 조율, 작업 할당, 위임을 처리합니다.

표시 모드 선택

에이전트 팀은 두 가지 표시 모드를 지원합니다:

- **In-process**: 모든 팀원이 메인 터미널 내에서 실행됩니다. Shift+Down을 사용하여 팀원들을 순환하고 입력하여 직접 메시지를 보냅니다. 모든 터미널에서 작동하며 추가 설정이 필요하지 않습니다.
- **분할 창**: 각 팀원이 자신의 창을 가집니다. 모든 사람의 출력을 한 번에 볼 수 있고 창을 클릭하여 직접 상호작용할 수 있습니다. tmux 또는 iTerm2가 필요합니다.

> **Note:**
>
> `tmux` 는 특정 운영 체제에서 알려진 제한 사항이 있으며 전통적으로 macOS에서 가장 잘 작동합니다. iTerm2에서 `tmux -CC` 를 사용하는 것이 `tmux` 로의 권장 진입점입니다.

기본값은 `"auto"` 이며, 이미 tmux 세션 내에서 실행 중이면 분할 창을 사용하고, 그렇지 않으면 in-process를 사용합니다. `"tmux"` 설정은 분할 창 모드를 활성화하고 터미널에 따라 tmux 또는 iTerm2를 사용할지 자동으로 감지합니다. 재정의하려면 settings.json에서 `teammateMode` 를 설정합니다:

```
{
  "teammateMode": "in-process"
}
```

단일 세션에 대해 in-process 모드를 강제하려면 플래그로 전달합니다:

```
claude --teammate-mode in-process
```

분할 창 모드는 tmux 또는 `it2` CLI가 있는 iTerm2가 필요합니다. 수동으로 설치하려면:

- **tmux**: 시스템의 패키지 관리자를 통해 설치합니다. 플랫폼별 지침은 tmux wiki를 참조합니다.
- **iTerm2**: `it2` CLI를 설치한 후, **iTerm2 → Settings → General → Magic → Enable Python API**에서 Python API를 활성화합니다.

팀원 및 모델 지정

Claude는 작업에 따라 생성할 팀원의 수를 결정하거나, 정확히 원하는 것을 지정할 수 있습니다:

```
Create a team with 4 teammates to refactor these modules in parallel.
Use Sonnet for each teammate.
```

팀원을 위한 계획 승인 요구

복잡하거나 위험한 작업의 경우, 팀원들이 구현하기 전에 계획하도록 요구할 수 있습니다. 팀원은 리더가 접근 방식을 승인할 때까지 읽기 전용 계획 모드에서 작동합니다:

```
Spawn an architect teammate to refactor the authentication module.
Require plan approval before they make any changes.
```

팀원이 계획을 마치면, 리더에게 계획 승인 요청을 보냅니다. 리더는 계획을 검토하고 승인하거나 피드백과 함께 거부합니다. 거부되면, 팀원은 계획 모드에 머물러 피드백에 따라 수정하고 다시 제출합니다. 승인되면, 팀원은 계획 모드를 종료하고 구현을 시작합니다.

리더는 자율적으로 승인 결정을 내립니다. 리더의 판단에 영향을 미치려면, 프롬프트에 "테스트 커버리지를 포함하는 계획만 승인" 또는 "데이터베이스 스키마를 수정하는 계획 거부"와 같은 기준을 제공합니다.

팀원과 직접 대화하기

각 팀원은 완전하고 독립적인 Claude Code 세션입니다. 모든 팀원에게 직접 메시지를 보내 추가 지시를 제공하고, 후속 질문을 하거나, 접근 방식을 재지정할 수 있습니다.

- **In-process 모드**: Shift+Down을 사용하여 팀원들을 순환한 후 입력하여 메시지를 보냅니다. Enter를 눌러 팀원의 세션을 보고, Escape를 눌러 현재 턴을 중단합니다. Ctrl+T를 눌러 작업 목록을 전환합니다.
- **분할 창 모드**: 팀원의 창을 클릭하여 세션과 직접 상호작용합니다. 각 팀원은 자신의 터미널의 전체 보기를 가집니다.

작업 할당 및 요청

공유 작업 목록은 팀 전체의 작업을 조율합니다. 리더는 작업을 만들고 팀원들이 이를 처리합니다. 작업은 세 가지 상태를 가집니다: 대기 중, 진행 중, 완료됨. 작업은 다른 작업에 종속될 수도 있습니다: 미해결 종속성이 있는 대기 중인 작업은 해당 종속성이 완료될 때까지 요청할 수 없습니다.

리더는 작업을 명시적으로 할당하거나 팀원들이 자체 요청할 수 있습니다:

- **리더 할당**: 리더에게 어느 작업을 어느 팀원에게 줄지 말합니다

- **자체 요청**: 작업을 마친 후, 팀원은 다음 미할당, 미차단 작업을 자체적으로 선택합니다

작업 요청은 파일 잠금을 사용하여 여러 팀원이 동시에 동일한 작업을 요청하려고 할 때 경합 조건을 방지합니다.

팀원 종료하기

팀원의 세션을 우아하게 종료하려면:

```
Ask the researcher teammate to shut down
```

리더는 종료 요청을 보냅니다. 팀원은 승인하여 우아하게 종료하거나 설명과 함께 거부할 수 있습니다.

팀 정리하기

완료되었을 때, 리더에게 정리하도록 요청합니다:

```
Clean up the team
```

이는 공유 팀 리소스를 제거합니다. 리더가 정리를 실행할 때, 활성 팀원을 확인하고 여전히 실행 중이면 실패하므로 먼저 종료합니다.

> **Warning:**
> 항상 리더를 사용하여 정리합니다. 팀원들은 정리를 실행하면 안 됩니다. 팀원의 팀 컨텍스트가 올바르게 해결되지 않아 리소스가 일관성 없는 상태로 남을 수 있기 때문입니다.

hooks로 품질 게이트 적용

hooks를 사용하여 팀원들이 작업을 마치거나 작업이 완료될 때 규칙을 적용합니다:

- `TeammateIdle`: 팀원이 유휴 상태가 되려고 할 때 실행됩니다. 종료 코드 2로 종료하여 피드백을 보내고 팀원을 계속 작동하게 합니다.
- `TaskCompleted`: 작업이 완료로 표시될 때 실행됩니다. 종료 코드 2로 종료하여 완료를 방지하고 피드백을 보냅니다.

에이전트 팀이 어떻게 작동하는지

이 섹션은 에이전트 팀 뒤의 아키텍처와 메커니즘을 다룹니다. 사용을 시작하려면 위의 에이전트 팀 제어하기를 참조합니다.

Claude가 에이전트 팀을 시작하는 방법

에이전트 팀이 시작되는 두 가지 방법이 있습니다:

- **팀 요청**: 병렬 작업의 이점이 있는 작업을 제공하고 명시적으로 에이전트 팀을 요청합니다. Claude는 지시에 따라 팀을 만듭니다.
- **Claude가 팀 제안**: Claude가 작업이 병렬 작업의 이점이 있다고 판단하면, 팀 생성을 제안할 수 있습니다. 진행하기 전에 확인합니다.

두 경우 모두 제어 권한을 유지합니다. Claude는 승인 없이 팀을 만들지 않습니다.

아키텍처

에이전트 팀은 다음으로 구성됩니다:

구성 요소	역할
팀 리더	팀을 만들고, 팀원들을 생성하며, 작업을 조율하는 메인 Claude Code 세션
팀원들	할당된 작업에서 각각 작동하는 별도의 Claude Code 인스턴스
작업 목록	팀원들이 요청하고 완료하는 공유 작업 항목 목록
메일박스	에이전트 간 통신을 위한 메시징 시스템

표시 구성 옵션은 표시 모드 선택을 참조합니다. 팀원 메시지는 리더에게 자동으로 도착합니다.

시스템은 작업 종속성을 자동으로 관리합니다. 팀원이 다른 작업이 종속된 작업을 완료하면, 차단된 작업은 수동 개입 없이 차단 해제됩니다.

팀과 작업은 로컬에 저장됩니다:

- **팀 구성**: `~/.claude/teams/{team-name}/config.json`
- **작업 목록**: `~/.claude/tasks/{team-name}/`

팀 구성에는 각 팀원의 이름, 에이전트 ID, 에이전트 유형이 있는 `members` 배열이 포함됩니다. 팀원들은 이 파일을 읽어 다른 팀 멤버를 발견할 수 있습니다.

권한

팀원들은 리더의 권한 설정으로 시작합니다. 리더가 `--dangerously-skip-permissions`로 실행되면, 모든 팀원도 그렇게 합니다. 생성 후, 개별 팀원 모드를 변경할 수 있지만, 생성 시 팀원별 모드를 설정할 수 없습니다.

컨텍스트 및 통신

각 팀원은 자신의 컨텍스트 윈도우를 가집니다. 생성될 때, 팀원은 일반 세션과 동일한 프로젝트 컨텍스트를 로드합니다: CLAUDE.md, MCP servers, skills. 또한 리더의 생성 프롬프트를 받습니다. 리더의 대화 기록은 전달되지 않습니다.

팀원들이 정보를 공유하는 방법:

- **자동 메시지 전달**: 팀원들이 메시지를 보낼 때, 자동으로 수신자에게 전달됩니다. 리더가 업데이트를 폴링할 필요가 없습니다.
- **유휴 알림**: 팀원이 완료되고 중지되면, 자동으로 리더에게 알립니다.
- **공유 작업 목록**: 모든 에이전트는 작업 상태를 보고 사용 가능한 작업을 요청할 수 있습니다.

팀원 메시징:

- **message**: 특정 팀원 한 명에게 메시지 보내기
- **broadcast**: 모든 팀원에게 동시에 보내기. 팀 크기에 따라 비용이 증가하므로 드물게 사용합니다.

토큰 사용

에이전트 팀은 단일 세션보다 훨씬 더 많은 토큰을 사용합니다. 각 팀원은 자신의 컨텍스트 윈도우를 가지며, 토큰 사용은 활성 팀원의 수에 따라 증가합니다. 연구, 검토, 새로운 기능 작업의 경우, 추가 토큰은 일반적으로 가치가 있습니다. 일상적인 작업의 경우, 단일 세션이 더 비용 효율적입니다. 사용 지침은 에이전트 팀 토큰 비용을 참조합니다.

사용 사례 예시

이 예시들은 병렬 탐색이 가치를 더하는 작업을 에이전트 팀이 어떻게 처리하는지 보여줍니다.

병렬 코드 검토 실행

단일 검토자는 한 번에 한 가지 유형의 문제로 기울어지는 경향이 있습니다. 검토 기준을 독립적인 도메인으로 분할하면 보안, 성능, 테스트 커버리지가 모두 동시에 철저한 주의를 받습니다. 프롬프트는 각 팀원에게 고유한 렌즈를 할당하여 겹치지 않도록 합니다:

```
Create an agent team to review PR #142. Spawn three reviewers:
- One focused on security implications
- One checking performance impact
- One validating test coverage
Have them each review and report findings.
```

각 검토자는 동일한 PR에서 작동하지만 다른 필터를 적용합니다. 리더는 모두 완료된 후 세 명 모두의 발견을 종합합니다.

경쟁하는 가설로 조사하기

근본 원인이 불명확할 때, 단일 에이전트는 그럴듯한 설명 하나를 찾고 멈추는 경향이 있습니다. 프롬프트는 팀원들을 명시적으로 적대적으로 만들어 이를 방지합니다: 각 팀원의 일은 자신의 이론을 조사하는 것뿐만 아니라 다른 팀원들에게 도전하는 것입니다.

```
Users report the app exits after one message instead of staying connected.
Spawn 5 agent teammates to investigate different hypotheses. Have them talk to
each other to try to disprove each other's theories, like a scientific
debate. Update the findings doc with whatever consensus emerges.
```

토론 구조가 여기서 핵심 메커니즘입니다. 순차적 조사는 앵커링으로 인해 고통받습니다: 한 이론이 탐색되면, 후속 조사는 그것에 편향됩니다.

여러 독립적인 조사자가 적극적으로 서로의 이론을 반박하려고 할 때, 생존하는 이론은 실제 근본 원인일 가능성이 훨씬 높습니다.

모범 사례

팀원에게 충분한 컨텍스트 제공

팀원들은 CLAUDE.md, MCP servers, skills를 포함한 프로젝트 컨텍스트를 자동으로 로드하지만, 리더의 대화 기록을 상속하지 않습니다. 자세한 내용은 컨텍스트 및 통신을 참조합니다. 생성 프롬프트에 작업별 세부 사항을 포함합니다:

```
Spawn a security reviewer teammate with the prompt: "Review the authentication
module
at src/auth/ for security vulnerabilities. Focus on token handling, session
management, and input validation. The app uses JWT tokens stored in
httpOnly cookies. Report any issues with severity ratings."
```

적절한 팀 크기 선택

팀원의 수에 대한 하드 제한은 없지만, 실질적인 제약이 적용됩니다:

- **토큰 비용이 선형으로 증가**: 각 팀원은 자신의 컨텍스트 윈도우를 가지며 독립적으로 토큰을 소비합니다. 자세한 내용은 에이전트 팀 토큰 비용을 참조합니다.
- **조율 오버헤드 증가**: 더 많은 팀원은 더 많은 통신, 작업 조율, 충돌 가능성을 의미합니다

- **수익 감소**: 특정 지점을 넘으면, 추가 팀원은 작업 속도를 비례적으로 높이지 않습니다

대부분의 워크플로우에 대해 3-5명의 팀원으로 시작합니다. 이는 병렬 작업과 관리 가능한 조율의 균형을 맞춥니다. 이 가이드의 예시들은 3-5명의 팀원을 사용합니다. 이 범위는 다양한 작업 유형에서 잘 작동하기 때문입니다.

팀원당 5-6개의 작업을 유지하면 과도한 컨텍스트 전환 없이 모두를 생산적으로 유지합니다. 15개의 독립적인 작업이 있으면, 3명의 팀원이 좋은 시작점입니다.

작업이 실제로 팀원들이 동시에 작동하는 것의 이점이 있을 때만 확장합니다. 세 명의 집중된 팀원은 종종 다섯 명의 산만한 팀원을 능가합니다.

작업을 적절히 크기 조정

- **너무 작음**: 조율 오버헤드가 이점을 초과합니다
- **너무 큼**: 팀원들이 체크인 없이 너무 오래 작동하여 낭비된 노력의 위험이 증가합니다
- **적절함**: 함수, 테스트 파일, 검토와 같은 명확한 결과물을 생성하는 자체 포함된 단위

> **Tip:**
>
> 리더는 작업을 작업으로 나누고 팀원들에게 자동으로 할당합니다. 충분한 작업을 만들지 않으면, 작업을 더 작은 조각으로 분할하도록 요청합니다. 팀원당 5-6개의 작업을 유지하면 모두를 생산적으로 유지하고 누군가 막히면 리더가 작업을 재할당할 수 있습니다.

팀원들이 완료될 때까지 기다리기

때때로 리더는 팀원들을 기다리지 않고 작업을 자체적으로 구현하기 시작합니다. 이를 알아차리면:

```
Wait for your teammates to complete their tasks before proceeding
```

연구 및 검토로 시작하기

에이전트 팀을 처음 사용하는 경우, 명확한 경계가 있고 코드 작성이 필요하지 않은 작업으로 시작합니다: PR 검토, 라이브러리 연구, 또는 버그 조사. 이러한 작업은 병렬 탐색의 가치를 보여주면서 병렬 구현과 함께 오는 조율 문제 없이 보여줍니다.

파일 충돌 피하기

두 팀원이 동일한 파일을 편집하면 덮어쓰기가 발생합니다. 각 팀원이 다른 파일 집합을 소유하도록 작업을 나눕니다.

모니터링 및 조율

팀원들의 진행 상황을 확인하고, 작동하지 않는 접근 방식을 재지정하며, 발견이 들어올 때 종합합니다. 팀을 무인으로 너무 오래 실행하면 낭비된 노력의 위험이 증가합니다.

문제 해결

팀원이 나타나지 않음

Claude에게 팀을 만들도록 요청한 후 팀원이 나타나지 않으면:

- In-process 모드에서, 팀원들이 이미 실행 중이지만 보이지 않을 수 있습니다. Shift+Down을 눌러 활성 팀원들을 순환합니다.
- Claude에게 준 작업이 팀을 보증할 만큼 복잡한지 확인합니다. Claude는 작업에 따라 팀원을 생성할지 결정합니다.
- 분할 창을 명시적으로 요청했으면, tmux가 설치되어 있고 PATH에서 사용 가능한지 확인합니다:

```
which tmux
```

- iTerm2의 경우, `it2` CLI가 설치되어 있고 Python API가 iTerm2 환경 설정에서 활성화되어 있는지 확인합니다.

너무 많은 권한 프롬프트

팀원 권한 요청이 리더로 버블업되어 마찰을 일으킬 수 있습니다. 팀원들을 생성하기 전에 권한 설정에서 일반적인 작업을 사전 승인하여 중단을 줄입니다.

팀원들이 오류에서 중지됨

팀원들은 오류를 만난 후 복구하지 않고 중지할 수 있습니다. In-process 모드에서 Shift+Down을 사용하거나 분할 모드에서 창을 클릭하여 출력을 확인한 후:

- 직접 추가 지시를 제공합니다
- 작업을 계속하기 위해 대체 팀원을 생성합니다

리더가 작업 완료 전에 종료됨

리더는 모든 작업이 실제로 완료되기 전에 팀이 완료되었다고 결정할 수 있습니다. 이 경우 계속하도록 말합니다. 또한 리더가 위임하지 않고 작업을 시작하면 팀원들이 완료될 때까지 기다리도록 말할 수 있습니다.

고아 tmux 세션

팀이 끝난 후 tmux 세션이 지속되면, 완전히 정리되지 않았을 수 있습니다. 세션을 나열하고 팀에서 만든 세션을 종료합니다:

```
tmux ls
tmux kill-session -t <session-name>
```

제한 사항

에이전트 팀은 실험적입니다. 주의할 현재 제한 사항:

- **In-process 팀원과의 세션 재개 없음**: `/resume`과 `/rewind`는 in-process 팀원을 복원하지 않습니다. 세션을 재개한 후, 리더는 더 이상 존재하지 않는 팀원에게 메시지를 보내려고 시도할 수 있습니다. 이 경우 리더에게 새 팀원을 생성하도록 말합니다.
- **작업 상태가 지연될 수 있음**: 팀원들이 때때로 작업을 완료로 표시하지 못하여 종속 작업을 차단합니다. 작업이 막혀 있는 것처럼 보이면, 작업이 실제로 완료되었는지 확인하고 작업 상태를 수동으로 업데이트하거나 리더에게 팀원을 밀도록 말합니다.
- **종료가 느릴 수 있음**: 팀원들은 현재 요청이나 도구 호출을 마친 후 종료되어 시간이 걸릴 수 있습니다.
- **세션당 한 팀**: 리더는 한 번에 한 팀만 관리할 수 있습니다. 새 팀을 시작하기 전에 현재 팀을 정리합니다.
- **중첩된 팀 없음**: 팀원들은 자신의 팀이나 팀원을 생성할 수 없습니다. 리더만 팀을 관리할 수 있습니다.
- **리더가 고정됨**: 팀을 만드는 세션은 수명 동안 리더입니다. 팀원을 리더로 승격하거나 리더십을 이전할 수 없습니다.
- **생성 시 권한 설정**: 모든 팀원은 리더의 권한 모드로 시작합니다. 생성 후 개별 팀원 모드를 변경할 수 있지만, 생성 시 팀원별 모드를 설정할 수 없습니다.
- **분할 창은 tmux 또는 iTerm2 필요**: 기본 in-process 모드는 모든 터미널에서 작동합니다. 분할 창 모드는 VS Code의 통합 터미널, Windows Terminal, Ghostty에서 지원되지 않습니다.

> **Tip:**
>
> **`CLAUDE.md` 는 정상적으로 작동합니다**: 팀원들은 작업 디렉토리에서 `CLAUDE.md` 파일을 읽습니다. 이를 사용하여 모든 팀원에게 프로젝트별 지침을 제공합니다.

다음 단계

병렬 작업 및 위임을 위한 관련 접근 방식을 탐색합니다:

- **경량 위임**: subagents는 세션 내에서 연구 또는 검증을 위해 도우미 에이전트를 생성하며, 에이전트 간 조율이 필요하지 않은 작업에 더 좋습니다
- **수동 병렬 세션**: Git worktrees를 사용하면 자동화된 팀 조율 없이 여러 Claude Code 세션을 직접 실행할 수 있습니다
- **접근 방식 비교**: subagent vs 에이전트 팀 비교를 참조하여 나란히 비교합니다

Claude Code를 프로그래밍 방식으로 실행하기

> Agent SDK를 사용하여 CLI, Python 또는 TypeScript에서 Claude Code를 프로그래밍 방식으로 실행합니다.

Agent SDK는 Claude Code를 구동하는 동일한 도구, 에이전트 루프 및 컨텍스트 관리를 제공합니다. 스크립트 및 CI/CD용 CLI로 사용하거나 완전한 프로그래밍 방식 제어를 위한 Python 및 TypeScript 패키지로 사용할 수 있습니다.

> **Note:**
>
> CLI는 이전에 "헤드리스 모드"라고 불렸습니다. `-p` 플래그 및 모든 CLI 옵션은 동일한 방식으로 작동합니다.

CLI에서 Claude Code를 프로그래밍 방식으로 실행하려면 프롬프트와 함께 `-p` 를 전달하고 CLI 옵션을 사용합니다:

```
claude -p "Find and fix the bug in auth.py" --allowedTools "Read,Edit,Bash"
```

이 페이지는 CLI(`claude -p`)를 통한 Agent SDK 사용을 다룹니다. 구조화된 출력, 도구 승인 콜백 및 기본 메시지 객체가 있는 Python 및 TypeScript SDK 패키지의 경우 전체 Agent SDK 문서를 참조하십시오.

기본 사용법

`claude` 명령에 `-p` (또는 `--print`) 플래그를 추가하여 비대화형으로 실행합니다. 모든 CLI 옵션은 `-p` 와 함께 작동합니다:

- `--continue` 는 대화 계속하기용
- `--allowedTools` 는 도구 자동 승인용
- `--output-format` 은 구조화된 출력용

이 예제는 코드베이스에 대해 Claude에 질문하고 응답을 출력합니다:

```
claude -p "What does the auth module do?"
```

예제

이 예제들은 일반적인 CLI 패턴을 강조합니다.

구조화된 출력 가져오기

`--output-format` 을 사용하여 응답이 반환되는 방식을 제어합니다:

- `text` (기본값): 일반 텍스트 출력
- `json` : 결과, 세션 ID 및 메타데이터가 포함된 구조화된 JSON
- `stream-json` : 실시간 스트리밍을 위한 줄 구분 JSON

이 예제는 세션 메타데이터와 함께 프로젝트 요약을 JSON으로 반환하며, 텍스트 결과는 `result` 필드에 있습니다:

```
claude -p "Summarize this project" --output-format json
```

특정 스키마를 준수하는 출력을 얻으려면 `--output-format json` 을 `--json-schema` 및 JSON Schema 정의와 함께 사용합니다. 응답에는 요청에 대한 메타데이터(세션 ID, 사용량 등)가 포함되며 구조화된 출력은 `structured_output` 필드에 있습니다.

이 예제는 함수 이름을 추출하고 문자열 배열로 반환합니다:

```
claude -p "Extract the main function names from auth.py" \
  --output-format json \
  --json-schema '{"type":"object","properties":{"functions":
{"type":"array","items":{"type":"string"}}},"required":["functions"]}'
```

> **Tip:**
>
> jq와 같은 도구를 사용하여 응답을 구문 분석하고 특정 필드를 추출합니다:

```
## 텍스트 결과 추출
claude -p "Summarize this project" --output-format json | jq -r '.result'

## 구조화된 출력 추출
claude -p "Extract function names from auth.py" \
  --output-format json \
  --json-schema '{"type":"object","properties":{"functions":
{"type":"array","items":{"type":"string"}}},"required":["functions"]}' \
  | jq '.structured_output'
```

응답 스트리밍

`--output-format stream-json` 을 `--verbose` 및 `--include-partial-messages` 와 함께 사용하여 생성되는 토큰을 수신합니다. 각 줄은 이벤트를 나타내는 JSON 객체입니다:

```
claude -p "Explain recursion" --output-format stream-json --verbose --include-
partial-messages
```

다음 예제는 jq를 사용하여 텍스트 델타를 필터링하고 스트리밍 텍스트만 표시합니다. `-r` 플래그는 원본 문자열(따옴표 없음)을 출력하고 `-j` 는 줄 바꿈 없이 조인하므로 토큰이 계속 스트리밍됩니다:

```
claude -p "Write a poem" --output-format stream-json --verbose --include-partial-
messages | \
  jq -rj 'select(.type == "stream_event" and .event.delta.type? == "text_delta")
| .event.delta.text'
```

콜백 및 메시지 객체를 사용한 프로그래밍 방식 스트리밍의 경우 Agent SDK 문서의 실시간 응답 스트리밍을 참조하십시오.

도구 자동 승인

`--allowedTools` 를 사용하여 Claude가 프롬프트 없이 특정 도구를 사용하도록 합니다. 이 예제는 테스트 스위트를 실행하고 실패를 수정하며, Claude가 권한을 요청하지 않고 Bash 명령을 실행하고 파일을 읽고 편집할 수 있도록 합니다:

```
claude -p "Run the test suite and fix any failures" \
  --allowedTools "Bash,Read,Edit"
```

커밋 생성

이 예제는 스테이징된 변경 사항을 검토하고 적절한 메시지로 커밋을 생성합니다:

```
claude -p "Look at my staged changes and create an appropriate commit" \
  --allowedTools "Bash(git diff *),Bash(git log *),Bash(git status *),Bash(git
commit *)"
```

`--allowedTools` 플래그는 권한 규칙 구문을 사용합니다. 뒤의 `*`는 접두사 일치를 활성화하므로 `Bash(git diff *)`는 `git diff`로 시작하는 모든 명령을 허용합니다. 공백이 중요합니다: 없으면 `Bash(git diff*)`도 `git diff-index`와 일치합니다.

> **Note:**
>
> 사용자가 호출한 skills와 같은 `/commit` 및 기본 제공 명령은 대화형 모드에서만 사용할 수 있습니다. `-p` 모드에서는 대신 수행하려는 작업을 설명합니다.

시스템 프롬프트 사용자 정의

`--append-system-prompt`를 사용하여 Claude Code의 기본 동작을 유지하면서 지침을 추가합니다. 이 예제는 PR diff를 Claude에 파이프하고 보안 취약점을 검토하도록 지시합니다:

```
gh pr diff "$1" | claude -p \
  --append-system-prompt "You are a security engineer. Review for
vulnerabilities." \
  --output-format json
```

기본 프롬프트를 완전히 바꾸는 `--system-prompt`를 포함한 더 많은 옵션은 시스템 프롬프트 플래그를 참조하십시오.

대화 계속하기

`--continue`를 사용하여 가장 최근 대화를 계속하거나 `--resume`을 세션 ID와 함께 사용하여 특정 대화를 계속합니다. 이 예제는 검토를 실행한 다음 후속 프롬프트를 보냅니다:

```
## 첫 번째 요청
claude -p "Review this codebase for performance issues"

## 가장 최근 대화 계속
claude -p "Now focus on the database queries" --continue
claude -p "Generate a summary of all issues found" --continue
```

여러 대화를 실행 중인 경우 세션 ID를 캡처하여 특정 대화를 재개합니다:

```
session_id=$(claude -p "Start a review" --output-format json | jq -r
'.session_id')
claude -p "Continue that review" --resume "$session_id"
```

다음 단계

- Agent SDK 빠른 시작: Python 또는 TypeScript로 첫 번째 에이전트 구축
- CLI 참조: 모든 CLI 플래그 및 옵션
- GitHub Actions: GitHub 워크플로우에서 Agent SDK 사용
- GitLab CI/CD: GitLab 파이프라인에서 Agent SDK 사용

모든 기기에서 로컬 세션 계속하기 (Remote Control)

> Remote Control을 사용하여 휴대폰, 태블릿 또는 모든 브라우저에서 로컬 Claude Code 세션을 계속할 수 있습니다. claude.ai/code 및 Claude 모바일 앱과 함께 작동합니다.

Note:

Remote Control은 모든 요금제에서 사용할 수 있습니다. Team 및 Enterprise 관리자는 먼저 관리자 설정에서 Claude Code를 활성화해야 합니다.

Remote Control은 claude.ai/code 또는 iOS 및 Android용 Claude 앱을 컴퓨터에서 실행 중인 Claude Code 세션에 연결합니다. 책상에서 작업을 시작한 다음 소파의 휴대폰이나 다른 컴퓨터의 브라우저에서 계속할 수 있습니다.

컴퓨터에서 Remote Control 세션을 시작하면 Claude는 전체 시간 동안 로컬에서 실행되므로 아무것도 클라우드로 이동하지 않습니다. Remote Control을 사용하면 다음을 수행할 수 있습니다.

- **전체 로컬 환경을 원격으로 사용**: 파일 시스템, MCP servers, 도구 및 프로젝트 구성이 모두 사용 가능하게 유지됩니다.
- **두 개의 표면에서 동시에 작업**: 대화가 모든 연결된 기기에서 동기화되므로 터미널, 브라우저 및 휴대폰에서 메시지를 교환 가능하게 보낼 수 있습니다.
- **중단 극복**: 노트북이 절전 모드로 전환되거나 네트워크가 끊어지면 컴퓨터가 다시 온라인 상태가 될 때 세션이 자동으로 다시 연결됩니다.

클라우드 인프라에서 실행되는 웹의 Claude Code와 달리 Remote Control 세션은 컴퓨터에서 직접 실행되며 로컬 파일 시스템과 상호 작용합니다. 웹 및 모바일 인터페이스는 단지 해당 로컬 세션으로의 창일 뿐입니다.

Note:

Remote Control에는 Claude Code v2.1.51 이상이 필요합니다. `claude --version` 으로 버전을 확인하세요.

이 페이지에서는 설정, 세션을 시작하고 연결하는 방법, 그리고 Remote Control이 웹의 Claude Code와 어떻게 비교되는지를 다룹니다.

요구 사항

Remote Control을 사용하기 전에 환경이 다음 조건을 충족하는지 확인하세요.

- **구독**: Pro, Max, Team 및 Enterprise 요금제에서 사용 가능합니다. Team 및 Enterprise 관리자는 먼저 관리자 설정에서 Claude Code를 활성화해야 합니다. API 키는 지원되지 않습니다.
- **인증**: `claude`를 실행하고 아직 로그인하지 않았다면 `/login`을 사용하여 claude.ai를 통해 로그인하세요.
- **작업 공간 신뢰**: 작업 공간 신뢰 대화를 수락하려면 프로젝트 디렉토리에서 최소한 한 번 `claude`를 실행하세요.

Remote Control 세션 시작

새 세션을 Remote Control에서 직접 시작하거나 이미 실행 중인 세션에 연결할 수 있습니다.

새 세션

프로젝트 디렉토리로 이동하여 다음을 실행하세요.

```
claude remote-control
```

프로세스는 터미널에서 계속 실행되며 원격 연결을 기다립니다. 다른 기기에서 연결하는 데 사용할 수 있는 세션 URL을 표시하며, 스페이스바를 눌러 휴대폰에서 빠른 액세스를 위한 QR 코드를 표시할 수 있습니다. 원격 세션이 활성화되어 있는 동안 터미널은 연결 상태 및 도구 활동을 표시합니다.

이 명령은 다음 플래그를 지원합니다.

- `--name "My Project"` : claude.ai/code의 세션 목록에 표시되는 사용자 정의 세션 제목을 설정합니다. 이름을 위치 인수로 전달할 수도 있습니다. `claude remote-control "My Project"`
- `--verbose` : 자세한 연결 및 세션 로그를 표시합니다.
- `--sandbox` / `--no-sandbox` : 세션 중 파일 시스템 및 네트워크 격리를 위해 sandboxing을 활성화하거나 비활성화합니다. 기본적으로 Sandboxing은 꺼져 있습니다.

기존 세션에서

이미 Claude Code 세션에 있고 원격으로 계속하려면 `/remote-control` (또는 `/rc`) 명령을 사용하세요.

```
/remote-control
```

이름을 인수로 전달하여 사용자 정의 세션 제목을 설정하세요.

```
/remote-control My Project
```

이는 현재 대화 기록을 이어받는 Remote Control 세션을 시작하며 다른 기기에서 연결하는 데 사용할 수 있는 세션 URL 및 QR 코드를 표시합니다. `--verbose`, `--sandbox` 및 `--no-sandbox` 플래그는 이 명령에서 사용할 수 없습니다.

다른 기기에서 연결

Remote Control 세션이 활성화되면 다른 기기에서 연결하는 몇 가지 방법이 있습니다.

- **세션 URL 열기**: 모든 브라우저에서 세션 URL을 열어 claude.ai/code의 세션으로 직접 이동합니다. `claude remote-control` 및 `/remote-control` 모두 이 URL을 터미널에 표시합니다.
- **QR 코드 스캔**: 세션 URL 옆에 표시된 QR 코드를 스캔하여 Claude 앱에서 직접 열 수 있습니다. `claude remote-control`을 사용하면 스페이스바를 눌러 QR 코드 표시를 전환할 수 있습니다.
- **claude.ai/code 또는 Claude 앱 열기**: 세션 목록에서 이름으로 세션을 찾습니다. Remote Control 세션은 온라인 상태일 때 녹색 상태 점이 있는 컴퓨터 아이콘을 표시합니다.

원격 세션은 `--name` 인수 (또는 `/remote-control`에 전달된 이름), 마지막 메시지, `/rename` 값 또는 대화 기록이 없으면 "Remote Control session"에서 이름을 가져옵니다. 환경에 이미 활성 세션이 있으면 계속할지 또는 새로 시작할지 묻는 메시지가 표시됩니다.

Claude 앱이 아직 없으면 Claude Code 내에서 `/mobile` 명령을 사용하여 iOS 또는 Android용 다운로드 QR 코드를 표시하세요.

모든 세션에 대해 Remote Control 활성화

기본적으로 Remote Control은 명시적으로 `claude remote-control` 또는 `/remote-control`을 실행할 때만 활성화됩니다. 모든 세션에 대해 자동으로 활성화하려면 Claude Code 내에서 `/config`를 실행하고 **Enable Remote Control for all sessions**을 `true`로 설정하세요. 비활성화하려면 `false`로 다시 설정하세요.

각 Claude Code 인스턴스는 한 번에 하나의 원격 세션을 지원합니다. 여러 인스턴스를 실행하면 각각 자체 환경 및 세션을 가집니다.

연결 및 보안

로컬 Claude Code 세션은 아웃바운드 HTTPS 요청만 수행하며 컴퓨터에서 인바운드 포트를 열지 않습니다. Remote Control을 시작하면 Anthropic API에 등록하고 작업을 폴링합니다. 다른 기기에서 연결하면 서버는 웹 또는 모바일 클라이언트와 로컬 세션 간의 메시지를 스트리밍 연결을 통해 라우팅합니다.

모든 트래픽은 TLS를 통해 Anthropic API를 통해 이동하며, 이는 모든 Claude Code 세션과 동일한 전송 보안입니다. 연결은 여러 단기 자격 증명을 사용하며, 각각은 단일 목적으로 범위가 지정되고 독립적으로 만료됩니다.

Remote Control vs 웹의 Claude Code

Remote Control과 웹의 Claude Code는 모두 claude.ai/code 인터페이스를 사용합니다. 주요 차이점은 세션이 실행되는 위치입니다. Remote Control은 컴퓨터에서 실행되므로 로컬 MCP servers, 도구 및 프로젝트 구성이 사용 가능하게 유지됩니다. 웹의 Claude Code는 Anthropic 관리 클라우드 인프라에서 실행됩니다.

로컬 작업 중간에 있고 다른 기기에서 계속하려면 Remote Control을 사용하세요. 로컬 설정 없이 작업을 시작하거나, 복제하지 않은 저장소에서 작업하거나, 여러 작업을 병렬로 실행하려면 웹의 Claude Code를 사용하세요.

제한 사항

- **한 번에 하나의 원격 세션**: 각 Claude Code 세션은 하나의 원격 연결을 지원합니다.
- **터미널은 열려 있어야 함**: Remote Control은 로컬 프로세스로 실행됩니다. 터미널을 닫거나 `claude` 프로세스를 중지하면 세션이 종료됩니다. 새 세션을 시작하려면 `claude remote-control`을 다시 실행하세요.
- **연장된 네트워크 중단**: 컴퓨터가 깨어 있지만 약 10분 이상 네트워크에 도달할 수 없으면 세션이 시간 초과되고 프로세스가 종료됩니다. 새 세션을 시작하려면 `claude remote-control`을 다시 실행하세요.

관련 리소스

- 웹의 Claude Code: 컴퓨터 대신 Anthropic 관리 클라우드 환경에서 세션을 실행합니다.
- 인증: `/login`을 설정하고 claude.ai의 자격 증명을 관리합니다.
- CLI 참조: `claude remote-control`을 포함한 플래그 및 명령의 전체 목록입니다.
- 보안: Remote Control 세션이 Claude Code 보안 모델에 어떻게 적합한지입니다.
- 데이터 사용: 로컬 및 원격 세션 중에 Anthropic API를 통해 흐르는 데이터입니다.

일정에 따라 프롬프트 실행하기

> /loop와 cron 스케줄링 도구를 사용하여 Claude Code 세션 내에서 프롬프트를 반복 실행하거나, 상태를 폴링하거나, 일회성 알림을 설정합니다.

Note:

스케줄된 작업을 사용하려면 Claude Code v2.1.72 이상이 필요합니다. `claude --version`으로 버전을 확인하십시오.

스케줄된 작업을 사용하면 Claude가 일정한 간격으로 프롬프트를 자동으로 다시 실행할 수 있습니다. 배포를 폴링하거나, PR을 감시하거나, 오래 실행되는 빌드를 확인하거나, 세션 후반에 무언가를 하도록 자신에게 알림을 설정하는 데 사용하십시오.

작업은 세션 범위입니다. 현재 Claude Code 프로세스에 존재하며 종료하면 사라집니다. 재시작을 견디고 활성 터미널 세션 없이 실행되는 지속적인 스케줄링의 경우 Desktop 스케줄된 작업 또는 GitHub Actions를 참조하십시오.

/loop로 반복 프롬프트 스케줄하기

`/loop` 번들 스킬은 반복 프롬프트를 스케줄하는 가장 빠른 방법입니다. 선택적 간격과 프롬프트를 전달하면 Claude가 세션이 열려 있는 동안 백그라운드에서 실행되는 cron 작업을 설정합니다.

```
/loop 5m check if the deployment finished and tell me what happened
```

Claude는 간격을 파싱하고 cron 표현식으로 변환하며 작업을 스케줄하고 주기와 작업 ID를 확인합니다.

간격 구문

간격은 선택 사항입니다. 앞에 둘 수도, 뒤에 둘 수도, 완전히 생략할 수도 있습니다.

형식	예시	파싱된 간격
선행 토큰	`/loop 30m check the build`	30분마다

형식	예시	파싱된 간격
후행 `every` 절	`/loop check the build every 2 hours`	2시간마다
간격 없음	`/loop check the build`	기본값: 10분마다

지원되는 단위는 초의 경우 `s`, 분의 경우 `m`, 시간의 경우 `h`, 일의 경우 `d` 입니다. cron은 1분 단위의 세분성을 가지므로 초는 가장 가까운 분으로 올림됩니다. `7m` 또는 `90m` 과 같이 단위로 균등하게 나누어지지 않는 간격은 가장 가까운 깔끔한 간격으로 반올림되며 Claude가 선택한 것을 알려줍니다.

다른 명령어에 대해 루프하기

스케줄된 프롬프트 자체가 명령어 또는 스킬 호출일 수 있습니다. 이는 이미 패키징한 워크플로우를 다시 실행하는 데 유용합니다.

```
/loop 20m /review-pr 1234
```

작업이 실행될 때마다 Claude는 마치 입력한 것처럼 `/review-pr 1234`를 실행합니다.

일회성 알림 설정하기

일회성 알림의 경우 `/loop`를 사용하는 대신 자연어로 원하는 것을 설명하십시오. Claude는 실행 후 자신을 삭제하는 일회성 작업을 스케줄합니다.

```
remind me at 3pm to push the release branch
```

```
in 45 minutes, check whether the integration tests passed
```

Claude는 cron 표현식을 사용하여 실행 시간을 특정 분과 시간으로 고정하고 실행 시간을 확인합니다.

스케줄된 작업 관리하기

Claude에게 자연어로 작업을 나열하거나 취소하도록 요청하거나 기본 도구를 직접 참조하십시오.

```
what scheduled tasks do I have?
```

```
cancel the deploy check job
```

내부적으로 Claude는 다음 도구를 사용합니다.

도구	목적
`CronCreate`	새 작업을 스케줄합니다. 5필드 cron 표현식, 실행할 프롬프트, 반복 여부 또는 일회성 실행 여부를 허용합니다.
`CronList`	ID, 스케줄, 프롬프트와 함께 모든 스케줄된 작업을 나열합니다.
`CronDelete`	ID로 작업을 취소합니다.

각 스케줄된 작업에는 `CronDelete`에 전달할 수 있는 8자 ID가 있습니다. 세션은 한 번에 최대 50개의 스케줄된 작업을 보유할 수 있습니다.

스케줄된 작업이 실행되는 방식

스케줄러는 매초 기한이 된 작업을 확인하고 낮은 우선순위로 큐에 넣습니다. 스케줄된 프롬프트는 차례 사이에 실행되며, Claude가 응답 중일 때는 실행되지 않습니다. Claude가 작업이 기한이 될 때 바쁘면 프롬프트는 현재 차례가 끝날 때까지 기다립니다.

모든 시간은 현지 시간대로 해석됩니다. `0 9 * * *`와 같은 cron 표현식은 UTC가 아니라 Claude Code를 실행 중인 곳의 오전 9시를 의미합니다.

지터

모든 세션이 동일한 벽시계 시간에 API에 도달하는 것을 방지하기 위해 스케줄러는 실행 시간에 작은 결정론적 오프셋을 추가합니다.

- 반복 작업은 기간의 최대 10% 늦게 실행되며, 최대 15분으로 제한됩니다. 시간별 작업은 `:00`에서 `:06` 사이의 어느 시점에서나 실행될 수 있습니다.
- 시간의 맨 위 또는 맨 아래에 스케줄된 일회성 작업은 최대 90초 일찍 실행됩니다.

오프셋은 작업 ID에서 파생되므로 동일한 작업은 항상 동일한 오프셋을 받습니다. 정확한 타이밍이 중요한 경우 `0 9 * * *` 대신 `3 9 * * *`와 같이 `:00` 또는 `:30`이 아닌 분을 선택하면 일회성 지터가 적용되지 않습니다.

3일 만료

반복 작업은 생성 후 3일 후 자동으로 만료됩니다. 작업은 마지막으로 한 번 실행된 후 자신을 삭제합니다. 이는 잊혀진 루프가 실행될 수 있는 기간을 제한합니다. 반복 작업이 더 오래 지속되어야 하는 경우 만료되기 전에 취소하고 다시 만들거나 지속적인 스케줄링을 위해 Desktop 스케줄된 작업을 사용하십시오.

Cron 표현식 참조

`CronCreate` 는 표준 5필드 cron 표현식을 허용합니다: `minute hour day-of-month month day-of-week` . 모든 필드는 와일드카드(`*`), 단일 값(`5`), 단계(`*/15`), 범위(`1-5`), 쉼표로 구분된 목록(`1,15,30`)을 지원합니다.

예시	의미
`*/5 * * * *`	5분마다
`0 * * * *`	매시간 정각
`7 * * * *`	매시간 7분
`0 9 * * *`	매일 오전 9시(현지 시간)
`0 9 * * 1-5`	평일 오전 9시(현지 시간)
`30 14 15 3 *`	3월 15일 오후 2시 30분(현지 시간)

요일은 일요일의 경우 `0` 또는 `7` , 토요일의 경우 `6` 을 사용합니다. `L` , `W` , `?` 와 같은 확장 구문 및 `MON` 또는 `JAN` 과 같은 이름 별칭은 지원되지 않습니다.

일-월과 요일이 모두 제한되면 두 필드 중 하나라도 일치하면 날짜가 일치합니다. 이는 표준 vixie-cron 의미론을 따릅니다.

스케줄된 작업 비활성화하기

환경에서 `CLAUDE_CODE_DISABLE_CRON=1` 을 설정하여 스케줄러를 완전히 비활성화합니다. cron 도구와 `/loop` 는 사용할 수 없게 되며 이미 스케줄된 모든 작업은 실행을 중지합니다. 비활성화 플래그의 전체 목록은 환경 변수를 참조하십시오.

제한 사항

세션 범위 스케줄링에는 고유한 제약이 있습니다.

- 작업은 Claude Code가 실행 중이고 유휴 상태일 때만 실행됩니다. 터미널을 닫거나 세션을 종료하면 모든 것이 취소됩니다.

- 놓친 실행에 대한 추적 없음. 작업의 스케줄된 시간이 Claude가 오래 실행되는 요청에 바쁠 때 지나가면 Claude가 유휴 상태가 될 때 한 번 실행되며, 놓친 각 간격마다 한 번씩 실행되지 않습니다.
- 재시작 간 지속성 없음. Claude Code를 다시 시작하면 모든 세션 범위 작업이 지워집니다.

무인으로 실행해야 하는 cron 기반 자동화의 경우 `schedule` 트리거가 있는 GitHub Actions 워크플로우를 사용하거나 그래픽 설정 흐름을 원하는 경우 Desktop 스케줄된 작업을 사용하십시오.

Code Review

> 다중 에이전트 분석을 통해 전체 코드베이스를 검토하여 논리 오류, 보안 취약점 및 회귀를 감지하는 자동화된 PR 검토를 설정합니다

> **Note:**
>
> Code Review는 연구 미리보기 상태이며 Teams 및 Enterprise 구독에서 사용 가능합니다. Zero Data Retention이 활성화된 조직에서는 사용할 수 없습니다.

Code Review는 GitHub 풀 요청을 분석하고 문제를 발견한 코드 라인에 인라인 댓글로 결과를 게시합니다. 전문화된 에이전트 집합이 전체 코드베이스의 맥락에서 코드 변경 사항을 검토하여 논리 오류, 보안 취약점, 손상된 엣지 케이스 및 미묘한 회귀를 찾습니다.

결과는 심각도별로 태그가 지정되며 PR을 승인하거나 차단하지 않으므로 기존 검토 워크플로우는 그대로 유지됩니다. 저장소에 `CLAUDE.md` 또는 `REVIEW.md` 파일을 추가하여 Claude가 플래그하는 항목을 조정할 수 있습니다.

관리되는 서비스 대신 자신의 CI 인프라에서 Claude를 실행하려면 GitHub Actions 또는 GitLab CI/CD를 참조하십시오.

이 페이지에서 다루는 내용:

- 검토 작동 방식
- 설정
- `CLAUDE.md` 및 `REVIEW.md`를 사용한 검토 사용자 정의
- 가격

검토 작동 방식

관리자가 조직에 대해 Code Review를 활성화하면 풀 요청이 열리거나 업데이트될 때 검토가 자동으로 실행됩니다. 여러 에이전트가 Anthropic 인프라에서 병렬로 diff 및 주변 코드를 분석합니다. 각 에이전트는 다른 클래스의 문제를 찾고, 검증 단계에서 후보를 실제 코드 동작과 비교하여 거짓 양성을 필터링합니다. 결과는 중복 제거되고 심각도별로 순위가 지정되며 문제가 발견된 특정 라인에 인라인 댓글로 게시됩니다. 문제가 발견되지 않으면 Claude는 PR에 짧은 확인 댓글을 게시합니다.

검토는 PR 크기 및 복잡도에 따라 비용이 증가하며 평균 20분 내에 완료됩니다. 관리자는 분석 대시보드를 통해 검토 활동 및 지출을 모니터링할 수 있습니다.

심각도 수준

각 결과는 심각도 수준으로 태그가 지정됩니다:

마커	심각도	의미
🔴	Normal	병합 전에 수정해야 하는 버그
🟡	Nit	사소한 문제, 수정할 가치가 있지만 차단하지는 않음
🟣	Pre-existing	코드베이스에 존재하지만 이 PR에서 도입되지 않은 버그

결과에는 Claude가 문제를 플래그한 이유와 문제를 어떻게 검증했는지 이해하기 위해 확장할 수 있는 축소 가능한 확장 추론 섹션이 포함됩니다.

Code Review가 확인하는 항목

기본적으로 Code Review는 정확성에 중점을 두고 있습니다: 형식 기본 설정이나 누락된 테스트 범위가 아닌 프로덕션을 중단할 버그입니다. 저장소에 지침 파일을 추가하여 확인하는 항목을 확장할 수 있습니다.

Code Review 설정

관리자가 조직에 대해 Code Review를 한 번 활성화하고 포함할 저장소를 선택합니다.

Step 1: Claude Code 관리자 설정 열기

claude.ai/admin-settings/claude-code로 이동하여 Code Review 섹션을 찾습니다. Claude 조직에 대한 관리자 액세스 권한과 GitHub 조직에 GitHub 앱을 설치할 수 있는 권한이 필요합니다.

Step 2: 설정 시작

Setup을 클릭합니다. 이렇게 하면 GitHub 앱 설치 흐름이 시작됩니다.

Step 3: Claude GitHub 앱 설치

프롬프트를 따라 Claude GitHub 앱을 GitHub 조직에 설치합니다. 앱은 다음 저장소 권한을 요청합니다:

- **Contents**: 읽기 및 쓰기
- **Issues**: 읽기 및 쓰기
- **Pull requests**: 읽기 및 쓰기

Code Review는 콘텐츠에 대한 읽기 액세스와 풀 요청에 대한 쓰기 액세스를 사용합니다. 더 광범위한 권한 집합은 나중에 활성화하는 경우 GitHub Actions도 지원합니다.

Step 4: 저장소 선택

Code Review를 활성화할 저장소를 선택합니다. 저장소가 보이지 않으면 설치 중에 Claude GitHub 앱에 액세스 권한을 부여했는지 확인하십시오. 나중에 더 많은 저장소를 추가할 수 있습니다.

Step 5: 저장소별 검토 트리거 설정

설정이 완료되면 Code Review 섹션에 저장소가 테이블에 표시됩니다. 각 저장소에 대해 드롭다운을 사용하여 검토가 실행되는 시기를 선택합니다:

- **After PR creation only**: PR이 열리거나 검토 준비 완료로 표시될 때 한 번 검토가 실행됩니다
- **After every push to PR branch**: 모든 푸시에서 검토가 실행되어 PR이 진화함에 따라 새로운 문제를 감지하고 플래그된 문제를 수정할 때 스레드를 자동으로 해결합니다

모든 푸시에서 검토하면 더 많은 검토가 실행되고 비용이 더 많이 듭니다. PR 생성만으로 시작하고 지속적인 범위 적용 및 자동 스레드 정리를 원하는 저장소의 경우 푸시 시로 전환합니다.

저장소 테이블은 또한 최근 활동을 기반으로 각 저장소의 평균 검토 비용을 표시합니다. 행 작업 메뉴를 사용하여 저장소별로 Code Review를 켜거나 끄거나 저장소를 완전히 제거합니다.

설정을 확인하려면 테스트 PR을 열어봅니다. **Claude Code Review**라는 이름의 확인 실행이 몇 분 내에 나타납니다. 나타나지 않으면 저장소가 관리자 설정에 나열되어 있고 Claude GitHub 앱이 액세스할 수 있는지 확인합니다.

검토 사용자 정의

Code Review는 저장소에서 두 파일을 읽어 플래그할 항목을 안내합니다. 둘 다 기본 정확성 확인 위에 추가됩니다:

- `CLAUDE.md`: Claude Code가 검토뿐만 아니라 모든 작업에 사용하는 공유 프로젝트 지침입니다. 지침이 대화형 Claude Code 세션에도 적용될 때 사용합니다.
- `REVIEW.md`: 검토 전용 지침으로 코드 검토 중에만 읽습니다. 검토 중에 플래그하거나 건너뛸 항목에 대한 규칙이 엄격하고 일반 `CLAUDE.md`를 복잡하게 할 규칙에 사용합니다.

CLAUDE.md

Code Review는 저장소의 `CLAUDE.md` 파일을 읽고 새로 도입된 위반을 nit 수준 결과로 취급합니다. 이는 양방향으로 작동합니다: PR이 `CLAUDE.md` 문을 오래된 것으로 만드는 방식으로 코드를 변경하면 Claude는 문서도 업데이트해야 한다고 플래그합니다.

Claude는 디렉토리 계층의 모든 수준에서 `CLAUDE.md` 파일을 읽으므로 하위 디렉토리의 `CLAUDE.md`의 규칙은 해당 경로 아래의 파일에만 적용됩니다. `CLAUDE.md` 작동 방식에 대한 자세한 내용은 메모리 설명서를 참조하십시오.

일반 Claude Code 세션에 적용하고 싶지 않은 검토 특정 지침의 경우 대신 `REVIEW.md`를 사용합니다.

REVIEW.md

검토 특정 규칙에 대해 저장소 루트에 `REVIEW.md` 파일을 추가합니다. 다음을 인코딩하는 데 사용합니다:

- 회사 또는 팀 스타일 지침: "중첩된 조건부보다 조기 반환 선호"
- 린터로 다루지 않는 언어 또는 프레임워크 특정 규칙
- Claude가 항상 플래그해야 할 항목: "새로운 API 경로에는 통합 테스트가 있어야 함"
- Claude가 건너뛸 항목: "생성된 코드 아래 `/gen/`의 형식 지정에 대해 댓글을 달지 마십시오"

`REVIEW.md` 예:

```
## Code Review Guidelines

### Always check
- New API endpoints have corresponding integration tests
- Database migrations are backward-compatible
- Error messages don't leak internal details to users

### Style
- Prefer `match` statements over chained `isinstance` checks
- Use structured logging, not f-string interpolation in log calls

### Skip
- Generated files under `src/gen/`
- Formatting-only changes in `*.lock` files
```

Claude는 저장소 루트에서 `REVIEW.md`를 자동으로 검색합니다. 구성이 필요하지 않습니다.

사용량 보기

claude.ai/analytics/code-review로 이동하여 조직 전체의 Code Review 활동을 확인합니다. 대시보드는 다음을 표시합니다:

섹션	표시 내용
PRs reviewed	선택한 시간 범위 동안 검토된 풀 요청의 일일 개수
Cost weekly	Code Review의 주간 지출
Feedback	개발자가 플래그된 문제를 해결했기 때문에 자동으로 해결된 검토 댓글의 개수
Repository breakdown	저장소별 검토된 PR 개수 및 해결된 댓글

관리자 설정의 저장소 테이블은 각 저장소의 검토당 평균 비용도 표시합니다.

가격

Code Review는 토큰 사용량을 기반으로 청구됩니다. 검토는 평균 $15-25이며 PR 크기, 코드베이스 복잡도 및 검증이 필요한 문제 수에 따라 확장됩니다. Code Review 사용량은 추가 사용량을 통해 별도로 청구되며 계획의 포함된 사용량에 포함되지 않습니다.

선택한 검토 트리거는 총 비용에 영향을 미칩니다:

- **After PR creation only**: PR당 한 번 실행됩니다
- **After every push**: 각 커밋에서 실행되어 푸시 수만큼 비용을 곱합니다

비용은 조직이 다른 Claude Code 기능에 AWS Bedrock 또는 Google Vertex AI를 사용하는지 여부와 관계없이 Anthropic 청구서에 나타납니다. Code Review의 월간 지출 한도를 설정하려면 claude.ai/admin-settings/usage로 이동하여 Claude Code Review 서비스의 한도를 구성합니다.

분석의 주간 비용 차트 또는 관리자 설정의 저장소별 평균 비용 열을 통해 지출을 모니터링합니다.

관련 리소스

Code Review는 Claude Code의 나머지 부분과 함께 작동하도록 설계되었습니다. PR을 열기 전에 로컬에서 검토를 실행하거나, 자체 호스팅 설정이 필요하거나, `CLAUDE.md`가 도구 전체에서 Claude의 동작을 어떻게 형성하는지에 대해 더 깊이 알고 싶다면 다음 페이지가 좋은 다음 단계입니다:

- Plugins: 푸시 전에 로컬에서 온디맨드 검토를 실행하기 위한 `code-review` 플러그인을 포함한 플러그인 마켓플레이스 찾아보기
- GitHub Actions: 코드 검토 이상의 사용자 정의 자동화를 위해 자신의 GitHub Actions 워크플로우에서 Claude 실행
- GitLab CI/CD: GitLab 파이프라인을 위한 자체 호스팅 Claude 통합
- Memory: Claude Code 전체에서 `CLAUDE.md` 파일이 작동하는 방식

- Analytics: 코드 검토 이상의 Claude Code 사용량 추적

Part 7: CI/CD & Integrations

Claude Code GitHub Actions

> Claude Code를 GitHub 워크플로우에 통합하는 방법에 대해 알아봅니다

Claude Code GitHub Actions는 GitHub 워크플로우에 AI 기반 자동화를 제공합니다. PR이나 이슈에서 간단한 `@claude` 멘션으로 Claude가 코드를 분석하고, 풀 리퀘스트를 생성하고, 기능을 구현하고, 버그를 수정할 수 있습니다. 모두 프로젝트의 표준을 따르면서 말입니다. 트리거 없이 모든 PR에 자동으로 게시되는 리뷰의 경우 GitHub Code Review를 참조하십시오.

Note:

Claude Code GitHub Actions는 Claude Agent SDK를 기반으로 구축되어 있으며, 이를 통해 Claude Code를 애플리케이션에 프로그래밍 방식으로 통합할 수 있습니다. SDK를 사용하여 GitHub Actions를 넘어서는 사용자 정의 자동화 워크플로우를 구축할 수 있습니다.

Info:

Claude Opus 4.6을 이제 사용할 수 있습니다. Claude Code GitHub Actions는 기본적으로 Sonnet을 사용합니다. Opus 4.6을 사용하려면 모델 파라미터를 `claude-opus-4-6` 을 사용하도록 구성하십시오.

Claude Code GitHub Actions를 사용하는 이유는 무엇입니까?

- **즉시 PR 생성**: 필요한 사항을 설명하면 Claude가 모든 필요한 변경 사항이 포함된 완전한 PR을 생성합니다
- **자동화된 코드 구현**: 이슈를 단일 명령으로 작동하는 코드로 변환합니다
- **표준 준수**: Claude는 `CLAUDE.md` 지침과 기존 코드 패턴을 존중합니다
- **간단한 설정**: 설치 프로그램과 API 키로 몇 분 안에 시작할 수 있습니다
- **기본적으로 안전**: 코드는 Github의 러너에 유지됩니다

Claude가 할 수 있는 것은 무엇입니까?

Claude Code는 코드 작업 방식을 변환하는 강력한 GitHub Action을 제공합니다:

Claude Code Action

이 GitHub Action을 사용하면 GitHub Actions 워크플로우 내에서 Claude Code를 실행할 수 있습니다. 이를 사용하여 Claude Code 위에 사용자 정의 워크플로우를 구축할 수 있습니다.

저장소 보기 →

설정

빠른 설정

이 작업을 설정하는 가장 쉬운 방법은 터미널에서 Claude Code를 통하는 것입니다. claude를 열고 `/install-github-app` 을 실행하면 됩니다.

이 명령은 GitHub 앱 및 필수 시크릿 설정을 안내합니다.

> **Note:**
>
> - GitHub 앱을 설치하고 시크릿을 추가하려면 저장소 관리자여야 합니다
> - GitHub 앱은 Contents, Issues 및 Pull requests에 대한 읽기 및 쓰기 권한을 요청합니다
> - 이 빠른 시작 방법은 직접 Claude API 사용자만 사용할 수 있습니다. AWS Bedrock 또는 Google Vertex AI를 사용 중인 경우 AWS Bedrock & Google Vertex AI 사용 섹션을 참조하십시오.

수동 설정

`/install-github-app` 명령이 실패하거나 수동 설정을 선호하는 경우 다음 수동 설정 지침을 따르십시오:

1. **Claude GitHub 앱을 저장소에 설치합니다**: https://github.com/apps/claude
   Claude GitHub 앱에는 다음 저장소 권한이 필요합니다:
   - **Contents**: 읽기 및 쓰기 (저장소 파일 수정)
   - **Issues**: 읽기 및 쓰기 (이슈에 응답)
   - **Pull requests**: 읽기 및 쓰기 (PR 생성 및 변경 사항 푸시)

   보안 및 권한에 대한 자세한 내용은 보안 설명서를 참조하십시오.
2. **ANTHROPIC_API_KEY를 저장소 시크릿에 추가합니다** (GitHub Actions에서 시크릿을 사용하는 방법 알아보기)
3. **워크플로우 파일을 복사합니다** examples/claude.yml에서 저장소의 `.github/workflows/` 로

> **Tip:**

빠른 시작 또는 수동 설정을 완료한 후 이슈 또는 PR 댓글에서 `@claude` 를 태그하여 작업을 테스트합니다.

베타에서 업그레이드

Warning:

Claude Code GitHub Actions v1.0은 베타 버전에서 v1.0으로 업그레이드하기 위해 워크플로우 파일을 업데이트해야 하는 주요 변경 사항을 도입합니다.

현재 Claude Code GitHub Actions의 베타 버전을 사용 중인 경우 워크플로우를 GA 버전을 사용하도록 업데이트하는 것이 좋습니다. 새 버전은 자동 모드 감지와 같은 강력한 새 기능을 추가하면서 구성을 단순화합니다.

필수 변경 사항

모든 베타 사용자는 업그레이드하기 위해 워크플로우 파일에서 다음 변경 사항을 수행해야 합니다:

1. **작업 버전 업데이트**: `@beta` 를 `@v1` 로 변경합니다
2. **모드 구성 제거**: `mode: "tag"` 또는 `mode: "agent"` 삭제 (이제 자동 감지됨)
3. **프롬프트 입력 업데이트**: `direct_prompt` 를 `prompt` 로 바꿉니다
4. **CLI 옵션 이동**: `max_turns` , `model` , `custom_instructions` 등을 `claude_args` 로 변환합니다

주요 변경 사항 참조

이전 베타 입력	새 v1.0 입력
`mode`	(제거됨 - 자동 감지됨)
`direct_prompt`	`prompt`
`override_prompt`	GitHub 변수가 있는 `prompt`
`custom_instructions`	`claude_args: --append-system-prompt`
`max_turns`	`claude_args: --max-turns`
`model`	`claude_args: --model`
`allowed_tools`	`claude_args: --allowedTools`

이전 베타 입력	새 v1.0 입력
`disallowed_tools`	`claude_args: --disallowedTools`
`claude_env`	`settings` JSON 형식

이전 및 이후 예제

베타 버전:

```yaml
- uses: anthropics/claude-code-action@beta
  with:
    mode: "tag"
    direct_prompt: "Review this PR for security issues"
    anthropic_api_key: ${{ secrets.ANTHROPIC_API_KEY }}
    custom_instructions: "Follow our coding standards"
    max_turns: "10"
    model: "claude-sonnet-4-6"
```

GA 버전 (v1.0):

```yaml
- uses: anthropics/claude-code-action@v1
  with:
    prompt: "Review this PR for security issues"
    anthropic_api_key: ${{ secrets.ANTHROPIC_API_KEY }}
    claude_args: |
      --append-system-prompt "Follow our coding standards"
      --max-turns 10
      --model claude-sonnet-4-6
```

> **Tip:**
>
> 작업은 이제 구성에 따라 대화형 모드(`@claude` 멘션에 응답) 또는 자동화 모드(프롬프트로 즉시 실행)에서 실행할지 여부를 자동으로 감지합니다.

예제 사용 사례

Claude Code GitHub Actions는 다양한 작업에 도움이 될 수 있습니다. examples 디렉토리에는 다양한 시나리오에 대한 즉시 사용 가능한 워크플로우가 포함되어 있습니다.

기본 워크플로우

```yaml
name: Claude Code
on:
  issue_comment:
    types: [created]
  pull_request_review_comment:
    types: [created]
jobs:
  claude:
    runs-on: ubuntu-latest
    steps:
      - uses: anthropics/claude-code-action@v1
        with:
          anthropic_api_key: ${{ secrets.ANTHROPIC_API_KEY }}
          # Responds to @claude mentions in comments
```

skills 사용

```yaml
name: Code Review
on:
  pull_request:
    types: [opened, synchronize]
jobs:
  review:
    runs-on: ubuntu-latest
    steps:
      - uses: anthropics/claude-code-action@v1
        with:
          anthropic_api_key: ${{ secrets.ANTHROPIC_API_KEY }}
          prompt: "Review this pull request for code quality, correctness, and
security. Analyze the diff, then post your findings as review comments."
          claude_args: "--max-turns 5"
```

프롬프트를 사용한 사용자 정의 자동화

```yaml
name: Daily Report
on:
  schedule:
    - cron: "0 9 * * *"
jobs:
  report:
    runs-on: ubuntu-latest
    steps:
      - uses: anthropics/claude-code-action@v1
        with:
          anthropic_api_key: ${{ secrets.ANTHROPIC_API_KEY }}
          prompt: "Generate a summary of yesterday's commits and open issues"
          claude_args: "--model opus"
```

일반적인 사용 사례

이슈 또는 PR 댓글에서:

```
@claude implement this feature based on the issue description
@claude how should I implement user authentication for this endpoint?
@claude fix the TypeError in the user dashboard component
```

Claude는 자동으로 컨텍스트를 분석하고 적절하게 응답합니다.

모범 사례

CLAUDE.md 구성

저장소 루트에 `CLAUDE.md` 파일을 생성하여 코드 스타일 지침, 리뷰 기준, 프로젝트별 규칙 및 선호하는 패턴을 정의합니다. 이 파일은 Claude의 프로젝트 표준 이해를 안내합니다.

보안 고려 사항

> **Warning:**
> API 키를 저장소에 직접 커밋하지 마십시오.

권한, 인증 및 모범 사례를 포함한 포괄적인 보안 지침은 Claude Code Action 보안 설명서를 참조하십시오.

항상 GitHub Secrets를 API 키에 사용합니다:

- API 키를 `ANTHROPIC_API_KEY` 라는 저장소 시크릿으로 추가합니다
- 워크플로우에서 참조합니다: `anthropic_api_key: ${{ secrets.ANTHROPIC_API_KEY }}`
- 작업 권한을 필요한 것으로만 제한합니다
- 병합하기 전에 Claude의 제안을 검토합니다

API 키를 워크플로우 파일에 직접 하드코딩하는 대신 항상 GitHub Secrets(예: `${{ secrets.ANTHROPIC_API_KEY }}`)를 사용합니다.

성능 최적화

이슈 템플릿을 사용하여 컨텍스트를 제공하고, `CLAUDE.md` 를 간결하고 집중적으로 유지하고, 워크플로우에 적절한 타임아웃을 구성합니다.

CI 비용

Claude Code GitHub Actions를 사용할 때 관련 비용을 인식합니다:

GitHub Actions 비용:

- Claude Code는 GitHub 호스팅 러너에서 실행되며, 이는 GitHub Actions 분을 소비합니다
- 자세한 가격 책정 및 분 제한은 GitHub의 청구 설명서를 참조하십시오

API 비용:

- 각 Claude 상호 작용은 프롬프트 및 응답의 길이에 따라 API 토큰을 소비합니다
- 토큰 사용량은 작업 복잡도 및 코드베이스 크기에 따라 다릅니다
- 현재 토큰 요금은 Claude의 가격 책정 페이지를 참조하십시오

비용 최적화 팁:

- 특정 `@claude` 명령을 사용하여 불필요한 API 호출을 줄입니다
- `claude_args` 에서 적절한 `--max-turns` 를 구성하여 과도한 반복을 방지합니다
- 워크플로우 수준 타임아웃을 설정하여 실행 중인 작업을 방지합니다
- GitHub의 동시성 제어를 사용하여 병렬 실행을 제한하는 것을 고려합니다

구성 예제

Claude Code Action v1은 통합 파라미터로 구성을 단순화합니다:

```
- uses: anthropics/claude-code-action@v1
  with:
    anthropic_api_key: ${{ secrets.ANTHROPIC_API_KEY }}
    prompt: "Your instructions here" # Optional
    claude_args: "--max-turns 5" # Optional CLI arguments
```

주요 기능:

- **통합 프롬프트 인터페이스** - 모든 지침에 `prompt` 사용
- **skills** - 프롬프트에서 설치된 skills를 직접 호출합니다
- **CLI 통과** - `claude_args` 를 통한 모든 Claude Code CLI 인수
- **유연한 트리거** - 모든 GitHub 이벤트와 함께 작동합니다

완전한 워크플로우 파일은 examples 디렉토리를 방문하십시오.

> **Tip:**
>
> 이슈 또는 PR 댓글에 응답할 때 Claude는 자동으로 @claude 멘션에 응답합니다. 다른 이벤트의 경우 `prompt` 파라미터를 사용하여 지침을 제공합니다.

AWS Bedrock & Google Vertex AI 사용

엔터프라이즈 환경의 경우 자신의 클라우드 인프라와 함께 Claude Code GitHub Actions를 사용할 수 있습니다. 이 접근 방식은 동일한 기능을 유지하면서 데이터 거주지 및 청구에 대한 제어를 제공합니다.

필수 조건

클라우드 공급자와 함께 Claude Code GitHub Actions를 설정하기 전에 다음이 필요합니다:

Google Cloud Vertex AI의 경우:

1. Vertex AI가 활성화된 Google Cloud 프로젝트
2. GitHub Actions에 대해 구성된 Workload Identity Federation
3. 필요한 권한이 있는 서비스 계정
4. GitHub 앱(권장) 또는 기본 GITHUB_TOKEN 사용

AWS Bedrock의 경우:

1. Amazon Bedrock이 활성화된 AWS 계정
2. AWS에서 구성된 GitHub OIDC Identity Provider

3. Bedrock 권한이 있는 IAM 역할
4. GitHub 앱(권장) 또는 기본 GITHUB_TOKEN 사용

Step 1: 사용자 정의 GitHub 앱 생성 (3P 공급자에 권장)

Vertex AI 또는 Bedrock과 같은 3P 공급자를 사용할 때 최상의 제어 및 보안을 위해 자신의 GitHub 앱을 생성하는 것이 좋습니다:

1. https://github.com/settings/apps/new로 이동합니다
2. 기본 정보를 입력합니다:

- **GitHub 앱 이름**: 고유한 이름을 선택합니다 (예: "YourOrg Claude Assistant")
- **홈페이지 URL**: 조직의 웹사이트 또는 저장소 URL

1. 앱 설정을 구성합니다:

- **Webhooks**: "Active" 선택 해제 (이 통합에는 필요하지 않음)

1. 필수 권한을 설정합니다:

- **저장소 권한**:
- Contents: 읽기 및 쓰기
- Issues: 읽기 및 쓰기
- Pull requests: 읽기 및 쓰기

1. "GitHub 앱 생성"을 클릭합니다
2. 생성 후 "개인 키 생성"을 클릭하고 다운로드한 `.pem` 파일을 저장합니다
3. 앱 설정 페이지에서 앱 ID를 기록합니다
4. 저장소에 앱을 설치합니다:

- 앱의 설정 페이지에서 왼쪽 사이드바의 "앱 설치"를 클릭합니다
- 계정 또는 조직을 선택합니다
- "선택한 저장소만"을 선택하고 특정 저장소를 선택합니다
- "설치"를 클릭합니다

1. 개인 키를 저장소 시크릿으로 추가합니다:

- 저장소의 설정 → 시크릿 및 변수 → Actions로 이동합니다
- `.pem` 파일의 내용으로 `APP_PRIVATE_KEY`라는 새 시크릿을 생성합니다

1. 앱 ID를 시크릿으로 추가합니다:

- GitHub 앱의 ID로 `APP_ID`라는 새 시크릿을 생성합니다

> **Note:**
>
> 이 앱은 actions/create-github-app-token 작업과 함께 사용되어 워크플로우에서 인증 토큰을 생성합니다.

Claude API의 경우 또는 자신의 Github 앱을 설정하지 않으려는 경우 대안: 공식 Anthropic 앱을 사용합니다:

1. 다음에서 설치합니다: https://github.com/apps/claude
2. 인증을 위한 추가 구성이 필요하지 않습니다

Step 2: 클라우드 공급자 인증 구성

클라우드 공급자를 선택하고 안전한 인증을 설정합니다:

자격 증명을 저장하지 않고 GitHub Actions가 안전하게 인증할 수 있도록 AWS를 구성합니다.

> **보안 참고**: 저장소별 구성을 사용하고 최소 필요 권한만 부여합니다.

필수 설정:

1. **Amazon Bedrock 활성화**:

- Amazon Bedrock에서 Claude 모델에 대한 액세스 요청
- 교차 지역 모델의 경우 모든 필요한 지역에서 액세스 요청

1. **GitHub OIDC Identity Provider 설정**:

- 공급자 URL: `https://token.actions.githubusercontent.com`
- 대상: `sts.amazonaws.com`

1. **GitHub Actions용 IAM 역할 생성**:

- 신뢰할 수 있는 엔티티 유형: 웹 ID
- ID 공급자: `token.actions.githubusercontent.com`
- 권한: `AmazonBedrockFullAccess` 정책
- 특정 저장소에 대한 신뢰 정책 구성

필수 값:

설정 후 다음이 필요합니다:

- **AWS_ROLE_TO_ASSUME**: 생성한 IAM 역할의 ARN

> **Tip:**

> OIDC는 자격 증명이 임시이고 자동으로 회전되기 때문에 정적 AWS 액세스 키를 사용하는 것보다 더 안전합니다.

자세한 OIDC 설정 지침은 AWS 설명서를 참조하십시오.

자격 증명을 저장하지 않고 GitHub Actions가 안전하게 인증할 수 있도록 Google Cloud를 구성합니다.

> **보안 참고**: 저장소별 구성을 사용하고 최소 필요 권한만 부여합니다.

필수 설정:

1. **Google Cloud 프로젝트에서 API 활성화**:

- IAM Credentials API
- Security Token Service (STS) API
- Vertex AI API

1. **Workload Identity Federation 리소스 생성**:

- Workload Identity Pool 생성
- 다음을 사용하여 GitHub OIDC 공급자 추가:
- 발급자: `https://token.actions.githubusercontent.com`
- 저장소 및 소유자에 대한 속성 매핑
- **보안 권장**: 저장소별 속성 조건 사용

1. **서비스 계정 생성**:

- `Vertex AI User` 역할만 부여
- **보안 권장**: 저장소당 전용 서비스 계정 생성

1. **IAM 바인딩 구성**:

- Workload Identity Pool이 서비스 계정을 가장하도록 허용
- **보안 권장**: 저장소별 주체 집합 사용

필수 값:

설정 후 다음이 필요합니다:

- **GCP_WORKLOAD_IDENTITY_PROVIDER**: 전체 공급자 리소스 이름
- **GCP_SERVICE_ACCOUNT**: 서비스 계정 이메일 주소

> **Tip:**

> Workload Identity Federation은 다운로드 가능한 서비스 계정 키의 필요성을 제거하여 보안을 개선합니다.

자세한 설정 지침은 Google Cloud Workload Identity Federation 설명서를 참조하십시오.

Step 3: 필수 시크릿 추가

저장소에 다음 시크릿을 추가합니다 (설정 → 시크릿 및 변수 → Actions):

Claude API의 경우 (직접):

1. **API 인증의 경우**:

- `ANTHROPIC_API_KEY`: console.anthropic.com의 Claude API 키

1. **GitHub 앱의 경우 (자신의 앱을 사용하는 경우)**:

- `APP_ID`: GitHub 앱의 ID
- `APP_PRIVATE_KEY`: 개인 키 (.pem) 내용

Google Cloud Vertex AI의 경우

1. **GCP 인증의 경우**:

- `GCP_WORKLOAD_IDENTITY_PROVIDER`
- `GCP_SERVICE_ACCOUNT`

1. **GitHub 앱의 경우 (자신의 앱을 사용하는 경우)**:

- `APP_ID`: GitHub 앱의 ID
- `APP_PRIVATE_KEY`: 개인 키 (.pem) 내용

AWS Bedrock의 경우

1. **AWS 인증의 경우**:

- `AWS_ROLE_TO_ASSUME`

1. **GitHub 앱의 경우 (자신의 앱을 사용하는 경우)**:

- `APP_ID`: GitHub 앱의 ID
- `APP_PRIVATE_KEY`: 개인 키 (.pem) 내용

Step 4: 워크플로우 파일 생성

클라우드 공급자와 통합되는 GitHub Actions 워크플로우 파일을 생성합니다. 아래 예제는 AWS Bedrock 및 Google Vertex AI 모두에 대한 완전한 구성을 보여줍니다:

필수 조건:

- AWS Bedrock 액세스가 Claude 모델 권한으로 활성화됨
- GitHub가 AWS에서 OIDC ID 공급자로 구성됨
- GitHub Actions를 신뢰하는 Bedrock 권한이 있는 IAM 역할

필수 GitHub 시크릿:

시크릿 이름	설명
`AWS_ROLE_TO_ASSUME`	Bedrock 액세스용 IAM 역할의 ARN
`APP_ID`	GitHub 앱 ID (앱 설정에서)
`APP_PRIVATE_KEY`	GitHub 앱에 대해 생성한 개인 키

```yaml
name: Claude PR Action

permissions:
  contents: write
  pull-requests: write
  issues: write
  id-token: write

on:
  issue_comment:
    types: [created]
  pull_request_review_comment:
    types: [created]
  issues:
    types: [opened, assigned]

jobs:
  claude-pr:
    if: |
      (github.event_name == 'issue_comment' && contains(github.event.comment.body,
'@claude')) ||
      (github.event_name == 'pull_request_review_comment' &&
contains(github.event.comment.body, '@claude')) ||
      (github.event_name == 'issues' && contains(github.event.issue.body,
'@claude'))
    runs-on: ubuntu-latest
    env:
      AWS_REGION: us-west-2
    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Generate GitHub App token
        id: app-token
        uses: actions/create-github-app-token@v2
        with:
          app-id: ${{ secrets.APP_ID }}
          private-key: ${{ secrets.APP_PRIVATE_KEY }}
```

```
      - name: Configure AWS Credentials (OIDC)
        uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume: ${{ secrets.AWS_ROLE_TO_ASSUME }}
          aws-region: us-west-2

      - uses: anthropics/claude-code-action@v1
        with:
          github_token: ${{ steps.app-token.outputs.token }}
          use_bedrock: "true"
          claude_args: '--model us.anthropic.claude-sonnet-4-6 --max-turns 10'
```

> **Tip:**
>
> Bedrock의 모델 ID 형식에는 지역 접두사가 포함됩니다 (예: `us.anthropic.claude-sonnet-4-6`).

필수 조건:

- GCP 프로젝트에서 Vertex AI API 활성화됨
- GitHub에 대해 구성된 Workload Identity Federation
- Vertex AI 권한이 있는 서비스 계정

필수 GitHub 시크릿:

시크릿 이름	설명
`GCP_WORKLOAD_IDENTITY_PROVIDER`	Workload Identity Provider 리소스 이름
`GCP_SERVICE_ACCOUNT`	Vertex AI 액세스 권한이 있는 서비스 계정 이메일
`APP_ID`	GitHub 앱 ID (앱 설정에서)
`APP_PRIVATE_KEY`	GitHub 앱에 대해 생성한 개인 키

```
name: Claude PR Action

permissions:
  contents: write
  pull-requests: write
  issues: write
  id-token: write

on:
  issue_comment:
    types: [created]
  pull_request_review_comment:
    types: [created]
  issues:
    types: [opened, assigned]

jobs:
  claude-pr:
    if: |
      (github.event_name == 'issue_comment' && contains(github.event.comment.body,
'@claude')) ||
      (github.event_name == 'pull_request_review_comment' &&
contains(github.event.comment.body, '@claude')) ||
      (github.event_name == 'issues' && contains(github.event.issue.body,
'@claude'))
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Generate GitHub App token
        id: app-token
        uses: actions/create-github-app-token@v2
        with:
          app-id: ${{ secrets.APP_ID }}
          private-key: ${{ secrets.APP_PRIVATE_KEY }}

      - name: Authenticate to Google Cloud
```

```
        id: auth
        uses: google-github-actions/auth@v2
        with:
          workload_identity_provider: $
{{ secrets.GCP_WORKLOAD_IDENTITY_PROVIDER }}
          service_account: ${{ secrets.GCP_SERVICE_ACCOUNT }}

      - uses: anthropics/claude-code-action@v1
        with:
          github_token: ${{ steps.app-token.outputs.token }}
          trigger_phrase: "@claude"
          use_vertex: "true"
          claude_args: '--model claude-sonnet-4@20250514 --max-turns 10'
        env:
          ANTHROPIC_VERTEX_PROJECT_ID: ${{ steps.auth.outputs.project_id }}
          CLOUD_ML_REGION: us-east5
          VERTEX_REGION_CLAUDE_3_7_SONNET: us-east5
```

> **Tip:**
>
> 프로젝트 ID는 Google Cloud 인증 단계에서 자동으로 검색되므로 하드코딩할 필요가 없습니다.

문제 해결

Claude가 @claude 명령에 응답하지 않음

GitHub 앱이 올바르게 설치되었는지 확인하고, 워크플로우가 활성화되었는지 확인하고, API 키가 저장소 시크릿에 설정되었는지 확인하고, 댓글에 `@claude`가 포함되어 있는지 확인합니다 (`/claude` 아님).

Claude의 커밋에서 CI가 실행되지 않음

GitHub 앱 또는 사용자 정의 앱을 사용 중인지 확인합니다 (Actions 사용자 아님), 워크플로우 트리거에 필요한 이벤트가 포함되어 있는지 확인하고, 앱 권한에 CI 트리거가 포함되어 있는지 확인합니다.

인증 오류

API 키가 유효하고 충분한 권한이 있는지 확인합니다. Bedrock/Vertex의 경우 자격 증명 구성을 확인하고 시크릿이 워크플로우에서 올바르게 명명되었는지 확인합니다.

고급 구성

작업 파라미터

Claude Code Action v1은 단순화된 구성을 사용합니다:

파라미터	설명	필수
`prompt`	Claude에 대한 지침 (일반 텍스트 또는 skill 이름)	아니오*
`claude_args`	Claude Code에 전달된 CLI 인수	아니오
`anthropic_api_key`	Claude API 키	예**
`github_token`	API 액세스용 GitHub 토큰	아니오
`trigger_phrase`	사용자 정의 트리거 구문 (기본값: “@claude”)	아니오
`use_bedrock`	Claude API 대신 AWS Bedrock 사용	아니오
`use_vertex`	Claude API 대신 Google Vertex AI 사용	아니오

*프롬프트는 선택 사항입니다. 이슈/PR 댓글에서 생략하면 Claude는 트리거 구문에 응답합니다
**직접 Claude API에 필수이며, Bedrock/Vertex에는 필수가 아닙니다

CLI 인수 전달

`claude_args` 파라미터는 모든 Claude Code CLI 인수를 허용합니다:

```
claude_args: "--max-turns 5 --model claude-sonnet-4-6 --mcp-config /path/to/
config.json"
```

일반적인 인수:

- `--max-turns` : 최대 대화 턴 (기본값: 10)
- `--model` : 사용할 모델 (예: `claude-sonnet-4-6`)
- `--mcp-config` : MCP 구성 경로
- `--allowed-tools` : 허용된 도구의 쉼표로 구분된 목록
- `--debug` : 디버그 출력 활성화

대체 통합 방법

`/install-github-app` 명령이 권장되는 접근 방식이지만 다음을 수행할 수도 있습니다:

- **사용자 정의 GitHub 앱**: 브랜드 사용자 이름 또는 사용자 정의 인증 흐름이 필요한 조직의 경우. 필요한 권한(contents, issues, pull requests)으로 자신의 GitHub 앱을 생성하고 actions/create-github-app-token 작업을 사용하여 워크플로우에서 토큰을 생성합니다.
- **수동 GitHub Actions**: 최대 유연성을 위한 직접 워크플로우 구성
- **MCP 구성**: Model Context Protocol 서버의 동적 로딩

자세한 인증, 보안 및 고급 구성 가이드는 Claude Code Action 설명서를 참조하십시오.

Claude의 동작 사용자 정의

두 가지 방법으로 Claude의 동작을 구성할 수 있습니다:

1. **CLAUDE.md**: 저장소의 루트에 `CLAUDE.md` 파일에서 코딩 표준, 리뷰 기준 및 프로젝트별 규칙을 정의합니다. Claude는 PR을 생성하고 요청에 응답할 때 이러한 지침을 따릅니다. 자세한 내용은 Memory 설명서를 확인하십시오.
2. **사용자 정의 프롬프트**: 워크플로우 파일의 `prompt` 파라미터를 사용하여 워크플로우별 지침을 제공합니다. 이를 통해 다양한 워크플로우 또는 작업에 대해 Claude의 동작을 사용자 정의할 수 있습니다.

Claude는 PR을 생성하고 요청에 응답할 때 이러한 지침을 따릅니다.

Claude Code GitLab CI/CD

> Claude Code를 GitLab CI/CD와 함께 개발 워크플로우에 통합하는 방법을 알아봅니다

Info:

Claude Code for GitLab CI/CD는 현재 베타 버전입니다. 경험을 개선하면서 기능과 기능성이 진화할 수 있습니다.

이 통합은 GitLab에서 유지 관리합니다. 지원을 받으려면 다음 GitLab 이슈를 참조하세요.

Note:

이 통합은 Claude Code CLI and Agent SDK 위에 구축되어 있으며, CI/CD 작업 및 사용자 정의 자동화 워크플로우에서 Claude를 프로그래밍 방식으로 사용할 수 있습니다.

GitLab에서 Claude Code를 사용하는 이유

- **즉시 MR 생성**: 필요한 사항을 설명하면 Claude가 변경 사항과 설명이 포함된 완전한 MR을 제안합니다
- **자동화된 구현**: 단일 명령 또는 언급으로 이슈를 작동하는 코드로 변환합니다
- **프로젝트 인식**: Claude는 `CLAUDE.md` 지침과 기존 코드 패턴을 따릅니다
- **간단한 설정**: `.gitlab-ci.yml` 에 하나의 작업과 마스킹된 CI/CD 변수를 추가합니다
- **엔터프라이즈 준비**: Claude API, AWS Bedrock 또는 Google Vertex AI를 선택하여 데이터 거주지 및 조달 요구 사항을 충족합니다
- **기본적으로 안전**: GitLab 러너에서 실행되며 브랜치 보호 및 승인이 적용됩니다

작동 방식

Claude Code는 GitLab CI/CD를 사용하여 격리된 작업에서 AI 작업을 실행하고 MR을 통해 결과를 다시 커밋합니다:

1. **이벤트 기반 오케스트레이션**: GitLab은 선택한 트리거(예: 이슈, MR 또는 검토 스레드에서 `@claude`를 언급하는 댓글)를 수신합니다. 작업은 스레드 및 저장소에서 컨텍스트를 수집하고, 해당 입력에서 프롬프트를 작성하고, Claude Code를 실행합니다.
2. **공급자 추상화**: 환경에 맞는 공급자를 사용합니다:
    - Claude API (SaaS)

    - AWS Bedrock (IAM 기반 액세스, 교차 지역 옵션)
    - Google Vertex AI (GCP 네이티브, Workload Identity Federation)

3. **샌드박스 실행**: 각 상호 작용은 엄격한 네트워크 및 파일 시스템 규칙이 있는 컨테이너에서 실행됩니다. Claude Code는 쓰기를 제한하기 위해 작업 공간 범위 권한을 적용합니다. 모든 변경 사항은 MR을 통해 흐르므로 검토자가 diff를 보고 승인이 여전히 적용됩니다.

지역 엔드포인트를 선택하여 지연 시간을 줄이고 기존 클라우드 계약을 사용하면서 데이터 주권 요구 사항을 충족합니다.

Claude가 할 수 있는 것

Claude Code는 코드 작업 방식을 변환하는 강력한 CI/CD 워크플로우를 활성화합니다:

- 이슈 설명 또는 댓글에서 MR 생성 및 업데이트
- 성능 회귀 분석 및 최적화 제안
- 브랜치에 직접 기능 구현 후 MR 열기
- 테스트 또는 댓글로 식별된 버그 및 회귀 수정
- 후속 댓글에 응답하여 요청된 변경 사항에 대해 반복

설정

빠른 설정

가장 빠른 시작 방법은 `.gitlab-ci.yml`에 최소 작업을 추가하고 API 키를 마스킹된 변수로 설정하는 것입니다.

1. **마스킹된 CI/CD 변수 추가**
    - **설정** → **CI/CD** → **변수**로 이동합니다
    - `ANTHROPIC_API_KEY` 추가 (마스킹됨, 필요에 따라 보호됨)
2. **`.gitlab-ci.yml`에 Claude 작업 추가**

```yaml
stages:
  - ai

claude:
  stage: ai
  image: node:24-alpine3.21
  # 작업을 트리거하는 방법에 맞게 규칙을 조정합니다:
  # - 수동 실행
  # - 병합 요청 이벤트
  # - '@claude'를 포함하는 댓글이 있을 때 웹/API 트리거
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web"'
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'
  variables:
    GIT_STRATEGY: fetch
  before_script:
    - apk update
    - apk add --no-cache git curl bash
    - curl -fsSL https://claude.ai/install.sh | bash
  script:
    # 선택 사항: 설정에서 제공하는 경우 GitLab MCP 서버 시작
    - /bin/gitlab-mcp-server || true
    # 웹/API 트리거를 통해 컨텍스트 페이로드로 호출할 때 AI_FLOW_* 변수 사용
    - echo "$AI_FLOW_INPUT for $AI_FLOW_CONTEXT on $AI_FLOW_EVENT"
    - >
      claude
      -p "${AI_FLOW_INPUT:-'Review this MR and implement the requested changes'}"
      --permission-mode acceptEdits
      --allowedTools "Bash Read Edit Write mcp__gitlab"
      --debug
```

작업과 `ANTHROPIC_API_KEY` 변수를 추가한 후 **CI/CD** → **파이프라인**에서 작업을 수동으로 실행하여 테스트하거나, MR에서 트리거하여 Claude가 브랜치에서 업데이트를 제안하고 필요한 경우 MR을 열도록 합니다.

> **Note:**

> Claude API 대신 AWS Bedrock 또는 Google Vertex AI에서 실행하려면 아래의 AWS Bedrock & Google Vertex AI 사용 섹션을 참조하여 인증 및 환경 설정을 확인하세요.

수동 설정 (프로덕션에 권장)

더 제어된 설정을 선호하거나 엔터프라이즈 공급자가 필요한 경우:

1. **공급자 액세스 구성**:
   - **Claude API**: `ANTHROPIC_API_KEY`를 생성하고 마스킹된 CI/CD 변수로 저장합니다
   - **AWS Bedrock**: **GitLab 구성 → AWS OIDC**를 구성하고 Bedrock용 IAM 역할을 생성합니다
   - **Google Vertex AI**: **GitLab용 Workload Identity Federation 구성 → GCP**
2. **GitLab API 작업을 위한 프로젝트 자격 증명 추가**:
   - 기본적으로 `CI_JOB_TOKEN`을 사용하거나 `api` 범위가 있는 프로젝트 액세스 토큰을 생성합니다
   - PAT를 사용하는 경우 `GITLAB_ACCESS_TOKEN` (마스킹됨)으로 저장합니다
3. **Claude 작업을 `.gitlab-ci.yml`에 추가** (아래 예제 참조)
4. **(선택 사항) 언급 기반 트리거 활성화**:
   - 이벤트 리스너에 "댓글 (노트)"에 대한 프로젝트 웹훅을 추가합니다 (사용하는 경우)
   - 댓글에 `@claude`가 포함될 때 `AI_FLOW_INPUT` 및 `AI_FLOW_CONTEXT`와 같은 변수로 파이프라인 트리거 API를 호출하도록 리스너를 설정합니다

예제 사용 사례

이슈를 MR로 변환

이슈 댓글에서:

```
@claude implement this feature based on the issue description
```

Claude는 이슈 및 코드베이스를 분석하고, 브랜치에서 변경 사항을 작성하고, 검토를 위해 MR을 엽니다.

구현 도움 받기

MR 토론에서:

```
@claude suggest a concrete approach to cache the results of this API call
```

Claude는 변경 사항을 제안하고, 적절한 캐싱으로 코드를 추가하고, MR을 업데이트합니다.

버그 빠르게 수정

이슈 또는 MR 댓글에서:

```
@claude fix the TypeError in the user dashboard component
```

Claude는 버그를 찾고, 수정을 구현하고, 브랜치를 업데이트하거나 새 MR을 엽니다.

AWS Bedrock & Google Vertex AI 사용

엔터프라이즈 환경의 경우 동일한 개발자 경험으로 클라우드 인프라에서 완전히 Claude Code를 실행할 수 있습니다.

AWS Bedrock

필수 조건

AWS Bedrock으로 Claude Code를 설정하기 전에 다음이 필요합니다:

1. 원하는 Claude 모델에 대한 Amazon Bedrock 액세스가 있는 AWS 계정
2. AWS IAM에서 OIDC 자격 증명 공급자로 구성된 GitLab
3. Bedrock 권한이 있는 IAM 역할 및 GitLab 프로젝트/참조로 제한된 신뢰 정책
4. 역할 가정을 위한 GitLab CI/CD 변수:

- `AWS_ROLE_TO_ASSUME` (역할 ARN)
- `AWS_REGION` (Bedrock 지역)

설정 지침

OIDC를 통해 GitLab CI 작업이 IAM 역할을 가정하도록 AWS를 구성합니다 (정적 키 없음).

필수 설정:

1. Amazon Bedrock을 활성화하고 대상 Claude 모델에 대한 액세스를 요청합니다
2. 아직 없는 경우 GitLab용 IAM OIDC 공급자를 생성합니다
3. GitLab OIDC 공급자를 신뢰하고 프로젝트 및 보호된 참조로 제한된 IAM 역할을 생성합니다
4. Bedrock 호출 API에 대한 최소 권한 권한을 연결합니다

CI/CD 변수에 저장할 필수 값:

- `AWS_ROLE_TO_ASSUME`
- `AWS_REGION`

설정 → CI/CD → 변수에서 변수를 추가합니다:

```
## AWS Bedrock의 경우:
- AWS_ROLE_TO_ASSUME
- AWS_REGION
```

위의 AWS Bedrock 작업 예제를 사용하여 런타임에 GitLab 작업 토큰을 임시 AWS 자격 증명으로 교환합니다.

Google Vertex AI

필수 조건

Google Vertex AI로 Claude Code를 설정하기 전에 다음이 필요합니다:

1. 다음이 포함된 Google Cloud 프로젝트:

- Vertex AI API 활성화됨
- GitLab OIDC를 신뢰하도록 구성된 Workload Identity Federation

1. 필요한 Vertex AI 역할만 있는 전용 서비스 계정
2. WIF용 GitLab CI/CD 변수:

- `GCP_WORKLOAD_IDENTITY_PROVIDER` (전체 리소스 이름)
- `GCP_SERVICE_ACCOUNT` (서비스 계정 이메일)

설정 지침

Workload Identity Federation을 통해 GitLab CI 작업이 서비스 계정을 가장하도록 Google Cloud를 구성합니다.

필수 설정:

1. IAM Credentials API, STS API 및 Vertex AI API 활성화
2. GitLab OIDC용 Workload Identity Pool 및 공급자 생성
3. Vertex AI 역할이 있는 전용 서비스 계정 생성
4. WIF 주체에 서비스 계정을 가장할 수 있는 권한 부여

CI/CD 변수에 저장할 필수 값:

- `GCP_WORKLOAD_IDENTITY_PROVIDER`
- `GCP_SERVICE_ACCOUNT`

설정 → CI/CD → 변수에서 변수를 추가합니다:

```
## Google Vertex AI의 경우:
- GCP_WORKLOAD_IDENTITY_PROVIDER
- GCP_SERVICE_ACCOUNT
- CLOUD_ML_REGION (예: us-east5)
```

위의 Google Vertex AI 작업 예제를 사용하여 키를 저장하지 않고 인증합니다.

구성 예제

파이프라인에 맞게 조정할 수 있는 즉시 사용 가능한 스니펫입니다.

기본 .gitlab-ci.yml (Claude API)

```
stages:
  - ai

claude:
  stage: ai
  image: node:24-alpine3.21
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web"'
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'
  variables:
    GIT_STRATEGY: fetch
  before_script:
    - apk update
    - apk add --no-cache git curl bash
    - curl -fsSL https://claude.ai/install.sh | bash
  script:
    - /bin/gitlab-mcp-server || true
    - >
      claude
      -p "${AI_FLOW_INPUT:-'Summarize recent changes and suggest improvements'}"
      --permission-mode acceptEdits
      --allowedTools "Bash Read Edit Write mcp__gitlab"
      --debug
  # Claude Code는 CI/CD 변수에서 ANTHROPIC_API_KEY를 사용합니다
```

AWS Bedrock 작업 예제 (OIDC)

필수 조건:

- Amazon Bedrock이 활성화되고 선택한 Claude 모델에 액세스 가능
- GitLab OIDC가 AWS에 구성되고 GitLab 프로젝트 및 참조를 신뢰하는 역할
- Bedrock 권한이 있는 IAM 역할 (최소 권한 권장)

필수 CI/CD 변수:

- `AWS_ROLE_TO_ASSUME` : Bedrock 액세스용 IAM 역할의 ARN
- `AWS_REGION` : Bedrock 지역 (예: `us-west-2`)

```
claude-bedrock:
  stage: ai
  image: node:24-alpine3.21
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web"'
  before_script:
    - apk add --no-cache bash curl jq git python3 py3-pip
    - pip install --no-cache-dir awscli
    - curl -fsSL https://claude.ai/install.sh | bash
    # GitLab OIDC 토큰을 AWS 자격 증명으로 교환
    - export AWS_WEB_IDENTITY_TOKEN_FILE="${CI_JOB_JWT_FILE:-/tmp/oidc_token}"
    - if [ -n "${CI_JOB_JWT_V2}" ]; then printf "%s" "$CI_JOB_JWT_V2" >
"$AWS_WEB_IDENTITY_TOKEN_FILE"; fi
    - >
      aws sts assume-role-with-web-identity
      --role-arn "$AWS_ROLE_TO_ASSUME"
      --role-session-name "gitlab-claude-$(date +%s)"
      --web-identity-token "file://$AWS_WEB_IDENTITY_TOKEN_FILE"
      --duration-seconds 3600 > /tmp/aws_creds.json
    - export AWS_ACCESS_KEY_ID="$(jq -r .Credentials.AccessKeyId /tmp/
aws_creds.json)"
    - export AWS_SECRET_ACCESS_KEY="$(jq -r .Credentials.SecretAccessKey /tmp/
aws_creds.json)"
    - export AWS_SESSION_TOKEN="$(jq -r .Credentials.SessionToken /tmp/
aws_creds.json)"
  script:
    - /bin/gitlab-mcp-server || true
    - >
      claude
      -p "${AI_FLOW_INPUT:-'Implement the requested changes and open an MR'}"
      --permission-mode acceptEdits
      --allowedTools "Bash Read Edit Write mcp__gitlab"
      --debug
  variables:
    AWS_REGION: "us-west-2"
```

Note:

> Bedrock의 모델 ID에는 지역별 접두사가 포함됩니다 (예: `us.anthropic.claude-sonnet-4-6`). 워크플로우에서 지원하는 경우 작업 구성 또는 프롬프트를 통해 원하는 모델을 전달합니다.

Google Vertex AI 작업 예제 (Workload Identity Federation)

필수 조건:

- GCP 프로젝트에서 Vertex AI API 활성화됨
- GitLab OIDC를 신뢰하도록 구성된 Workload Identity Federation
- Vertex AI 권한이 있는 서비스 계정

필수 CI/CD 변수:

- `GCP_WORKLOAD_IDENTITY_PROVIDER` : 전체 공급자 리소스 이름
- `GCP_SERVICE_ACCOUNT` : 서비스 계정 이메일
- `CLOUD_ML_REGION` : Vertex 지역 (예: `us-east5`)

```
claude-vertex:
  stage: ai
  image: gcr.io/google.com/cloudsdktool/google-cloud-cli:slim
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web"'
  before_script:
    - apt-get update && apt-get install -y git && apt-get clean
    - curl -fsSL https://claude.ai/install.sh | bash
    # WIF를 통해 Google Cloud에 인증 (다운로드된 키 없음)
    - >
      gcloud auth login --cred-file=<(cat <<EOF
      {
        "type": "external_account",
        "audience": "${GCP_WORKLOAD_IDENTITY_PROVIDER}",
        "subject_token_type": "urn:ietf:params:oauth:token-type:jwt",
        "service_account_impersonation_url": "https://
iamcredentials.googleapis.com/v1/projects/-/serviceAccounts/$
{GCP_SERVICE_ACCOUNT}:generateAccessToken",
        "token_url": "https://sts.googleapis.com/v1/token"
      }
      EOF
      )
    - gcloud config set project "$(gcloud projects list --
format='value(projectId)' --filter="name:${CI_PROJECT_NAMESPACE}" | head -n1)" ||
true
  script:
    - /bin/gitlab-mcp-server || true
    - >
      CLOUD_ML_REGION="${CLOUD_ML_REGION:-us-east5}"
      claude
      -p "${AI_FLOW_INPUT:-'Review and update code as requested'}"
      --permission-mode acceptEdits
      --allowedTools "Bash Read Edit Write mcp__gitlab"
      --debug
  variables:
    CLOUD_ML_REGION: "us-east5"
```

> **Note:**
>
> Workload Identity Federation을 사용하면 서비스 계정 키를 저장할 필요가 없습니다. 저장소별 신뢰 조건 및 최소 권한 서비스 계정을 사용합니다.

모범 사례

CLAUDE.md 구성

저장소 루트에 `CLAUDE.md` 파일을 생성하여 코딩 표준, 검토 기준 및 프로젝트별 규칙을 정의합니다. Claude는 실행 중에 이 파일을 읽고 변경 사항을 제안할 때 규칙을 따릅니다.

보안 고려 사항

API 키 또는 클라우드 자격 증명을 저장소에 커밋하지 마세요. 항상 GitLab CI/CD 변수를 사용합니다:

- `ANTHROPIC_API_KEY`를 마스킹된 변수로 추가합니다 (필요한 경우 보호)
- 가능한 경우 공급자별 OIDC를 사용합니다 (장기 키 없음)
- 작업 권한 및 네트워크 송신 제한
- 다른 기여자처럼 Claude의 MR을 검토합니다

성능 최적화

- `CLAUDE.md`를 집중적이고 간결하게 유지합니다
- 명확한 이슈/MR 설명을 제공하여 반복을 줄입니다
- 작업 시간 초과를 구성하여 실행 중단을 방지합니다
- 가능한 경우 러너에서 npm 및 패키지 설치를 캐시합니다

CI 비용

GitLab CI/CD와 함께 Claude Code를 사용할 때 관련 비용을 인식합니다:

- **GitLab Runner 시간**:
  - Claude는 GitLab 러너에서 실행되고 컴퓨팅 분을 소비합니다
  - GitLab 플랜의 러너 청구 세부 정보를 참조하세요
- **API 비용**:
  - 각 Claude 상호 작용은 프롬프트 및 응답 크기에 따라 토큰을 소비합니다
  - 토큰 사용량은 작업 복잡도 및 코드베이스 크기에 따라 다릅니다
  - Anthropic 가격 책정 세부 정보를 참조하세요

- **비용 최적화 팁**:
  - 특정 `@claude` 명령을 사용하여 불필요한 턴을 줄입니다
  - 적절한 `max_turns` 및 작업 시간 초과 값을 설정합니다
  - 동시성을 제한하여 병렬 실행을 제어합니다

보안 및 거버넌스

- 각 작업은 제한된 네트워크 액세스가 있는 격리된 컨테이너에서 실행됩니다
- Claude의 변경 사항은 MR을 통해 흐르므로 검토자가 모든 diff를 봅니다
- 브랜치 보호 및 승인 규칙이 AI 생성 코드에 적용됩니다
- Claude Code는 쓰기를 제한하기 위해 작업 공간 범위 권한을 사용합니다
- 자신의 공급자 자격 증명을 가져오기 때문에 비용이 제어됩니다

문제 해결

Claude가 @claude 명령에 응답하지 않음

- 파이프라인이 트리거되고 있는지 확인합니다 (수동, MR 이벤트 또는 노트 이벤트 리스너/웹훅을 통해)
- CI/CD 변수 (`ANTHROPIC_API_KEY` 또는 클라우드 공급자 설정)가 있고 마스킹 해제되어 있는지 확인합니다
- 댓글에 `@claude` (not `/claude`)가 포함되어 있고 언급 트리거가 구성되어 있는지 확인합니다

작업이 댓글을 쓰거나 MR을 열 수 없음

- `CI_JOB_TOKEN` 이 프로젝트에 대한 충분한 권한이 있거나 `api` 범위가 있는 프로젝트 액세스 토큰을 사용하는지 확인합니다
- `mcp__gitlab` 도구가 `--allowedTools` 에서 활성화되어 있는지 확인합니다
- 작업이 MR의 컨텍스트에서 실행되거나 `AI_FLOW_*` 변수를 통해 충분한 컨텍스트가 있는지 확인합니다

인증 오류

- **Claude API의 경우**: `ANTHROPIC_API_KEY` 가 유효하고 만료되지 않았는지 확인합니다
- **Bedrock/Vertex의 경우**: OIDC/WIF 구성, 역할 가장 및 비밀 이름을 확인합니다. 지역 및 모델 가용성을 확인합니다

고급 구성

일반적인 매개변수 및 변수

Claude Code는 다음과 같이 일반적으로 사용되는 입력을 지원합니다:

- `prompt` / `prompt_file` : 인라인 (`-p`) 또는 파일을 통해 지침을 제공합니다
- `max_turns` : 왕복 반복 횟수를 제한합니다
- `timeout_minutes` : 총 실행 시간을 제한합니다
- `ANTHROPIC_API_KEY` : Claude API에 필요합니다 (Bedrock/Vertex에는 사용되지 않음)
- 공급자별 환경: `AWS_REGION` , Vertex용 프로젝트/지역 변수

> **Note:**
>
> 정확한 플래그 및 매개변수는 `@anthropic-ai/claude-code` 버전에 따라 다를 수 있습니다. 작업에서 `claude --help` 를 실행하여 지원되는 옵션을 확인합니다.

Claude의 동작 사용자 정의

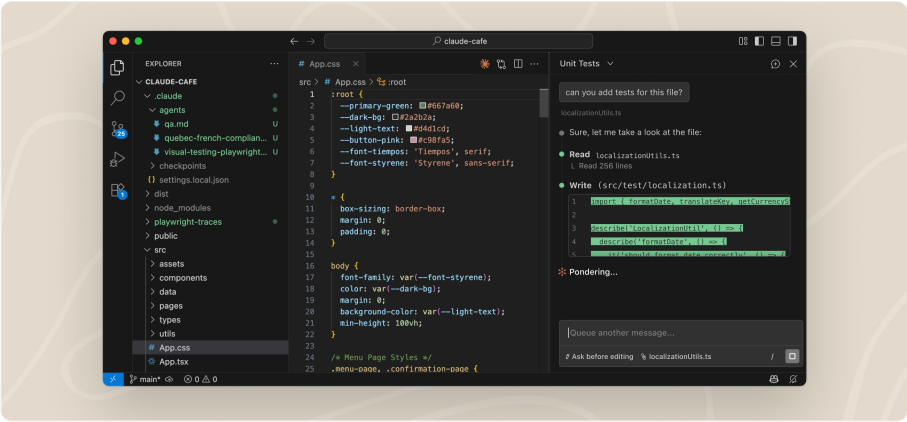
두 가지 주요 방법으로 Claude를 안내할 수 있습니다:

1. **CLAUDE.md**: 코딩 표준, 보안 요구 사항 및 프로젝트 규칙을 정의합니다. Claude는 실행 중에 이를 읽고 규칙을 따릅니다.
2. **사용자 정의 프롬프트**: 작업에서 `prompt` / `prompt_file` 을 통해 작업별 지침을 전달합니다. 다양한 작업에 다양한 프롬프트를 사용합니다 (예: 검토, 구현, 리팩토링).

Part 8: IDE & Platform Integration

VS Code에서 Claude Code 사용하기

VS Code용 Claude Code 확장 프로그램을 설치하고 구성합니다. 인라인 diff, @-멘션, 계획 검토 및 키보드 단축키를 통해 AI 코딩 지원을 받습니다.



VS Code 편집기와 오른쪽에 열린 Claude Code 확장 프로그램 패널, Claude와의 대화를 표시

VS Code 확장 프로그램은 Claude Code를 위한 기본 그래픽 인터페이스를 제공하며, IDE에 직접 통합됩니다. 이것이 VS Code에서 Claude Code를 사용하는 권장 방법입니다.

확장 프로그램을 사용하면 Claude의 계획을 수락하기 전에 검토하고 편집할 수 있으며, 편집이 이루어질 때 자동으로 수락하고, 선택 항목에서 특정 줄 범위가 있는 파일을 @-멘션하고, 대화 기록에 액세스하고, 별도의 탭이나 창에서 여러 대화를 열 수 있습니다.

필수 조건

설치하기 전에 다음을 확인하십시오:

- VS Code 1.98.0 이상
- Anthropic 계정(확장 프로그램을 처음 열 때 로그인합니다). Amazon Bedrock이나 Google Vertex AI와 같은 타사 제공자를 사용하는 경우 대신 타사 제공자 사용을 참조하십시오.

Tip:

확장 프로그램에는 CLI(명령줄 인터페이스)가 포함되어 있으며, VS Code의 통합 터미널에서 고급 기능에 액세스할 수 있습니다. 자세한 내용은 VS Code 확장 프로그램 vs. Claude Code CLI를 참조하십시오.

확장 프로그램 설치

IDE에 대한 링크를 클릭하여 직접 설치합니다:

- VS Code용 설치
- Cursor용 설치

또는 VS Code에서 `Cmd+Shift+X` (Mac) 또는 `Ctrl+Shift+X` (Windows/Linux)를 눌러 확장 프로그램 보기를 열고, "Claude Code"를 검색한 후 **설치**를 클릭합니다.

Note:

설치 후 확장 프로그램이 나타나지 않으면 VS Code를 다시 시작하거나 명령 팔레트에서 "Developer: Reload Window" 를 실행합니다.

시작하기

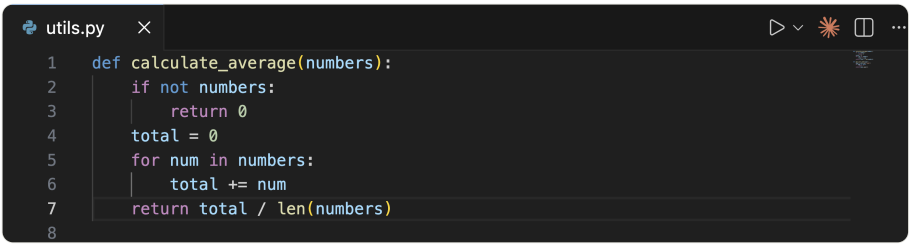
설치 후 VS Code 인터페이스를 통해 Claude Code를 사용할 수 있습니다:

Step 1: Claude Code 패널 열기

VS Code 전체에서 Spark 아이콘은 Claude Code를 나타냅니다:

Spark 아이콘

Claude를 여는 가장 빠른 방법은 **편집기 도구 모음**(편집기의 오른쪽 위 모서리)에서 Spark 아이콘을 클릭하는 것입니다. 이 아이콘은 파일을 열었을 때만 나타납니다.



VS Code 편집기 도구 모음에서 Spark 아이콘을 표시하는 VS Code 편집기

Claude Code를 여는 다른 방법:

- **활동 표시줄**: 왼쪽 사이드바에서 Spark 아이콘을 클릭하여 세션 목록을 엽니다. 모든 세션을 클릭하여 전체 편집기 탭으로 열거나 새 세션을 시작합니다. 이 아이콘은 항상 활동 표시줄에 표시됩니다.
- **명령 팔레트**: `Cmd+Shift+P` (Mac) 또는 `Ctrl+Shift+P` (Windows/Linux), "Claude Code"를 입력하고 "새 탭에서 열기"와 같은 옵션을 선택합니다.
- **상태 표시줄**: 창의 오른쪽 아래 모서리에서 **Claude Code**를 클릭합니다. 파일을 열지 않았을 때도 작동합니다.

패널을 처음 열 때 **Learn Claude Code** 체크리스트가 나타납니다. **Show me**를 클릭하여 각 항목을 진행하거나 X로 닫습니다. 나중에 다시 열려면 VS Code 설정의 확장 프로그램 → Claude Code에서 **Hide Onboarding**을 선택 해제합니다.

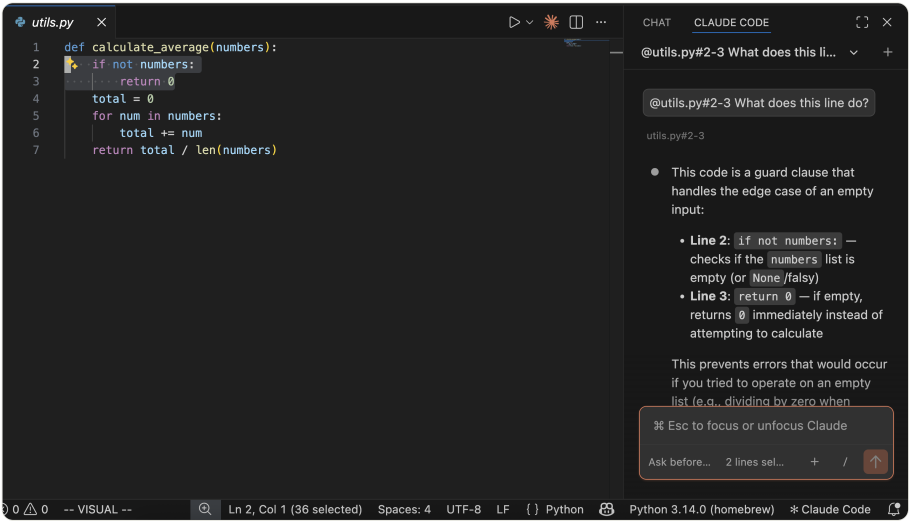
Claude 패널을 드래그하여 VS Code의 어느 곳이든 다시 배치할 수 있습니다. 자세한 내용은 워크플로우 사용자 정의를 참조하십시오.

Step 2: 프롬프트 보내기

Claude에게 코드나 파일을 도와달라고 요청합니다. 작동 방식 설명, 문제 디버깅 또는 변경 사항 만들기 등이 있습니다.

> **Tip:**
>
> Claude는 자동으로 선택한 텍스트를 봅니다. `Option+K` (Mac) / `Alt+K` (Windows/Linux)를 눌러 프롬프트에 @-멘션 참조(예: `@file.ts#5-10`)를 삽입합니다.

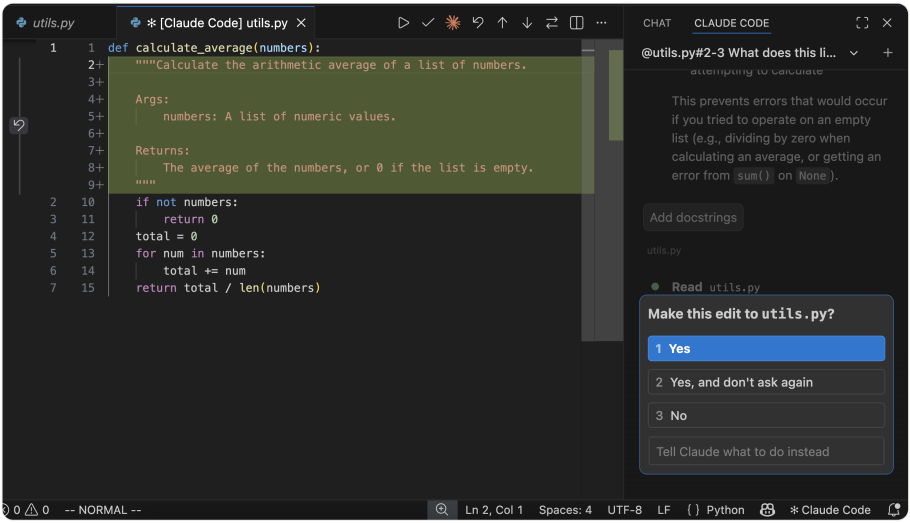
파일의 특정 줄에 대해 묻는 예제입니다:



VS Code 편집기에서 Python 파일의 2-3줄이 선택되고, Claude Code 패널에서 @-멘션 참조가 있는 해당 줄에 대한 질문을 표시

Step 3: 변경 사항 검토

Claude가 파일을 편집하려고 할 때, 원본과 제안된 변경 사항을 나란히 비교하고 권한을 요청합니다. 수락하거나 거부하거나 Claude에게 대신 수행할 작업을 알릴 수 있습니다.



VS Code에서 Claude의 제안된 변경 사항의 diff를 표시하고 편집을 수행할지 여부를 묻는 권한 프롬프트

Claude Code로 수행할 수 있는 작업에 대한 더 많은 아이디어는 일반적인 워크플로우를 참조하십시오.

> **Tip:**
>
> 명령 팔레트에서 "Claude Code: Open Walkthrough"를 실행하여 기본 사항에 대한 안내 투어를 받습니다.

프롬프트 상자 사용

프롬프트 상자는 여러 기능을 지원합니다:

- **권한 모드**: 프롬프트 상자 하단의 모드 표시기를 클릭하여 모드를 전환합니다. 일반 모드에서 Claude는 각 작업 전에 권한을 요청합니다. Plan Mode에서 Claude는 수행할 작업을 설명하고 변경을 수행하기 전에 승인을 기다립니다. VS Code는 자동으로 계획을 전체 마크다운 문서로 열어서 Claude가 시작하기 전에 피드백을 제공하기 위해 인라인 주석을 추가할 수 있습니다. 자동 수락 모드에서 Claude는 요청 없이 편집을 수행합니다. VS Code 설정의 `claudeCode.initialPermissionMode`에서 기본값을 설정합니다.
- **명령 메뉴**: `/`를 클릭하거나 `/`를 입력하여 명령 메뉴를 엽니다. 옵션에는 파일 첨부, 모델 전환, 확장 사고 토글 및 계획 사용량 보기(`/usage`)가 포함됩니다. 사용자 정의 섹션은 MCP 서버, hooks, 메모리, 권한 및 플러그인에 대한 액세스를 제공합니다. 터미널 아이콘이 있는 항목은 통합 터미널에서 열립니다.
- **컨텍스트 표시기**: 프롬프트 상자는 Claude의 context window를 얼마나 사용하고 있는지 표시합니다. Claude는 필요할 때 자동으로 압축하거나 `/compact`를 수동으로 실행할 수 있습니다.
- **확장 사고**: Claude가 복잡한 문제를 추론하는 데 더 많은 시간을 소비할 수 있습니다. 명령 메뉴(`/`)를 통해 켭니다. 자세한 내용은 확장 사고를 참조하십시오.
- **여러 줄 입력**: `Shift+Enter`를 눌러 보내지 않고 새 줄을 추가합니다. 이것은 질문 대화의 "기타" 자유 텍스트 입력에서도 작동합니다.

파일 및 폴더 참조

@-멘션을 사용하여 특정 파일이나 폴더에 대한 컨텍스트를 Claude에게 제공합니다. `@` 다음에 파일 또는 폴더 이름을 입력하면 Claude는 해당 콘텐츠를 읽고 이에 대해 질문하거나 변경할 수 있습니다. Claude Code는 fuzzy matching을 지원하므로 부분 이름을 입력하여 필요한 것을 찾을 수 있습니다:

```
> Explain the logic in @auth (fuzzy matches auth.js, AuthService.ts, etc.)
> What's in @src/components/ (include a trailing slash for folders)
```

큰 PDF의 경우 Claude에게 전체 파일 대신 특정 페이지를 읽도록 요청할 수 있습니다: 단일 페이지, 1-10페이지와 같은 범위 또는 3페이지 이상과 같은 개방형 범위입니다.

편집기에서 텍스트를 선택하면 Claude는 강조 표시된 코드를 자동으로 볼 수 있습니다. 프롬프트 상자 바닥글은 선택된 줄 수를 표시합니다. `Option+K` (Mac) / `Alt+K` (Windows/Linux)를 눌러 파일 경로 및 줄 번호(예: `@app.ts#5-10`)가 있는 @-멘션을 삽입합니다. 선택 표시기를 클릭하여 Claude가 강조 표시된 텍스트를 볼 수 있는지 여부를 전환합니다. 눈 슬래시 아이콘은 선택이 Claude에서 숨겨져 있음을 의미합니다.

또한 `Shift`를 누른 상태에서 파일을 프롬프트 상자로 드래그하여 첨부 파일로 추가할 수 있습니다. 모든 첨부 파일의 X를 클릭하여 컨텍스트에서 제거합니다.

과거 대화 재개

Claude Code 패널 상단의 드롭다운을 클릭하여 대화 기록에 액세스합니다. 키워드로 검색하거나 시간별로 찾아볼 수 있습니다(오늘, 어제, 지난 7일 등). 모든 대화를 클릭하여 전체 메시지 기록으로 재개합니다. 세션 위에 마우스를 올려 이름 바꾸기 및 제거 작업을 표시합니다: 설명적인 제목을 지정하도록 이름을 바꾸거나 목록에서 삭제하도록 제거합니다. 세션 재개에 대한 자세한 내용은 일반적인 워크플로우를 참조하십시오.

Claude.ai에서 원격 세션 재개

웹에서 Claude Code를 사용하는 경우 VS Code에서 직접 해당 원격 세션을 재개할 수 있습니다. 이를 위해서는 Anthropic Console이 아닌 **Claude.ai Subscription**으로 로그인해야 합니다.

Step 1: 과거 대화 열기

Claude Code 패널 상단의 **과거 대화** 드롭다운을 클릭합니다.

Step 2: 원격 탭 선택

대화 상자에는 로컬 및 원격의 두 탭이 표시됩니다. **원격**을 클릭하여 claude.ai의 세션을 봅니다.

Step 3: 재개할 세션 선택

원격 세션을 찾아보거나 검색합니다. 모든 세션을 클릭하여 다운로드하고 대화를 로컬에서 계속합니다.

> **Note:**

원격 탭에는 GitHub 저장소로 시작된 웹 세션만 나타납니다. 재개하면 대화 기록이 로컬로 로드됩니다. 변경 사항은 claude.ai로 다시 동기화되지 않습니다.

워크플로우 사용자 정의

실행 중이면 Claude 패널을 다시 배치하거나 여러 세션을 실행하거나 터미널 모드로 전환할 수 있습니다.

Claude가 있는 위치 선택

Claude 패널을 드래그하여 VS Code의 어느 곳이든 다시 배치할 수 있습니다. 패널의 탭이나 제목 표시줄을 잡고 다음으로 드래그합니다:

- **보조 사이드바**: 창의 오른쪽. 코딩하는 동안 Claude를 표시 상태로 유지합니다.
- **기본 사이드바**: 탐색기, 검색 등의 아이콘이 있는 왼쪽 사이드바입니다.
- **편집기 영역**: Claude를 파일 옆의 탭으로 엽니다. 부작업에 유용합니다.

Tip:

주 Claude 세션에 사이드바를 사용하고 부작업을 위해 추가 탭을 엽니다. Claude는 선호하는 위치를 기억합니다. 활동 표시줄 세션 목록 아이콘은 Claude 패널과 별개입니다: 세션 목록은 항상 활동 표시줄에 표시되지만 Claude 패널 아이콘은 패널이 왼쪽 사이드바에 도킹될 때만 나타납니다.

여러 대화 실행

명령 팔레트에서 **새 탭에서 열기** 또는 **새 창에서 열기**를 사용하여 추가 대화를 시작합니다. 각 대화는 자체 기록 및 컨텍스트를 유지하므로 다양한 작업을 병렬로 작업할 수 있습니다.

탭을 사용할 때 spark 아이콘의 작은 색상 점은 상태를 나타냅니다: 파란색은 권한 요청이 보류 중임을 의미하고 주황색은 탭이 숨겨진 동안 Claude가 완료되었음을 의미합니다.

터미널 모드로 전환

기본적으로 확장 프로그램은 그래픽 채팅 패널을 엽니다. CLI 스타일 인터페이스를 선호하는 경우 Use Terminal 설정을 열고 상자를 선택합니다.

또한 VS Code 설정(`Cmd+,` Mac 또는 `Ctrl+,` Windows/Linux)을 열고 확장 프로그램 → Claude Code로 이동한 후 **Use Terminal**을 선택합니다.

플러그인 관리

VS Code 확장 프로그램에는 플러그인을 설치하고 관리하기 위한 그래픽 인터페이스가 포함되어 있습니다. 프롬프트 상자에 `/plugins`를 입력하여 **플러그인 관리** 인터페이스를 엽니다.

플러그인 설치

플러그인 대화 상자에는 **플러그인** 및 **마켓플레이스**의 두 탭이 표시됩니다.

플러그인 탭에서:

- **설치된 플러그인**은 토글 스위치와 함께 상단에 나타나 활성화 또는 비활성화합니다.
- **구성된 마켓플레이스의 사용 가능한 플러그인**이 아래에 나타납니다.
- 이름 또는 설명으로 플러그인을 필터링하도록 검색합니다.
- 사용 가능한 플러그인에서 **설치**를 클릭합니다.

플러그인을 설치할 때 설치 범위를 선택합니다:

- **당신을 위해 설치**: 모든 프로젝트에서 사용 가능(사용자 범위)
- **이 프로젝트를 위해 설치**: 프로젝트 협력자와 공유(프로젝트 범위)
- **로컬로 설치**: 당신만, 이 저장소에서만(로컬 범위)

마켓플레이스 관리

마켓플레이스 탭으로 전환하여 플러그인 소스를 추가하거나 제거합니다:

- GitHub 저장소, URL 또는 로컬 경로를 입력하여 새 마켓플레이스를 추가합니다.
- 새로 고침 아이콘을 클릭하여 마켓플레이스의 플러그인 목록을 업데이트합니다.
- 휴지통 아이콘을 클릭하여 마켓플레이스를 제거합니다.

변경 후 배너가 Claude Code를 다시 시작하여 업데이트를 적용하도록 요청합니다.

> **Note:**
>
> VS Code의 플러그인 관리는 내부적으로 동일한 CLI 명령을 사용합니다. 확장 프로그램에서 구성한 플러그인 및 마켓플레이스는 CLI에서도 사용 가능하며 그 반대도 마찬가지입니다.

플러그인 시스템에 대한 자세한 내용은 플러그인 및 플러그인 마켓플레이스를 참조하십시오.

Chrome으로 브라우저 작업 자동화

Claude를 Chrome 브라우저에 연결하여 웹 앱을 테스트하고, 콘솔 로그로 디버깅하고, VS Code를 떠나지 않고 브라우저 워크플로우를 자동화합니다. 이를 위해서는 Chrome의 Claude 확장 프로그램 버전 1.0.36 이상이 필요합니다.

프롬프트 상자에 `@browser`를 입력한 후 Claude가 수행할 작업을 입력합니다:

```
@browser go to localhost:3000 and check the console for errors
```

또한 첨부 메뉴를 열어 새 탭 열기 또는 페이지 콘텐츠 읽기와 같은 특정 브라우저 도구를 선택할 수 있습니다.

Claude는 브라우저 작업을 위해 새 탭을 열고 브라우저의 로그인 상태를 공유하므로 이미 로그인한 모든 사이트에 액세스할 수 있습니다.

설정 지침, 전체 기능 목록 및 문제 해결은 Chrome에서 Claude Code 사용을 참조하십시오.

VS Code 명령 및 단축키

명령 팔레트(`Cmd+Shift+P` Mac 또는 `Ctrl+Shift+P` Windows/Linux)를 열고 “Claude Code” 를 입력하여 Claude Code 확장 프로그램에 사용 가능한 모든 VS Code 명령을 봅니다.

일부 단축키는 어느 패널이 “포커스”(키보드 입력을 받음)되는지에 따라 다릅니다. 커서가 코드 파일에 있으면 편집기가 포커스됩니다. 커서가 Claude의 프롬프트 상자에 있으면 Claude가 포커스됩니다. `Cmd+Esc` / `Ctrl+Esc` 를 사용하여 둘 사이를 전환합니다.

> **Note:**
>
> 이는 확장 프로그램을 제어하기 위한 VS Code 명령입니다. 모든 기본 제공 Claude Code 명령을 확장 프로그램에서 사용할 수 있는 것은 아닙니다. 자세한 내용은 VS Code 확장 프로그램 vs. Claude Code CLI를 참조하십시오.

명령	단축키	설명
Focus Input	`Cmd+Esc` (Mac) / `Ctrl+Esc` (Windows/Linux)	편집기와 Claude 사이의 포커스 전환
Open in Side Bar	-	Claude를 왼쪽 사이드바에서 열기
Open in Terminal	-	Claude를 터미널 모드에서 열기
Open in New Tab	`Cmd+Shift+Esc` (Mac) / `Ctrl+Shift+Esc` (Windows/Linux)	새 대화를 편집기 탭으로 열기
Open in New Window	-	새 대화를 별도 창에서 열기
New Conversation	`Cmd+N` (Mac) / `Ctrl+N` (Windows/Linux)	새 대화 시작(Claude가 포커스되어야 함)

명령	단축키	설명
Insert @-Mention Reference	`Option+K` (Mac) / `Alt+K` (Windows/Linux)	현재 파일 및 선택에 대한 참조 삽입(편집기가 포커스되어야 함)
Show Logs	-	확장 프로그램 디버그 로그 보기
Logout	-	Anthropic 계정에서 로그아웃

설정 구성

확장 프로그램에는 두 가지 유형의 설정이 있습니다:

- **확장 프로그램 설정** VS Code에서: VS Code 내에서 확장 프로그램의 동작을 제어합니다. `Cmd+,` (Mac) 또는 `Ctrl+,` (Windows/Linux)로 열고 확장 프로그램 → Claude Code로 이동합니다. 또한 `/` 를 입력하고 **General Config**를 선택하여 설정을 열 수 있습니다.
- **Claude Code 설정** `~/.claude/settings.json` 에서: 확장 프로그램과 CLI 간에 공유됩니다. 허용된 명령, 환경 변수, hooks 및 MCP 서버에 사용합니다. 자세한 내용은 설정을 참조하십시오.

> **Tip:**
>
> `"$schema": "https://json.schemastore.org/claude-code-settings.json"` 을 `settings.json` 에 추가하여 VS Code에서 직접 사용 가능한 모든 설정에 대한 자동 완성 및 인라인 유효성 검사를 받습니다.

확장 프로그램 설정

설정	기본값	설명
`selectedModel`	`default`	새 대화를 위한 모델. `/model` 로 세션별로 변경합니다.
`useTerminal`	`false`	그래픽 패널 대신 터미널 모드에서 Claude 시작

설정	기본값	설명
`initialPermissionMode`	`default`	승인 프롬프트 제어: `default` (매번 요청), `plan`, `acceptEdits` 또는 `bypassPermissions`
`preferredLocation`	`panel`	Claude가 열리는 위치: `sidebar` (오른쪽) 또는 `panel` (새 탭)
`autosave`	`true`	Claude가 파일을 읽거나 쓰기 전에 자동 저장
`useCtrlEnterToSend`	`false`	Enter 대신 Ctrl/Cmd+Enter를 사용하여 프롬프트 보내기
`enableNewConversationShortcut`	`true`	Cmd/Ctrl+N을 활성화하여 새 대화 시작
`hideOnboarding`	`false`	온보딩 체크리스트 숨기기(졸업 모자 아이콘)
`respectGitIgnore`	`true`	파일 검색에서 .gitignore 패턴 제외
`environmentVariables`	`[]`	Claude 프로세스에 대한 환경 변수 설정. 공유 구성을 위해 Claude Code 설정을 대신 사용합니다.
`disableLoginPrompt`	`false`	인증 프롬프트 건너뛰기(타사 제공자 설정용)
`allowDangerouslySkipPermissions`	`false`	모든 권한 프롬프트 무시. **극도의 주의를 기울여 사용합니다.**
`claudeProcessWrapper`	-	Claude 프로세스를 시작하는 데 사용되는 실행 파일 경로

VS Code 확장 프로그램 vs. Claude Code CLI

Claude Code는 VS Code 확장 프로그램(그래픽 패널)과 CLI(터미널의 명령줄 인터페이스) 모두로 사용 가능합니다. 일부 기능은 CLI에서만 사용 가능합니다. CLI 전용 기능이 필요한 경우 VS Code의 통합 터미널에서 `claude`를 실행합니다.

기능	CLI	VS Code 확장 프로그램
명령 및 skills	모두	부분 집합(`/`를 입력하여 사용 가능한 항목 보기)
MCP 서버 구성	예	부분(CLI를 통해 서버 추가; 채팅 패널에서 `/mcp`로 기존 서버 관리)
Checkpoints	예	예
`!` bash 단축키	예	아니요
Tab 완성	예	아니요

Checkpoints로 되감기

VS Code 확장 프로그램은 Claude의 파일 편집을 추적하고 이전 상태로 되감을 수 있는 checkpoints를 지원합니다. 모든 메시지 위에 마우스를 올려 되감기 버튼을 표시한 후 세 가지 옵션 중에서 선택합니다:

- **여기서 대화 분기**: 모든 코드 변경 사항을 유지하면서 이 메시지에서 새 대화 분기 시작
- **여기로 코드 되감기**: 전체 대화 기록을 유지하면서 파일 변경 사항을 이 지점으로 되돌리기
- **대화 분기 및 코드 되감기**: 새 대화 분기를 시작하고 파일 변경 사항을 이 지점으로 되돌리기

checkpoints 작동 방식 및 제한 사항에 대한 전체 세부 정보는 Checkpointing을 참조하십시오.

VS Code에서 CLI 실행

VS Code에 머물면서 CLI를 사용하려면 통합 터미널(Windows/Linux에서 `Ctrl+`` 또는 Mac에서 `Cmd+``)을 열고 `claude`를 실행합니다. CLI는 diff 보기 및 진단 공유와 같은 기능을 위해 IDE와 자동으로 통합됩니다.

외부 터미널을 사용하는 경우 Claude Code 내에서 `/ide`를 실행하여 VS Code에 연결합니다.

확장 프로그램과 CLI 간 전환

확장 프로그램과 CLI는 동일한 대화 기록을 공유합니다. 확장 프로그램 대화를 CLI에서 계속하려면 터미널에서 `claude --resume`을 실행합니다. 이렇게 하면 대화를 검색하고 선택할 수 있는 대화형 선택기가 열립니다.

프롬프트에 터미널 출력 포함

`@terminal:name`을 사용하여 프롬프트에서 터미널 출력을 참조합니다. 여기서 `name`은 터미널의 제목입니다. 이를 통해 Claude는 복사 붙여넣기 없이 명령 출력, 오류 메시지 또는 로그를 볼 수 있습니다.

백그라운드 프로세스 모니터링

Claude가 장기 실행 명령을 실행할 때 확장 프로그램은 상태 표시줄에 진행 상황을 표시합니다. 그러나 백그라운드 작업의 가시성은 CLI에 비해 제한적입니다. 더 나은 가시성을 위해 Claude가 명령을 출력하도록 하여 VS Code의 통합 터미널에서 실행할 수 있습니다.

MCP를 사용하여 외부 도구에 연결

MCP(Model Context Protocol) 서버는 Claude에게 외부 도구, 데이터베이스 및 API에 대한 액세스를 제공합니다.

MCP 서버를 추가하려면 통합 터미널(`Ctrl+`` 또는 `Cmd+``)을 열고 다음을 실행합니다:

```
claude mcp add --transport http github https://api.githubcopilot.com/mcp/
```

구성되면 Claude에게 도구를 사용하도록 요청합니다(예: “Review PR #456”).

VS Code를 떠나지 않고 MCP 서버를 관리하려면 채팅 패널에 `/mcp`를 입력합니다. MCP 관리 대화 상자를 사용하면 서버를 활성화 또는 비활성화하고, 서버에 다시 연결하고, OAuth 인증을 관리할 수 있습니다. 사용 가능한 서버는 MCP 문서를 참조하십시오.

git으로 작업

Claude Code는 git과 통합되어 VS Code에서 직접 버전 제어 워크플로우를 도와줍니다. Claude에게 변경 사항을 커밋하거나, 풀 요청을 생성하거나, 분기 간에 작업하도록 요청합니다.

커밋 및 풀 요청 생성

Claude는 변경 사항을 스테이징하고, 커밋 메시지를 작성하고, 작업을 기반으로 풀 요청을 생성할 수 있습니다:

```
> commit my changes with a descriptive message
> create a pr for this feature
> summarize the changes I've made to the auth module
```

풀 요청을 생성할 때 Claude는 실제 코드 변경을 기반으로 설명을 생성하고 테스트 또는 구현 결정에 대한 컨텍스트를 추가할 수 있습니다.

병렬 작업을 위해 git worktrees 사용

`--worktree` (`-w`) 플래그를 사용하여 자체 파일 및 분기가 있는 격리된 worktree에서 Claude를 시작합니다:

```
claude --worktree feature-auth
```

각 worktree는 git 기록을 공유하면서 독립적인 파일 상태를 유지합니다. 이는 다양한 작업에서 작업할 때 Claude 인스턴스가 서로 간섭하는 것을 방지합니다. 자세한 내용은 Git worktrees를 사용하여 병렬 세션 실행을 참조하십시오.

타사 제공자 사용

기본적으로 Claude Code는 Anthropic의 API에 직접 연결됩니다. 조직에서 Amazon Bedrock, Google Vertex AI 또는 Microsoft Foundry를 사용하여 Claude에 액세스하는 경우 대신 제공자를 사용하도록 확장 프로그램을 구성합니다:

Step 1: 로그인 프롬프트 비활성화

로그인 프롬프트 비활성화 설정을 열고 상자를 선택합니다.

또한 VS Code 설정(`Cmd+,` Mac 또는 `Ctrl+,` Windows/Linux)을 열고 “Claude Code login” 을 검색한 후 **로그인 프롬프트 비활성화**를 선택합니다.

Step 2: 제공자 구성

제공자에 대한 설정 가이드를 따릅니다:

- Amazon Bedrock의 Claude Code
- Google Vertex AI의 Claude Code
- Microsoft Foundry의 Claude Code

이 가이드는 `~/.claude/settings.json` 에서 제공자를 구성하는 방법을 다루며, 이는 VS Code 확장 프로그램과 CLI 간에 설정이 공유되도록 합니다.

보안 및 개인 정보

코드는 비공개로 유지됩니다. Claude Code는 코드를 처리하여 지원을 제공하지만 모델 학습에 사용하지 않습니다. 데이터 처리 및 로깅을 거부하는 방법에 대한 자세한 내용은 데이터 및 개인 정보를 참조하십시오.

자동 편집 권한이 활성화된 경우 Claude Code는 VS Code가 자동으로 실행할 수 있는 VS Code 구성 파일(예: `settings.json` 또는 `tasks.json`)을 수정할 수 있습니다. 신뢰할 수 없는 코드로 작업할 때 위험을 줄이려면:

- 신뢰할 수 없는 작업 공간에 대해 VS Code 제한 모드를 활성화합니다.
- 편집에 대해 자동 수락 대신 수동 승인 모드를 사용합니다.
- 변경 사항을 수락하기 전에 신중하게 검토합니다.

일반적인 문제 해결

확장 프로그램이 설치되지 않음

- VS Code의 호환 버전(1.98.0 이상)이 있는지 확인합니다.
- VS Code에 확장 프로그램을 설치할 권한이 있는지 확인합니다.
- VS Code Marketplace에서 직접 설치를 시도합니다.

Spark 아이콘이 표시되지 않음

Spark 아이콘은 파일을 열었을 때 **편집기 도구 모음**(편집기의 오른쪽 위)에 나타납니다. 표시되지 않으면:

1. **파일 열기**: 아이콘에는 파일을 열어야 합니다. 폴더만 열어서는 충분하지 않습니다.
2. **VS Code 버전 확인**: 1.98.0 이상 필요(도움말 → 정보)
3. **VS Code 다시 시작**: 명령 팔레트에서 “Developer: Reload Window” 실행
4. **충돌하는 확장 프로그램 비활성화**: 다른 AI 확장 프로그램(Cline, Continue 등)을 일시적으로 비활성화합니다.
5. **작업 공간 신뢰 확인**: 확장 프로그램은 제한 모드에서 작동하지 않습니다.

또는 **상태 표시줄**(오른쪽 아래 모서리)에서 “ Claude Code”를 클릭합니다. 파일을 열지 않았을 때도 작동합니다. 또한 **명령 팔레트**(`Cmd+Shift+P` / `Ctrl+Shift+P`)를 사용하고 “Claude Code” 를 입력할 수 있습니다.

Claude Code가 응답하지 않음

Claude Code가 프롬프트에 응답하지 않으면:

1. **인터넷 연결 확인**: 안정적인 인터넷 연결이 있는지 확인합니다.
2. **새 대화 시작**: 새 대화를 시작하여 문제가 지속되는지 확인합니다.
3. **CLI 시도**: 터미널에서 `claude`를 실행하여 더 자세한 오류 메시지를 받는지 확인합니다.

문제가 지속되면 오류에 대한 세부 정보와 함께 GitHub에서 문제를 제출합니다.

확장 프로그램 제거

Claude Code 확장 프로그램을 제거하려면:

1. 확장 프로그램 보기 열기(`Cmd+Shift+X` Mac 또는 `Ctrl+Shift+X` Windows/Linux)
2. “Claude Code” 검색
3. **제거** 클릭

확장 프로그램 데이터를 제거하고 모든 설정을 재설정하려면:

```
rm -rf ~/.vscode/globalStorage/anthropic.claude-code
```

추가 도움말은 문제 해결 가이드를 참조하십시오.

다음 단계

이제 VS Code에서 Claude Code를 설정했습니다:

- 일반적인 워크플로우 탐색하여 Claude Code를 최대한 활용합니다.
- MCP 서버 설정하여 외부 도구로 Claude의 기능을 확장합니다. CLI를 사용하여 서버를 추가한 후 채팅 패널에서 `/mcp` 로 관리합니다.
- Claude Code 설정 구성하여 허용된 명령, hooks 등을 사용자 정의합니다. 이 설정은 확장 프로그램과 CLI 간에 공유됩니다.

JetBrains IDEs

> Claude Code를 IntelliJ, PyCharm, WebStorm 등 JetBrains IDE와 함께 사용합니다

Claude Code는 전용 플러그인을 통해 JetBrains IDE와 통합되며, 대화형 diff 보기, 선택 영역 컨텍스트 공유 등의 기능을 제공합니다.

지원되는 IDE

Claude Code 플러그인은 다음을 포함한 대부분의 JetBrains IDE와 호환됩니다:

- IntelliJ IDEA
- PyCharm
- Android Studio
- WebStorm
- PhpStorm
- GoLand

기능

- **빠른 실행**: `Cmd+Esc` (Mac) 또는 `Ctrl+Esc` (Windows/Linux)를 사용하여 편집기에서 직접 Claude Code를 열거나, UI의 Claude Code 버튼을 클릭합니다
- **Diff 보기**: 코드 변경 사항을 터미널 대신 IDE diff 뷰어에 직접 표시할 수 있습니다
- **선택 영역 컨텍스트**: IDE의 현재 선택 영역/탭이 Claude Code와 자동으로 공유됩니다
- **파일 참조 바로가기**: `Cmd+Option+K` (Mac) 또는 `Alt+Ctrl+K` (Linux/Windows)를 사용하여 파일 참조를 삽입합니다 (예: @File#L1-99)
- **진단 공유**: IDE의 진단 오류 (lint, 구문 등)가 작업할 때 Claude와 자동으로 공유됩니다

설치

마켓플레이스 설치

JetBrains 마켓플레이스에서 Claude Code 플러그인을 찾아 설치하고 IDE를 다시 시작합니다.

Claude Code를 아직 설치하지 않았다면, 빠른 시작 가이드에서 설치 지침을 참조하세요.

> **Note:**

> 플러그인을 설치한 후 IDE를 완전히 다시 시작해야 적용될 수 있습니다.

사용법

IDE에서

IDE의 통합 터미널에서 `claude` 를 실행하면 모든 통합 기능이 활성화됩니다.

외부 터미널에서

모든 외부 터미널에서 `/ide` 명령을 사용하여 Claude Code를 JetBrains IDE에 연결하고 모든 기능을 활성화합니다:

```
claude
```

```
/ide
```

Claude가 IDE와 동일한 파일에 액세스하도록 하려면, IDE 프로젝트 루트와 동일한 디렉터리에서 Claude Code를 시작합니다.

구성

Claude Code 설정

Claude Code의 설정을 통해 IDE 통합을 구성합니다:

1. `claude` 실행
2. `/config` 명령 입력
3. diff 도구를 `auto` 로 설정하여 자동 IDE 감지

플러그인 설정

설정 → 도구 → Claude Code [Beta]로 이동하여 Claude Code 플러그인을 구성합니다:

일반 설정

- **Claude 명령**: Claude를 실행할 사용자 정의 명령을 지정합니다 (예: `claude` , `/usr/local/bin/claude` , 또는 `npx @anthropic/claude`)
- **Claude 명령을 찾을 수 없음에 대한 알림 표시 안 함**: Claude 명령을 찾을 수 없다는 알림을 건너뜁니다

- **다중 줄 프롬프트에 Option+Enter 사용 활성화** (macOS만 해당): 활성화되면 Option+Enter가 Claude Code 프롬프트에 새 줄을 삽입합니다. Option 키가 예기치 않게 캡처되는 문제가 발생하면 비활성화합니다 (터미널 다시 시작 필요)
- **자동 업데이트 활성화**: 플러그인 업데이트를 자동으로 확인하고 설치합니다 (다시 시작 시 적용)

> **Tip:**
>
> WSL 사용자의 경우: Claude 명령으로 `wsl -d Ubuntu -- bash -lic "claude"` 를 설정합니다 (`Ubuntu` 를 WSL 배포판 이름으로 바꿉니다)

ESC 키 구성

ESC 키가 JetBrains 터미널에서 Claude Code 작업을 중단하지 않는 경우:

1. **설정 → 도구 → 터미널**로 이동합니다
2. 다음 중 하나를 수행합니다:
   - “Escape로 편집기에 포커스 이동” 선택 해제, 또는
   - “터미널 키 바인딩 구성”을 클릭하고 “편집기로 포커스 전환” 바로가기 삭제
3. 변경 사항을 적용합니다

이렇게 하면 ESC 키가 Claude Code 작업을 제대로 중단할 수 있습니다.

특수 구성

원격 개발

> **Warning:**
>
> JetBrains 원격 개발을 사용할 때는 **설정 → 플러그인 (호스트)**를 통해 원격 호스트에 플러그인을 설치해야 합니다.

플러그인은 로컬 클라이언트 머신이 아닌 원격 호스트에 설치해야 합니다.

WSL 구성

> **Warning:**
>
> WSL 사용자는 IDE 감지가 제대로 작동하도록 추가 구성이 필요할 수 있습니다. 자세한 설정 지침은 WSL 문제 해결 가이드를 참조하세요.

WSL 구성에는 다음이 필요할 수 있습니다:

- 적절한 터미널 구성
- 네트워킹 모드 조정
- 방화벽 설정 업데이트

문제 해결

플러그인이 작동하지 않음

- 프로젝트 루트 디렉터리에서 Claude Code를 실행 중인지 확인합니다
- JetBrains 플러그인이 IDE 설정에서 활성화되어 있는지 확인합니다
- IDE를 완전히 다시 시작합니다 (여러 번 수행해야 할 수 있습니다)
- 원격 개발의 경우 플러그인이 원격 호스트에 설치되어 있는지 확인합니다

IDE가 감지되지 않음

- 플러그인이 설치되고 활성화되어 있는지 확인합니다
- IDE를 완전히 다시 시작합니다
- 통합 터미널에서 Claude Code를 실행 중인지 확인합니다
- WSL 사용자의 경우 WSL 문제 해결 가이드를 참조하세요

명령을 찾을 수 없음

Claude 아이콘을 클릭하면 "명령을 찾을 수 없음"이 표시되는 경우:

1. Claude Code가 설치되어 있는지 확인합니다: `npm list -g @anthropic-ai/claude-code`
2. 플러그인 설정에서 Claude 명령 경로를 구성합니다
3. WSL 사용자의 경우 구성 섹션에서 언급한 WSL 명령 형식을 사용합니다

보안 고려 사항

Claude Code가 자동 편집 권한이 활성화된 JetBrains IDE에서 실행될 때, IDE에서 자동으로 실행될 수 있는 IDE 구성 파일을 수정할 수 있습니다. 이는 자동 편집 모드에서 Claude Code를 실행하는 위험을 증가시킬 수 있으며 bash 실행에 대한 Claude Code의 권한 프롬프트를 우회할 수 있습니다.

JetBrains IDE에서 실행할 때 다음을 고려합니다:

- 편집에 대한 수동 승인 모드 사용
- Claude가 신뢰할 수 있는 프롬프트로만 사용되도록 각별히 주의
- Claude Code가 수정할 수 있는 파일이 무엇인지 인식

추가 도움말은 문제 해결 가이드를 참조하세요.

Claude Code Desktop 사용하기

> Claude Code Desktop을 더 활용하세요: Git 격리를 통한 병렬 세션, 시각적 diff 검토, 앱 미리보기, PR 모니터링, 권한 모드, 커넥터, 엔터프라이즈 구성.

Claude Desktop 앱 내의 Code 탭을 사용하면 터미널 대신 그래픽 인터페이스를 통해 Claude Code를 사용할 수 있습니다.

Desktop은 표준 Claude Code 경험 위에 다음과 같은 기능을 추가합니다:

- 시각적 diff 검토 (인라인 댓글 포함)
- 라이브 앱 미리보기 (개발 서버 포함)
- GitHub PR 모니터링 (자동 수정 및 자동 병합)
- 병렬 세션 (자동 Git worktree 격리)
- 예약된 작업 (반복 일정으로 Claude 실행)
- 커넥터 (GitHub, Slack, Linear 등)
- 로컬, SSH, 클라우드 환경

> **Tip:**
>
> Desktop을 처음 사용하시나요? 시작하기에서 앱을 설치하고 첫 번째 편집을 해보세요.

이 페이지는 코드 작업, 세션 관리, Claude Code 확장, 예약된 작업, 구성을 다룹니다. 또한 CLI 비교와 문제 해결도 포함되어 있습니다.

세션 시작하기

첫 번째 메시지를 보내기 전에 프롬프트 영역에서 네 가지를 구성하세요:

- **환경**: Claude가 실행되는 위치를 선택합니다. 자신의 머신의 경우 **Local**, Anthropic 호스팅 클라우드 세션의 경우 **Remote**, 관리하는 원격 머신의 경우 **SSH 연결**을 선택합니다. 환경 구성을 참조하세요.
- **프로젝트 폴더**: Claude가 작업할 폴더 또는 저장소를 선택합니다. 원격 세션의 경우 여러 저장소를 추가할 수 있습니다.
- **모델**: 전송 버튼 옆의 드롭다운에서 모델을 선택합니다. 세션이 시작되면 모델이 잠깁니다.
- **권한 모드**: 모드 선택기에서 Claude가 가질 자율성을 선택합니다. 세션 중에 이를 변경할 수 있습니다.

작업을 입력하고 **Enter**를 눌러 시작합니다. 각 세션은 자신의 컨텍스트와 변경 사항을 독립적으로 추적합니다.

코드 작업하기

Claude에게 올바른 컨텍스트를 제공하고, 자동으로 수행할 작업의 양을 제어하고, 변경 사항을 검토합니다.

프롬프트 상자 사용하기

Claude가 수행할 작업을 입력하고 **Enter**를 눌러 보냅니다. Claude는 프로젝트 파일을 읽고, 변경 사항을 만들고, 권한 모드에 따라 명령을 실행합니다. 언제든지 Claude를 중단할 수 있습니다: 중지 버튼을 클릭하거나 수정 사항을 입력하고 **Enter**를 누릅니다. Claude는 작업을 중지하고 입력에 따라 조정합니다.

프롬프트 상자 옆의 **+** 버튼을 클릭하면 파일 첨부, skills, 커넥터, 플러그인에 액세스할 수 있습니다.

프롬프트에 파일 및 컨텍스트 추가하기

프롬프트 상자는 외부 컨텍스트를 가져오는 두 가지 방법을 지원합니다:

- **@mention 파일**: `@` 다음에 파일 이름을 입력하여 파일을 대화 컨텍스트에 추가합니다. Claude는 그 파일을 읽고 참조할 수 있습니다.
- **파일 첨부**: 첨부 버튼을 사용하거나 파일을 프롬프트에 직접 드래그 앤 드롭하여 이미지, PDF 및 기타 파일을 프롬프트에 첨부합니다. 이는 버그 스크린샷, 디자인 목업 또는 참고 문서를 공유하는 데 유용합니다.

권한 모드 선택하기

권한 모드는 세션 중에 Claude가 가질 자율성을 제어합니다: 파일 편집, 명령 실행 또는 둘 다 전에 묻는지 여부입니다. 전송 버튼 옆의 모드 선택기를 사용하여 언제든지 모드를 전환할 수 있습니다. Claude가 정확히 무엇을 하는지 보려면 권한 요청으로 시작한 다음, 편하면 자동 수락 편집 또는 Plan 모드로 이동합니다.

모드	설정 키	동작
권한 요청	`default`	Claude는 파일을 편집하거나 명령을 실행하기 전에 묻습니다. 각 변경 사항에 대해 diff를 보고 수락하거나 거부할 수 있습니다. 새 사용자에게 권장됩니다.

모드	설정 키	동작
자동 수락 편집	`acceptEdits`	Claude는 파일 편집을 자동으로 수락하지만 터미널 명령 실행 전에는 여전히 묻습니다. 파일 변경을 신뢰하고 더 빠른 반복을 원할 때 사용합니다.
Plan 모드	`plan`	Claude는 코드를 분석하고 파일을 수정하거나 명령을 실행하지 않고 계획을 만듭니다. 먼저 접근 방식을 검토하고 싶은 복잡한 작업에 좋습니다.
권한 무시	`bypassPermissions`	Claude는 권한 프롬프트 없이 실행되며, CLI의 `--dangerously-skip-permissions`과 동일합니다. Settings → Claude Code에서 "권한 무시 모드 허용"에서 활성화합니다. 샌드박스 컨테이너 또는 VM에서만 사용합니다. 엔터프라이즈 관리자는 이 옵션을 비활성화할 수 있습니다.

`dontAsk` 권한 모드는 CLI에서만 사용 가능합니다.

> **Tip:**
>
> 복잡한 작업을 Plan 모드에서 시작하여 Claude가 변경하기 전에 접근 방식을 매핑하도록 합니다. 계획을 승인한 후 자동 수락 편집 또는 권한 요청으로 전환하여 실행합니다. 이 워크플로우에 대한 자세한 내용은 먼저 탐색, 그 다음 계획, 그 다음 코드를 참조하세요.

원격 세션은 자동 수락 편집 및 Plan 모드를 지원합니다. 원격 세션이 기본적으로 파일 편집을 자동으로 수락하므로 권한 요청은 사용할 수 없으며, 원격 환경이 이미 샌드박스되어 있으므로 권한 무시는 사용할 수 없습니다.

엔터프라이즈 관리자는 사용 가능한 권한 모드를 제한할 수 있습니다. 자세한 내용은 엔터프라이즈 구성을 참조하세요.

앱 미리보기

Claude는 개발 서버를 시작하고 임베드된 브라우저를 열어 변경 사항을 확인할 수 있습니다. 이는 프론트엔드 웹 앱뿐만 아니라 백엔드 서버에도 작동합니다: Claude는 API 엔드포인트를 테스트하고, 서버 로그를 보고, 발견한 문제를 반복할 수 있습니다. 대부분의 경우 Claude는 프로젝트 파일을 편집한 후 자동으로 서버를 시작합니다. 언제든지 Claude에게 미리보기를 요청할 수도 있습니다. 기본적으로 Claude는 모든 편집 후 자동으로 변경 사항을 확인합니다.

미리보기 패널에서 다음을 수행할 수 있습니다:

- 임베드된 브라우저에서 실행 중인 앱과 직접 상호작용
- Claude가 자동으로 자신의 변경 사항을 확인하는 것을 봅니다: 스크린샷을 찍고, DOM을 검사하고, 요소를 클릭하고, 양식을 채우고, 발견한 문제를 수정합니다
- 세션 도구 모음의 **Preview** 드롭다운에서 서버 시작 또는 중지
- **Persist sessions**을 드롭다운에서 선택하여 서버 재시작 시 쿠키 및 로컬 스토리지를 유지하므로 개발 중에 다시 로그인할 필요가 없습니다
- 서버 구성을 편집하거나 모든 서버를 한 번에 중지

Claude는 프로젝트를 기반으로 초기 서버 구성을 만듭니다. 앱이 사용자 정의 개발 명령을 사용하는 경우 `.claude/launch.json` 을 편집하여 설정과 일치시킵니다. 전체 참조는 미리보기 서버 구성을 참조하세요.

저장된 세션 데이터를 지우려면 Settings → Claude Code에서 **Persist preview sessions**을 토글 해제합니다. 미리보기를 완전히 비활성화하려면 Settings → Claude Code에서 **Preview**를 토글 해제합니다.

diff 보기로 변경 사항 검토하기

Claude가 코드를 변경한 후 diff 보기를 사용하면 pull request를 만들기 전에 파일별로 수정 사항을 검토할 수 있습니다.

Claude가 파일을 변경하면 `+12 -1` 과 같이 추가 및 제거된 줄 수를 표시하는 diff 통계 표시기가 나타납니다. 이 표시기를 클릭하여 diff 뷰어를 열면 왼쪽에 파일 목록이 표시되고 오른쪽에 각 파일의 변경 사항이 표시됩니다.

특정 줄에 댓글을 달려면 diff의 모든 줄을 클릭하여 댓글 상자를 엽니다. 피드백을 입력하고 **Enter**를 눌러 댓글을 추가합니다. 여러 줄에 댓글을 추가한 후 모든 댓글을 한 번에 제출합니다:

- **macOS**: **Cmd+Enter** 누르기
- **Windows**: **Ctrl+Enter** 누르기

Claude는 댓글을 읽고 요청된 변경 사항을 만들며, 이는 검토할 수 있는 새로운 diff로 나타납니다.

코드 검토하기

diff 보기에서 오른쪽 상단 도구 모음의 **Review code**를 클릭하여 Claude에게 커밋하기 전에 변경 사항을 평가하도록 요청합니다. Claude는 현재 diff를 검토하고 diff 보기에 직접 댓글을 남깁니다. 모든 댓글에 응답하거나 Claude에게 수정을 요청할 수 있습니다.

검토는 높은 신호 문제에 중점을 둡니다: 컴파일 오류, 명확한 논리 오류, 보안 취약점, 명백한 버그입니다. 스타일, 형식, 기존 문제 또는 린터가 포착할 수 있는 것은 플래그하지 않습니다.

pull request 상태 모니터링하기

pull request를 연 후 CI 상태 표시줄이 세션에 나타납니다. Claude Code는 GitHub CLI를 사용하여 확인 결과를 폴링하고 실패를 표시합니다.

- **자동 수정**: 활성화되면 Claude는 실패 출력을 읽고 반복하여 실패한 CI 확인을 자동으로 수정하려고 시도합니다.
- **자동 병합**: 활성화되면 모든 확인이 통과하면 Claude가 PR을 병합합니다. 병합 방법은 squash입니다. 자동 병합이 이 작업을 수행하려면 GitHub 저장소 설정에서 활성화되어야 합니다.

CI 상태 표시줄의 **Auto-fix** 및 **Auto-merge** 토글을 사용하여 옵션을 활성화합니다. Claude Code는 CI가 완료되면 데스크톱 알림도 보냅니다.

> **Note:**
>
> PR 모니터링에는 GitHub CLI (`gh`)가 머신에 설치되고 인증되어야 합니다. `gh` 가 설치되지 않은 경우 Desktop은 처음으로 PR을 만들려고 할 때 설치하라는 메시지를 표시합니다.

세션 관리하기

각 세션은 자신의 컨텍스트와 변경 사항을 가진 독립적인 대화입니다. 여러 세션을 병렬로 실행하거나 작업을 클라우드로 보낼 수 있습니다.

세션으로 병렬 작업하기

사이드바에서 **+ New session**을 클릭하여 여러 작업을 병렬로 작업합니다. Git 저장소의 경우 각 세션은 Git worktrees를 사용하여 프로젝트의 자신의 격리된 복사본을 가져오므로 한 세션의 변경 사항이 커밋할 때까지 다른 세션에 영향을 주지 않습니다.

Worktrees는 기본적으로 `<project-root>/.claude/worktrees/` 에 저장됩니다. Settings → Claude Code의 “Worktree location”에서 사용자 정의 디렉토리로 변경할 수 있습니다. 또한 모든 worktree 분기 이름 앞에 추가되는 분기 접두사를 설정할 수 있으며, 이는 Claude가 만든 분기를 정리하는 데 유용합니다. 완료되면 사이드바의 세션 위에 마우스를 올리고 아카이브 아이콘을 클릭하여 worktree를 제거합니다.

> **Note:**
>
> 세션 격리에는 Git이 필요합니다. 대부분의 Mac에는 기본적으로 Git이 포함되어 있습니다. Terminal에서 `git --version` 을 실행하여 확인합니다. Windows에서는 Code 탭이 작동하려면 Git이 필요합니다: Windows용 Git 다운로드, 설치 및 앱 재시작. Git 오류가 발생하면 Cowork 세션을 시도하여 설정을 문제 해결하세요.

사이드바 상단의 필터 아이콘을 사용하여 상태(Active, Archived) 및 환경(Local, Cloud)별로 세션을 필터링합니다. 세션 이름을 바꾸거나 컨텍스트 사용량을 확인하려면 활성 세션 상단의 도구 모음에서 세션 제목을 클릭합니다. 컨텍스트가 가득 차면 Claude는 자동으로 대화를 요약하고 계속 작업합니다. `/compact` 를 입력하여 요약을 더 일찍 트리거하고 컨텍스트 공간을 확보할 수도 있습니다. 압축이 작동하는 방식에 대한 자세한 내용은 컨텍스트 윈도우를 참조하세요.

원격으로 장기 실행 작업 실행하기

대규모 리팩토링, 테스트 스위트, 마이그레이션 또는 기타 장기 실행 작업의 경우 세션을 시작할 때 **Local** 대신 **Remote**를 선택합니다. 원격 세션은 Anthropic의 클라우드 인프라에서 실행되며 앱을 닫거나 컴퓨터를 종료해도 계속됩니다. 언제든지 다시 확인하여 진행 상황을 보거나 Claude를 다른 방향으로 조종할 수 있습니다. claude.ai/code에서 또는 Claude iOS 앱에서 원격 세션을 모니터링할 수도 있습니다.

원격 세션은 또한 여러 저장소를 지원합니다. 클라우드 환경을 선택한 후 저장소 pill 옆의 **+** 버튼을 클릭하여 세션에 추가 저장소를 추가합니다. 각 저장소는 자신의 분기 선택기를 가집니다. 이는 공유 라이브러리와 그 소비자를 업데이트하는 것과 같이 여러 코드베이스에 걸친 작업에 유용합니다.

원격 세션이 작동하는 방식에 대한 자세한 내용은 웹의 Claude Code를 참조하세요.

다른 표면에서 계속하기

세션 도구 모음의 오른쪽 아래에 있는 VS Code 아이콘에서 액세스할 수 있는 **Continue in** 메뉴를 사용하면 세션을 다른 표면으로 이동할 수 있습니다:

- **Claude Code on the Web**: 로컬 세션을 원격으로 계속 실행하도록 보냅니다. Desktop은 분기를 푸시하고, 대화 요약을 생성하고, 전체 컨텍스트를 사용하여 새로운 원격 세션을 만듭니다. 그 다음 로컬 세션을 아카이브하거나 유지하도록 선택할 수 있습니다. 이는 깨끗한 작업 트리가 필요하며 SSH 세션에는 사용할 수 없습니다.
- **Your IDE**: 현재 작업 디렉토리에서 지원되는 IDE에서 프로젝트를 엽니다.

Claude Code 확장하기

외부 서비스를 연결하고, 재사용 가능한 워크플로우를 추가하고, Claude의 동작을 사용자 정의하고, 미리보기 서버를 구성합니다.

외부 도구 연결하기

로컬 및 SSH 세션의 경우 프롬프트 상자 옆의 **+** 버튼을 클릭하고 **Connectors**를 선택하여 Google Calendar, Slack, GitHub, Linear, Notion 등과 같은 통합을 추가합니다. 세션 전이나 중에 커넥터를 추가할 수 있습니다. 커넥터는 원격 세션에는 사용할 수 없습니다.

커넥터를 관리하거나 연결을 해제하려면 데스크톱 앱의 Settings → Connectors로 이동하거나 프롬프트 상자의 Connectors 메뉴에서 **Manage connectors**를 선택합니다.

연결되면 Claude는 캘린더를 읽고, 메시지를 보내고, 문제를 만들고, 도구와 직접 상호작용할 수 있습니다. Claude에게 세션에 구성된 커넥터가 무엇인지 물어볼 수 있습니다.

커넥터는 그래픽 설정 흐름이 있는 MCP servers입니다. 지원되는 서비스와의 빠른 통합을 위해 사용합니다. Connectors에 나열되지 않은 통합의 경우 설정 파일을 통해 MCP 서버를 수동으로 추가합니다. 사용자 정의 커넥터를 만들 수도 있습니다.

skills 사용하기

Skills는 Claude가 할 수 있는 것을 확장합니다. Claude는 관련이 있을 때 자동으로 로드하거나 직접 호출할 수 있습니다: 프롬프트 상자에서 `/` 를 입력하거나 **+** 버튼을 클릭하고 **Slash commands**를 선택하여 사용 가능한 것을 찾아봅니다. 여기에는 내장 명령, 사용자 정의 skills, 코드베이스의 프로젝트 skills, 설치된 플러그인의 skills가 포함됩니다. 하나를 선택하면 입력 필드에 강조 표시됩니다. 그 다음에 작업을 입력하고 평소대로 보냅니다.

플러그인 설치하기

Plugins는 Claude Code에 skills, agents, hooks, MCP servers, LSP 구성을 추가하는 재사용 가능한 패키지입니다. 터미널을 사용하지 않고 데스크톱 앱에서 플러그인을 설치할 수 있습니다.

로컬 및 SSH 세션의 경우 프롬프트 상자 옆의 **+** 버튼을 클릭하고 **Plugins**를 선택하여 설치된 플러그인과 해당 명령을 봅니다. 플러그인을 추가하려면 하위 메뉴에서 **Add plugin**을 선택하여 플러그인 브라우저를 열면 공식 Anthropic 마켓플레이스를 포함한 구성된 마켓플레이스의 사용 가능한 플러그인을 표시합니다. **Manage plugins**를 선택하여 플러그인을 활성화, 비활성화 또는 제거합니다.

플러그인은 사용자 계정, 특정 프로젝트 또는 로컬 전용으로 범위를 지정할 수 있습니다. 플러그인은 원격 세션에는 사용할 수 없습니다. 자신의 플러그인을 만드는 것을 포함한 전체 플러그인 참조는 플러그인을 참조하세요.

미리보기 서버 구성하기

Claude는 개발 서버 설정을 자동으로 감지하고 세션을 시작할 때 선택한 폴더의 루트에 있는 `.claude/launch.json`에 구성을 저장합니다. Preview는 이 폴더를 작업 디렉토리로 사용하므로 부모 폴더를 선택한 경우 자신의 개발 서버가 있는 하위 폴더는 자동으로 감지되지 않습니다. 하위 폴더의 서버로 작업하려면 해당 폴더에서 직접 세션을 시작하거나 구성을 수동으로 추가합니다.

예를 들어 `npm run dev` 대신 `yarn dev`를 사용하거나 포트를 변경하도록 서버가 시작되는 방식을 사용자 정의하려면 파일을 수동으로 편집하거나 Preview 드롭다운에서 **Edit configuration**을 클릭하여 코드 편집기에서 엽니다. 파일은 주석이 있는 JSON을 지원합니다.

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "my-app",
      "runtimeExecutable": "npm",
      "runtimeArgs": ["run", "dev"],
      "port": 3000
    }
  ]
}
```

동일한 프로젝트에서 다양한 서버를 실행하기 위해 여러 구성을 정의할 수 있습니다(예: 프론트엔드 및 API). 아래의 예제를 참조하세요.

자동 변경 사항 확인

`autoVerify`가 활성화되면 Claude는 파일을 편집한 후 자동으로 코드 변경 사항을 확인합니다. 스크린샷을 찍고, 오류를 확인하고, 응답을 완료하기 전에 변경 사항이 작동하는지 확인합니다.

자동 확인은 기본적으로 켜져 있습니다. `.claude/launch.json`에 `"autoVerify": false`를 추가하여 프로젝트별로 비활성화하거나 **Preview** 드롭다운 메뉴에서 토글합니다.

```
{
  "version": "0.0.1",
  "autoVerify": false,
  "configurations": [...]
}
```

비활성화되면 미리보기 도구는 여전히 사용 가능하며 언제든지 Claude에게 확인을 요청할 수 있습니다. 자동 확인은 모든 편집 후 자동으로 만듭니다.

구성 필드

`configurations` 배열의 각 항목은 다음 필드를 허용합니다:

필드	유형	설명
`name`	string	이 서버의 고유 식별자
`runtimeExecutable`	string	실행할 명령(예: `npm`, `yarn`, `node`)
`runtimeArgs`	string[]	`runtimeExecutable`에 전달되는 인수(예: `["run", "dev"]`)
`port`	number	서버가 수신하는 포트. 기본값은 3000
`cwd`	string	프로젝트 루트에 상대적인 작업 디렉토리. 기본값은 프로젝트 루트입니다. 프로젝트 루트를 명시적으로 참조하려면 `${workspaceFolder}`를 사용합니다
`env`	object	`{ "NODE_ENV": "development" }`와 같은 키-값 쌍으로 추가 환경 변수. 이 파일이 저장소에 커밋되므로 여기에 비밀을 넣지 마세요. 셸 프로필에 설정된 비밀은 자동으로 상속됩니다.
`autoPort`	boolean	포트 충돌을 처리하는 방법. 아래를 참조하세요
`program`	string	`node`로 실행할 스크립트. 언제 `program` vs `runtimeExecutable`을 사용할지 참조

필드	유형	설명
`args`	string[]	`program` 에 전달되는 인수. `program` 이 설정된 경우에만 사용됨

`program` vs `runtimeExecutable` 사용 시기

패키지 관리자를 통해 개발 서버를 시작하려면 `runtimeExecutable` 을 `runtimeArgs` 와 함께 사용합니다. 예를 들어 `"runtimeExecutable": "npm"` 과 `"runtimeArgs": ["run", "dev"]` 는 `npm run dev` 를 실행합니다.

`node` 로 직접 실행하려는 독립 실행형 스크립트가 있을 때 `program` 을 사용합니다. 예를 들어 `"program": "server.js"` 는 `node server.js` 를 실행합니다. `args` 로 추가 플래그를 전달합니다.

포트 충돌

`autoPort` 필드는 선호하는 포트가 이미 사용 중일 때 발생하는 상황을 제어합니다:

- `true` : Claude는 자동으로 사용 가능한 포트를 찾아 사용합니다. 대부분의 개발 서버에 적합합니다.
- `false` : Claude는 오류로 실패합니다. OAuth 콜백 또는 CORS 허용 목록과 같이 서버가 특정 포트를 사용해야 할 때 사용합니다.
- **설정되지 않음(기본값)**: Claude는 서버가 정확한 포트가 필요한지 묻고 답변을 저장합니다.

Claude가 다른 포트를 선택하면 할당된 포트를 `PORT` 환경 변수를 통해 서버에 전달합니다.

예제

이러한 구성은 다양한 프로젝트 유형에 대한 일반적인 설정을 보여줍니다:

Next.js

이 구성은 Yarn을 사용하여 포트 3000에서 Next.js 앱을 실행합니다:

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "web",
      "runtimeExecutable": "yarn",
      "runtimeArgs": ["dev"],
      "port": 3000
    }
  ]
}
```

Multiple servers

프론트엔드 및 API 서버가 있는 모노레포의 경우 여러 구성을 정의합니다. 프론트엔드는 `autoPort: true`를 사용하므로 3000이 사용 중이면 사용 가능한 포트를 선택하고, API 서버는 포트 8080을 정확히 요구합니다:

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "frontend",
      "runtimeExecutable": "npm",
      "runtimeArgs": ["run", "dev"],
      "cwd": "apps/web",
      "port": 3000,
      "autoPort": true
    },
    {
      "name": "api",
      "runtimeExecutable": "npm",
      "runtimeArgs": ["run", "start"],
      "cwd": "server",
      "port": 8080,
      "env": { "NODE_ENV": "development" },
      "autoPort": false
    }
  ]
}
```

Node.js script

패키지 관리자 명령 대신 Node.js 스크립트를 직접 실행하려면 `program` 필드를 사용합니다:

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "server",
      "program": "server.js",
      "args": ["--verbose"],
      "port": 4000
    }
  ]
}
```

반복 작업 예약하기

예약된 작업은 선택한 시간과 빈도에 자동으로 새로운 로컬 세션을 시작합니다. 일일 코드 검토, 종속성 업데이트 확인 또는 캘린더 및 받은 편지함에서 가져오는 아침 브리핑과 같은 반복 작업에 사용합니다.

작업은 머신에서 실행되므로 데스크톱 앱이 열려 있고 컴퓨터가 깨어 있어야 실행됩니다. 놓친 실행 및 따라잡기 동작에 대한 자세한 내용은 예약된 작업이 실행되는 방식을 참조하세요.

> **Note:**
>
> 기본적으로 예약된 작업은 커밋되지 않은 변경 사항을 포함한 작업 디렉토리의 현재 상태에 대해 실행됩니다. 프롬프트 입력에서 worktree 토글을 활성화하여 각 실행에 자신의 격리된 Git worktree를 제공합니다(병렬 세션과 동일한 방식).

예약된 작업을 만들려면 사이드바에서 **Schedule**을 클릭한 다음 **+ New task**를 클릭합니다. 다음 필드를 구성합니다:

필드	설명
Name	작업의 식별자. 소문자 kebab-case로 변환되고 디스크의 폴더 이름으로 사용됩니다. 작업 전체에서 고유해야 합니다.
Description	작업 목록에 표시되는 짧은 요약.
Prompt	작업이 실행될 때 Claude에게 전송되는 지침. 프롬프트 상자에서 메시지를 작성하는 것과 동일한 방식으로 작성합니다. 프롬프트 입력에는 모델, 권한 모드, 작업 폴더, worktree에 대한 컨트롤도 포함됩니다.

필드	설명
Frequency	작업이 실행되는 빈도. 아래의 빈도 옵션을 참조하세요.

또한 모든 세션에서 원하는 것을 설명하여 작업을 만들 수 있습니다. 예를 들어 “매일 아침 9시에 실행되는 일일 코드 검토를 설정합니다.”

빈도 옵션

- **Manual**: 일정 없음, **Run now**를 클릭할 때만 실행됩니다. 온디맨드로 트리거하는 프롬프트를 저장하는 데 유용합니다
- **Hourly**: 매시간 실행됩니다. 각 작업은 API 트래픽을 분산하기 위해 시간 상단에서 최대 10분의 고정 오프셋을 가집니다
- **Daily**: 시간 선택기를 표시하고 기본값은 오전 9:00 현지 시간입니다
- **Weekdays**: Daily와 동일하지만 토요일과 일요일을 건너뜁니다
- **Weekly**: 시간 선택기와 요일 선택기를 표시합니다

선택기가 제공하지 않는 간격(15분마다, 매월 첫 번째 등)의 경우 모든 Desktop 세션에서 Claude에게 일정을 설정하도록 요청합니다. 일반 언어를 사용합니다. 예를 들어 “6시간마다 모든 테스트를 실행하는 작업을 예약합니다.”

예약된 작업이 실행되는 방식

예약된 작업은 머신에서 로컬로 실행됩니다. Desktop은 앱이 열려 있는 동안 매분 일정을 확인하고 열려 있는 수동 세션과 독립적으로 작업이 만료되면 새로운 세션을 시작합니다. 각 작업은 API 트래픽을 분산하기 위해 예약된 시간 후 최대 10분의 고정 지연을 가집니다. 지연은 결정론적입니다: 동일한 작업은 항상 동일한 오프셋에서 시작됩니다.

작업이 실행되면 데스크톱 알림을 받고 새로운 세션이 사이드바의 **Scheduled** 섹션 아래에 나타납니다. 이를 열어 Claude가 수행한 작업을 보고, 변경 사항을 검토하거나, 권한 프롬프트에 응답합니다. 세션은 다른 것처럼 작동합니다: Claude는 파일을 편집하고, 명령을 실행하고, 커밋을 만들고, pull request를 열 수 있습니다.

작업은 데스크톱 앱이 실행 중이고 컴퓨터가 깨어 있을 때만 실행됩니다. 컴퓨터가 예약된 시간을 통해 절전 모드로 전환되면 실행이 건너뜁니다. 유휴 절전을 방지하려면 Settings의 **Desktop app → General** 아래에서 **Keep computer awake**를 활성화합니다. 노트북 뚜껑을 닫으면 여전히 절전 모드로 전환됩니다.

놓친 실행

앱이 시작되거나 컴퓨터가 깨어나면 Desktop은 지난 7일 동안 각 작업이 놓친 실행이 있는지 확인합니다. 그렇다면 Desktop은 가장 최근에 놓친 시간에 대해 정확히 하나의 따라잡기 실행을 시작하고 더 오래된 것은 버립니다. 6일을 놓친 일일 작업은 한 번 깨어날 때 실행됩니다. Desktop은 따라잡기 실행이 시작될 때 알림을 표시합니다.

프롬프트를 작성할 때 이를 염두에 두세요. 오전 9시에 예약된 작업은 컴퓨터가 하루 종일 절전 모드였다면 오후 11시에 실행될 수 있습니다. 타이밍이 중요하면 프롬프트 자체에 가드레일을 추가합니다. 예를 들어 "오늘의 커밋만 검토합니다. 오후 5시 이후라면 검토를 건너뛰고 놓친 것의 요약만 게시합니다."

예약된 작업에 대한 권한

각 작업은 자신의 권한 모드를 가지며, 작업을 만들거나 편집할 때 설정합니다. `~/.claude/settings.json`의 Allow 규칙도 예약된 작업 세션에 적용됩니다. 작업이 Ask 모드에서 실행되고 권한이 없는 도구를 실행해야 하면 실행이 승인할 때까지 정지됩니다. 세션은 사이드바에서 열려 있으므로 나중에 답변할 수 있습니다.

정지를 피하려면 작업을 만든 후 **Run now**를 클릭하고 권한 프롬프트를 확인하고 각각에 대해 "항상 허용"을 선택합니다. 해당 작업의 향후 실행은 프롬프트 없이 동일한 도구를 자동으로 승인합니다. 작업의 세부 정보 페이지에서 이러한 승인을 검토하고 취소할 수 있습니다.

예약된 작업 관리하기

Schedule 목록의 작업을 클릭하여 세부 정보 페이지를 엽니다. 여기에서 다음을 수행할 수 있습니다:

- **Run now**: 다음 예약된 시간을 기다리지 않고 즉시 작업을 시작합니다
- **Toggle repeats**: 작업을 삭제하지 않고 예약된 실행을 일시 중지하거나 재개합니다
- **Edit**: 프롬프트, 빈도, 폴더 또는 기타 설정을 변경합니다
- **Review history**: 컴퓨터가 절전 모드였기 때문에 건너뛴 것을 포함한 모든 과거 실행을 봅니다
- **Review allowed permissions**: **Always allowed** 패널에서 이 작업에 대해 저장된 도구 승인을 보고 취소합니다
- **Delete**: 작업을 제거하고 만든 모든 세션을 아카이브합니다

또한 모든 Desktop 세션에서 Claude에게 요청하여 작업을 관리할 수 있습니다. 예를 들어 "내 dependency-audit 작업을 일시 중지합니다", "standup-prep 작업을 삭제합니다" 또는 "예약된 작업을 보여줍니다."

디스크에서 작업의 프롬프트를 편집하려면 `~/.claude/scheduled-tasks/<task-name>/SKILL.md`를 엽니다(`CLAUDE_CONFIG_DIR`이 설정된 경우 그 아래). 파일은 `name` 및 `description`에 대한 YAML frontmatter를 사용하고 프롬프트를 본문으로 사용합니다. 변경 사항은 다음 실행에 적용됩니다. 일정, 폴더, 모델, 활성화 상태는 이 파일에 없습니다: Edit 양식을 통해 또는 Claude에게 요청하여 변경합니다.

환경 구성

세션을 시작할 때 선택한 환경은 Claude가 실행되는 위치와 연결 방식을 결정합니다:

- **Local**: 머신에서 실행되며 파일에 직접 액세스합니다
- **Remote**: Anthropic의 클라우드 인프라에서 실행됩니다. 앱을 닫아도 세션이 계속됩니다.
- **SSH**: SSH를 통해 연결하는 원격 머신(예: 자신의 서버, 클라우드 VM 또는 개발 컨테이너)에서 실행됩니다

로컬 세션

로컬 세션은 셸에서 환경 변수를 상속합니다. 추가 변수가 필요하면 `~/.zshrc` 또는 `~/.bashrc`와 같은 셸 프로필에 설정하고 데스크톱 앱을 재시작합니다. 지원되는 변수의 전체 목록은 환경 변수를 참조하세요.

Extended thinking은 기본적으로 활성화되어 있으며, 복잡한 추론 작업의 성능을 향상시키지만 추가 토큰을 사용합니다. 사고를 완전히 비활성화하려면 셸 프로필에서 `MAX_THINKING_TOKENS=0`을 설정합니다. Opus에서는 적응형 추론이 사고 깊이를 제어하므로 `MAX_THINKING_TOKENS`은 `0`을 제외하고는 무시됩니다.

원격 세션

원격 세션은 앱을 닫아도 백그라운드에서 계속됩니다. 사용량은 별도의 컴퓨팅 요금 없이 구독 계획 한도에 포함됩니다.

다양한 네트워크 액세스 수준 및 환경 변수를 가진 사용자 정의 클라우드 환경을 만들 수 있습니다. 원격 세션을 시작할 때 환경 드롭다운을 선택하고 **Add environment**를 선택합니다. 네트워크 액세스 및 환경 변수 구성에 대한 자세한 내용은 클라우드 환경을 참조하세요.

SSH 세션

SSH 세션을 사용하면 데스크톱 앱을 인터페이스로 사용하면서 원격 머신에서 Claude Code를 실행할 수 있습니다. 이는 클라우드 VM, 개발 컨테이너 또는 특정 하드웨어 또는 종속성이 있는 서버에 있는 코드베이스로 작업할 때 유용합니다.

SSH 연결을 추가하려면 세션을 시작하기 전에 환경 드롭다운을 클릭하고 **+ Add SSH connection**을 선택합니다. 대화 상자는 다음을 요청합니다:

- **Name**: 이 연결의 친화적인 레이블

- **SSH Host**: `user@hostname` 또는 `~/.ssh/config`에 정의된 호스트
- **SSH Port**: 비워두면 기본값은 22이거나 SSH 구성의 포트를 사용합니다
- **Identity File**: `~/.ssh/id_rsa`와 같은 개인 키의 경로. 기본 키 또는 SSH 구성을 사용하려면 비워둡니다.

추가되면 연결이 환경 드롭다운에 나타납니다. 이를 선택하여 해당 머신에서 세션을 시작합니다. Claude는 원격 머신에서 실행되며 파일 및 도구에 액세스합니다.

Claude Code는 원격 머신에 설치되어야 합니다. 연결되면 SSH 세션은 권한 모드, 커넥터, 플러그인, MCP 서버를 지원합니다.

엔터프라이즈 구성

Teams 또는 Enterprise 계획의 조직은 관리 콘솔 컨트롤, 관리 설정 파일, 장치 관리 정책을 통해 데스크톱 앱 동작을 관리할 수 있습니다.

관리 콘솔 컨트롤

이러한 설정은 관리 설정 콘솔을 통해 구성됩니다:

- **Code 탭 활성화 또는 비활성화**: 조직의 사용자가 데스크톱 앱에서 Claude Code에 액세스할 수 있는지 제어합니다
- **권한 무시 모드 비활성화**: 조직의 사용자가 권한 무시 모드를 활성화하지 못하도록 방지합니다
- **웹의 Claude Code 비활성화**: 조직의 원격 세션을 활성화하거나 비활성화합니다

관리 설정

관리 설정은 프로젝트 및 사용자 설정을 재정의하고 Desktop이 CLI 세션을 생성할 때 적용됩니다. 조직의 관리 설정 파일에서 이러한 키를 설정하거나 관리 콘솔을 통해 원격으로 푸시할 수 있습니다.

키	설명
`disableBypassPermissionsMode`	사용자가 권한 무시 모드를 활성화하지 못하도록 하려면 `"disable"`로 설정합니다. 관리 설정을 참조하세요.

`allowManagedPermissionRulesOnly` 및 `allowManagedHooksOnly`를 포함한 관리 전용 설정의 전체 목록은 관리 전용 설정을 참조하세요.

관리 콘솔을 통해 업로드된 원격 관리 설정은 현재 CLI 및 IDE 세션에만 적용됩니다. Desktop 특정 제한의 경우 위의 관리 콘솔 컨트롤을 사용합니다.

장치 관리 정책

IT 팀은 macOS의 MDM 또는 Windows의 그룹 정책을 통해 데스크톱 앱을 관리할 수 있습니다. 사용 가능한 정책에는 Claude Code 기능 활성화 또는 비활성화, 자동 업데이트 제어, 사용자 정의 배포 URL 설정이 포함됩니다.

- **macOS**: Jamf 또는 Kandji와 같은 도구를 사용하여 `com.anthropic.Claude` 기본 설정 도메인을 통해 구성합니다
- **Windows**: `SOFTWARE\Policies\Claude`의 레지스트리를 통해 구성합니다

인증 및 SSO

엔터프라이즈 조직은 모든 사용자에게 SSO를 요구할 수 있습니다. 계획 수준 세부 정보는 인증을 참조하고 SAML 및 OIDC 구성은 SSO 설정을 참조하세요.

데이터 처리

Claude Code는 로컬 세션에서 코드를 로컬로 처리하거나 원격 세션에서 Anthropic의 클라우드 인프라에서 처리합니다. 대화 및 코드 컨텍스트는 처리를 위해 Anthropic의 API로 전송됩니다. 데이터 보존, 개인 정보 보호, 규정 준수에 대한 자세한 내용은 데이터 처리를 참조하세요.

배포

Desktop은 엔터프라이즈 배포 도구를 통해 배포할 수 있습니다:

- **macOS**: Jamf 또는 Kandji와 같은 MDM을 통해 `.dmg` 설치 프로그램을 사용하여 배포합니다
- **Windows**: MSIX 패키지 또는 `.exe` 설치 프로그램을 통해 배포합니다. 자동 설치를 포함한 엔터프라이즈 배포 옵션은 Windows용 Claude Desktop 배포를 참조하세요

프록시 설정, 방화벽 허용 목록, LLM 게이트웨이와 같은 네트워크 구성은 네트워크 구성을 참조하세요.

전체 엔터프라이즈 구성 참조는 엔터프라이즈 구성 가이드를 참조하세요.

CLI에서 오셨나요?

이미 Claude Code CLI를 사용하고 있다면 Desktop은 그래픽 인터페이스를 사용하여 동일한 기본 엔진을 실행합니다. 동일한 머신에서, 심지어 동일한 프로젝트에서도 둘 다 동시에 실행할 수 있습니다. 각각은 별도의 세션 기록을 유지하지만 CLAUDE.md 파일을 통해 구성 및 프로젝트 메모리를 공유합니다.

CLI 세션을 Desktop으로 이동하려면 터미널에서 `/desktop`을 실행합니다. Claude는 세션을 저장하고 데스크톱 앱에서 열고 CLI를 종료합니다. 이 명령은 macOS 및 Windows에서만 사용 가능합니다.

> **Tip:**
>
> Desktop vs CLI를 사용할 때: 시각적 diff 검토, 파일 첨부 또는 사이드바의 세션 관리를 원할 때 Desktop을 사용합니다. 스크립팅, 자동화, 타사 공급자 또는 터미널 워크플로우를 선호할 때 CLI를 사용합니다.

CLI 플래그 동등물

이 표는 일반적인 CLI 플래그에 대한 데스크톱 앱 동등물을 보여줍니다. 나열되지 않은 플래그는 스크립팅 또는 자동화를 위해 설계되었으므로 데스크톱 동등물이 없습니다.

CLI	Desktop 동등물
`--model sonnet`	세션을 시작하기 전에 전송 버튼 옆의 모델 드롭다운
`--resume`, `--continue`	사이드바의 세션을 클릭합니다
`--permission-mode`	전송 버튼 옆의 모드 선택기
`--dangerously-skip-permissions`	권한 무시 모드. Settings → Claude Code → "권한 무시 모드 허용"에서 활성화합니다. 엔터프라이즈 관리자는 이 설정을 비활성화할 수 있습니다.
`--add-dir`	원격 세션에서 + 버튼으로 여러 저장소 추가
`--allowedTools`, `--disallowedTools`	Desktop에서 사용할 수 없음
`--verbose`	사용할 수 없음. 시스템 로그 확인: macOS의 Console.app, Windows의 Event Viewer → Windows Logs → Application
`--print`, `--output-format`	사용할 수 없음. Desktop은 대화형 전용입니다.
`ANTHROPIC_MODEL` env var	전송 버튼 옆의 모델 드롭다운
`MAX_THINKING_TOKENS` env var	셸 프로필에 설정; 로컬 세션에 적용됩니다. 환경 구성을 참조하세요.

공유 구성

Desktop과 CLI는 동일한 구성 파일을 읽으므로 설정이 이월됩니다:

- **CLAUDE.md** 프로젝트의 파일은 둘 다에서 사용됩니다

- **MCP servers** `~/.claude.json` 또는 `.mcp.json`에 구성된 것은 둘 다에서 작동합니다
- **Hooks** 및 **skills** 설정에 정의된 것은 둘 다에 적용됩니다
- **Settings** `~/.claude.json` 및 `~/.claude/settings.json`은 공유됩니다. `settings.json`의 권한 규칙, 허용된 도구 및 기타 설정은 Desktop 세션에 적용됩니다.
- **Models**: Sonnet, Opus, Haiku는 둘 다에서 사용 가능합니다. Desktop에서 세션을 시작하기 전에 전송 버튼 옆의 드롭다운에서 모델을 선택합니다. 활성 세션 중에는 모델을 변경할 수 없습니다.

> **Note:**
>
> **MCP servers: desktop chat app vs Claude Code**: Claude Desktop chat 앱의 `claude_desktop_config.json`에 구성된 MCP 서버는 Claude Code와 별개이며 Code 탭에 나타나지 않습니다. Claude Code에서 MCP 서버를 사용하려면 `~/.claude.json` 또는 프로젝트의 `.mcp.json` 파일에 구성합니다. 자세한 내용은 MCP 구성을 참조하세요.

기능 비교

이 표는 CLI와 Desktop 간의 핵심 기능을 비교합니다. CLI 플래그의 전체 목록은 CLI 참조를 참조하세요.

기능	CLI	Desktop
권한 모드	`dontAsk`를 포함한 모든 모드	권한 요청, 자동 수락 편집, Plan 모드, Settings를 통한 권한 무시
`--dangerously-skip-permissions`	CLI 플래그	권한 무시 모드. Settings → Claude Code → "권한 무시 모드 허용"에서 활성화합니다
Third-party providers	Bedrock, Vertex, Foundry	사용할 수 없음. Desktop은 Anthropic의 API에 직접 연결됩니다.
MCP servers	설정 파일에 구성	로컬 및 SSH 세션의 Connectors UI 또는 설정 파일
Plugins	`/plugin` 명령	플러그인 관리자 UI
@mention 파일	텍스트 기반	자동 완성 포함

기능	CLI	Desktop
파일 첨부	사용할 수 없음	이미지, PDF
세션 격리	`--worktree` 플래그	자동 worktrees
여러 세션	별도 터미널	사이드바 탭
반복 작업	cron 작업, CI 파이프라인	예약된 작업
스크립팅 및 자동화	`--print`, Agent SDK	사용할 수 없음

Desktop에서 사용할 수 없는 것

다음 기능은 CLI 또는 VS Code 확장에서만 사용 가능합니다:

- **Third-party providers**: Desktop은 Anthropic의 API에 직접 연결됩니다. 대신 CLI를 Bedrock, Vertex 또는 Foundry와 함께 사용합니다.
- **Linux**: 데스크톱 앱은 macOS 및 Windows에서만 사용 가능합니다.
- **Inline code suggestions**: Desktop은 자동 완성 스타일 제안을 제공하지 않습니다. 대화형 프롬프트 및 명시적 코드 변경을 통해 작동합니다.
- **Agent teams**: 다중 에이전트 오케스트레이션은 CLI 및 Agent SDK를 통해 사용 가능하며 Desktop에서는 사용할 수 없습니다.

문제 해결

버전 확인하기

실행 중인 데스크톱 앱의 버전을 보려면:

- **macOS**: 메뉴 모음에서 **Claude**를 클릭한 다음 **About Claude**를 클릭합니다
- **Windows**: **Help**를 클릭한 다음 **About**을 클릭합니다

버전 번호를 클릭하여 클립보드에 복사합니다.

Code 탭의 403 또는 인증 오류

Code 탭을 사용할 때 `Error 403: Forbidden` 또는 기타 인증 실패가 표시되면:

1. 앱 메뉴에서 로그아웃했다가 다시 로그인합니다. 이것이 가장 일반적인 수정입니다.
2. 활성 유료 구독(Pro, Max, Teams 또는 Enterprise)이 있는지 확인합니다.
3. CLI는 작동하지만 Desktop은 작동하지 않으면 데스크톱 앱을 완전히 종료하고(창만 닫지 말고) 다시 열고 로그인합니다.
4. 인터넷 연결 및 프록시 설정을 확인합니다.

시작 시 빈 화면 또는 정지된 화면

앱이 열리지만 빈 화면이나 응답하지 않는 화면을 표시하면:

1. 앱을 다시 시작합니다.
2. 보류 중인 업데이트를 확인합니다. 앱은 시작 시 자동으로 업데이트됩니다.
3. Windows에서 **Windows Logs → Application** 아래의 Event Viewer에서 충돌 로그를 확인합니다.

"Failed to load session"

`Failed to load session` 이 표시되면 선택한 폴더가 더 이상 존재하지 않거나 Git 저장소에 설치되지 않은 Git LFS가 필요하거나 파일 권한이 액세스를 방지할 수 있습니다. 다른 폴더를 선택하거나 앱을 다시 시작해 보세요.

세션이 설치된 도구를 찾지 못함

Claude가 `npm`, `node` 또는 기타 CLI 명령과 같은 도구를 찾을 수 없으면 도구가 일반 터미널에서 작동하는지 확인하고, 셸 프로필이 PATH를 올바르게 설정하는지 확인하고, 데스크톱 앱을 다시 시작하여 환경 변수를 다시 로드합니다.

Git 및 Git LFS 오류

Windows에서 Git은 로컬 세션을 시작하기 위해 Code 탭에 필요합니다. "Git is required"가 표시되면 Windows용 Git을 설치하고 앱을 다시 시작합니다.

"Git LFS is required by this repository but is not installed" 가 표시되면 git-lfs.com에서 Git LFS를 설치하고 `git lfs install` 을 실행한 다음 앱을 다시 시작합니다.

Windows에서 MCP 서버가 작동하지 않음

Windows에서 MCP 서버 토글이 응답하지 않거나 서버가 연결되지 않으면 서버가 설정에서 올바르게 구성되었는지 확인하고, 앱을 다시 시작하고, Task Manager에서 서버 프로세스가 실행 중인지 확인하고, 연결 오류에 대한 서버 로그를 검토합니다.

앱이 종료되지 않음

- **macOS**: Cmd+Q를 누릅니다. 앱이 응답하지 않으면 Cmd+Option+Esc로 강제 종료를 사용하고 Claude를 선택한 다음 강제 종료를 클릭합니다.
- **Windows**: Ctrl+Shift+Esc로 Task Manager를 사용하여 Claude 프로세스를 종료합니다.

Windows 특정 문제

- **설치 후 PATH가 업데이트되지 않음**: 새 터미널 창을 엽니다. PATH 업데이트는 새 터미널 세션에만 적용됩니다.

- **동시 설치 오류**: 진행 중인 다른 설치에 대한 오류가 표시되지만 없으면 관리자로 설치 프로그램을 실행해 보세요.
- **ARM64**: Windows ARM64 장치는 완전히 지원됩니다.

Intel Mac에서 Cowork 탭을 사용할 수 없음

Cowork 탭은 macOS에서 Apple Silicon(M1 이상)이 필요합니다. Windows에서는 Cowork를 모든 지원되는 하드웨어에서 사용할 수 있습니다. Chat 및 Code 탭은 Intel Mac에서 정상적으로 작동합니다.

CLI에서 열 때 "Branch doesn't exist yet"

원격 세션은 로컬 머신에 존재하지 않는 분기를 만들 수 있습니다. 세션 도구 모음의 분기 이름을 클릭하여 복사한 다음 로컬로 가져옵니다:

```
git fetch origin <branch-name>
git checkout <branch-name>
```

여전히 막혔나요?

- GitHub Issues에서 검색하거나 버그를 제출합니다
- Claude 지원 센터를 방문합니다

버그를 제출할 때 데스크톱 앱 버전, 운영 체제, 정확한 오류 메시지, 관련 로그를 포함합니다. macOS에서는 Console.app을 확인합니다. Windows에서는 Event Viewer → Windows Logs → Application을 확인합니다.

Chrome에서 Claude Code 사용하기 (베타)

> Claude Code를 Chrome 브라우저에 연결하여 웹 앱을 테스트하고, 콘솔 로그로 디버깅하며, 양식 작성을 자동화하고, 웹 페이지에서 데이터를 추출합니다.

Claude Code는 Claude in Chrome 브라우저 확장 프로그램과 통합되어 CLI 또는 VS Code 확장 프로그램에서 브라우저 자동화 기능을 제공합니다. 코드를 작성한 후 컨텍스트를 전환하지 않고 브라우저에서 테스트하고 디버깅합니다.

Claude는 브라우저 작업을 위해 새 탭을 열고 브라우저의 로그인 상태를 공유하므로 이미 로그인한 모든 사이트에 액세스할 수 있습니다. 브라우저 작업은 실시간으로 표시되는 Chrome 창에서 실행됩니다. Claude가 로그인 페이지나 CAPTCHA를 만나면 일시 중지하고 수동으로 처리하도록 요청합니다.

> **Note:**
>
> Chrome 통합은 베타 버전이며 현재 Google Chrome에서만 작동합니다. Brave, Arc 또는 기타 Chromium 기반 브라우저에서는 아직 지원되지 않습니다. WSL(Windows Subsystem for Linux)도 지원되지 않습니다.

기능

Chrome이 연결되면 단일 워크플로우에서 브라우저 작업과 코딩 작업을 연결할 수 있습니다.

- **라이브 디버깅**: 콘솔 오류 및 DOM 상태를 직접 읽은 후 이를 유발한 코드를 수정합니다.
- **디자인 검증**: Figma 목업에서 UI를 빌드한 후 브라우저에서 열어 일치하는지 확인합니다.
- **웹 앱 테스트**: 양식 유효성 검사를 테스트하고, 시각적 회귀를 확인하거나, 사용자 흐름을 검증합니다.
- **인증된 웹 앱**: API 커넥터 없이 Google Docs, Gmail, Notion 또는 로그인한 모든 앱과 상호작용합니다.
- **데이터 추출**: 웹 페이지에서 구조화된 정보를 가져와 로컬에 저장합니다.
- **작업 자동화**: 데이터 입력, 양식 작성 또는 다중 사이트 워크플로우와 같은 반복적인 브라우저 작업을 자동화합니다.
- **세션 기록**: 브라우저 상호작용을 GIF로 기록하여 발생한 상황을 문서화하거나 공유합니다.

필수 요구사항

Chrome에서 Claude Code를 사용하기 전에 다음이 필요합니다.

- Google Chrome 브라우저
- Claude in Chrome 확장 프로그램 버전 1.0.36 이상
- Claude Code 버전 2.0.73 이상
- 직접 Anthropic 플랜 (Pro, Max, Team 또는 Enterprise)

> **Note:**
>
> Chrome 통합은 Amazon Bedrock, Google Cloud Vertex AI 또는 Microsoft Foundry와 같은 타사 제공자를 통해 사용할 수 없습니다. 타사 제공자를 통해서만 Claude에 액세스하는 경우 이 기능을 사용하려면 별도의 claude.ai 계정이 필요합니다.

CLI에서 시작하기

Step 1: Chrome으로 Claude Code 시작

`--chrome` 플래그로 Claude Code를 시작합니다.

```
claude --chrome
```

기존 세션 내에서 `/chrome`을 실행하여 Chrome을 활성화할 수도 있습니다.

Step 2: Claude에게 브라우저 사용을 요청

이 예제는 페이지로 이동하고, 상호작용하며, 터미널이나 편집기에서 모두 발견한 내용을 보고합니다.

```
code.claude.com/docs로 이동하여 검색 상자를 클릭하고,
"hooks"를 입력한 후 나타나는 결과를 알려주세요.
```

언제든지 `/chrome`을 실행하여 연결 상태를 확인하고, 권한을 관리하거나, 확장 프로그램을 다시 연결합니다.

VS Code의 경우 VS Code에서 브라우저 자동화를 참조하세요.

기본적으로 Chrome 활성화

각 세션마다 `--chrome`을 전달하지 않으려면 `/chrome`을 실행하고 "기본적으로 활성화"를 선택합니다.

VS Code 확장 프로그램에서는 Chrome 확장 프로그램이 설치되어 있으면 Chrome을 사용할 수 있습니다. 추가 플래그가 필요하지 않습니다.

> **Note:**
>
> CLI에서 기본적으로 Chrome을 활성화하면 브라우저 도구가 항상 로드되므로 컨텍스트 사용량이 증가합니다. 컨텍스트 소비가 증가하는 것을 발견하면 이 설정을 비활성화하고 필요할 때만 `--chrome` 을 사용합니다.

사이트 권한 관리

사이트 수준 권한은 Chrome 확장 프로그램에서 상속됩니다. Chrome 확장 프로그램 설정에서 권한을 관리하여 Claude가 탐색하고, 클릭하고, 입력할 수 있는 사이트를 제어합니다.

예제 워크플로우

이 예제들은 브라우저 작업과 코딩 작업을 결합하는 일반적인 방법을 보여줍니다. `/mcp` 를 실행하고 `claude-in-chrome` 을 선택하여 사용 가능한 브라우저 도구의 전체 목록을 확인합니다.

로컬 웹 애플리케이션 테스트

웹 앱을 개발할 때 Claude에게 변경 사항이 올바르게 작동하는지 확인하도록 요청합니다.

```
방금 로그인 양식 유효성 검사를 업데이트했습니다. localhost:3000을 열고,
잘못된 데이터로 양식을 제출해 보고, 오류 메시지가 올바르게
나타나는지 확인해 주시겠어요?
```

Claude는 로컬 서버로 이동하여 양식과 상호작용하고 관찰한 내용을 보고합니다.

콘솔 로그로 디버깅

Claude는 콘솔 출력을 읽어 문제 진단을 도울 수 있습니다. 로그가 상세할 수 있으므로 모든 콘솔 출력을 요청하는 대신 Claude에게 찾을 패턴을 알려줍니다.

```
대시보드 페이지를 열고 페이지가 로드될 때 콘솔에서 오류를
확인해 주세요.
```

Claude는 콘솔 메시지를 읽고 특정 패턴이나 오류 유형을 필터링할 수 있습니다.

양식 작성 자동화

반복적인 데이터 입력 작업을 가속화합니다.

```
contacts.csv에 고객 연락처 스프레드시트가 있습니다. 각 행에 대해
crm.example.com의 CRM으로 이동하여 "연락처 추가"를 클릭하고
이름, 이메일 및 전화 필드를 작성해 주세요.
```

Claude는 로컬 파일을 읽고, 웹 인터페이스를 탐색하며, 각 레코드에 대한 데이터를 입력합니다.

Google Docs에서 콘텐츠 작성

API 설정 없이 Claude를 사용하여 문서에 직접 작성합니다.

```
최근 커밋을 기반으로 프로젝트 업데이트를 작성하고
docs.google.com/document/d/abc123의 Google Doc에 추가해 주세요.
```

Claude는 문서를 열고, 편집기를 클릭한 후 콘텐츠를 입력합니다. 이는 로그인한 모든 웹 앱에서 작동합니다. Gmail, Notion, Sheets 등입니다.

웹 페이지에서 데이터 추출

웹사이트에서 구조화된 정보를 가져옵니다.

```
제품 목록 페이지로 이동하여 각 항목의 이름, 가격 및 가용성을
추출합니다. 결과를 CSV 파일로 저장해 주세요.
```

Claude는 페이지로 이동하여 콘텐츠를 읽고 데이터를 구조화된 형식으로 컴파일합니다.

다중 사이트 워크플로우 실행

여러 웹사이트에서 작업을 조정합니다.

```
내 캘린더에서 내일의 회의를 확인한 후, 외부 참석자가 있는 각
회의에 대해 해당 회사 웹사이트를 찾아보고 그들이 하는 일에 대한
메모를 추가해 주세요.
```

Claude는 탭 전체에서 작업하여 정보를 수집하고 워크플로우를 완료합니다.

데모 GIF 기록

브라우저 상호작용의 공유 가능한 기록을 만듭니다.

```
장바구니에 항목을 추가하는 것부터 확인 페이지까지 체크아웃
흐름을 완료하는 방법을 보여주는 GIF를 기록해 주세요.
```

Claude는 상호작용 시퀀스를 기록하고 GIF 파일로 저장합니다.

문제 해결

확장 프로그램이 감지되지 않음

Claude Code에 “Chrome 확장 프로그램이 감지되지 않음”이 표시되는 경우:

1. Chrome 확장 프로그램이 설치되어 있고 `chrome://extensions`에서 활성화되어 있는지 확인합니다.
2. `claude --version`을 실행하여 Claude Code가 최신 버전인지 확인합니다.
3. Chrome이 실행 중인지 확인합니다.
4. `/chrome`을 실행하고 “확장 프로그램 다시 연결”을 선택하여 연결을 다시 설정합니다.
5. 문제가 지속되면 Claude Code와 Chrome을 모두 다시 시작합니다.

Chrome 통합을 처음 활성화할 때 Claude Code는 네이티브 메시징 호스트 구성 파일을 설치합니다. Chrome은 시작 시 이 파일을 읽으므로 첫 번째 시도에서 확장 프로그램이 감지되지 않으면 Chrome을 다시 시작하여 새 구성을 선택합니다.

연결이 계속 실패하면 다음 위치에 호스트 구성 파일이 있는지 확인합니다.

- **macOS**: `~/Library/Application Support/Google/Chrome/NativeMessagingHosts/com.anthropic.claude_code_browser_extension.json`
- **Linux**: `~/.config/google-chrome/NativeMessagingHosts/com.anthropic.claude_code_browser_extension.json`
- **Windows**: Windows 레지스트리에서 `HKCU\Software\Google\Chrome\NativeMessagingHosts\`를 확인합니다.

브라우저가 응답하지 않음

Claude의 브라우저 명령이 작동하지 않는 경우:

1. 모달 대화 상자(경고, 확인, 프롬프트)가 페이지를 차단하고 있는지 확인합니다. JavaScript 대화 상자는 브라우저 이벤트를 차단하고 Claude가 명령을 수신하지 못하게 합니다. 대화 상자를 수동으로 닫은 후 Claude에게 계속하도록 알립니다.
2. Claude에게 새 탭을 만들고 다시 시도하도록 요청합니다.
3. `chrome://extensions`에서 Chrome 확장 프로그램을 비활성화했다가 다시 활성화하여 다시 시작합니다.

긴 세션 중 연결 끊김

Chrome 확장 프로그램의 서비스 워커는 확장 세션 중에 유휴 상태가 될 수 있으며, 이는 연결을 끊습니다. 비활성 기간 후 브라우저 도구가 작동하지 않으면 `/chrome`을 실행하고 "확장 프로그램 다시 연결"을 선택합니다.

Windows 관련 문제

Windows에서 다음을 만날 수 있습니다.

- **명명된 파이프 충돌 (EADDRINUSE)**: 다른 프로세스가 동일한 명명된 파이프를 사용 중인 경우 Claude Code를 다시 시작합니다. Chrome을 사용 중일 수 있는 다른 Claude Code 세션을 모두 닫습니다.
- **네이티브 메시징 호스트 오류**: 시작 시 네이티브 메시징 호스트가 충돌하면 Claude Code를 다시 설치하여 호스트 구성을 다시 생성해 봅니다.

일반적인 오류 메시지

가장 자주 발생하는 오류와 해결 방법은 다음과 같습니다.

오류	원인	해결 방법
"브라우저 확장 프로그램이 연결되지 않음"	네이티브 메시징 호스트가 확장 프로그램에 도달할 수 없음	Chrome과 Claude Code를 다시 시작한 후 `/chrome`을 실행하여 다시 연결합니다.
"확장 프로그램이 감지되지 않음"	Chrome 확장 프로그램이 설치되지 않았거나 비활성화됨	`chrome://extensions`에서 확장 프로그램을 설치하거나 활성화합니다.
"사용 가능한 탭 없음"	Claude가 탭이 준비되기 전에 작동하려고 시도함	Claude에게 새 탭을 만들고 다시 시도하도록 요청합니다.
"수신 끝이 존재하지 않음"	확장 프로그램 서비스 워커가 유휴 상태가 됨	`/chrome`을 실행하고 "확장 프로그램 다시 연결"을 선택합니다.

참고 항목

- VS Code에서 Claude Code 사용: VS Code 확장 프로그램의 브라우저 자동화
- CLI 참조: `--chrome`을 포함한 명령줄 플래그
- 일반적인 워크플로우: Claude Code를 사용하는 더 많은 방법
- 데이터 및 개인정보: Claude Code가 데이터를 처리하는 방법

- Chrome에서 Claude 시작하기: 바로 가기, 일정 예약 및 권한을 포함한 Chrome 확장 프로그램의 전체 문서

Slack의 Claude Code

> Slack 워크스페이스에서 직접 코딩 작업 위임

Slack의 Claude Code는 Claude Code의 강력한 기능을 Slack 워크스페이스에 직접 가져옵니다. `@Claude`를 언급하여 코딩 작업을 요청하면, Claude는 자동으로 의도를 감지하고 웹에서 Claude Code 세션을 생성하여 팀 대화를 떠나지 않고도 개발 작업을 위임할 수 있습니다.

이 통합은 기존 Slack용 Claude 앱을 기반으로 하지만 코딩 관련 요청에 대해 웹의 Claude Code로 지능형 라우팅을 추가합니다.

사용 사례

- **버그 조사 및 수정**: Claude에 Slack 채널에서 보고된 버그를 조사하고 수정하도록 요청합니다.
- **빠른 코드 검토 및 수정**: Claude가 팀 피드백을 기반으로 작은 기능을 구현하거나 코드를 리팩토링하도록 합니다.
- **협업 디버깅**: 팀 토론에서 중요한 컨텍스트(예: 오류 재현 또는 사용자 보고)를 제공할 때, Claude는 이 정보를 사용하여 디버깅 접근 방식을 알릴 수 있습니다.
- **병렬 작업 실행**: Slack에서 코딩 작업을 시작하면서 다른 작업을 계속하고, 완료되면 알림을 받습니다.

필수 조건

Slack에서 Claude Code를 사용하기 전에 다음을 확인하세요:

요구 사항	세부 정보
Claude 플랜	Claude Code 액세스가 있는 Pro, Max, Team 또는 Enterprise(프리미엄 시트)
웹의 Claude Code	웹의 Claude Code 액세스가 활성화되어야 함
GitHub 계정	웹의 Claude Code에 연결되어 있으며 최소 하나의 저장소가 인증됨
Slack 인증	Claude 앱을 통해 Claude 계정에 연결된 Slack 계정

Slack에서 Claude Code 설정

Step 1: Slack에 Claude 앱 설치

워크스페이스 관리자는 Slack 앱 마켓플레이스에서 Claude 앱을 설치해야 합니다. Slack 앱 마켓플레이스를 방문하여 “Slack에 추가”를 클릭하여 설치 프로세스를 시작합니다.

Step 2: Claude 계정 연결

앱이 설치된 후 개별 Claude 계정을 인증합니다:

1. 앱 섹션에서 “Claude”를 클릭하여 Slack에서 Claude 앱을 엽니다
2. 앱 홈 탭으로 이동합니다
3. “연결”을 클릭하여 Slack 계정을 Claude 계정과 연결합니다
4. 브라우저에서 인증 흐름을 완료합니다

Step 3: 웹의 Claude Code 구성

웹의 Claude Code가 제대로 구성되어 있는지 확인합니다:

- claude.ai/code를 방문하여 Slack에 연결한 동일한 계정으로 로그인합니다
- 아직 연결되지 않은 경우 GitHub 계정을 연결합니다
- Claude가 작업할 수 있도록 최소 하나의 저장소를 인증합니다

Step 4: 라우팅 모드 선택

계정을 연결한 후 Claude가 Slack의 메시지를 처리하는 방식을 구성합니다. Slack의 Claude 앱 홈으로 이동하여 **라우팅 모드** 설정을 찾습니다.

모드	동작
코드만	Claude는 모든 @mentions을 Claude Code 세션으로 라우팅합니다. Claude를 Slack에서 개발 작업 전용으로 사용하는 팀에 가장 적합합니다.
코드 + 채팅	Claude는 각 메시지를 분석하고 Claude Code(코딩 작업용)와 Claude Chat(작성, 분석 및 일반 질문용) 간에 지능형으로 라우팅합니다. 모든 유형의 작업에 대해 단일 @Claude 진입점을 원하는 팀에 가장 적합합니다.

> **Note:**
>
> 코드 + 채팅 모드에서 Claude가 메시지를 채팅으로 라우팅했지만 코딩 세션을 원했다면 “코드로 다시 시도”를 클릭하여 Claude Code 세션을 대신 생성할 수 있습니다. 마찬가지로 코드로 라우팅되었지만 채팅 세션을 원했다면 해당 스레드에서 해당 옵션을 선택할 수 있습니다.

작동 방식

자동 감지

Slack 채널이나 스레드에서 @Claude를 언급하면, Claude는 자동으로 메시지를 분석하여 코딩 작업인지 여부를 결정합니다. Claude가 코딩 의도를 감지하면 일반 채팅 어시스턴트로 응답하는 대신 요청을 웹의 Claude Code로 라우팅합니다.

자동으로 감지되지 않더라도 Claude에 요청을 코딩 작업으로 처리하도록 명시적으로 지시할 수 있습니다.

> **Note:**
>
> Slack의 Claude Code는 채널(공개 또는 비공개)에서만 작동합니다. 직접 메시지(DM)에서는 작동하지 않습니다.

컨텍스트 수집

스레드에서: 스레드에서 @Claude를 언급하면 전체 대화를 이해하기 위해 해당 스레드의 모든 메시지에서 컨텍스트를 수집합니다.

채널에서: 채널에서 직접 언급되면 Claude는 관련 컨텍스트를 위해 최근 채널 메시지를 살펴봅니다.

이 컨텍스트는 Claude가 문제를 이해하고, 적절한 저장소를 선택하고, 작업에 대한 접근 방식을 알리는 데 도움이 됩니다.

> **Warning:**
>
> Slack에서 @Claude가 호출되면 Claude는 요청을 더 잘 이해하기 위해 대화 컨텍스트에 액세스할 수 있습니다. Claude는 컨텍스트의 다른 메시지의 지시를 따를 수 있으므로 사용자는 Claude를 신뢰할 수 있는 Slack 대화에서만 사용해야 합니다.

세션 흐름

1. **시작**: @Claude를 코딩 요청과 함께 언급합니다
2. **감지**: Claude가 메시지를 분석하고 코딩 의도를 감지합니다
3. **세션 생성**: claude.ai/code에서 새로운 Claude Code 세션이 생성됩니다
4. **진행 상황 업데이트**: Claude는 작업이 진행됨에 따라 Slack 스레드에 상태 업데이트를 게시합니다
5. **완료**: 완료되면 Claude는 요약 및 작업 버튼과 함께 @mentions을 합니다
6. **검토**: “세션 보기”를 클릭하여 전체 기록을 보거나 “PR 생성”을 클릭하여 풀 요청을 엽니다

사용자 인터페이스 요소

앱 홈

앱 홈 탭은 연결 상태를 표시하고 Claude 계정을 Slack에서 연결하거나 연결 해제할 수 있습니다.

메시지 작업

- **세션 보기**: 수행된 모든 작업을 볼 수 있고, 세션을 계속하거나 추가 요청을 할 수 있는 브라우저에서 전체 Claude Code 세션을 엽니다.
- **PR 생성**: 세션의 변경 사항에서 직접 풀 요청을 생성합니다.
- **코드로 다시 시도**: Claude가 처음에 채팅 어시스턴트로 응답했지만 코딩 세션을 원했다면 이 버튼을 클릭하여 요청을 Claude Code 작업으로 다시 시도합니다.
- **저장소 변경**: Claude가 잘못 선택한 경우 다른 저장소를 선택할 수 있습니다.

저장소 선택

Claude는 Slack 대화의 컨텍스트를 기반으로 저장소를 자동으로 선택합니다. 여러 저장소가 적용될 수 있는 경우 Claude는 올바른 저장소를 선택할 수 있는 드롭다운을 표시할 수 있습니다.

액세스 및 권한

사용자 수준 액세스

액세스 유형	요구 사항
Claude Code 세션	각 사용자는 자신의 Claude 계정에서 세션을 실행합니다
사용량 및 속도 제한	세션은 개별 사용자의 플랜 제한에 대해 계산됩니다
저장소 액세스	사용자는 개인적으로 연결한 저장소에만 액세스할 수 있습니다
세션 기록	세션은 claude.ai/code의 Claude Code 기록에 나타납니다

워크스페이스 관리자 권한

Slack 워크스페이스 관리자는 Claude 앱을 워크스페이스에 설치할 수 있는지 여부를 제어합니다. 개별 사용자는 자신의 Claude 계정으로 인증하여 통합을 사용합니다.

어디서 액세스할 수 있는지

Slack에서: 상태 업데이트, 완료 요약 및 작업 버튼이 표시됩니다. 전체 기록은 보존되며 항상 액세스할 수 있습니다.

웹에서: 전체 대화 기록, 모든 코드 변경, 파일 작업 및 세션을 계속하거나 풀 요청을 생성할 수 있는 기능이 있는 완전한 Claude Code 세션입니다.

모범 사례

효과적인 요청 작성

- **구체적으로**: 관련이 있을 때 파일 이름, 함수 이름 또는 오류 메시지를 포함합니다.
- **컨텍스트 제공**: 대화에서 명확하지 않은 경우 저장소 또는 프로젝트를 언급합니다.
- **성공 정의**: "완료"가 무엇인지 설명합니다. Claude가 테스트를 작성해야 합니까? 문서를 업데이트합니까? PR을 생성합니까?
- **스레드 사용**: 버그나 기능을 논의할 때 스레드에서 회신하여 Claude가 전체 컨텍스트를 수집할 수 있도록 합니다.

Slack과 웹 사용 시기

Slack을 사용할 때: 컨텍스트가 이미 Slack 토론에 있을 때, 작업을 비동기적으로 시작하려고 할 때, 또는 가시성이 필요한 팀원과 협업할 때입니다.

웹을 직접 사용할 때: 파일을 업로드해야 할 때, 개발 중 실시간 상호 작용을 원할 때, 또는 더 길고 복잡한 작업을 수행할 때입니다.

문제 해결

세션이 시작되지 않음

1. Claude 앱 홈에서 Claude 계정이 연결되어 있는지 확인합니다
2. 웹의 Claude Code 액세스가 활성화되어 있는지 확인합니다
3. Claude Code에 연결된 GitHub 저장소가 최소 하나 있는지 확인합니다

저장소가 표시되지 않음

1. claude.ai/code에서 웹의 Claude Code에 저장소를 연결합니다
2. 해당 저장소에 대한 GitHub 권한을 확인합니다
3. GitHub 계정을 연결 해제했다가 다시 연결해봅니다

잘못된 저장소 선택됨

1. "저장소 변경" 버튼을 클릭하여 다른 저장소를 선택합니다
2. 더 정확한 선택을 위해 요청에 저장소 이름을 포함합니다

인증 오류

1. 앱 홈에서 Claude 계정을 연결 해제했다가 다시 연결합니다

2. 브라우저에서 올바른 Claude 계정으로 로그인했는지 확인합니다
3. Claude 플랜에 Claude Code 액세스가 포함되어 있는지 확인합니다

세션 만료

1. 세션은 웹의 Claude Code 기록에서 액세스할 수 있습니다
2. claude.ai/code에서 과거 세션을 계속하거나 참조할 수 있습니다

현재 제한 사항

- **GitHub만**: 현재 GitHub의 저장소만 지원합니다.
- **한 번에 하나의 PR**: 각 세션은 하나의 풀 요청을 생성할 수 있습니다.
- **속도 제한 적용**: 세션은 개별 Claude 플랜의 속도 제한을 사용합니다.
- **웹 액세스 필요**: 사용자는 웹의 Claude Code 액세스가 있어야 합니다. 없으면 표준 Claude 채팅 응답만 받습니다.

관련 리소스

- 웹의 Claude Code 웹의 Claude Code에 대해 자세히 알아보기
- Slack용 Claude 일반 Slack용 Claude 문서
- Slack 앱 마켓플레이스 Slack 마켓플레이스에서 Claude 앱 설치
- Claude 도움말 센터 추가 지원 받기

웹에서 Claude Code 사용하기

> 안전한 클라우드 인프라에서 Claude Code 작업을 비동기적으로 실행합니다

> **Note:**
> 웹에서 Claude Code는 현재 연구 미리보기 상태입니다.

Claude Code on the web란 무엇입니까?

웹에서 Claude Code를 사용하면 개발자가 Claude 앱에서 Claude Code를 시작할 수 있습니다. 이는 다음과 같은 경우에 완벽합니다:

- **질문에 답변하기**: 코드 아키텍처 및 기능 구현 방식에 대해 질문하기
- **버그 수정 및 일상적인 작업**: 자주 조정할 필요가 없는 잘 정의된 작업
- **병렬 작업**: 여러 버그 수정을 동시에 처리하기
- **로컬 머신에 없는 저장소**: 로컬에 체크아웃하지 않은 코드 작업하기
- **백엔드 변경**: Claude Code가 테스트를 작성한 다음 해당 테스트를 통과하는 코드를 작성할 수 있는 경우

Claude Code는 iOS 및 Android용 Claude 앱에서도 사용 가능하므로 이동 중에 작업을 시작하고 진행 중인 작업을 모니터링할 수 있습니다.

`--remote` 를 사용하여 터미널에서 웹으로 새 작업을 시작하거나, 웹 세션을 터미널로 텔레포트하여 로컬에서 계속할 수 있습니다. 클라우드 인프라 대신 자신의 머신에서 Claude Code를 실행하면서 웹 인터페이스를 사용하려면 Remote Control을 참조하세요.

Claude Code on the web을 누가 사용할 수 있습니까?

웹에서 Claude Code는 연구 미리보기로 다음에 사용 가능합니다:

- **Pro 사용자**
- **Max 사용자**
- **Team 사용자**
- **Enterprise 사용자** (프리미엄 시트 또는 Chat + Claude Code 시트 포함)

시작하기

1. claude.ai/code 방문
2. GitHub 계정 연결
3. 저장소에 Claude GitHub 앱 설치
4. 기본 환경 선택
5. 코딩 작업 제출
6. diff 보기에서 변경 사항 검토, 주석으로 반복, pull request 생성

작동 방식

웹에서 Claude Code on the web에서 작업을 시작할 때:

1. **저장소 복제**: 저장소가 Anthropic 관리 가상 머신으로 복제됩니다
2. **환경 설정**: Claude가 코드를 포함한 안전한 클라우드 환경을 준비한 다음 구성된 경우 설정 스크립트를 실행합니다
3. **네트워크 구성**: 인터넷 액세스는 설정에 따라 구성됩니다
4. **작업 실행**: Claude가 코드를 분석하고, 변경을 수행하고, 테스트를 실행하고, 작업을 확인합니다
5. **완료**: 완료되면 알림을 받고 변경 사항으로 PR을 생성할 수 있습니다
6. **결과**: 변경 사항이 분기로 푸시되어 pull request 생성 준비가 됩니다

diff 보기로 변경 사항 검토하기

Diff 보기를 사용하면 pull request를 생성하기 전에 Claude가 정확히 무엇을 변경했는지 볼 수 있습니다. GitHub에서 변경 사항을 검토하기 위해 “Create PR”을 클릭하는 대신 앱에서 직접 diff를 보고 변경 사항이 준비될 때까지 Claude와 반복합니다.

Claude가 파일을 변경하면 추가 및 제거된 줄 수를 표시하는 diff 통계 표시기가 나타납니다(예: `+12 -1`). 이 표시기를 선택하여 diff 뷰어를 열면 왼쪽에 파일 목록이 표시되고 오른쪽에 각 파일의 변경 사항이 표시됩니다.

diff 보기에서 다음을 수행할 수 있습니다:

- 파일별로 변경 사항 검토
- 특정 변경 사항에 주석을 달아 수정 요청
- 보이는 내용을 기반으로 Claude와 계속 반복

이를 통해 draft PR을 생성하거나 GitHub로 전환하지 않고도 여러 라운드의 피드백을 통해 변경 사항을 개선할 수 있습니다.

웹과 터미널 간 작업 이동

터미널에서 웹으로 새 작업을 시작하거나 웹 세션을 터미널로 가져와 로컬에서 계속할 수 있습니다. 웹 세션은 노트북을 닫아도 유지되며 Claude 모바일 앱을 포함한 어디서나 모니터링할 수 있습니다.

> **Note:**
>
> 세션 핸드오프는 일방향입니다: 웹 세션을 터미널로 가져올 수 있지만 기존 터미널 세션을 웹으로 푸시할 수 없습니다. `--remote` 플래그는 현재 저장소에 대한 새로운 웹 세션을 생성합니다.

터미널에서 웹으로

`--remote` 플래그를 사용하여 명령줄에서 웹 세션을 시작합니다:

```
claude --remote "Fix the authentication bug in src/auth/login.ts"
```

이렇게 하면 claude.ai에서 새 웹 세션이 생성됩니다. 작업은 클라우드에서 실행되는 동안 로컬에서 계속 작업할 수 있습니다. `/tasks`를 사용하여 진행 상황을 확인하거나 claude.ai 또는 Claude 모바일 앱에서 세션을 열어 직접 상호 작용합니다. 여기서 Claude를 조종하고, 피드백을 제공하거나, 다른 대화처럼 질문에 답변할 수 있습니다.

원격 작업 팁

로컬에서 계획하고 원격으로 실행: 복잡한 작업의 경우 Claude를 plan mode에서 시작하여 접근 방식을 협력한 다음 작업을 웹으로 보냅니다:

```
claude --permission-mode plan
```

Plan mode에서 Claude는 파일만 읽고 코드베이스를 탐색할 수 있습니다. 계획에 만족하면 자율 실행을 위해 원격 세션을 시작합니다:

```
claude --remote "Execute the migration plan in docs/migration-plan.md"
```

이 패턴은 Claude가 클라우드에서 자율적으로 실행되도록 하면서 전략에 대한 제어를 제공합니다.

작업을 병렬로 실행: 각 `--remote` 명령은 독립적으로 실행되는 자체 웹 세션을 생성합니다. 여러 작업을 시작할 수 있으며 모두 별도의 세션에서 동시에 실행됩니다:

```
claude --remote "Fix the flaky test in auth.spec.ts"
claude --remote "Update the API documentation"
claude --remote "Refactor the logger to use structured output"
```

`/tasks` 로 모든 세션을 모니터링합니다. 세션이 완료되면 웹 인터페이스에서 PR을 생성하거나 세션을 텔레포트하여 터미널에서 계속 작업할 수 있습니다.

웹에서 터미널로

웹 세션을 터미널로 가져오는 방법은 여러 가지입니다:

- **`/teleport` 사용**: Claude Code 내에서 `/teleport` (또는 `/tp`)를 실행하여 웹 세션의 대화형 선택기를 봅니다. 커밋되지 않은 변경 사항이 있으면 먼저 stash하라는 메시지가 표시됩니다.
- **`--teleport` 사용**: 명령줄에서 `claude --teleport` 를 실행하여 대화형 세션 선택기를 사용하거나 `claude --teleport <session-id>` 를 실행하여 특정 세션을 직접 재개합니다.
- **`/tasks` 에서**: `/tasks` 를 실행하여 백그라운드 세션을 보고 `t` 를 눌러 하나로 텔레포트합니다
- **웹 인터페이스에서**: “Open in CLI”를 클릭하여 터미널에 붙여넣을 수 있는 명령을 복사합니다

세션을 텔레포트하면 Claude가 올바른 저장소에 있는지 확인하고, 원격 세션에서 분기를 가져와 체크아웃하고, 전체 대화 기록을 터미널에 로드합니다.

텔레포트 요구 사항

텔레포트는 세션을 재개하기 전에 이러한 요구 사항을 확인합니다. 요구 사항이 충족되지 않으면 오류가 표시되거나 문제를 해결하라는 메시지가 표시됩니다.

요구 사항	세부 정보
Clean git state	작업 디렉토리에 커밋되지 않은 변경 사항이 없어야 합니다. 필요한 경우 텔레포트가 변경 사항을 stash하라는 메시지를 표시합니다.
올바른 저장소	fork가 아닌 동일한 저장소의 체크아웃에서 `--teleport` 를 실행해야 합니다.
분기 사용 가능	웹 세션의 분기가 원격으로 푸시되어야 합니다. 텔레포트가 자동으로 가져와 체크아웃합니다.
동일한 계정	웹 세션에서 사용한 Claude.ai 계정과 동일한 계정으로 인증되어야 합니다.

세션 공유

세션을 공유하려면 아래 계정 유형에 따라 가시성을 전환합니다. 그 후 세션 링크를 그대로 공유합니다. 공유 세션을 열 수 있는 수신자는 로드 시 세션의 최신 상태를 보게 되지만 수신자의 페이지는 실시간으로 업데이트되지 않습니다.

Enterprise 또는 Teams 계정에서 공유

Enterprise 및 Teams 계정의 경우 두 가지 가시성 옵션은 **Private** 및 **Team**입니다. Team 가시성은 Claude.ai 조직의 다른 구성원에게 세션을 표시합니다. 저장소 액세스 확인은 기본적으로 수신자의 계정에 연결된 GitHub 계정을 기반으로 활성화됩니다. 계정의 표시 이름은 액세스 권한이 있는 모든 수신자에게 표시됩니다. Claude in Slack 세션은 자동으로 Team 가시성으로 공유됩니다.

Max 또는 Pro 계정에서 공유

Max 및 Pro 계정의 경우 두 가지 가시성 옵션은 **Private** 및 **Public**입니다. Public 가시성은 claude.ai에 로그인한 모든 사용자에게 세션을 표시합니다.

공유하기 전에 민감한 내용이 있는지 세션을 확인합니다. 세션에는 개인 GitHub 저장소의 코드 및 자격 증명이 포함될 수 있습니다. 저장소 액세스 확인은 기본적으로 활성화되지 않습니다.

Settings > Claude Code > Sharing settings로 이동하여 저장소 액세스 확인을 활성화하거나 공유 세션에서 이름을 숨길 수 있습니다.

세션 관리

세션 보관

세션을 보관하여 세션 목록을 정리할 수 있습니다. 보관된 세션은 기본 세션 목록에서 숨겨지지만 보관된 세션을 필터링하여 볼 수 있습니다.

세션을 보관하려면 사이드바의 세션 위에 마우스를 올리고 보관 아이콘을 클릭합니다.

세션 삭제

세션을 삭제하면 세션과 해당 데이터가 영구적으로 제거됩니다. 이 작업은 실행 취소할 수 없습니다. 두 가지 방법으로 세션을 삭제할 수 있습니다:

- **사이드바에서**: 보관된 세션을 필터링한 다음 삭제할 세션 위에 마우스를 올리고 삭제 아이콘을 클릭합니다
- **세션 메뉴에서**: 세션을 열고 세션 제목 옆의 드롭다운을 클릭한 다음 **Delete**를 선택합니다

세션이 삭제되기 전에 확인하라는 메시지가 표시됩니다.

클라우드 환경

기본 이미지

일반적인 도구 체인 및 언어 생태계가 사전 설치된 범용 이미지를 구축하고 유지합니다. 이 이미지에는 다음이 포함됩니다:

- 인기 있는 프로그래밍 언어 및 런타임
- 일반적인 빌드 도구 및 패키지 관리자
- 테스트 프레임워크 및 린터

사용 가능한 도구 확인

환경에 사전 설치된 항목을 확인하려면 Claude Code에 다음을 실행하도록 요청합니다:

```
check-tools
```

이 명령은 다음을 표시합니다:

- 프로그래밍 언어 및 해당 버전
- 사용 가능한 패키지 관리자
- 설치된 개발 도구

언어별 설정

범용 이미지에는 다음에 대해 사전 구성된 환경이 포함됩니다:

- **Python**: pip, poetry 및 일반적인 과학 라이브러리가 포함된 Python 3.x
- **Node.js**: npm, yarn, pnpm 및 bun이 포함된 최신 LTS 버전
- **Ruby**: 버전 3.1.6, 3.2.6, 3.3.6 (기본값: 3.3.6) (gem, bundler 및 버전 관리용 rbenv 포함)
- **PHP**: 버전 8.4.14
- **Java**: Maven 및 Gradle이 포함된 OpenJDK
- **Go**: 모듈 지원이 포함된 최신 안정 버전
- **Rust**: cargo가 포함된 Rust 도구 체인
- **C++**: GCC 및 Clang 컴파일러

데이터베이스

범용 이미지에는 다음 데이터베이스가 포함됩니다:

- **PostgreSQL**: 버전 16
- **Redis**: 버전 7.0

환경 구성

웹에서 Claude Code on the web에서 세션을 시작할 때 내부적으로 다음이 발생합니다:

1. **환경 준비**: 저장소를 복제하고 구성된 설정 스크립트를 실행합니다. 저장소는 GitHub 저장소의 기본 분기로 복제됩니다. 특정 분기를 체크아웃하려면 프롬프트에서 지정할 수 있습니다.
2. **네트워크 구성**: 에이전트에 대한 인터넷 액세스를 구성합니다. 인터넷 액세스는 기본적으로 제한되지만 필요에 따라 인터넷 액세스가 없거나 전체 인터넷 액세스를 갖도록 환경을 구성할 수 있습니다.
3. **Claude Code 실행**: Claude Code가 실행되어 작업을 완료하고, 코드를 작성하고, 테스트를 실행하고, 작업을 확인합니다. 웹 인터페이스를 통해 세션 전체에서 Claude를 안내하고 조종할 수 있습니다. Claude는 `CLAUDE.md` 에서 정의한 컨텍스트를 존중합니다.
4. **결과**: Claude가 작업을 완료하면 분기를 원격으로 푸시합니다. 분기에 대한 PR을 생성할 수 있습니다.

> **Note:**
>
> Claude는 환경에서 사용 가능한 터미널 및 CLI 도구를 통해 전적으로 작동합니다. 범용 이미지의 사전 설치된 도구와 hooks 또는 종속성 관리를 통해 설치하는 추가 도구를 사용합니다.

새 환경을 추가하려면: 현재 환경을 선택하여 환경 선택기를 열고 “Add environment”를 선택합니다. 이렇게 하면 환경 이름, 네트워크 액세스 수준, 환경 변수 및 설정 스크립트를 지정할 수 있는 대화 상자가 열립니다.

기존 환경을 업데이트하려면: 현재 환경을 선택하고 환경 이름의 오른쪽에서 설정 버튼을 선택합니다. 이렇게 하면 환경 이름, 네트워크 액세스, 환경 변수 및 설정 스크립트를 업데이트할 수 있는 대화 상자가 열립니다.

터미널에서 기본 환경을 선택하려면: 여러 환경이 구성된 경우 `/remote-env` 를 실행하여 `--remote` 를 사용하여 터미널에서 웹 세션을 시작할 때 사용할 환경을 선택합니다. 단일 환경의 경우 이 명령은 현재 구성을 표시합니다.

> **Note:**
>
> 환경 변수는 `.env` 형식으로 키-값 쌍으로 지정해야 합니다. 예를 들어:
>
> ```
> API_KEY=your_api_key
> DEBUG=true
> ```

설정 스크립트

설정 스크립트는 새 클라우드 세션이 시작될 때 Claude Code가 시작되기 전에 실행되는 Bash 스크립트입니다. 설정 스크립트를 사용하여 종속성을 설치하고, 도구를 구성하거나, 클라우드 환경이 필요하지만 기본 이미지에 없는 항목을 준비합니다.

스크립트는 Ubuntu 24.04에서 root로 실행되므로 `apt install` 및 대부분의 언어 패키지 관리자가 작동합니다.

> **Tip:**
>
> 설정 스크립트에 추가하기 전에 이미 설치된 항목을 확인하려면 Claude에 클라우드 세션에서 `check-tools`를 실행하도록 요청합니다.

설정 스크립트를 추가하려면 환경 설정 대화 상자를 열고 **Setup script** 필드에 스크립트를 입력합니다.

이 예제는 기본 이미지에 없는 `gh` CLI를 설치합니다:

```
#!/bin/bash
apt update && apt install -y gh
```

설정 스크립트는 새 세션을 생성할 때만 실행됩니다. 기존 세션을 재개할 때는 건너뜁니다.

스크립트가 0이 아닌 값으로 종료되면 세션이 시작되지 않습니다. 세션을 차단하지 않으려면 중요하지 않은 명령에 `|| true`를 추가합니다.

> **Note:**
>
> 패키지를 설치하는 설정 스크립트는 레지스트리에 도달하기 위해 네트워크 액세스가 필요합니다. 기본 네트워크 액세스는 npm, PyPI, RubyGems 및 crates.io를 포함한 일반적인 패키지 레지스트리에 대한 연결을 허용합니다. 환경에 네트워크 액세스가 비활성화되어 있으면 스크립트가 패키지 설치에 실패합니다.

설정 스크립트 vs. SessionStart hooks

클라우드가 필요하지만 노트북에 이미 있는 것(예: 언어 런타임 또는 CLI 도구)을 설치하려면 설정 스크립트를 사용합니다. 클라우드 및 로컬 모두에서 실행되어야 하는 프로젝트 설정(예: `npm install`)의 경우 SessionStart hook을 사용합니다.

둘 다 세션 시작 시 실행되지만 다른 위치에 속합니다:

	설정 스크립트	SessionStart hooks
첨부 대상	클라우드 환경	저장소
구성 위치	클라우드 환경 UI	저장소의 `.claude/settings.json`
실행	Claude Code 시작 전, 새 세션에서만	Claude Code 시작 후, 재개된 세션을 포함한 모든 세션에서
범위	클라우드 환경만	로컬 및 클라우드 모두

SessionStart hooks는 로컬의 사용자 수준 `~/.claude/settings.json`에서도 정의할 수 있지만 사용자 수준 설정은 클라우드 세션으로 이월되지 않습니다. 클라우드에서는 저장소에 커밋된 hooks만 실행됩니다.

종속성 관리

사용자 정의 환경 이미지 및 스냅샷은 아직 지원되지 않습니다. 설정 스크립트를 사용하여 세션이 시작될 때 패키지를 설치하거나 SessionStart hooks를 사용하여 로컬 환경에서도 실행되어야 하는 종속성 설치를 수행합니다. SessionStart hooks에는 알려진 제한 사항이 있습니다.

설정 스크립트를 사용하여 자동 종속성 설치를 구성하려면 환경 설정을 열고 스크립트를 추가합니다:

```
#!/bin/bash
npm install
pip install -r requirements.txt
```

또는 저장소의 `.claude/settings.json` 파일에서 SessionStart hooks를 사용하여 로컬 환경에서도 실행되어야 하는 종속성 설치를 수행할 수 있습니다:

```
{
  "hooks": {
    "SessionStart": [
      {
        "matcher": "startup",
        "hooks": [
          {
            "type": "command",
            "command": "\"$CLAUDE_PROJECT_DIR\"/scripts/install_pkgs.sh"
          }
        ]
      }
    ]
  }
}
```

`scripts/install_pkgs.sh`에서 해당 스크립트를 생성합니다:

```
#!/bin/bash

## Only run in remote environments
if [ "$CLAUDE_CODE_REMOTE" != "true" ]; then
  exit 0
fi

npm install
pip install -r requirements.txt
exit 0
```

실행 가능하게 만듭니다: `chmod +x scripts/install_pkgs.sh`

환경 변수 유지

SessionStart hooks는 `CLAUDE_ENV_FILE` 환경 변수에 지정된 파일에 쓰는 방식으로 후속 Bash 명령에 대한 환경 변수를 유지할 수 있습니다. 자세한 내용은 hooks 참조의 SessionStart hooks를 참조하세요.

종속성 관리 제한 사항

- **모든 세션에 대해 Hooks 실행**: SessionStart hooks는 로컬 및 원격 환경 모두에서 실행됩니다. 원격 세션에만 hook을 범위 지정하는 hook 구성이 없습니다. 로컬 실행을 건너뛰려면 위에 표시된 대로 스크립트에서 `CLAUDE_CODE_REMOTE` 환경 변수를 확인합니다.
- **네트워크 액세스 필요**: 설치 명령은 패키지 레지스트리에 도달하기 위해 네트워크 액세스가 필요합니다. 환경이 "No internet" 액세스로 구성된 경우 이러한 hooks가 실패합니다. "Limited" (기본값) 또는 "Full" 네트워크 액세스를 사용합니다. 기본 허용 목록에는 npm, PyPI, RubyGems 및 crates.io와 같은 일반적인 레지스트리가 포함됩니다.
- **프록시 호환성**: 원격 환경의 모든 아웃바운드 트래픽은 보안 프록시를 통과합니다. 일부 패키지 관리자는 이 프록시에서 제대로 작동하지 않습니다. Bun은 알려진 예입니다.
- **모든 세션 시작 시 실행**: Hooks는 세션이 시작되거나 재개될 때마다 실행되어 시작 지연을 추가합니다. 재설치하기 전에 종속성이 이미 있는지 확인하여 설치 스크립트를 빠르게 유지합니다.

네트워크 액세스 및 보안

네트워크 정책

GitHub 프록시

보안을 위해 모든 GitHub 작업은 모든 git 상호 작용을 투명하게 처리하는 전용 프록시 서비스를 통해 진행됩니다. 샌드박스 내에서 git 클라이언트는 사용자 정의 빌드 범위 자격 증명을 사용하여 인증합니다. 이 프록시는:

- GitHub 인증을 안전하게 관리합니다 - git 클라이언트는 샌드박스 내에서 범위 자격 증명을 사용하며, 프록시가 이를 확인하고 실제 GitHub 인증 토큰으로 변환합니다
- 안전을 위해 git push 작업을 현재 작업 분기로 제한합니다
- 보안 경계를 유지하면서 원활한 복제, 가져오기 및 PR 작업을 활성화합니다

보안 프록시

환경은 보안 및 남용 방지를 위해 HTTP/HTTPS 네트워크 프록시 뒤에서 실행됩니다. 모든 아웃바운드 인터넷 트래픽은 다음을 제공하는 이 프록시를 통과합니다:

- 악의적인 요청으로부터의 보호
- 속도 제한 및 남용 방지
- 향상된 보안을 위한 콘텐츠 필터링

액세스 수준

기본적으로 네트워크 액세스는 허용 목록 도메인으로 제한됩니다.

네트워크 액세스 비활성화를 포함한 사용자 정의 네트워크 액세스를 구성할 수 있습니다.

기본 허용 도메인

"Limited" 네트워크 액세스를 사용할 때 다음 도메인이 기본적으로 허용됩니다:

Anthropic 서비스

- api.anthropic.com
- statsig.anthropic.com
- platform.claude.com
- code.claude.com
- claude.ai

버전 제어

- github.com
- www.github.com
- api.github.com
- npm.pkg.github.com
- raw.githubusercontent.com
- pkg-npm.githubusercontent.com
- objects.githubusercontent.com
- codeload.github.com
- avatars.githubusercontent.com
- camo.githubusercontent.com
- gist.github.com
- gitlab.com
- www.gitlab.com
- registry.gitlab.com
- bitbucket.org
- www.bitbucket.org
- api.bitbucket.org

컨테이너 레지스트리

- registry-1.docker.io
- auth.docker.io
- index.docker.io

- hub.docker.com
- www.docker.com
- production.cloudflare.docker.com
- download.docker.com
- gcr.io
- *.gcr.io
- ghcr.io
- mcr.microsoft.com
- *.data.mcr.microsoft.com
- public.ecr.aws

클라우드 플랫폼

- cloud.google.com
- accounts.google.com
- gcloud.google.com
- *.googleapis.com
- storage.googleapis.com
- compute.googleapis.com
- container.googleapis.com
- azure.com
- portal.azure.com
- microsoft.com
- www.microsoft.com
- *.microsoftonline.com
- packages.microsoft.com
- dotnet.microsoft.com
- dot.net
- visualstudio.com
- dev.azure.com
- *.amazonaws.com
- *.api.aws
- oracle.com
- www.oracle.com
- java.com

- www.java.com
- java.net
- www.java.net
- download.oracle.com
- yum.oracle.com

패키지 관리자 - JavaScript/Node

- registry.npmjs.org
- www.npmjs.com
- www.npmjs.org
- npmjs.com
- npmjs.org
- yarnpkg.com
- registry.yarnpkg.com

패키지 관리자 - Python

- pypi.org
- www.pypi.org
- files.pythonhosted.org
- pythonhosted.org
- test.pypi.org
- pypi.python.org
- pypa.io
- www.pypa.io

패키지 관리자 - Ruby

- rubygems.org
- www.rubygems.org
- api.rubygems.org
- index.rubygems.org
- ruby-lang.org
- www.ruby-lang.org
- rubyforge.org
- www.rubyforge.org

- rubyonrails.org
- www.rubyonrails.org
- rvm.io
- get.rvm.io

패키지 관리자 - Rust

- crates.io
- www.crates.io
- index.crates.io
- static.crates.io
- rustup.rs
- static.rust-lang.org
- www.rust-lang.org

패키지 관리자 - Go

- proxy.golang.org
- sum.golang.org
- index.golang.org
- golang.org
- www.golang.org
- goproxy.io
- pkg.go.dev

패키지 관리자 - JVM

- maven.org
- repo.maven.org
- central.maven.org
- repo1.maven.org
- jcenter.bintray.com
- gradle.org
- www.gradle.org
- services.gradle.org
- plugins.gradle.org
- kotlin.org

- www.kotlin.org
- spring.io
- repo.spring.io

패키지 관리자 - 기타 언어

- packagist.org (PHP Composer)
- www.packagist.org
- repo.packagist.org
- nuget.org (.NET NuGet)
- www.nuget.org
- api.nuget.org
- pub.dev (Dart/Flutter)
- api.pub.dev
- hex.pm (Elixir/Erlang)
- www.hex.pm
- cpan.org (Perl CPAN)
- www.cpan.org
- metacpan.org
- www.metacpan.org
- api.metacpan.org
- cocoapods.org (iOS/macOS)
- www.cocoapods.org
- cdn.cocoapods.org
- haskell.org
- www.haskell.org
- hackage.haskell.org
- swift.org
- www.swift.org

Linux 배포판

- archive.ubuntu.com
- security.ubuntu.com
- ubuntu.com
- www.ubuntu.com

- *.ubuntu.com
- ppa.launchpad.net
- launchpad.net
- www.launchpad.net

개발 도구 및 플랫폼

- dl.k8s.io (Kubernetes)
- pkgs.k8s.io
- k8s.io
- www.k8s.io
- releases.hashicorp.com (HashiCorp)
- apt.releases.hashicorp.com
- rpm.releases.hashicorp.com
- archive.releases.hashicorp.com
- hashicorp.com
- www.hashicorp.com
- repo.anaconda.com (Anaconda/Conda)
- conda.anaconda.org
- anaconda.org
- www.anaconda.com
- anaconda.com
- continuum.io
- apache.org (Apache)
- www.apache.org
- archive.apache.org
- downloads.apache.org
- eclipse.org (Eclipse)
- www.eclipse.org
- download.eclipse.org
- nodejs.org (Node.js)
- www.nodejs.org

클라우드 서비스 및 모니터링

- statsig.com

- www.statsig.com
- api.statsig.com
- sentry.io
- *.sentry.io
- http-intake.logs.datadoghq.com
- *.datadoghq.com
- *.datadoghq.eu

콘텐츠 전달 및 미러

- sourceforge.net
- *.sourceforge.net
- packagecloud.io
- *.packagecloud.io

스키마 및 구성

- json-schema.org
- www.json-schema.org
- json.schemastore.org
- www.schemastore.org

Model Context Protocol

- *.modelcontextprotocol.io

> **Note:**
>
> `*` 로 표시된 도메인은 와일드카드 하위 도메인 일치를 나타냅니다. 예를 들어 `*.gcr.io` 는 `gcr.io` 의 모든 하위 도메인에 대한 액세스를 허용합니다.

사용자 정의 네트워크 액세스에 대한 보안 모범 사례

1. **최소 권한 원칙**: 필요한 최소 네트워크 액세스만 활성화합니다
2. **정기적으로 감사**: 허용된 도메인을 정기적으로 검토합니다
3. **HTTPS 사용**: 항상 HTTP 끝점보다 HTTPS 끝점을 선호합니다

보안 및 격리

웹에서 Claude Code는 강력한 보안 보장을 제공합니다:

- **격리된 가상 머신**: 각 세션은 격리된 Anthropic 관리 VM에서 실행됩니다
- **네트워크 액세스 제어**: 네트워크 액세스는 기본적으로 제한되며 비활성화할 수 있습니다

> **Note:**
>
> 네트워크 액세스가 비활성화된 상태에서 실행할 때 Claude Code는 Anthropic API와 통신할 수 있으며, 이는 여전히 격리된 Claude Code VM에서 데이터가 나갈 수 있습니다.

- **자격 증명 보호**: 민감한 자격 증명(예: git 자격 증명 또는 서명 키)은 Claude Code가 있는 샌드박스 내부에 없습니다. 인증은 범위 자격 증명을 사용하는 보안 프록시를 통해 처리됩니다
- **안전한 분석**: 코드는 PR을 생성하기 전에 격리된 VM 내에서 분석 및 수정됩니다

가격 및 속도 제한

웹에서 Claude Code는 계정 내의 다른 모든 Claude 및 Claude Code 사용과 속도 제한을 공유합니다. 여러 작업을 병렬로 실행하면 비례적으로 더 많은 속도 제한을 소비합니다.

제한 사항

- **저장소 인증**: 웹에서 로컬로 세션을 이동할 때 동일한 계정으로 인증된 경우에만 가능합니다
- **플랫폼 제한**: 웹에서 Claude Code는 GitHub에서 호스팅되는 코드에서만 작동합니다. GitLab 및 기타 비 GitHub 저장소는 클라우드 세션에서 사용할 수 없습니다

모범 사례

1. **환경 설정 자동화**: 설정 스크립트를 사용하여 Claude Code가 시작되기 전에 종속성을 설치하고 도구를 구성합니다. 더 고급 시나리오의 경우 SessionStart hooks를 구성합니다.
2. **요구 사항 문서화**: `CLAUDE.md` 파일에서 종속성 및 명령을 명확하게 지정합니다. `AGENTS.md` 파일이 있는 경우 `@AGENTS.md` 를 사용하여 `CLAUDE.md` 에서 소싱하여 단일 정보 소스를 유지할 수 있습니다.

관련 리소스

- Hooks 구성
- 설정 참조
- 보안
- 데이터 사용

Part 9: Security & Privacy

보안

> Claude Code의 보안 보호 기능과 안전한 사용을 위한 모범 사례에 대해 알아봅니다.

보안에 대한 우리의 접근 방식

보안 기초

코드의 보안은 매우 중요합니다. Claude Code는 보안을 핵심으로 구축되었으며, Anthropic의 포괄적인 보안 프로그램에 따라 개발되었습니다. Anthropic Trust Center에서 자세히 알아보고 리소스(SOC 2 Type 2 보고서, ISO 27001 인증서 등)에 접근할 수 있습니다.

권한 기반 아키텍처

Claude Code는 기본적으로 엄격한 읽기 전용 권한을 사용합니다. 추가 작업이 필요한 경우(파일 편집, 테스트 실행, 명령 실행), Claude Code는 명시적 권한을 요청합니다. 사용자는 작업을 한 번만 승인할지 또는 자동으로 허용할지 제어할 수 있습니다.

Claude Code는 투명하고 안전하도록 설계되었습니다. 예를 들어, bash 명령을 실행하기 전에 승인을 요구하여 직접 제어할 수 있습니다. 이 접근 방식을 통해 사용자와 조직은 권한을 직접 구성할 수 있습니다.

자세한 권한 구성은 Permissions를 참조하십시오.

기본 제공 보호

에이전트 시스템의 위험을 완화하기 위해:

- **샌드박스 bash 도구**: Sandbox를 사용하여 bash 명령을 파일 시스템 및 네트워크 격리로 실행하여 권한 프롬프트를 줄이면서 보안을 유지합니다. `/sandbox` 를 사용하여 Claude Code가 자율적으로 작업할 수 있는 경계를 정의하도록 활성화합니다.
- **쓰기 액세스 제한**: Claude Code는 시작된 폴더와 그 하위 폴더에만 쓸 수 있으며, 명시적 권한 없이 상위 디렉토리의 파일을 수정할 수 없습니다. Claude Code는 작업 디렉토리 외부의 파일을 읽을 수 있지만(시스템 라이브러리 및 종속성에 액세스하는 데 유용함), 쓰기 작업은 프로젝트 범위로 엄격히 제한되어 명확한 보안 경계를 만듭니다.
- **프롬프트 피로 완화**: 사용자별, 코드베이스별 또는 조직별로 자주 사용되는 안전한 명령을 허용 목록에 추가하는 지원

- **Accept Edits 모드**: 여러 편집을 일괄 수락하면서 부작용이 있는 명령에 대한 권한 프롬프트를 유지합니다.

사용자 책임

Claude Code는 사용자가 부여한 권한만 가집니다. 승인 전에 제안된 코드와 명령의 안전성을 검토할 책임이 있습니다.

프롬프트 주입으로부터 보호

프롬프트 주입은 공격자가 악의적인 텍스트를 삽입하여 AI 어시스턴트의 지시사항을 무시하거나 조작하려는 기법입니다. Claude Code는 이러한 공격에 대한 여러 보호 기능을 포함합니다:

핵심 보호

- **권한 시스템**: 민감한 작업에는 명시적 승인이 필요합니다.
- **컨텍스트 인식 분석**: 전체 요청을 분석하여 잠재적으로 해로운 지시사항을 감지합니다.
- **입력 살균**: 사용자 입력을 처리하여 명령 주입을 방지합니다.
- **명령 차단 목록**: `curl` 및 `wget`과 같이 웹에서 임의의 콘텐츠를 가져오는 위험한 명령을 기본적으로 차단합니다. 명시적으로 허용된 경우 권한 패턴 제한을 인식하십시오.

개인정보 보호 장치

데이터를 보호하기 위해 다음을 포함한 여러 보호 기능을 구현했습니다:

- 민감한 정보에 대한 제한된 보관 기간(Privacy Center에서 자세히 알아보기)
- 사용자 세션 데이터에 대한 제한된 액세스
- 데이터 학습 기본 설정에 대한 사용자 제어. 소비자 사용자는 언제든지 개인정보 보호 설정을 변경할 수 있습니다.

전체 세부 사항은 Commercial Terms of Service(Team, Enterprise 및 API 사용자용) 또는 Consumer Terms(Free, Pro 및 Max 사용자용) 및 Privacy Policy를 검토하십시오.

추가 보호 기능

- **네트워크 요청 승인**: 네트워크 요청을 하는 도구는 기본적으로 사용자 승인이 필요합니다.
- **격리된 컨텍스트 윈도우**: 웹 가져오기는 별도의 컨텍스트 윈도우를 사용하여 잠재적으로 악의적인 프롬프트 주입을 방지합니다.
- **신뢰 확인**: 첫 번째 코드베이스 실행 및 새 MCP 서버는 신뢰 확인이 필요합니다.
  - 참고: `-p` 플래그를 사용하여 비대화형으로 실행할 때 신뢰 확인이 비활성화됩니다.
- **명령 주입 감지**: 의심스러운 bash 명령은 이전에 허용 목록에 있었더라도 수동 승인이 필요합니다.

- **폐쇄형 매칭 실패**: 일치하지 않는 명령은 기본적으로 수동 승인이 필요합니다.
- **자연어 설명**: 복잡한 bash 명령에는 사용자 이해를 위한 설명이 포함됩니다.
- **보안 자격증명 저장소**: API 키 및 토큰은 암호화됩니다. Credential Management를 참조하십시오.

> **Warning:**
>
> **Windows WebDAV 보안 위험**: Windows에서 Claude Code를 실행할 때 WebDAV를 활성화하거나 Claude Code가 WebDAV 하위 디렉토리를 포함할 수 있는 `\\*` 와 같은 경로에 액세스하도록 허용하지 않는 것이 좋습니다. WebDAV는 Microsoft에서 보안 위험으로 인해 더 이상 사용되지 않습니다. WebDAV를 활성화하면 Claude Code가 원격 호스트에 대한 네트워크 요청을 트리거하여 권한 시스템을 우회할 수 있습니다.

신뢰할 수 없는 콘텐츠로 작업하기 위한 모범 사례:

1. 승인 전에 제안된 명령 검토
2. 신뢰할 수 없는 콘텐츠를 Claude에 직접 파이프하지 않기
3. 중요한 파일에 대한 제안된 변경 사항 확인
4. 가상 머신(VM)을 사용하여 스크립트를 실행하고 도구 호출을 수행합니다. 특히 외부 웹 서비스와 상호 작용할 때
5. `/bug` 를 사용하여 의심스러운 동작 보고

> **Warning:**
>
> 이러한 보호 기능이 위험을 크게 줄이지만, 모든 공격에 완전히 면역인 시스템은 없습니다. 모든 AI 도구로 작업할 때 항상 좋은 보안 관행을 유지하십시오.

MCP 보안

Claude Code를 사용하면 사용자가 Model Context Protocol(MCP) 서버를 구성할 수 있습니다. 허용된 MCP 서버 목록은 소스 코드에서 구성되며, Claude Code 설정의 일부로 엔지니어가 소스 제어에 체크인합니다.

자신의 MCP 서버를 작성하거나 신뢰하는 제공자의 MCP 서버를 사용할 것을 권장합니다.
Claude Code 권한을 MCP 서버에 대해 구성할 수 있습니다. Anthropic은 MCP 서버를 관리하거나 감사하지 않습니다.

IDE 보안

IDE에서 Claude Code를 실행하는 방법에 대한 자세한 내용은 VS Code 보안 및 개인정보 보호를 참조하십시오.

클라우드 실행 보안

웹에서 Claude Code를 사용할 때 추가 보안 제어가 적용됩니다:

- **격리된 가상 머신**: 각 클라우드 세션은 격리된 Anthropic 관리 VM에서 실행됩니다.
- **네트워크 액세스 제어**: 네트워크 액세스는 기본적으로 제한되며 비활성화되거나 특정 도메인만 허용하도록 구성할 수 있습니다.
- **자격증명 보호**: 인증은 샌드박스 내에서 범위가 지정된 자격증명을 사용하는 보안 프록시를 통해 처리되며, 이는 실제 GitHub 인증 토큰으로 변환됩니다.
- **분기 제한**: Git 푸시 작업은 현재 작업 분기로 제한됩니다.
- **감사 로깅**: 클라우드 환경의 모든 작업은 규정 준수 및 감사 목적으로 기록됩니다.
- **자동 정리**: 클라우드 환경은 세션 완료 후 자동으로 종료됩니다.

클라우드 실행에 대한 자세한 내용은 Claude Code on the web을 참조하십시오.

Remote Control 세션은 다르게 작동합니다: 웹 인터페이스는 로컬 머신에서 실행 중인 Claude Code 프로세스에 연결됩니다. 모든 코드 실행 및 파일 액세스는 로컬에 유지되며, 모든 로컬 Claude Code 세션 중에 흐르는 동일한 데이터는 TLS를 통해 Anthropic API를 통해 이동합니다. 클라우드 VM 또는 샌드박싱이 관련되지 않습니다. 연결은 각각 특정 목적으로 제한되고 독립적으로 만료되는 여러 단기 범위 자격증명을 사용하여 손상된 단일 자격증명의 영향 범위를 제한합니다.

보안 모범 사례

민감한 코드로 작업

- 승인 전에 제안된 모든 변경 사항 검토
- 민감한 저장소에 프로젝트별 권한 설정 사용
- 추가 격리를 위해 devcontainers 사용 고려
- `/permissions`를 사용하여 권한 설정을 정기적으로 감사합니다.

팀 보안

- managed settings를 사용하여 조직 표준 적용
- 버전 제어를 통해 승인된 권한 구성 공유
- 팀 구성원에게 보안 모범 사례 교육

- OpenTelemetry metrics를 통해 Claude Code 사용 모니터링
- `ConfigChange` hooks를 사용하여 세션 중 설정 변경 감사 또는 차단

보안 문제 보고

Claude Code에서 보안 취약점을 발견한 경우:

1. 공개적으로 공개하지 마십시오.
2. HackerOne 프로그램을 통해 보고합니다.
3. 자세한 재현 단계 포함
4. 공개 공개 전에 문제를 해결할 시간을 허용합니다.

관련 리소스

- Sandboxing - bash 명령에 대한 파일 시스템 및 네트워크 격리
- Permissions - 권한 및 액세스 제어 구성
- Monitoring usage - Claude Code 활동 추적 및 감사
- Development containers - 보안, 격리된 환경
- Anthropic Trust Center - 보안 인증 및 규정 준수

샌드박싱

> Claude Code의 샌드박싱된 bash 도구가 파일시스템 및 네트워크 격리를 제공하여 더 안전하고 자율적인 에이전트 실행을 가능하게 하는 방법을 알아봅니다.

개요

Claude Code는 에이전트 실행을 위한 더 안전한 환경을 제공하고 지속적인 권한 프롬프트의 필요성을 줄이기 위해 기본 샌드박싱 기능을 제공합니다. 각 bash 명령에 대해 권한을 요청하는 대신, 샌드박싱은 Claude Code가 위험을 줄이면서 더 자유롭게 작업할 수 있는 정의된 경계를 미리 생성합니다.

샌드박싱된 bash 도구는 OS 수준의 기본 요소를 사용하여 파일시스템 및 네트워크 격리를 모두 적용합니다.

샌드박싱이 중요한 이유

기존의 권한 기반 보안은 bash 명령에 대한 지속적인 사용자 승인이 필요합니다. 이는 제어를 제공하지만 다음과 같은 문제를 야기할 수 있습니다:

- **승인 피로**: "승인" 버튼을 반복적으로 클릭하면 사용자가 승인하는 내용에 덜 주의를 기울이게 될 수 있습니다
- **생산성 감소**: 지속적인 중단으로 인해 개발 워크플로우가 느려집니다
- **제한된 자율성**: Claude Code는 승인을 기다릴 때 효율적으로 작업할 수 없습니다

샌드박싱은 다음과 같은 방식으로 이러한 문제를 해결합니다:

1. **명확한 경계 정의**: Claude Code가 액세스할 수 있는 정확한 디렉토리 및 네트워크 호스트를 지정합니다
2. **권한 프롬프트 감소**: 샌드박스 내의 안전한 명령은 승인이 필요하지 않습니다
3. **보안 유지**: 샌드박스 외부의 리소스에 액세스하려는 시도는 즉시 알림을 트리거합니다
4. **자율성 활성화**: Claude Code는 정의된 제한 내에서 더 독립적으로 실행할 수 있습니다

> **Warning:**

> 효과적인 샌드박싱은 **파일시스템 및 네트워크 격리 모두**를 필요로 합니다. 네트워크 격리가 없으면 손상된 에이전트가 SSH 키와 같은 민감한 파일을 유출할 수 있습니다. 파일시스템 격리가 없으면 손상된 에이전트가 시스템 리소스를 백도어하여 네트워크 액세스를 얻을 수 있습니다. 샌드박싱을 구성할 때 구성된 설정이 이러한 시스템에서 우회를 생성하지 않도록 하는 것이 중요합니다.

작동 방식

파일시스템 격리

샌드박싱된 bash 도구는 파일 시스템 액세스를 특정 디렉토리로 제한합니다:

- **기본 쓰기 동작**: 현재 작업 디렉토리 및 그 하위 디렉토리에 대한 읽기 및 쓰기 액세스
- **기본 읽기 동작**: 특정 거부된 디렉토리를 제외한 전체 컴퓨터에 대한 읽기 액세스
- **차단된 액세스**: 명시적 권한 없이 현재 작업 디렉토리 외부의 파일을 수정할 수 없습니다
- **구성 가능**: 설정을 통해 사용자 정의 허용 및 거부 경로를 정의합니다

설정에서 `sandbox.filesystem.allowWrite`를 사용하여 추가 경로에 대한 쓰기 액세스를 부여할 수 있습니다. 이러한 제한은 OS 수준(macOS의 Seatbelt, Linux의 bubblewrap)에서 적용되므로 Claude의 파일 도구뿐만 아니라 `kubectl`, `terraform`, `npm`과 같은 도구를 포함한 모든 하위 프로세스 명령에 적용됩니다.

네트워크 격리

네트워크 액세스는 샌드박스 외부에서 실행되는 프록시 서버를 통해 제어됩니다:

- **도메인 제한**: 승인된 도메인만 액세스할 수 있습니다
- **사용자 확인**: 새 도메인 요청은 권한 프롬프트를 트리거합니다(`allowManagedDomainsOnly`가 활성화된 경우 제외, 이는 허용되지 않은 도메인을 자동으로 차단합니다)
- **사용자 정의 프록시 지원**: 고급 사용자는 나가는 트래픽에 대한 사용자 정의 규칙을 구현할 수 있습니다
- **포괄적 범위**: 제한은 명령으로 생성된 모든 스크립트, 프로그램 및 하위 프로세스에 적용됩니다

OS 수준 적용

샌드박싱된 bash 도구는 운영 체제 보안 기본 요소를 활용합니다:

- **macOS**: 샌드박스 적용을 위해 Seatbelt를 사용합니다
- **Linux**: 격리를 위해 bubblewrap을 사용합니다
- **WSL2**: Linux와 동일하게 bubblewrap을 사용합니다

WSL1은 bubblewrap이 WSL2에서만 사용 가능한 커널 기능을 필요로 하기 때문에 지원되지 않습니다.

이러한 OS 수준의 제한은 Claude Code의 명령으로 생성된 모든 자식 프로세스가 동일한 보안 경계를 상속하도록 보장합니다.

시작하기

필수 조건

macOS에서는 기본 제공 Seatbelt 프레임워크를 사용하여 샌드박싱이 기본적으로 작동합니다.

Linux 및 WSL2에서는 먼저 필수 패키지를 설치합니다:

Ubuntu/Debian

```
sudo apt-get install bubblewrap socat
```

Fedora

```
sudo dnf install bubblewrap socat
```

샌드박싱 활성화

`/sandbox` 명령을 실행하여 샌드박싱을 활성화할 수 있습니다:

```
/sandbox
```

이는 샌드박스 모드 중에서 선택할 수 있는 메뉴를 엽니다. Linux에서 `bubblewrap` 또는 `socat`과 같은 필수 종속성이 누락된 경우 메뉴에 플랫폼에 대한 설치 지침이 표시됩니다.

샌드박스 모드

Claude Code는 두 가지 샌드박스 모드를 제공합니다:

자동 허용 모드: Bash 명령은 샌드박스 내에서 실행을 시도하며 권한 없이 자동으로 허용됩니다. 샌드박싱할 수 없는 명령(허용되지 않은 호스트에 대한 네트워크 액세스가 필요한 경우 등)은 일반 권한 흐름으로 폴백됩니다. 구성한 명시적 요청/거부 규칙은 항상 존중됩니다.

일반 권한 모드: 모든 bash 명령은 샌드박싱되었더라도 표준 권한 흐름을 거칩니다. 이는 더 많은 제어를 제공하지만 더 많은 승인이 필요합니다.

두 모드 모두에서 샌드박스는 동일한 파일시스템 및 네트워크 제한을 적용합니다. 차이점은 샌드박싱된 명령이 자동 승인되는지 또는 명시적 권한이 필요한지 여부뿐입니다.

> **Info:**
>
> 자동 허용 모드는 권한 모드 설정과 독립적으로 작동합니다. "편집 수락" 모드에 있지 않더라도 자동 허용이 활성화되면 샌드박싱된 bash 명령이 자동으로 실행됩니다. 이는 샌드박스 경계 내에서 파일을 수정하는 bash 명령이 파일 편집 도구가 일반적으로 승인을 요구할 때도 프롬프트 없이 실행됨을 의미합니다.

샌드박싱 구성

`settings.json` 파일을 통해 샌드박스 동작을 사용자 정의합니다. 전체 구성 참조는 설정을 참조하세요.

특정 경로에 대한 하위 프로세스 쓰기 액세스 부여

기본적으로 샌드박싱된 명령은 현재 작업 디렉토리에만 쓸 수 있습니다. `kubectl`, `terraform` 또는 `npm`과 같은 하위 프로세스 명령이 프로젝트 디렉토리 외부에 쓰기해야 하는 경우 `sandbox.filesystem.allowWrite`를 사용하여 특정 경로에 대한 액세스를 부여합니다:

```
{
  "sandbox": {
    "enabled": true,
    "filesystem": {
      "allowWrite": ["~/.kube", "//tmp/build"]
    }
  }
}
```

이러한 경로는 OS 수준에서 적용되므로 샌드박스 내에서 실행되는 모든 명령(자식 프로세스 포함)이 이를 존중합니다. 이는 도구를 `excludedCommands`로 샌드박스에서 완전히 제외하는 것보다 도구가 특정 위치에 쓰기 액세스가 필요할 때 권장되는 방법입니다.

`allowWrite`(또는 `denyWrite` / `denyRead`)가 여러 설정 범위에서 정의된 경우 배열이 **병합**됩니다. 즉, 모든 범위의 경로가 결합되며 대체되지 않습니다. 예를 들어 관리 설정이 `//opt/company-tools`에 대한 쓰기를 허용하고 사용자가 개인 설정에서 `~/.kube`를 추가하면 두 경로 모두 최종 샌드박스 구성에 포함됩니다. 이는 사용자와 프로젝트가 더 높은 우선순위 범위에서 설정한 경로를 복제하거나 재정의하지 않고 목록을 확장할 수 있음을 의미합니다.

경로 접두사는 경로가 해석되는 방식을 제어합니다:

접두사	의미	예시
`//`	파일시스템 루트의 절대 경로	`//tmp/build` 는 `/tmp/build` 가 됩니다
`~/`	홈 디렉토리에 상대적	`~/.kube` 는 `$HOME/.kube` 가 됩니다
`/`	설정 파일의 디렉토리에 상대적	`/build` 는 `$SETTINGS_DIR/build` 가 됩니다
`./` 또는 접두사 없음	상대 경로(샌드박스 런타임에 의해 해석됨)	`./output`

`sandbox.filesystem.denyWrite` 및 `sandbox.filesystem.denyRead` 를 사용하여 쓰기 또는 읽기 액세스를 거부할 수도 있습니다. 이들은 `Edit(...)` 및 `Read(...)` 권한 규칙의 모든 경로와 병합됩니다.

Tip:

모든 명령이 기본적으로 샌드박싱과 호환되는 것은 아닙니다. 샌드박스를 최대한 활용하는 데 도움이 될 수 있는 몇 가지 참고 사항:

- 많은 CLI 도구는 특정 호스트에 액세스해야 합니다. 이러한 도구를 사용하면서 특정 호스트에 액세스할 권한을 요청합니다. 권한을 부여하면 지금과 앞으로 이러한 호스트에 액세스할 수 있으므로 샌드박스 내에서 안전하게 실행할 수 있습니다.
- `watchman` 은 샌드박스에서 실행하는 것과 호환되지 않습니다. `jest` 를 실행 중인 경우 `jest --no-watchman` 사용을 고려하세요
- `docker` 는 샌드박스에서 실행하는 것과 호환되지 않습니다. `excludedCommands` 에서 `docker` 를 지정하여 샌드박스 외부에서 실행하도록 강제하는 것을 고려하세요.

Note:

Claude Code는 필요할 때 명령이 샌드박스 외부에서 실행될 수 있도록 하는 의도적인 탈출 해치 메커니즘을 포함합니다. 명령이 샌드박스 제한으로 인해 실패할 때(예: 네트워크 연결 문제 또는 호환되지 않는 도구), Claude는 실패를 분석하도록 프롬프트되며 `dangerouslyDisableSandbox` 매개변수로 명령을 다시 시도할 수 있습니다. 이 매개변수를 사용하는 명령은 실행을 위해 사용자 권한이 필요한 일반 Claude Code 권한 흐름을 거칩니다. 이를 통해 Claude Code는 특정 도구 또는 네트워크 작업이 샌드박스 제약 내에서 작동할 수 없는 경우를 처리할 수 있습니다.

샌드박스 설정에서 `"allowUnsandboxedCommands": false` 를 설정하여 이 탈출 해치를 비활성화할 수 있습니다. 비활성화되면 `dangerouslyDisableSandbox` 매개변수가 완전히 무시되고 모든 명령은 샌드박싱되거나 `excludedCommands` 에 명시적으로 나열되어야 합니다.

보안 이점

프롬프트 주입으로부터의 보호

공격자가 프롬프트 주입을 통해 Claude Code의 동작을 성공적으로 조작하더라도 샌드박스는 시스템이 안전하게 유지되도록 보장합니다:

파일시스템 보호:

- `~/.bashrc` 와 같은 중요한 구성 파일을 수정할 수 없습니다
- `/bin/` 의 시스템 수준 파일을 수정할 수 없습니다
- Claude 권한 설정에서 거부된 파일을 읽을 수 없습니다

네트워크 보호:

- 공격자가 제어하는 서버로 데이터를 유출할 수 없습니다
- 승인되지 않은 도메인에서 악성 스크립트를 다운로드할 수 없습니다
- 승인되지 않은 서비스에 예상치 못한 API 호출을 할 수 없습니다
- 명시적으로 허용된 도메인이 아닌 다른 도메인에 연락할 수 없습니다

모니터링 및 제어:

- 샌드박스 외부의 모든 액세스 시도는 OS 수준에서 차단됩니다
- 경계가 테스트될 때 즉시 알림을 받습니다
- 요청을 거부하거나, 한 번만 허용하거나, 구성을 영구적으로 업데이트하도록 선택할 수 있습니다

공격 표면 감소

샌드박싱은 다음으로 인한 잠재적 피해를 제한합니다:

- **악성 종속성**: 해로운 코드가 있는 NPM 패키지 또는 기타 종속성
- **손상된 스크립트**: 보안 취약점이 있는 빌드 스크립트 또는 도구
- **사회 공학**: 사용자를 속여 위험한 명령을 실행하도록 하는 공격
- **프롬프트 주입**: Claude를 속여 위험한 명령을 실행하도록 하는 공격

투명한 작동

Claude Code가 샌드박스 외부의 네트워크 리소스에 액세스하려고 시도할 때:

1. 작업이 OS 수준에서 차단됩니다
2. 즉시 알림을 받습니다
3. 다음을 선택할 수 있습니다:
    - 요청 거부

- 한 번만 허용
- 샌드박스 구성을 업데이트하여 영구적으로 허용

보안 제한 사항

- 네트워크 샌드박싱 제한: 네트워크 필터링 시스템은 프로세스가 연결할 수 있는 도메인을 제한하여 작동합니다. 프록시를 통과하는 트래픽을 검사하지 않으며 사용자는 정책에서 신뢰할 수 있는 도메인만 허용하도록 해야 합니다.

> **Warning:**
>
> 사용자는 데이터 유출을 허용할 수 있는 `github.com` 과 같은 광범위한 도메인을 허용하는 것과 관련된 잠재적 위험을 인식해야 합니다. 또한 경우에 따라 도메인 프론팅을 통해 네트워크 필터링을 우회할 수 있습니다.

- Unix 소켓을 통한 권한 상승: `allowUnixSockets` 구성은 실수로 샌드박스 우회로 이어질 수 있는 강력한 시스템 서비스에 대한 액세스를 부여할 수 있습니다. 예를 들어 `/var/run/docker.sock` 에 대한 액세스를 허용하는 데 사용되면 docker 소켓을 악용하여 호스트 시스템에 대한 액세스를 효과적으로 부여합니다. 사용자는 샌드박스를 통해 허용하는 모든 unix 소켓을 신중하게 고려하도록 권장됩니다.
- 파일시스템 권한 상승: 과도하게 광범위한 파일시스템 쓰기 권한은 권한 상승 공격을 가능하게 할 수 있습니다. `$PATH` 의 실행 파일을 포함하는 디렉토리, 시스템 구성 디렉토리 또는 사용자 셸 구성 파일(`.bashrc` , `.zshrc`)에 대한 쓰기를 허용하면 다른 사용자 또는 시스템 프로세스가 이러한 파일에 액세스할 때 다른 보안 컨텍스트에서 코드 실행으로 이어질 수 있습니다.
- Linux 샌드박스 강도: Linux 구현은 강력한 파일시스템 및 네트워크 격리를 제공하지만 권한 있는 네임스페이스 없이 Docker 환경 내에서 작동할 수 있도록 하는 `enableWeakerNestedSandbox` 모드를 포함합니다. 이 옵션은 보안을 상당히 약화시키며 추가 격리가 다른 방식으로 적용되는 경우에만 사용해야 합니다.

샌드박싱이 권한과 어떻게 관련되는지

샌드박싱과 권한은 함께 작동하는 상호 보완적인 보안 계층입니다:

- **권한**은 Claude Code가 사용할 수 있는 도구를 제어하며 도구가 실행되기 전에 평가됩니다. 이들은 모든 도구에 적용됩니다: Bash, Read, Edit, WebFetch, MCP 등.
- **샌드박싱**은 Bash 명령이 파일시스템 및 네트워크 수준에서 액세스할 수 있는 것을 제한하는 OS 수준의 적용을 제공합니다. Bash 명령 및 그 자식 프로세스에만 적용됩니다.

파일시스템 및 네트워크 제한은 샌드박스 설정 및 권한 규칙을 통해 모두 구성됩니다:

- `sandbox.filesystem.allowWrite`를 사용하여 작업 디렉토리 외부의 경로에 대한 하위 프로세스 쓰기 액세스를 부여합니다
- `sandbox.filesystem.denyWrite` 및 `sandbox.filesystem.denyRead`를 사용하여 특정 경로에 대한 하위 프로세스 액세스를 차단합니다
- `Read` 및 `Edit` 거부 규칙을 사용하여 특정 파일 또는 디렉토리에 대한 액세스를 차단합니다
- `WebFetch` 허용/거부 규칙을 사용하여 도메인 액세스를 제어합니다
- 샌드박스 `allowedDomains`를 사용하여 Bash 명령이 도달할 수 있는 도메인을 제어합니다

`sandbox.filesystem` 설정 및 권한 규칙의 경로는 최종 샌드박스 구성으로 병합됩니다.

이 저장소에는 샌드박스 관련 예제를 포함한 일반적인 배포 시나리오에 대한 시작 설정 구성이 포함되어 있습니다. 이를 시작점으로 사용하고 필요에 맞게 조정합니다.

고급 사용

사용자 정의 프록시 구성

고급 네트워크 보안이 필요한 조직의 경우 사용자 정의 프록시를 구현하여 다음을 수행할 수 있습니다:

- HTTPS 트래픽 복호화 및 검사
- 사용자 정의 필터링 규칙 적용
- 모든 네트워크 요청 로깅
- 기존 보안 인프라와 통합

```
{
  "sandbox": {
    "network": {
      "httpProxyPort": 8080,
      "socksProxyPort": 8081
    }
  }
}
```

기존 보안 도구와의 통합

샌드박싱된 bash 도구는 다음과 함께 작동합니다:

- **권한 규칙**: 권한 설정과 결합하여 심층 방어를 수행합니다

- **개발 컨테이너**: devcontainers와 함께 사용하여 추가 격리를 수행합니다
- **엔터프라이즈 정책**: 관리 설정을 통해 샌드박스 구성을 적용합니다

모범 사례

1. **제한적으로 시작**: 최소 권한으로 시작하여 필요에 따라 확장합니다
2. **로그 모니터링**: 샌드박스 위반 시도를 검토하여 Claude Code의 필요 사항을 이해합니다
3. **환경별 구성 사용**: 개발 및 프로덕션 컨텍스트에 대해 다른 샌드박스 규칙을 사용합니다
4. **권한과 결합**: 포괄적인 보안을 위해 샌드박싱을 IAM 정책과 함께 사용합니다
5. **구성 테스트**: 샌드박스 설정이 합법적인 워크플로우를 차단하지 않는지 확인합니다

오픈 소스

샌드박스 런타임은 자신의 에이전트 프로젝트에서 사용할 수 있는 오픈 소스 npm 패키지로 제공됩니다. 이를 통해 더 넓은 AI 에이전트 커뮤니티가 더 안전하고 보안이 강화된 자율 시스템을 구축할 수 있습니다. 이를 사용하여 실행하려는 다른 프로그램을 샌드박싱할 수도 있습니다. 예를 들어 MCP 서버를 샌드박싱하려면 다음을 실행할 수 있습니다:

```
npx @anthropic-ai/sandbox-runtime <command-to-sandbox>
```

구현 세부 사항 및 소스 코드는 GitHub 저장소를 방문하세요.

제한 사항

- **성능 오버헤드**: 최소이지만 일부 파일시스템 작업이 약간 더 느릴 수 있습니다
- **호환성**: 특정 시스템 액세스 패턴이 필요한 일부 도구는 구성 조정이 필요할 수 있으며 샌드박스 외부에서 실행해야 할 수도 있습니다
- **플랫폼 지원**: macOS, Linux 및 WSL2를 지원합니다. WSL1은 지원되지 않습니다. 기본 Windows 지원이 계획되어 있습니다.

참고 항목

- 보안 - 포괄적인 보안 기능 및 모범 사례
- 권한 - 권한 구성 및 액세스 제어
- 설정 - 전체 구성 참조
- CLI 참조 - 명령줄 옵션

Checkpointing

> Claude의 편집을 자동으로 추적하고 원하지 않는 변경 사항에서 빠르게 복구하기 위해 되감기합니다.

Claude Code는 작업하면서 Claude의 파일 편집을 자동으로 추적하므로, 문제가 발생하면 변경 사항을 빠르게 실행 취소하고 이전 상태로 되돌릴 수 있습니다.

Checkpointing의 작동 방식

Claude와 함께 작업할 때, checkpointing은 각 편집 전에 코드의 상태를 자동으로 캡처합니다. 이 안전망을 통해 항상 이전 코드 상태로 돌아갈 수 있다는 것을 알면서 야심 찬 대규모 작업을 수행할 수 있습니다.

자동 추적

Claude Code는 파일 편집 도구로 수행된 모든 변경 사항을 추적합니다:

- 모든 사용자 프롬프트는 새로운 checkpoint를 생성합니다
- Checkpoint는 세션 전체에 걸쳐 유지되므로 재개된 대화에서 액세스할 수 있습니다
- 30일 후 세션과 함께 자동으로 정리됩니다(구성 가능)

변경 사항 되감기

`Esc` 두 번(`Esc` + `Esc`)을 누르거나 `/rewind` 명령을 사용하여 rewind 메뉴를 엽니다. 다음을 복구하도록 선택할 수 있습니다:

- **대화만**: 코드 변경 사항을 유지하면서 사용자 메시지로 되감기
- **코드만**: 대화를 유지하면서 파일 변경 사항 되돌리기
- **코드와 대화 모두**: 세션의 이전 지점으로 둘 다 복구

일반적인 사용 사례

Checkpoint는 다음과 같은 경우에 특히 유용합니다:

- **대안 탐색**: 시작점을 잃지 않으면서 다양한 구현 방식을 시도합니다
- **실수에서 복구**: 버그를 도입하거나 기능을 손상시킨 변경 사항을 빠르게 실행 취소합니다
- **기능 반복**: 작동하는 상태로 되돌릴 수 있다는 것을 알면서 변형을 실험합니다

제한 사항

Bash 명령 변경 사항이 추적되지 않음

Checkpointing은 bash 명령으로 수정된 파일을 추적하지 않습니다. 예를 들어, Claude Code가 다음을 실행하는 경우:

```
rm file.txt
mv old.txt new.txt
cp source.txt dest.txt
```

이러한 파일 수정 사항은 rewind를 통해 실행 취소할 수 없습니다. Claude의 파일 편집 도구를 통해 직접 수행된 파일 편집만 추적됩니다.

외부 변경 사항이 추적되지 않음

Checkpointing은 현재 세션 내에서 편집된 파일만 추적합니다. Claude Code 외부에서 수동으로 수행한 파일 변경 사항과 다른 동시 세션의 편집은 일반적으로 캡처되지 않습니다. 단, 현재 세션과 동일한 파일을 수정하는 경우는 예외입니다.

버전 관리의 대체가 아님

Checkpoint는 빠른 세션 수준의 복구를 위해 설계되었습니다. 영구적인 버전 기록 및 협업을 위해:

- 커밋, 분기 및 장기 기록을 위해 버전 관리(예: Git)를 계속 사용합니다
- Checkpoint는 적절한 버전 관리를 보완하지만 대체하지 않습니다
- Checkpoint를 “로컬 실행 취소”로, Git을 “영구 기록”으로 생각합니다

참고 항목

- Interactive mode - 키보드 단축키 및 세션 제어
- Built-in commands - `/rewind`를 사용하여 checkpoint에 액세스
- CLI reference - 명령줄 옵션

데이터 사용

> Anthropic의 Claude 데이터 사용 정책에 대해 알아봅니다

데이터 정책

데이터 학습 정책

소비자 사용자(Free, Pro, Max 플랜): 향후 Claude 모델 개선을 위해 데이터 사용을 허용할 수 있는 선택권을 제공합니다. 이 설정이 켜져 있을 때 Free, Pro, Max 계정의 데이터를 사용하여 새로운 모델을 학습합니다(이러한 계정에서 Claude Code를 사용할 때 포함).

상업용 사용자: (Team 및 Enterprise 플랜, API, 타사 플랫폼, Claude Gov)는 기존 정책을 유지합니다: Anthropic은 상업 약관에 따라 Claude Code로 전송된 코드 또는 프롬프트를 사용하여 생성형 모델을 학습하지 않습니다. 단, 고객이 모델 개선을 위해 데이터를 제공하기로 선택한 경우는 예외입니다(예: Developer Partner Program).

Development Partner Program

Development Partner Program을 통해 학습할 자료를 제공하는 방법에 명시적으로 옵트인하는 경우, 제공된 자료를 사용하여 모델을 학습할 수 있습니다. 조직 관리자는 조직에 대해 Development Partner Program에 명시적으로 옵트인할 수 있습니다. 이 프로그램은 Anthropic 자체 API에만 사용 가능하며 Bedrock 또는 Vertex 사용자는 이용할 수 없습니다.

`/bug` 명령을 사용한 피드백

`/bug` 명령을 사용하여 Claude Code에 대한 피드백을 보내기로 선택한 경우, 피드백을 사용하여 제품 및 서비스를 개선할 수 있습니다. `/bug` 를 통해 공유된 대화 기록은 5년 동안 보관됩니다.

세션 품질 설문조사

Claude Code에서 “Claude가 이 세션을 어떻게 수행하고 있나요?”라는 메시지가 표시될 때, 이 설문조사에 응답하면(“Dismiss” 선택 포함) 숫자 등급(1, 2, 3 또는 dismiss)만 기록됩니다. 이 설문조사의 일부로 대화 기록, 입력, 출력 또는 기타 세션 데이터를 수집하거나 저장하지 않습니다. 엄지손가락 위/아래 피드백이나 `/bug` 보고서와 달리, 이 세션 품질 설문조사는 간단한 제품 만족도 지표입니다. 이 설문조사에 대한 응답은 데이터 학습 선호도에 영향을 주지 않으며 AI 모델을 학습하는 데 사용될 수 없습니다.

이러한 설문조사를 비활성화하려면 `CLAUDE_CODE_DISABLE_FEEDBACK_SURVEY=1` 을 설정합니다. 타사 제공자(Bedrock, Vertex, Foundry)를 사용하거나 원격 측정이 비활성화된 경우에도 설문조사가 자동으로 비활성화됩니다.

데이터 보관

Anthropic은 계정 유형 및 선호도에 따라 Claude Code 데이터를 보관합니다.

소비자 사용자(Free, Pro, Max 플랜):

- 모델 개선을 위한 데이터 사용을 허용하는 사용자: 모델 개발 및 안전 개선을 지원하기 위한 5년 보관 기간
- 모델 개선을 위한 데이터 사용을 허용하지 않는 사용자: 30일 보관 기간
- 개인정보 보호 설정은 claude.ai/settings/data-privacy-controls에서 언제든지 변경할 수 있습니다.

상업용 사용자(Team, Enterprise, API):

- 표준: 30일 보관 기간
- Zero data retention: Claude for Enterprise의 Claude Code에서 사용 가능합니다. ZDR은 조직별로 활성화되며, 각 새로운 조직은 계정 팀에서 별도로 ZDR을 활성화해야 합니다.
- 로컬 캐싱: Claude Code 클라이언트는 세션 재개를 활성화하기 위해 최대 30일 동안 세션을 로컬에 저장할 수 있습니다(구성 가능).

웹에서 개별 Claude Code 세션을 언제든지 삭제할 수 있습니다. 세션을 삭제하면 세션의 이벤트 데이터가 영구적으로 제거됩니다. 세션 삭제 방법에 대한 지침은 세션 관리를 참조하세요.

Privacy Center에서 데이터 보관 관행에 대해 자세히 알아보세요.

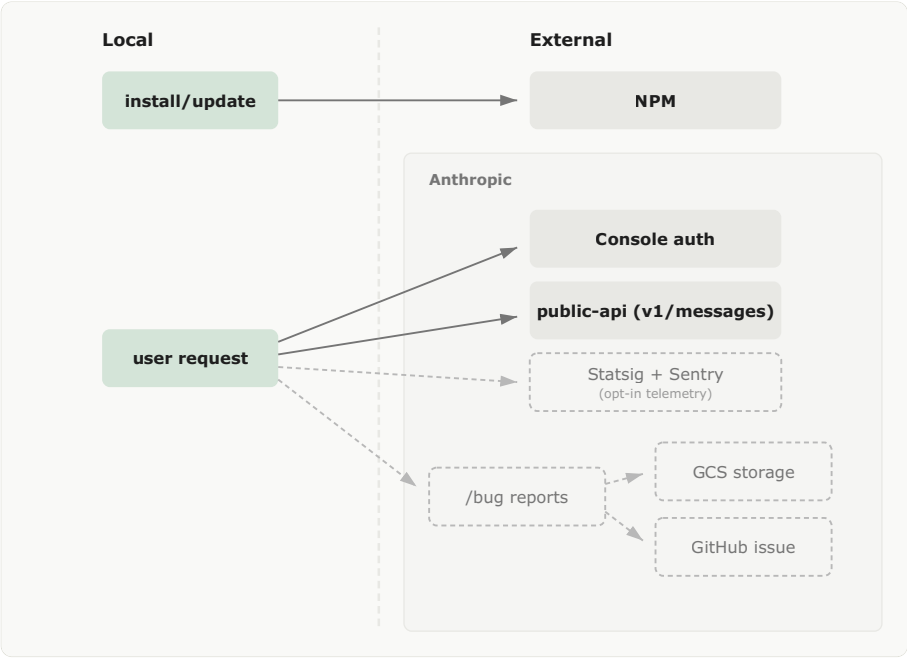
전체 세부 사항은 Commercial Terms of Service(Team, Enterprise, API 사용자용) 또는 Consumer Terms(Free, Pro, Max 사용자용) 및 Privacy Policy를 검토하세요.

데이터 액세스

모든 자체 플랫폼 사용자의 경우, 로컬 Claude Code 및 원격 Claude Code에 대해 기록되는 데이터에 대해 자세히 알아볼 수 있습니다. Remote Control 세션은 모든 실행이 사용자의 머신에서 발생하므로 로컬 데이터 흐름을 따릅니다. 원격 Claude Code의 경우 Claude는 Claude Code 세션을 시작한 저장소에 액세스합니다. Claude는 연결했지만 세션을 시작하지 않은 저장소에는 액세스하지 않습니다.

로컬 Claude Code: 데이터 흐름 및 종속성

아래 다이어그램은 설치 및 정상 작동 중에 Claude Code가 외부 서비스에 어떻게 연결되는지 보여줍니다. 실선은 필수 연결을 나타내고, 점선은 선택적 또는 사용자가 시작한 데이터 흐름을 나타냅니다.



Claude Code의 외부 연결을 보여주는 다이어그램: 설치/업데이트는 NPM에 연결되고, 사용자 요청은 Console auth, public-api, 그리고 선택적으로 Statsig, Sentry, 버그 보고를 포함한 Anthropic 서비스에 연결됩니다

Claude Code는 NPM에서 설치됩니다. Claude Code는 로컬에서 실행됩니다. LLM과 상호작용하기 위해 Claude Code는 네트워크를 통해 데이터를 전송합니다. 이 데이터에는 모든 사용자 프롬프트 및 모델 출력이 포함됩니다. 데이터는 TLS를 통해 전송 중에 암호화되며 저장 시에는 암호화되지 않습니다. Claude Code는 대부분의 인기 있는 VPN 및 LLM 프록시와 호환됩니다.

Claude Code는 Anthropic의 API를 기반으로 구축되었습니다. API 로깅 절차를 포함한 API의 보안 제어에 대한 자세한 내용은 Anthropic Trust Center에서 제공하는 규정 준수 아티팩트를 참조하세요.

클라우드 실행: 데이터 흐름 및 종속성

Claude Code on the web을 사용할 때, 세션은 로컬이 아닌 Anthropic 관리 가상 머신에서 실행됩니다. 클라우드 환경에서:

- **코드 및 데이터 저장소:** 저장소가 격리된 VM으로 복제됩니다. 코드 및 세션 데이터는 계정 유형에 대한 보관 및 사용 정책의 적용을 받습니다(위의 데이터 보관 섹션 참조).
- **자격 증명:** GitHub 인증은 보안 프록시를 통해 처리되며, GitHub 자격 증명은 샌드박스에 절대 입력되지 않습니다.

- **네트워크 트래픽:** 모든 아웃바운드 트래픽은 감사 로깅 및 악용 방지를 위해 보안 프록시를 통해 이동합니다.
- **세션 데이터:** 프롬프트, 코드 변경 및 출력은 로컬 Claude Code 사용과 동일한 데이터 정책을 따릅니다.

클라우드 실행의 보안 세부 사항은 Security를 참조하세요.

원격 측정 서비스

Claude Code는 사용자의 머신에서 Statsig 서비스에 연결하여 지연 시간, 안정성 및 사용 패턴과 같은 운영 메트릭을 기록합니다. 이 로깅에는 코드 또는 파일 경로가 포함되지 않습니다. 데이터는 TLS를 사용하여 전송 중에 암호화되고 256비트 AES 암호화를 사용하여 저장 시에 암호화됩니다. Statsig 보안 문서에서 자세히 알아보세요. Statsig 원격 측정을 거부하려면 `DISABLE_TELEMETRY` 환경 변수를 설정합니다.

Claude Code는 사용자의 머신에서 Sentry에 연결하여 운영 오류 로깅을 수행합니다. 데이터는 TLS를 사용하여 전송 중에 암호화되고 256비트 AES 암호화를 사용하여 저장 시에 암호화됩니다. Sentry 보안 문서에서 자세히 알아보세요. 오류 로깅을 거부하려면 `DISABLE_ERROR_REPORTING` 환경 변수를 설정합니다.

사용자가 `/bug` 명령을 실행하면 코드를 포함한 전체 대화 기록의 복사본이 Anthropic으로 전송됩니다. 데이터는 전송 중 및 저장 시에 암호화됩니다. 선택적으로 공개 저장소에 Github 이슈가 생성됩니다. 버그 보고를 거부하려면 `DISABLE_BUG_COMMAND` 환경 변수를 설정합니다.

API 제공자별 기본 동작

기본적으로 Bedrock, Vertex 또는 Foundry를 사용할 때 모든 필수가 아닌 트래픽(오류 보고, 원격 측정, 버그 보고 기능 및 세션 품질 설문조사 포함)을 비활성화합니다. `CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC` 환경 변수를 설정하여 이 모든 항목을 한 번에 거부할 수도 있습니다. 다음은 전체 기본 동작입니다:

서비스	Claude API	Vertex API	Bedrock API	Foundry API
Statsig (메트릭)	기본 켜짐. `DISABLE_TELEMETRY=1`로 비활성화합니다.	기본 꺼짐. `CLAUDE_CODE_USE_VERTEX`는 1이어야 합니다.	기본 꺼짐. `CLAUDE_CODE_USE_BEDROCK`은 1이어야 합니다.	기본 꺼짐. `CLAUDE_CODE_USE_FOUNDRY`는 1이어야 합니다.
Sentry (오류)	기본 켜짐. `DISABLE_ERROR_REPORTING=1`로 비활성화합니다.	기본 꺼짐. `CLAUDE_CODE_USE_VERTEX`는 1이어야 합니다.	기본 꺼짐. `CLAUDE_CODE_USE_BEDROCK`은 1이어야 합니다.	기본 꺼짐. `CLAUDE_CODE_USE_FOUNDRY`는 1이어야 합니다.

서비스	Claude API	Vertex API	Bedrock API	Foundry API
Claude API (`/bug` 보고)	기본 켜짐. `DISABLE_BUG_COMMAND=1` 로 비활성화합니다.	기본 꺼짐. `CLAUDE_CODE_USE_VERTEX` 는 1이어야 합니다.	기본 꺼짐. `CLAUDE_CODE_USE_BEDROCK` 은 1이어야 합니다.	기본 꺼짐. `CLAUDE_CODE_USE_FOUNDRY` 는 1이어야 합니다.
세션 품질 설문 조사	기본 켜짐. `CLAUDE_CODE_DISABLE_FEEDBACK_SURVEY=1` 로 비활성화합니다.	기본 꺼짐. `CLAUDE_CODE_USE_VERTEX` 는 1이어야 합니다.	기본 꺼짐. `CLAUDE_CODE_USE_BEDROCK` 은 1이어야 합니다.	기본 꺼짐. `CLAUDE_CODE_USE_FOUNDRY` 는 1이어야 합니다.

모든 환경 변수는 `settings.json` 에 체크인할 수 있습니다(자세히 알아보기).

법률 및 규정 준수

> Claude Code에 대한 법률 계약, 규정 준수 인증 및 보안 정보.

법률 계약

라이선스

Claude Code 사용은 다음의 적용을 받습니다:

- 상용 약관 - Team, Enterprise 및 Claude API 사용자용
- 소비자 약관 - Free, Pro 및 Max 사용자용

상용 계약

Claude API를 직접 사용하든(1P) AWS Bedrock 또는 Google Vertex를 통해 액세스하든(3P), 기존 상용 계약이 Claude Code 사용에 적용되며, 달리 상호 합의하지 않는 한 그렇습니다.

규정 준수

의료 규정 준수(BAA)

고객이 당사와 비즈니스 어소시에이트 계약(BAA)을 체결했고 Claude Code를 사용하려는 경우, 고객이 BAA를 체결했고 Zero Data Retention(ZDR)이 활성화되어 있으면 BAA가 자동으로 Claude Code를 포함하도록 확장됩니다. BAA는 Claude Code를 통해 흐르는 해당 고객의 API 트래픽에 적용됩니다.

보안 및 신뢰

신뢰 및 안전

Anthropic Trust Center 및 Transparency Hub에서 더 많은 정보를 찾을 수 있습니다.

보안 취약점 보고

Anthropic은 HackerOne을 통해 보안 프로그램을 관리합니다. 이 양식을 사용하여 취약점을 보고하세요.

Zero data retention

> Claude for Enterprise에서 Claude Code의 Zero Data Retention(ZDR)에 대해 알아보세요. 범위, 비활성화된 기능, 활성화 요청 방법을 포함합니다.

Zero Data Retention(ZDR)은 Claude for Enterprise를 통해 사용할 때 Claude Code에서 사용 가능합니다. ZDR이 활성화되면 Claude Code 세션 중에 생성된 프롬프트와 모델 응답은 실시간으로 처리되며 응답이 반환된 후 Anthropic에서 저장되지 않습니다. 단, 법률 준수 또는 오용 방지가 필요한 경우는 제외합니다.

Claude for Enterprise의 ZDR은 엔터프라이즈 고객에게 Zero Data Retention으로 Claude Code를 사용하고 관리 기능에 액세스할 수 있는 기능을 제공합니다:

- 사용자별 비용 제어
- 분석 대시보드
- 서버 관리 설정
- 감사 로그

Claude for Enterprise의 Claude Code에 대한 ZDR은 Anthropic의 직접 플랫폼에만 적용됩니다. AWS Bedrock, Google Vertex AI 또는 Microsoft Foundry의 Claude 배포의 경우 해당 플랫폼의 데이터 보존 정책을 참조하세요.

ZDR 범위

ZDR은 Claude for Enterprise의 Claude Code 추론을 포함합니다.

> **Warning:**
>
> ZDR은 조직별로 활성화됩니다. 각 새로운 조직은 Anthropic 계정 팀에서 별도로 ZDR을 활성화해야 합니다. ZDR은 동일한 계정 아래에 생성된 새로운 조직에 자동으로 적용되지 않습니다. 새로운 조직에 대해 ZDR을 활성화하려면 계정 팀에 문의하세요.

ZDR이 포함하는 것

ZDR은 Claude for Enterprise의 Claude Code를 통해 이루어진 모델 추론 호출을 포함합니다. 터미널에서 Claude Code를 사용할 때 전송하는 프롬프트와 Claude가 생성하는 응답은 Anthropic에서 보존되지 않습니다. 이는 사용되는 Claude 모델에 관계없이 적용됩니다.

ZDR이 포함하지 않는 것

ZDR은 ZDR이 활성화된 조직의 경우에도 다음을 포함하지 않습니다. 이러한 기능은 표준 데이터 보존 정책을 따릅니다:

기능	세부 정보
claude.ai의 채팅	Claude for Enterprise 웹 인터페이스를 통한 채팅 대화는 ZDR에 포함되지 않습니다.
Cowork	Cowork 세션은 ZDR에 포함되지 않습니다.
Claude Code 분석	프롬프트 또는 모델 응답을 저장하지 않지만 계정 이메일 및 사용 통계와 같은 생산성 메타데이터를 수집합니다. 기여도 메트릭은 ZDR 조직에서 사용할 수 없습니다. 분석 대시보드는 사용 메트릭만 표시합니다.
사용자 및 시트 관리	계정 이메일 및 시트 할당과 같은 관리 데이터는 표준 정책에 따라 보존됩니다.
타사 통합	타사 도구, MCP servers 또는 기타 외부 통합에서 처리한 데이터는 ZDR에 포함되지 않습니다. 해당 서비스의 데이터 처리 관행을 독립적으로 검토하세요.

ZDR에서 비활성화된 기능

Claude for Enterprise의 Claude Code 조직에 대해 ZDR이 활성화되면 프롬프트 또는 완성을 저장해야 하는 특정 기능이 백엔드 수준에서 자동으로 비활성화됩니다:

기능	이유
웹의 Claude Code	대화 기록의 서버 측 저장이 필요합니다.
Desktop 앱의 원격 세션	프롬프트 및 완성을 포함하는 지속적인 세션 데이터가 필요합니다.
피드백 제출(`/feedback`)	피드백을 제출하면 대화 데이터가 Anthropic으로 전송됩니다.

이러한 기능은 클라이언트 측 표시에 관계없이 백엔드에서 차단됩니다. 시작 중에 Claude Code 터미널에서 비활성화된 기능이 표시되면 이를 사용하려고 시도하면 조직의 정책이 해당 작업을 허용하지 않음을 나타내는 오류가 반환됩니다.

향후 기능도 프롬프트 또는 완성을 저장해야 하는 경우 비활성화될 수 있습니다.

정책 위반에 대한 데이터 보존

ZDR이 활성화된 경우에도 Anthropic은 법률에서 요구하거나 Usage Policy 위반을 해결하기 위해 필요한 경우 데이터를 보존할 수 있습니다. 세션이 정책 위반으로 플래그되면 Anthropic은 관련 입력 및 출력을 최대 2년 동안 보존할 수 있으며, 이는 Anthropic의 표준 ZDR 정책과 일치합니다.

ZDR 요청

Claude for Enterprise의 Claude Code에 대해 ZDR을 요청하려면 Anthropic 계정 팀에 문의하세요. 계정 팀이 내부적으로 요청을 제출하면 Anthropic이 적격성을 확인한 후 조직에서 ZDR을 검토하고 활성화합니다. 모든 활성화 작업은 감사 로그에 기록됩니다.

현재 종량제 API 키를 통해 Claude Code에 대해 ZDR을 사용 중인 경우 Claude for Enterprise로 전환하여 Claude Code에 대한 ZDR을 유지하면서 관리 기능에 액세스할 수 있습니다. 마이그레이션을 조정하려면 계정 팀에 문의하세요.

Part 10: Enterprise & Monitoring

팀 사용량을 분석으로 추적하기

> Claude Code 사용량 지표를 확인하고, 채택 현황을 추적하며, 분석 대시보드에서 엔지니어링 속도를 측정합니다.

Claude Code는 조직이 개발자 사용 패턴을 이해하고, 기여도 지표를 추적하며, Claude Code가 엔지니어링 속도에 미치는 영향을 측정할 수 있도록 분석 대시보드를 제공합니다. 귀사의 플랜에 맞는 대시보드에 접근하세요:

플랜	대시보드 URL	포함 사항	자세히 알아보기
Claude for Teams / Enterprise	claude.ai/analytics/claude-code	사용량 지표, GitHub 통합을 포함한 기여도 지표, 리더보드, 데이터 내보내기	세부 정보
API (Claude Console)	platform.claude.com/claude-code	사용량 지표, 지출 추적, 팀 인사이트	세부 정보

Teams 및 Enterprise를 위한 분석 접근

claude.ai/analytics/claude-code로 이동합니다. 관리자 및 소유자가 대시보드를 볼 수 있습니다.

Teams 및 Enterprise 대시보드에는 다음이 포함됩니다:

- **사용량 지표**: 수락된 코드 라인, 제안 수락률, 일일 활성 사용자 및 세션
- **기여도 지표**: Claude Code 지원으로 배포된 PR 및 코드 라인(GitHub 통합 포함)
- **리더보드**: Claude Code 사용량으로 순위가 매겨진 상위 기여자
- **데이터 내보내기**: 사용자 정의 보고를 위해 기여도 데이터를 CSV로 다운로드

기여도 지표 활성화

> **Note:**

> 기여도 지표는 공개 베타 상태이며 Claude for Teams 및 Claude for Enterprise 플랜에서 사용할 수 있습니다. 이러한 지표는 claude.ai 조직 내의 사용자만 포함합니다. Claude Console API 또는 타사 통합을 통한 사용량은 포함되지 않습니다.

사용량 및 채택 데이터는 모든 Claude for Teams 및 Claude for Enterprise 계정에서 사용할 수 있습니다. 기여도 지표는 GitHub 조직을 연결하기 위해 추가 설정이 필요합니다.

분석 설정을 구성하려면 소유자 역할이 필요합니다. GitHub 관리자가 GitHub 앱을 설치해야 합니다.

> **Warning:**
>
> Zero Data Retention이 활성화된 조직에서는 기여도 지표를 사용할 수 없습니다. 분석 대시보드는 사용량 지표만 표시합니다.

Step 1: GitHub 앱 설치

GitHub 관리자가 github.com/apps/claude에서 조직의 GitHub 계정에 Claude GitHub 앱을 설치합니다.

Step 2: Claude Code 분석 활성화

Claude 소유자가 claude.ai/admin-settings/claude-code로 이동하여 Claude Code 분석 기능을 활성화합니다.

Step 3: GitHub 분석 활성화

같은 페이지에서 “GitHub 분석” 토글을 활성화합니다.

Step 4: GitHub로 인증

GitHub 인증 흐름을 완료하고 분석에 포함할 GitHub 조직을 선택합니다.

활성화 후 일반적으로 24시간 이내에 데이터가 나타나며, 매일 업데이트됩니다. 데이터가 나타나지 않으면 다음 메시지 중 하나가 표시될 수 있습니다:

- **“GitHub 앱 필수”** : 기여도 지표를 보려면 GitHub 앱을 설치하세요
- **“데이터 처리 진행 중”**: 며칠 후 다시 확인하고 데이터가 나타나지 않으면 GitHub 앱이 설치되었는지 확인하세요

기여도 지표는 GitHub Cloud 및 GitHub Enterprise Server를 지원합니다.

요약 지표 검토

> **Note:**

> 이러한 지표는 의도적으로 보수적이며 Claude Code의 실제 영향을 과소평가합니다. Claude Code의 관여도가 높은 라인 및 PR만 계산됩니다.

대시보드는 상단에 다음 요약 지표를 표시합니다:

- **CC가 포함된 PR**: Claude Code로 작성된 코드 라인이 하나 이상 포함된 병합된 풀 요청의 총 개수
- **CC가 포함된 코드 라인**: Claude Code 지원으로 작성된 모든 병합된 PR의 총 코드 라인 수입니다. "효과적인 라인"만 계산됩니다: 정규화 후 3자 이상의 라인, 빈 라인 및 괄호나 사소한 구두점만 있는 라인 제외.
- **Claude Code가 포함된 PR (%)**: Claude Code 지원 코드를 포함하는 모든 병합된 PR의 백분율
- **제안 수락률**: 사용자가 Claude Code의 코드 편집 제안을 수락하는 횟수의 백분율(Edit, Write, NotebookEdit 도구 사용 포함)
- **수락된 코드 라인**: 사용자가 세션에서 수락한 Claude Code로 작성된 총 코드 라인 수입니다. 거부된 제안은 제외되며 후속 삭제는 추적하지 않습니다.

차트 탐색

대시보드에는 시간 경과에 따른 추세를 시각화하는 여러 차트가 포함되어 있습니다.

채택 추적

채택 차트는 일일 사용 추세를 보여줍니다:

- **사용자**: 일일 활성 사용자
- **세션**: 일일 활성 Claude Code 세션 수

사용자당 PR 측정

이 차트는 시간 경과에 따른 개별 개발자 활동을 표시합니다:

- **사용자당 PR**: 일일 병합된 PR의 총 개수를 일일 활성 사용자로 나눈 값
- **사용자**: 일일 활성 사용자

이를 사용하여 Claude Code 채택이 증가함에 따라 개별 생산성이 어떻게 변하는지 이해할 수 있습니다.

풀 요청 분석 보기

풀 요청 차트는 병합된 PR의 일일 분석을 보여줍니다:

- **CC가 포함된 PR**: Claude Code 지원 코드를 포함하는 풀 요청
- **CC가 포함되지 않은 PR**: Claude Code 지원 코드를 포함하지 않는 풀 요청

코드 라인 보기로 전환하여 PR 개수가 아닌 코드 라인으로 동일한 분석을 확인합니다.

상위 기여자 찾기

리더보드는 기여도 볼륨으로 순위가 매겨진 상위 10명의 사용자를 보여줍니다. 다음 사이를 전환합니다:

- **풀 요청**: 각 사용자에 대해 Claude Code가 포함된 PR 대 모든 PR을 표시합니다
- **코드 라인**: 각 사용자에 대해 Claude Code가 포함된 라인 대 모든 라인을 표시합니다

모든 사용자 내보내기를 클릭하여 모든 사용자의 완전한 기여도 데이터를 CSV 파일로 다운로드합니다. 내보내기에는 표시된 상위 10명뿐만 아니라 모든 사용자가 포함됩니다.

PR 속성

기여도 지표가 활성화되면 Claude Code는 병합된 풀 요청을 분석하여 Claude Code 지원으로 작성된 코드를 결정합니다. 이는 Claude Code 세션 활동을 각 PR의 코드와 일치시켜 수행됩니다.

태깅 기준

PR은 Claude Code 세션 중에 작성된 코드 라인이 하나 이상 포함되어 있으면 "Claude Code 포함"으로 태깅됩니다. 시스템은 보수적인 일치를 사용합니다: Claude Code의 관여도가 높은 코드만 지원되는 것으로 계산됩니다.

속성 프로세스

풀 요청이 병합될 때:

1. 추가된 라인이 PR diff에서 추출됩니다
2. 일치하는 파일을 편집한 Claude Code 세션이 시간 창 내에서 식별됩니다
3. PR 라인이 여러 전략을 사용하여 Claude Code 출력과 일치합니다
4. AI 지원 라인 및 총 라인에 대한 지표가 계산됩니다

비교 전에 라인이 정규화됩니다: 공백이 제거되고, 여러 공백이 축약되며, 따옴표가 표준화되고, 텍스트가 소문자로 변환됩니다.

Claude Code 지원 라인을 포함하는 병합된 풀 요청은 GitHub에서 `claude-code-assisted` 로 레이블이 지정됩니다.

시간 창

PR 병합 날짜 21일 전부터 2일 후까지의 세션이 속성 일치를 위해 고려됩니다.

제외된 파일

특정 파일은 자동 생성되기 때문에 분석에서 자동으로 제외됩니다:

- 잠금 파일: package-lock.json, yarn.lock, Cargo.lock 등
- 생성된 코드: Protobuf 출력, 빌드 아티팩트, 축소된 파일
- 빌드 디렉토리: dist/, build/, node_modules/, target/
- 테스트 픽스처: 스냅샷, 카세트, 모의 데이터
- 1,000자 이상의 라인(축소되거나 생성된 가능성이 높음)

속성 참고 사항

속성 데이터를 해석할 때 다음 추가 세부 정보를 염두에 두세요:

- 개발자가 20% 이상의 차이로 실질적으로 다시 작성한 코드는 Claude Code에 속성되지 않습니다
- 21일 창 외의 세션은 고려되지 않습니다
- 알고리즘은 속성을 수행할 때 PR 소스 또는 대상 분기를 고려하지 않습니다

분석에서 최대한 활용하기

기여도 지표를 사용하여 ROI를 입증하고, 채택 패턴을 식별하며, 다른 사람이 시작하도록 도울 수 있는 팀 구성원을 찾습니다.

채택 모니터링

채택 차트 및 사용자 수를 추적하여 다음을 식별합니다:

- 모범 사례를 공유할 수 있는 활성 사용자
- 조직 전체의 전반적인 채택 추세
- 마찰이나 문제를 나타낼 수 있는 사용량 감소

ROI 측정

기여도 지표는 자신의 코드베이스의 데이터로 “이 도구가 투자할 가치가 있는가?”라는 질문에 답하는 데 도움이 됩니다:

- 채택이 증가함에 따라 시간 경과에 따른 사용자당 PR의 변화를 추적합니다
- Claude Code 포함 및 미포함으로 배포된 PR 및 코드 라인을 비교합니다
- DORA 지표, 스프린트 속도 또는 기타 엔지니어링 KPI와 함께 사용하여 Claude Code 채택으로 인한 변화를 이해합니다

파워 사용자 식별

리더보드는 높은 Claude Code 채택을 가진 팀 구성원을 찾는 데 도움이 되며, 이들은 다음을 수행할 수 있습니다:

- 팀과 프롬프팅 기법 및 워크플로우 공유
- 잘 작동하는 것에 대한 피드백 제공
- 새 사용자 온보딩 지원

프로그래매틱 방식으로 데이터 접근

GitHub를 통해 이 데이터를 쿼리하려면 `claude-code-assisted` 레이블이 지정된 PR을 검색합니다.

API 고객을 위한 분석 접근

Claude Console을 사용하는 API 고객은 platform.claude.com/claude-code에서 분석에 접근할 수 있습니다. 대시보드에 접근하려면 UsageView 권한이 필요하며, 이는 개발자, 청구, 관리자, 소유자 및 기본 소유자 역할에 부여됩니다.

> **Note:**
>
> GitHub 통합을 포함한 기여도 지표는 현재 API 고객에게 사용할 수 없습니다. Console 대시보드는 사용량 및 지출 지표만 표시합니다.

Console 대시보드는 다음을 표시합니다:

- **수락된 코드 라인**: 사용자가 세션에서 수락한 Claude Code로 작성된 총 코드 라인 수입니다. 거부된 제안은 제외되며 후속 삭제는 추적하지 않습니다.
- **제안 수락률**: 사용자가 코드 편집 도구 사용을 수락하는 횟수의 백분율(Edit, Write, NotebookEdit 도구 포함).
- **활동**: 차트에 표시된 일일 활성 사용자 및 세션.
- **지출**: 사용자 수와 함께 일일 API 비용(달러).

팀 인사이트 보기

팀 인사이트 테이블은 사용자별 지표를 표시합니다:

- **구성원**: Claude Code에 인증한 모든 사용자. API 키 사용자는 키 식별자로 표시되고, OAuth 사용자는 이메일 주소로 표시됩니다.
- **이번 달 지출**: 현재 달의 사용자별 총 API 비용.
- **이번 달 라인**: 현재 달의 사용자별 수락된 코드 라인의 총합.

> **Note:**
>
> Console 대시보드의 지출 수치는 분석 목적의 추정치입니다. 실제 비용은 청구 페이지를 참조하세요.

관련 리소스

- OpenTelemetry를 사용한 모니터링: 실시간 지표 및 이벤트를 관찰성 스택으로 내보내기
- 비용 효과적으로 관리하기: 지출 한도 설정 및 토큰 사용량 최적화
- 권한: 역할 및 권한 구성

모니터링

> Claude Code에 대한 OpenTelemetry를 활성화하고 구성하는 방법을 알아봅니다.

Claude Code는 모니터링 및 관찰성을 위해 OpenTelemetry(OTel) 메트릭 및 이벤트를 지원합니다.

모든 메트릭은 OpenTelemetry의 표준 메트릭 프로토콜을 통해 내보내지는 시계열 데이터이며, 이벤트는 OpenTelemetry의 로그/이벤트 프로토콜을 통해 내보내집니다. 메트릭 및 로그 백엔드가 올바르게 구성되어 있고 집계 세분성이 모니터링 요구 사항을 충족하는지 확인하는 것은 사용자의 책임입니다.

빠른 시작

환경 변수를 사용하여 OpenTelemetry를 구성합니다:

```
## 1. 원격 측정 활성화
export CLAUDE_CODE_ENABLE_TELEMETRY=1

## 2. 내보내기 선택 (둘 다 선택 사항 - 필요한 것만 구성)
export OTEL_METRICS_EXPORTER=otlp       # 옵션: otlp, prometheus, console
export OTEL_LOGS_EXPORTER=otlp          # 옵션: otlp, console

## 3. OTLP 엔드포인트 구성 (OTLP 내보내기용)
export OTEL_EXPORTER_OTLP_PROTOCOL=grpc
export OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317

## 4. 인증 설정 (필요한 경우)
export OTEL_EXPORTER_OTLP_HEADERS="Authorization=Bearer your-token"

## 5. 디버깅용: 내보내기 간격 단축
export OTEL_METRIC_EXPORT_INTERVAL=10000  # 10초 (기본값: 60000ms)
export OTEL_LOGS_EXPORT_INTERVAL=5000     # 5초 (기본값: 5000ms)

## 6. Claude Code 실행
claude
```

> **Note:**
>
> 기본 내보내기 간격은 메트릭의 경우 60초, 로그의 경우 5초입니다. 설정 중에 디버깅 목적으로 더 짧은 간격을 사용할 수 있습니다. 프로덕션 사용을 위해 이를 재설정하는 것을 잊지 마세요.

전체 구성 옵션은 OpenTelemetry 사양을 참조하세요.

관리자 구성

관리자는 관리 설정 파일을 통해 모든 사용자에 대한 OpenTelemetry 설정을 구성할 수 있습니다. 이를 통해 조직 전체에서 원격 측정 설정을 중앙에서 제어할 수 있습니다. 설정이 적용되는 방식에 대한 자세한 내용은 설정 우선순위를 참조하세요.

관리 설정 구성 예:

```
{
  "env": {
    "CLAUDE_CODE_ENABLE_TELEMETRY": "1",
    "OTEL_METRICS_EXPORTER": "otlp",
    "OTEL_LOGS_EXPORTER": "otlp",
    "OTEL_EXPORTER_OTLP_PROTOCOL": "grpc",
    "OTEL_EXPORTER_OTLP_ENDPOINT": "http://collector.company.com:4317",
    "OTEL_EXPORTER_OTLP_HEADERS": "Authorization=Bearer company-token"
  }
}
```

> **Note:**
>
> 관리 설정은 MDM(Mobile Device Management) 또는 기타 장치 관리 솔루션을 통해 배포할 수 있습니다. 관리 설정 파일에 정의된 환경 변수는 높은 우선순위를 가지며 사용자가 재정의할 수 없습니다.

구성 세부 정보

일반적인 구성 변수

환경 변수	설명	예제 값
`CLAUDE_CODE_ENABLE_TELEMETRY`	원격 측정 수집 활성화 (필수)	`1`

환경 변수	설명	예제 값
`OTEL_METRICS_EXPORTER`	메트릭 내보내기 유형 (쉼표로 구분)	`console`, `otlp`, `prometheus`
`OTEL_LOGS_EXPORTER`	로그/이벤트 내보내기 유형 (쉼표로 구분)	`console`, `otlp`
`OTEL_EXPORTER_OTLP_PROTOCOL`	OTLP 내보내기 프로토콜 (모든 신호)	`grpc`, `http/json`, `http/protobuf`
`OTEL_EXPORTER_OTLP_ENDPOINT`	OTLP 수집기 엔드포인트 (모든 신호)	`http://localhost:4317`
`OTEL_EXPORTER_OTLP_METRICS_PROTOCOL`	메트릭 프로토콜 (일반 설정 재정의)	`grpc`, `http/json`, `http/protobuf`
`OTEL_EXPORTER_OTLP_METRICS_ENDPOINT`	OTLP 메트릭 엔드포인트 (일반 설정 재정의)	`http://localhost:4318/v1/metrics`
`OTEL_EXPORTER_OTLP_LOGS_PROTOCOL`	로그 프로토콜 (일반 설정 재정의)	`grpc`, `http/json`, `http/protobuf`
`OTEL_EXPORTER_OTLP_LOGS_ENDPOINT`	OTLP 로그 엔드포인트 (일반 설정 재정의)	`http://localhost:4318/v1/logs`
`OTEL_EXPORTER_OTLP_HEADERS`	OTLP용 인증 헤더	`Authorization=Bearer token`
`OTEL_EXPORTER_OTLP_METRICS_CLIENT_KEY`	mTLS 인증용 클라이언트 키	클라이언트 키 파일 경로
`OTEL_EXPORTER_OTLP_METRICS_CLIENT_CERTIFICATE`	mTLS 인증용 클라이언트 인증서	클라이언트 인증서 파일 경로
`OTEL_METRIC_EXPORT_INTERVAL`	내보내기 간격 (밀리초 단위, 기본값: 60000)	`5000`, `60000`
`OTEL_LOGS_EXPORT_INTERVAL`	로그 내보내기 간격 (밀리초 단위, 기본값: 5000)	`1000`, `10000`

환경 변수	설명	예제 값
`OTEL_LOG_USER_PROMPTS`	사용자 프롬프트 콘텐츠 로깅 활성화 (기본값: 비활성화)	`1` 로 활성화
`CLAUDE_CODE_OTEL_HEADERS_HELPER_DEBOUNCE_MS`	동적 헤더 새로 고침 간격 (기본값: 1740000ms / 29분)	`900000`

메트릭 카디널리티 제어

다음 환경 변수는 카디널리티를 관리하기 위해 메트릭에 포함되는 속성을 제어합니다:

환경 변수	설명	기본값	비활성화 예
`OTEL_METRICS_INCLUDE_SESSION_ID`	메트릭에 session.id 속성 포함	`true`	`false`
`OTEL_METRICS_INCLUDE_VERSION`	메트릭에 app.version 속성 포함	`false`	`true`
`OTEL_METRICS_INCLUDE_ACCOUNT_UUID`	메트릭에 user.account_uuid 속성 포함	`true`	`false`

이러한 변수는 메트릭의 카디널리티를 제어하는 데 도움이 되며, 이는 메트릭 백엔드의 저장소 요구 사항 및 쿼리 성능에 영향을 미칩니다. 낮은 카디널리티는 일반적으로 더 나은 성능과 낮은 저장소 비용을 의미하지만 분석을 위한 세분화된 데이터는 적습니다.

동적 헤더

동적 인증이 필요한 엔터프라이즈 환경의 경우 스크립트를 구성하여 헤더를 동적으로 생성할 수 있습니다:

설정 구성

`.claude/settings.json` 에 추가:

```
{
  "otelHeadersHelper": "/bin/generate_opentelemetry_headers.sh"
}
```

스크립트 요구 사항

스크립트는 HTTP 헤더를 나타내는 문자열 키-값 쌍이 있는 유효한 JSON을 출력해야 합니다:

```
#!/bin/bash
## 예: 여러 헤더
echo "{\"Authorization\": \"Bearer $(get-token.sh)\", \"X-API-Key\": \"$(get-api-key.sh)\"}"
```

새로 고침 동작

헤더 도우미 스크립트는 시작 시 그리고 그 이후 주기적으로 실행되어 토큰 새로 고침을 지원합니다. 기본적으로 스크립트는 29분마다 실행됩니다.
`CLAUDE_CODE_OTEL_HEADERS_HELPER_DEBOUNCE_MS` 환경 변수로 간격을 사용자 정의합니다.

다중 팀 조직 지원

여러 팀 또는 부서가 있는 조직은 `OTEL_RESOURCE_ATTRIBUTES` 환경 변수를 사용하여 다양한 그룹을 구분하기 위한 사용자 정의 속성을 추가할 수 있습니다:

```
## 팀 식별을 위한 사용자 정의 속성 추가
export OTEL_RESOURCE_ATTRIBUTES="department=engineering,team.id=platform,cost_center=eng-123"
```

이러한 사용자 정의 속성은 모든 메트릭 및 이벤트에 포함되어 다음을 수행할 수 있습니다:

- 팀 또는 부서별로 메트릭 필터링
- 비용 센터별 비용 추적
- 팀별 대시보드 생성
- 특정 팀에 대한 경고 설정

> **Warning:**
>
> **OTEL_RESOURCE_ATTRIBUTES에 대한 중요한 형식 요구 사항:**

`OTEL_RESOURCE_ATTRIBUTES` 환경 변수는 W3C Baggage 사양을 따르며, 엄격한 형식 요구 사항이 있습니다:

- **공백 허용 안 함**: 값에 공백이 포함될 수 없습니다. 예를 들어 `user.organizationName=My Company` 는 유효하지 않습니다
- **형식**: 쉼표로 구분된 키=값 쌍이어야 합니다: `key1=value1,key2=value2`
- **허용된 문자**: 제어 문자, 공백, 큰따옴표, 쉼표, 세미콜론 및 백슬래시를 제외한 US-ASCII 문자만 허용됩니다
- **특수 문자**: 허용된 범위 외의 문자는 퍼센트 인코딩되어야 합니다

예:

```
## ❌유효하지 않음 - 공백 포함
export OTEL_RESOURCE_ATTRIBUTES="org.name=John's Organization"

## ✅유효함 - 대신 언더스코어 또는 camelCase 사용
export OTEL_RESOURCE_ATTRIBUTES="org.name=Johns_Organization"
export OTEL_RESOURCE_ATTRIBUTES="org.name=JohnsOrganization"

## ✅유효함 - 필요한 경우 특수 문자를 퍼센트 인코딩
export OTEL_RESOURCE_ATTRIBUTES="org.name=John%27s%20Organization"
```

참고: 값을 따옴표로 감싸도 공백이 이스케이프되지 않습니다. 예를 들어 `org.name="My Company"` 는 `My Company` 가 아닌 리터럴 값 `"My Company"` (따옴표 포함)를 생성합니다.

예제 구성

```
## 콘솔 디버깅 (1초 간격)
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=console
export OTEL_METRIC_EXPORT_INTERVAL=1000

## OTLP/gRPC
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=otlp
export OTEL_EXPORTER_OTLP_PROTOCOL=grpc
export OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317

## Prometheus
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=prometheus

## 여러 내보내기
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=console,otlp
export OTEL_EXPORTER_OTLP_PROTOCOL=http/json

## 메트릭 및 로그에 대한 다양한 엔드포인트/백엔드
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=otlp
export OTEL_LOGS_EXPORTER=otlp
export OTEL_EXPORTER_OTLP_METRICS_PROTOCOL=http/protobuf
export OTEL_EXPORTER_OTLP_METRICS_ENDPOINT=http://metrics.company.com:4318
export OTEL_EXPORTER_OTLP_LOGS_PROTOCOL=grpc
export OTEL_EXPORTER_OTLP_LOGS_ENDPOINT=http://logs.company.com:4317

## 메트릭만 (이벤트/로그 없음)
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=otlp
export OTEL_EXPORTER_OTLP_PROTOCOL=grpc
export OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317

## 이벤트/로그만 (메트릭 없음)
export CLAUDE_CODE_ENABLE_TELEMETRY=1
```

```
export OTEL_LOGS_EXPORTER=otlp
export OTEL_EXPORTER_OTLP_PROTOCOL=grpc
export OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317
```

사용 가능한 메트릭 및 이벤트

표준 속성

모든 메트릭 및 이벤트는 다음 표준 속성을 공유합니다:

속성	설명	제어 대상
`session.id`	고유 세션 식별자	`OTEL_METRICS_INCLUDE_SESSION_ID` (기본값: true)
`app.version`	현재 Claude Code 버전	`OTEL_METRICS_INCLUDE_VERSION` (기본값: false)
`organization.id`	조직 UUID (인증된 경우)	사용 가능할 때 항상 포함됨
`user.account_uuid`	계정 UUID (인증된 경우)	`OTEL_METRICS_INCLUDE_ACCOUNT_UUID` (기본값: true)
`terminal.type`	터미널 유형 (예: `iTerm.app`, `vscode`, `cursor`, `tmux`)	감지될 때 항상 포함됨

메트릭

Claude Code는 다음 메트릭을 내보냅니다:

메트릭 이름	설명	단위
`claude_code.session.count`	시작된 CLI 세션 수	count
`claude_code.lines_of_code.count`	수정된 코드 라인 수	count
`claude_code.pull_request.count`	생성된 풀 요청 수	count
`claude_code.commit.count`	생성된 git 커밋 수	count

메트릭 이름	설명	단위
`claude_code.cost.usage`	Claude Code 세션의 비용	USD
`claude_code.token.usage`	사용된 토큰 수	tokens
`claude_code.code_edit_tool.decision`	코드 편집 도구 권한 결정 수	count
`claude_code.active_time.total`	총 활성 시간 (초)	s

메트릭 세부 정보

세션 카운터

각 세션 시작 시 증가합니다.

속성:

- 모든 표준 속성

코드 라인 카운터

코드가 추가되거나 제거될 때 증가합니다.

속성:

- 모든 표준 속성
- `type` : (`"added"`, `"removed"`)

풀 요청 카운터

Claude Code를 통해 풀 요청을 생성할 때 증가합니다.

속성:

- 모든 표준 속성

커밋 카운터

Claude Code를 통해 git 커밋을 생성할 때 증가합니다.

속성:

- 모든 표준 속성

비용 카운터

각 API 요청 후 증가합니다.

속성:

- 모든 표준 속성
- `model`: 모델 식별자 (예: "claude-sonnet-4-5-20250929")

토큰 카운터

각 API 요청 후 증가합니다.

속성:

- 모든 표준 속성
- `type`: (`"input"`, `"output"`, `"cacheRead"`, `"cacheCreation"`)
- `model`: 모델 식별자 (예: "claude-sonnet-4-5-20250929")

코드 편집 도구 결정 카운터

사용자가 Edit, Write 또는 NotebookEdit 도구 사용을 수락하거나 거부할 때 증가합니다.

속성:

- 모든 표준 속성
- `tool`: 도구 이름 (`"Edit"`, `"Write"`, `"NotebookEdit"`)
- `decision`: 사용자 결정 (`"accept"`, `"reject"`)
- `language`: 편집된 파일의 프로그래밍 언어 (예: `"TypeScript"`, `"Python"`, `"JavaScript"`, `"Markdown"`). 인식되지 않는 파일 확장자의 경우 `"unknown"`을 반환합니다.

활성 시간 카운터

Claude Code를 적극적으로 사용하는 실제 시간을 추적합니다 (유휴 시간 아님). 이 메트릭은 프롬프트 입력 또는 응답 수신과 같은 사용자 상호 작용 중에 증가합니다.

속성:

- 모든 표준 속성

이벤트

Claude Code는 OpenTelemetry 로그/이벤트를 통해 다음 이벤트를 내보냅니다 (`OTEL_LOGS_EXPORTER`가 구성된 경우):

사용자 프롬프트 이벤트

사용자가 프롬프트를 제출할 때 기록됩니다.

이벤트 이름: `claude_code.user_prompt`

속성:

- 모든 표준 속성
- `event.name`: `"user_prompt"`
- `event.timestamp`: ISO 8601 타임스탬프
- `prompt_length`: 프롬프트의 길이
- `prompt`: 프롬프트 콘텐츠 (기본적으로 수정됨, `OTEL_LOG_USER_PROMPTS=1` 로 활성화)

도구 결과 이벤트

도구가 실행을 완료할 때 기록됩니다.

이벤트 이름: `claude_code.tool_result`

속성:

- 모든 표준 속성
- `event.name`: `"tool_result"`
- `event.timestamp`: ISO 8601 타임스탬프
- `tool_name`: 도구의 이름
- `success`: `"true"` 또는 `"false"`
- `duration_ms`: 실행 시간 (밀리초)
- `error`: 오류 메시지 (실패한 경우)
- `decision`: `"accept"` 또는 `"reject"`
- `source`: 결정 출처 - `"config"`, `"user_permanent"`, `"user_temporary"`, `"user_abort"` 또는 `"user_reject"`
- `tool_parameters`: 도구별 매개변수를 포함하는 JSON 문자열 (사용 가능한 경우)
  - Bash 도구의 경우: `bash_command`, `full_command`, `timeout`, `description`, `sandbox` 포함

API 요청 이벤트

Claude에 대한 각 API 요청에 대해 기록됩니다.

이벤트 이름: `claude_code.api_request`

속성:

- 모든 표준 속성
- `event.name`: `"api_request"`
- `event.timestamp`: ISO 8601 타임스탬프
- `model`: 사용된 모델 (예: “claude-sonnet-4-5-20250929”)
- `cost_usd`: USD 단위의 예상 비용
- `duration_ms`: 요청 지속 시간 (밀리초)
- `input_tokens`: 입력 토큰 수
- `output_tokens`: 출력 토큰 수
- `cache_read_tokens`: 캐시에서 읽은 토큰 수
- `cache_creation_tokens`: 캐시 생성에 사용된 토큰 수

API 오류 이벤트

Claude에 대한 API 요청이 실패할 때 기록됩니다.

이벤트 이름: `claude_code.api_error`

속성:

- 모든 표준 속성
- `event.name`: `"api_error"`
- `event.timestamp`: ISO 8601 타임스탬프
- `model`: 사용된 모델 (예: “claude-sonnet-4-5-20250929”)
- `error`: 오류 메시지
- `status_code`: HTTP 상태 코드 (해당하는 경우)
- `duration_ms`: 요청 지속 시간 (밀리초)
- `attempt`: 시도 번호 (재시도된 요청의 경우)

도구 결정 이벤트

도구 권한 결정이 내려질 때 기록됩니다 (수락/거부).

이벤트 이름: `claude_code.tool_decision`

속성:

- 모든 표준 속성
- `event.name`: `"tool_decision"`
- `event.timestamp`: ISO 8601 타임스탬프

- `tool_name`: 도구의 이름 (예: “Read”, “Edit”, “Write”, “NotebookEdit”)
- `decision`: `"accept"` 또는 `"reject"`
- `source`: 결정 출처 - `"config"`, `"user_permanent"`, `"user_temporary"`, `"user_abort"` 또는 `"user_reject"`

메트릭 및 이벤트 데이터 해석

Claude Code에서 내보낸 메트릭은 사용 패턴 및 생산성에 대한 귀중한 통찰력을 제공합니다. 다음은 만들 수 있는 일반적인 시각화 및 분석입니다:

사용 모니터링

메트릭	분석 기회
`claude_code.token.usage`	`type` (입력/출력), 사용자, 팀 또는 모델별로 분류
`claude_code.session.count`	시간 경과에 따른 채택 및 참여 추적
`claude_code.lines_of_code.count`	코드 추가/제거를 추적하여 생산성 측정
`claude_code.commit.count` & `claude_code.pull_request.count`	개발 워크플로우에 미치는 영향 이해

비용 모니터링

`claude_code.cost.usage` 메트릭은 다음에 도움이 됩니다:

- 팀 또는 개인 전체의 사용 추세 추적
- 최적화를 위한 높은 사용 세션 식별

> **Note:**
>
> 비용 메트릭은 근사값입니다. 공식 청구 데이터는 API 제공자 (Claude Console, AWS Bedrock 또는 Google Cloud Vertex)를 참조하세요.

경고 및 세분화

고려할 일반적인 경고:

- 비용 급증
- 비정상적인 토큰 소비

- 특정 사용자의 높은 세션 볼륨

모든 메트릭은 `user.account_uuid`, `organization.id`, `session.id`, `model` 및 `app.version`으로 세분화할 수 있습니다.

이벤트 분석

이벤트 데이터는 Claude Code 상호 작용에 대한 자세한 통찰력을 제공합니다:

도구 사용 패턴: 도구 결과 이벤트를 분석하여 다음을 식별합니다:

- 가장 자주 사용되는 도구
- 도구 성공률
- 평균 도구 실행 시간
- 도구 유형별 오류 패턴

성능 모니터링: API 요청 지속 시간 및 도구 실행 시간을 추적하여 성능 병목 현상을 식별합니다.

백엔드 고려 사항

메트릭 및 로그 백엔드 선택은 수행할 수 있는 분석 유형을 결정합니다:

메트릭의 경우

- **시계열 데이터베이스 (예: Prometheus)**: 비율 계산, 집계된 메트릭
- **컬럼형 저장소 (예: ClickHouse)**: 복잡한 쿼리, 고유 사용자 분석
- **완전한 기능의 관찰성 플랫폼 (예: Honeycomb, Datadog)**: 고급 쿼리, 시각화, 경고

이벤트/로그의 경우

- **로그 집계 시스템 (예: Elasticsearch, Loki)**: 전체 텍스트 검색, 로그 분석
- **컬럼형 저장소 (예: ClickHouse)**: 구조화된 이벤트 분석
- **완전한 기능의 관찰성 플랫폼 (예: Honeycomb, Datadog)**: 메트릭과 이벤트 간의 상관 관계

일일/주간/월간 활성 사용자 (DAU/WAU/MAU) 메트릭이 필요한 조직의 경우 효율적인 고유 값 쿼리를 지원하는 백엔드를 고려하세요.

서비스 정보

모든 메트릭 및 이벤트는 다음 리소스 속성과 함께 내보내집니다:

- `service.name`: `claude-code`
- `service.version`: 현재 Claude Code 버전
- `os.type`: 운영 체제 유형 (예: `linux`, `darwin`, `windows`)

- `os.version`: 운영 체제 버전 문자열
- `host.arch`: 호스트 아키텍처 (예: `amd64`, `arm64`)
- `wsl.version`: WSL 버전 번호 (Windows Subsystem for Linux에서 실행할 때만 표시)
- 미터 이름: `com.anthropic.claude_code`

ROI 측정 리소스

원격 측정 설정, 비용 분석, 생산성 메트릭 및 자동화된 보고를 포함하여 Claude Code의 투자 수익률 측정에 대한 포괄적인 가이드는 Claude Code ROI 측정 가이드를 참조하세요. 이 저장소는 즉시 사용 가능한 Docker Compose 구성, Prometheus 및 OpenTelemetry 설정, Linear와 같은 도구와 통합된 생산성 보고서 생성 템플릿을 제공합니다.

보안/개인 정보 보호 고려 사항

- 원격 측정은 선택 사항이며 명시적 구성이 필요합니다
- API 키 또는 파일 콘텐츠와 같은 민감한 정보는 메트릭 또는 이벤트에 포함되지 않습니다
- 사용자 프롬프트 콘텐츠는 기본적으로 수정됩니다 - 프롬프트 길이만 기록됩니다. 사용자 프롬프트 로깅을 활성화하려면 `OTEL_LOG_USER_PROMPTS=1`을 설정하세요

Amazon Bedrock에서 Claude Code 모니터링

Amazon Bedrock의 Claude Code 사용 모니터링에 대한 자세한 지침은 Claude Code 모니터링 구현 (Bedrock)을 참조하세요.

엔터프라이즈 배포 개요

> Claude Code가 다양한 타사 서비스 및 인프라와 통합되어 엔터프라이즈 배포 요구사항을 충족하는 방법을 알아봅니다.

이 페이지는 사용 가능한 배포 옵션의 개요를 제공하며 조직에 맞는 올바른 구성을 선택하는 데 도움을 줍니다.

제공자 비교

기능 Anthropic Amazon Bedrock Google Vertex AI Microsoft Foundry

지역 지원되는 국가 여러 AWS 지역 여러 GCP 지역 여러 Azure 지역

프롬프트 캐싱 기본적으로 활성화됨 기본적으로 활성화됨 기본적으로 활성화됨 기본적으로 활성화됨

인증 API 키 API 키 또는 AWS 자격증명 GCP 자격증명 API 키 또는 Microsoft Entra ID

비용 추적 대시보드 AWS Cost Explorer GCP Billing Azure Cost Management

엔터프라이즈 기능 팀, 사용량 모니터링 IAM 정책, CloudTrail IAM 역할, Cloud Audit Logs RBAC 정책, Azure Monitor

클라우드 제공자

- Amazon Bedrock API 키 또는 IAM 기반 인증 및 AWS 네이티브 모니터링을 통해 AWS 인프라를 통해 Claude 모델 사용
- Google Vertex AI 엔터프라이즈급 보안 및 규정 준수를 통해 Google Cloud Platform을 통해 Claude 모델에 액세스
- Microsoft Foundry API 키 또는 Microsoft Entra ID 인증 및 Azure 청구를 통해 Azure를 통해 Claude에 액세스

기업 인프라

- 엔터프라이즈 네트워크 조직의 프록시 서버 및 SSL/TLS 요구사항과 함께 작동하도록 Claude Code 구성
- LLM Gateway 사용량 추적, 예산 책정 및 감사 로깅을 통해 중앙 집중식 모델 액세스 배포

구성 개요

Claude Code는 다양한 제공자 및 인프라를 결합할 수 있는 유연한 구성 옵션을 지원합니다:

> **Note:**
>
> 다음의 차이점을 이해하세요:
>
> - **기업 프록시**: 트래픽 라우팅을 위한 HTTP/HTTPS 프록시 (`HTTPS_PROXY` 또는 `HTTP_PROXY` 를 통해 설정)
> - **LLM Gateway**: 인증을 처리하고 제공자 호환 엔드포인트를 제공하는 서비스 (`ANTHROPIC_BASE_URL` , `ANTHROPIC_BEDROCK_BASE_URL` 또는 `ANTHROPIC_VERTEX_BASE_URL` 을 통해 설정)
>
> 두 구성을 함께 사용할 수 있습니다.

기업 프록시를 사용한 Bedrock

기업 HTTP/HTTPS 프록시를 통해 Bedrock 트래픽 라우팅:

```
## Bedrock 활성화
export CLAUDE_CODE_USE_BEDROCK=1
export AWS_REGION=us-east-1

## 기업 프록시 구성
export HTTPS_PROXY='https://proxy.example.com:8080'
```

LLM Gateway를 사용한 Bedrock

Bedrock 호환 엔드포인트를 제공하는 게이트웨이 서비스 사용:

```
## Bedrock 활성화
export CLAUDE_CODE_USE_BEDROCK=1

## LLM 게이트웨이 구성
export ANTHROPIC_BEDROCK_BASE_URL='https://your-llm-gateway.com/bedrock'
export CLAUDE_CODE_SKIP_BEDROCK_AUTH=1  # 게이트웨이가 AWS 인증을 처리하는 경우
```

기업 프록시를 사용한 Foundry

기업 HTTP/HTTPS 프록시를 통해 Azure 트래픽 라우팅:

```
## Microsoft Foundry 활성화
export CLAUDE_CODE_USE_FOUNDRY=1
export ANTHROPIC_FOUNDRY_RESOURCE=your-resource
export ANTHROPIC_FOUNDRY_API_KEY=your-api-key  # 또는 Entra ID 인증의 경우 생략

## 기업 프록시 구성
export HTTPS_PROXY='https://proxy.example.com:8080'
```

LLM Gateway를 사용한 Foundry

Azure 호환 엔드포인트를 제공하는 게이트웨이 서비스 사용:

```
## Microsoft Foundry 활성화
export CLAUDE_CODE_USE_FOUNDRY=1

## LLM 게이트웨이 구성
export ANTHROPIC_FOUNDRY_BASE_URL='https://your-llm-gateway.com'
export CLAUDE_CODE_SKIP_FOUNDRY_AUTH=1  # 게이트웨이가 Azure 인증을 처리하는 경우
```

기업 프록시를 사용한 Vertex AI

기업 HTTP/HTTPS 프록시를 통해 Vertex AI 트래픽 라우팅:

```
## Vertex 활성화
export CLAUDE_CODE_USE_VERTEX=1
export CLOUD_ML_REGION=us-east5
export ANTHROPIC_VERTEX_PROJECT_ID=your-project-id

## 기업 프록시 구성
export HTTPS_PROXY='https://proxy.example.com:8080'
```

LLM Gateway를 사용한 Vertex AI

중앙 집중식 관리를 위해 Google Vertex AI 모델을 LLM 게이트웨이와 결합:

```
## Vertex 활성화
export CLAUDE_CODE_USE_VERTEX=1

## LLM 게이트웨이 구성
export ANTHROPIC_VERTEX_BASE_URL='https://your-llm-gateway.com/vertex'
export CLAUDE_CODE_SKIP_VERTEX_AUTH=1  # 게이트웨이가 GCP 인증을 처리하는 경우
```

인증 구성

Claude Code는 필요할 때 `Authorization` 헤더에 대해 `ANTHROPIC_AUTH_TOKEN` 을 사용합니다. `SKIP_AUTH` 플래그 (`CLAUDE_CODE_SKIP_BEDROCK_AUTH` , `CLAUDE_CODE_SKIP_VERTEX_AUTH`)는 게이트웨이가 제공자 인증을 처리하는 LLM 게이트웨이 시나리오에서 사용됩니다.

올바른 배포 구성 선택

배포 방식을 선택할 때 다음 요소를 고려하세요:

직접 제공자 액세스

다음과 같은 조직에 최적:

- 가장 간단한 설정을 원함
- 기존 AWS 또는 GCP 인프라 보유
- 제공자 네이티브 모니터링 및 규정 준수 필요

기업 프록시

다음과 같은 조직에 최적:

- 기존 기업 프록시 요구사항 보유
- 트래픽 모니터링 및 규정 준수 필요
- 모든 트래픽을 특정 네트워크 경로를 통해 라우팅해야 함

LLM Gateway

다음과 같은 조직에 최적:

- 팀 전체의 사용량 추적 필요
- 모델 간 동적 전환 원함
- 사용자 정의 속도 제한 또는 예산 필요
- 중앙 집중식 인증 관리 필요

디버깅

배포를 디버깅할 때:

- `claude /status` 슬래시 명령어를 사용하세요. 이 명령어는 적용된 인증, 프록시 및 URL 설정에 대한 관찰성을 제공합니다.
- 환경 변수 `export ANTHROPIC_LOG=debug`를 설정하여 요청을 로깅합니다.

조직을 위한 모범 사례

1. 문서화 및 메모리에 투자

Claude Code가 코드베이스를 이해하도록 문서화에 투자할 것을 강력히 권장합니다. 조직은 여러 수준에서 CLAUDE.md 파일을 배포할 수 있습니다:

- **조직 전체**: 회사 전체 표준을 위해 `/Library/Application Support/ClaudeCode/CLAUDE.md` (macOS)와 같은 시스템 디렉토리에 배포
- **저장소 수준**: 프로젝트 아키텍처, 빌드 명령 및 기여 지침을 포함하는 저장소 루트에 `CLAUDE.md` 파일 생성. 소스 제어에 체크인하여 모든 사용자가 이점을 얻도록 함
  자세히 알아보기.

2. 배포 단순화

사용자 정의 개발 환경이 있는 경우, Claude Code를 설치하는 "원클릭" 방식을 만드는 것이 조직 전체에서 채택을 늘리는 핵심이라고 생각합니다.

3. 안내된 사용으로 시작

새 사용자가 코드베이스 Q&A 또는 더 작은 버그 수정이나 기능 요청에 Claude Code를 시도하도록 권장합니다. Claude Code에 계획을 세우도록 요청하세요. Claude의 제안을 확인하고 잘못된 경우 피드백을 제공하세요. 시간이 지남에 따라 사용자가 이 새로운 패러다임을 더 잘 이해하게 되면, Claude Code를 더 자율적으로 실행하는 데 더 효과적이 될 것입니다.

4. 보안 정책 구성

보안 팀은 Claude Code가 할 수 있는 것과 할 수 없는 것에 대한 관리 권한을 구성할 수 있으며, 이는 로컬 구성으로 덮어쓸 수 없습니다. 자세히 알아보기.

5. MCP를 통합에 활용

MCP는 Claude Code에 더 많은 정보를 제공하는 좋은 방법입니다. 예를 들어 티켓 관리 시스템이나 오류 로그에 연결할 수 있습니다. 한 중앙 팀이 MCP 서버를 구성하고 `.mcp.json` 구성을 코드베이스에 체크인하여 모든 사용자가 이점을 얻도록 할 것을 권장합니다. 자세히 알아보기.

Anthropic에서는 Claude Code가 모든 Anthropic 코드베이스 전체에서 개발을 강화하도록 신뢰합니다. Claude Code를 우리만큼 즐기시기를 바랍니다.

다음 단계

- Amazon Bedrock 설정 - AWS 네이티브 배포
- Google Vertex AI 구성 - GCP 배포
- Microsoft Foundry 설정 - Azure 배포
- 엔터프라이즈 네트워크 구성 - 네트워크 요구사항
- LLM Gateway 배포 - 엔터프라이즈 관리
- 설정 - 구성 옵션 및 환경 변수

서버 관리 설정 구성(공개 베타)

> 장치 관리 인프라 없이 Claude.ai의 웹 기반 인터페이스를 통해 조직을 위해 Claude Code를 중앙에서 구성합니다.

서버 관리 설정을 통해 관리자는 Claude.ai의 웹 기반 인터페이스를 통해 Claude Code를 중앙에서 구성할 수 있습니다. Claude Code 클라이언트는 사용자가 조직 자격증명으로 인증할 때 이러한 설정을 자동으로 수신합니다.

이 방식은 장치 관리 인프라가 없거나 관리되지 않는 장치의 사용자를 위해 설정을 관리해야 하는 조직을 위해 설계되었습니다.

> **Note:**
>
> 서버 관리 설정은 공개 베타 상태이며 Claude for Teams 및 Claude for Enterprise 고객에게 제공됩니다. 일반 공급 전에 기능이 변경될 수 있습니다.

요구사항

서버 관리 설정을 사용하려면 다음이 필요합니다.

- Claude for Teams 또는 Claude for Enterprise 플랜
- Claude for Teams의 경우 Claude Code 버전 2.1.38 이상, Claude for Enterprise의 경우 버전 2.1.30 이상
- `api.anthropic.com`에 대한 네트워크 액세스

서버 관리 설정과 엔드포인트 관리 설정 중 선택

Claude Code는 중앙 집중식 구성을 위한 두 가지 방식을 지원합니다. 서버 관리 설정은 Anthropic의 서버에서 구성을 전달합니다. 엔드포인트 관리 설정은 기본 OS 정책(macOS 관리 기본 설정, Windows 레지스트리) 또는 관리 설정 파일을 통해 장치에 직접 배포됩니다.

방식	최적 대상	보안 모델
서버 관리 설정	MDM이 없는 조직 또는 관리되지 않는 장치의 사용자	인증 시 Anthropic의 서버에서 전달되는 설정

방식	최적 대상	보안 모델
엔드포인트 관리 설정	MDM 또는 엔드포인트 관리가 있는 조직	MDM 구성 프로필, 레지스트리 정책 또는 관리 설정 파일을 통해 장치에 배포되는 설정

장치가 MDM 또는 엔드포인트 관리 솔루션에 등록된 경우, 엔드포인트 관리 설정은 설정 파일을 OS 수준에서 사용자 수정으로부터 보호할 수 있으므로 더 강력한 보안 보장을 제공합니다.

서버 관리 설정 구성

Step 1: 관리 콘솔 열기

Claude.ai에서 **관리 설정 > Claude Code > 관리 설정**으로 이동합니다.

Step 2: 설정 정의

구성을 JSON으로 추가합니다. `settings.json` 에서 사용 가능한 모든 설정이 지원되며, `disableBypassPermissionsMode` 와 같은 관리 전용 설정도 포함됩니다.

이 예제는 권한 거부 목록을 적용하고 사용자가 권한을 우회하는 것을 방지합니다.

```
{
  "permissions": {
    "deny": [
      "Bash(curl *)",
      "Read(./.env)",
      "Read(./.env.*)",
      "Read(./secrets/**)"
    ]
  },
  "disableBypassPermissionsMode": "disable"
}
```

Step 3: 저장 및 배포

변경 사항을 저장합니다. Claude Code 클라이언트는 다음 시작 또는 시간별 폴링 주기에 업데이트된 설정을 수신합니다.

설정 전달 확인

설정이 적용되고 있는지 확인하려면 사용자에게 Claude Code를 다시 시작하도록 요청합니다. 구성에 보안 승인 대화를 트리거하는 설정이 포함된 경우, 사용자는 시작 시 관리 설정을 설명하는 프롬프트를 봅니다. 사용자가 `/permissions`를 실행하여 유효한 권한 규칙을 확인하도록 하여 관리 권한 규칙이 활성화되어 있는지 확인할 수도 있습니다.

액세스 제어

다음 역할이 서버 관리 설정을 관리할 수 있습니다.

- **주 소유자**
- **소유자**

설정 변경이 조직의 모든 사용자에게 적용되므로 신뢰할 수 있는 담당자에게만 액세스를 제한합니다.

현재 제한사항

서버 관리 설정은 베타 기간 동안 다음과 같은 제한사항이 있습니다.

- 설정은 조직의 모든 사용자에게 균일하게 적용됩니다. 그룹별 구성은 아직 지원되지 않습니다.
- MCP 서버 구성은 서버 관리 설정을 통해 배포할 수 없습니다.

설정 전달

설정 우선순위

서버 관리 설정과 엔드포인트 관리 설정은 모두 Claude Code 설정 계층의 최상위 계층을 차지합니다. 명령줄 인수를 포함한 다른 설정 수준은 이를 재정의할 수 없습니다. 둘 다 있을 때 서버 관리 설정이 우선순위를 가지며 엔드포인트 관리 설정은 사용되지 않습니다.

가져오기 및 캐싱 동작

Claude Code는 시작 시 Anthropic의 서버에서 설정을 가져오고 활성 세션 중에 시간별로 업데이트를 폴링합니다.

캐시된 설정 없이 처음 시작:

- Claude Code는 비동기적으로 설정을 가져옵니다.
- 가져오기가 실패하면 Claude Code는 관리 설정 없이 계속됩니다.
- 설정이 로드되기 전에 제한이 아직 적용되지 않는 짧은 시간이 있습니다.

캐시된 설정으로 이후 시작:

- 캐시된 설정은 시작 시 즉시 적용됩니다.

- Claude Code는 백그라운드에서 새로운 설정을 가져옵니다.
- 캐시된 설정은 네트워크 장애를 통해 유지됩니다.

Claude Code는 OpenTelemetry 구성과 같은 고급 설정을 제외하고 재시작 없이 설정 업데이트를 자동으로 적용하며, 이는 적용되려면 전체 재시작이 필요합니다.

보안 승인 대화

보안 위험을 초래할 수 있는 특정 설정은 적용되기 전에 명시적인 사용자 승인이 필요합니다.

- **셸 명령 설정**: 셸 명령을 실행하는 설정
- **사용자 정의 환경 변수**: 알려진 안전 허용 목록에 없는 변수
- **Hook 구성**: 모든 hook 정의

이러한 설정이 있을 때 사용자는 구성되는 내용을 설명하는 보안 대화를 봅니다. 사용자는 진행하려면 승인해야 합니다. 사용자가 설정을 거부하면 Claude Code가 종료됩니다.

> **Note:**
>
> `-p` 플래그를 사용한 비대화형 모드에서 Claude Code는 보안 대화를 건너뛰고 사용자 승인 없이 설정을 적용합니다.

플랫폼 가용성

서버 관리 설정은 `api.anthropic.com`에 대한 직접 연결이 필요하며 타사 모델 공급자를 사용할 때는 사용할 수 없습니다.

- Amazon Bedrock
- Google Vertex AI
- Microsoft Foundry
- `ANTHROPIC_BASE_URL` 또는 LLM 게이트웨이를 통한 사용자 정의 API 엔드포인트

감사 로깅

설정 변경에 대한 감사 로그 이벤트는 규정 준수 API 또는 감사 로그 내보내기를 통해 사용할 수 있습니다. 액세스를 위해 Anthropic 계정 팀에 문의합니다.

감사 이벤트는 수행된 작업 유형, 작업을 수행한 계정 및 장치, 이전 값과 새 값에 대한 참조를 포함합니다.

보안 고려사항

서버 관리 설정은 중앙 집중식 정책 적용을 제공하지만 클라이언트 측 제어로 작동합니다. 관리되지 않는 장치에서 관리자 또는 sudo 액세스 권한이 있는 사용자는 Claude Code 바이너리, 파일 시스템 또는 네트워크 구성을 수정할 수 있습니다.

시나리오	동작
사용자가 캐시된 설정 파일을 편집함	변조된 파일이 시작 시 적용되지만 다음 서버 가져오기에서 올바른 설정이 복원됩니다.
사용자가 캐시된 설정 파일을 삭제함	첫 시작 동작이 발생합니다. 설정이 비동기적으로 가져오지며 짧은 적용되지 않은 시간이 있습니다.
API를 사용할 수 없음	캐시된 설정이 있으면 적용되고, 그렇지 않으면 다음 성공적인 가져오기까지 관리 설정이 적용되지 않습니다.
사용자가 다른 조직으로 인증함	관리 조직 외부의 계정에 대해 설정이 전달되지 않습니다.
사용자가 기본이 아닌 `ANTHROPIC_BASE_URL` 을 설정함	타사 API 공급자를 사용할 때 서버 관리 설정이 우회됩니다.

런타임 구성 변경을 감지하려면 `ConfigChange` hooks를 사용하여 수정 사항을 기록하거나 적용되기 전에 무단 변경을 차단합니다.

더 강력한 적용 보장을 위해 MDM 솔루션에 등록된 장치에서 엔드포인트 관리 설정을 사용합니다.

참고 항목

Claude Code 구성 관리를 위한 관련 페이지:

- 설정: 사용 가능한 모든 설정을 포함한 완전한 구성 참조
- 엔드포인트 관리 설정: IT에서 장치에 배포하는 관리 설정
- 인증: Claude Code에 대한 사용자 액세스 설정
- 보안: 보안 보호 및 모범 사례

Part 11: Cloud Providers

Amazon Bedrock의 Claude Code

> Amazon Bedrock을 통한 Claude Code 구성, 설정, IAM 구성 및 문제 해결에 대해 알아봅니다.

필수 조건

Claude Code를 Bedrock으로 구성하기 전에 다음을 확인하십시오:

- Bedrock 액세스가 활성화된 AWS 계정
- Bedrock에서 원하는 Claude 모델(예: Claude Sonnet 4.6)에 대한 액세스
- AWS CLI 설치 및 구성(선택 사항 - 자격 증명을 얻을 다른 메커니즘이 없는 경우에만 필요)
- 적절한 IAM 권한

> **Note:**
>
> Claude Code를 여러 사용자에게 배포하는 경우 Anthropic이 새 모델을 출시할 때 중단을 방지하기 위해 모델 버전을 고정하십시오.

설정

1. 사용 사례 세부 정보 제출

Anthropic 모델의 첫 사용자는 모델을 호출하기 전에 사용 사례 세부 정보를 제출해야 합니다. 이는 계정당 한 번 수행됩니다.

1. 올바른 IAM 권한이 있는지 확인하십시오(아래에서 자세히 알아보기)
2. Amazon Bedrock 콘솔로 이동하십시오
3. **Chat/Text playground**를 선택하십시오
4. 모든 Anthropic 모델을 선택하면 사용 사례 양식을 작성하라는 메시지가 표시됩니다

2. AWS 자격 증명 구성

Claude Code는 기본 AWS SDK 자격 증명 체인을 사용합니다. 다음 방법 중 하나를 사용하여 자격 증명을 설정하십시오:

옵션 A: AWS CLI 구성

```
aws configure
```

옵션 B: 환경 변수(액세스 키)

```
export AWS_ACCESS_KEY_ID=your-access-key-id
export AWS_SECRET_ACCESS_KEY=your-secret-access-key
export AWS_SESSION_TOKEN=your-session-token
```

옵션 C: 환경 변수(SSO 프로필)

```
aws sso login --profile=<your-profile-name>

export AWS_PROFILE=your-profile-name
```

옵션 D: AWS Management Console 자격 증명

```
aws login
```

`aws login`에 대해 자세히 알아보기.

옵션 E: Bedrock API 키

```
export AWS_BEARER_TOKEN_BEDROCK=your-bedrock-api-key
```

Bedrock API 키는 전체 AWS 자격 증명이 필요 없는 더 간단한 인증 방법을 제공합니다.
Bedrock API 키에 대해 자세히 알아보기.

고급 자격 증명 구성

Claude Code는 AWS SSO 및 회사 ID 공급자에 대한 자동 자격 증명 새로 고침을 지원합니다. Claude Code 설정 파일에 이러한 설정을 추가하십시오(설정에서 파일 위치 참조).

Claude Code가 AWS 자격 증명이 만료되었음을 감지하면(타임스탬프를 기반으로 로컬에서 또는 Bedrock이 자격 증명 오류를 반환할 때), 요청을 다시 시도하기 전에 새 자격 증명을 얻기 위해 구성된 `awsAuthRefresh` 및/또는 `awsCredentialExport` 명령을 자동으로 실행합니다.

예제 구성

```
{
  "awsAuthRefresh": "aws sso login --profile myprofile",
  "env": {
    "AWS_PROFILE": "myprofile"
  }
}
```

구성 설정 설명

`awsAuthRefresh`: `.aws` 디렉토리를 수정하는 명령(예: 자격 증명, SSO 캐시 또는 구성 파일 업데이트)에 사용하십시오. 명령의 출력이 사용자에게 표시되지만 대화형 입력은 지원되지 않습니다. 이는 CLI가 URL 또는 코드를 표시하고 브라우저에서 인증을 완료하는 브라우저 기반 SSO 흐름에 적합합니다.

`awsCredentialExport`: `.aws` 를 수정할 수 없고 자격 증명을 직접 반환해야 하는 경우에만 사용하십시오. 출력은 자동으로 캡처되며 사용자에게 표시되지 않습니다. 명령은 다음 형식으로 JSON을 출력해야 합니다:

```
{
  "Credentials": {
    "AccessKeyId": "value",
    "SecretAccessKey": "value",
    "SessionToken": "value"
  }
}
```

3. Claude Code 구성

Bedrock을 활성화하려면 다음 환경 변수를 설정하십시오:

```
## Bedrock 통합 활성화
export CLAUDE_CODE_USE_BEDROCK=1
export AWS_REGION=us-east-1  # 또는 선호하는 지역

## 선택 사항: 소형/빠른 모델(Haiku)의 지역 재정의
export ANTHROPIC_SMALL_FAST_MODEL_AWS_REGION=us-west-2
```

Claude Code에 대해 Bedrock을 활성화할 때 다음을 염두에 두십시오:

- `AWS_REGION` 은 필수 환경 변수입니다. Claude Code는 이 설정에 대해 `.aws` 구성 파일을 읽지 않습니다.
- Bedrock을 사용할 때 `/login` 및 `/logout` 명령은 AWS 자격 증명을 통해 인증이 처리되므로 비활성화됩니다.
- 다른 프로세스에 유출되지 않도록 하려는 `AWS_PROFILE` 과 같은 환경 변수에 설정 파일을 사용할 수 있습니다. 자세한 내용은 설정을 참조하십시오.

4. 모델 버전 고정

> **Warning:**
>
> 모든 배포에 대해 특정 모델 버전을 고정하십시오. 모델 별칭(`sonnet` , `opus` , `haiku`)을 고정하지 않고 사용하면 Claude Code가 Bedrock 계정에서 사용할 수 없는 최신 모델 버전을 사용하려고 시도하여 Anthropic이 업데이트를 출시할 때 기존 사용자가 중단될 수 있습니다.

이러한 환경 변수를 특정 Bedrock 모델 ID로 설정하십시오:

```
export ANTHROPIC_DEFAULT_OPUS_MODEL='us.anthropic.claude-opus-4-6-v1'
export ANTHROPIC_DEFAULT_SONNET_MODEL='us.anthropic.claude-sonnet-4-6'
export ANTHROPIC_DEFAULT_HAIKU_MODEL='us.anthropic.claude-haiku-4-5-20251001-v1:0'
```

이러한 변수는 교차 지역 추론 프로필 ID(`us.` 접두사 포함)를 사용합니다. 다른 지역 접두사 또는 애플리케이션 추론 프로필을 사용하는 경우 적절히 조정하십시오. 현재 및 레거시 모델 ID는 모델 개요를 참조하십시오. 전체 환경 변수 목록은 모델 구성을 참조하십시오.

고정 변수가 설정되지 않은 경우 Claude Code는 이러한 기본 모델을 사용합니다:

모델 유형	기본값
기본 모델	`global.anthropic.claude-sonnet-4-6`
소형/빠른 모델	`us.anthropic.claude-haiku-4-5-20251001-v1:0`

모델을 추가로 사용자 정의하려면 다음 방법 중 하나를 사용하십시오:

```
## 추론 프로필 ID 사용
export ANTHROPIC_MODEL='global.anthropic.claude-sonnet-4-6'
export ANTHROPIC_SMALL_FAST_MODEL='us.anthropic.claude-haiku-4-5-20251001-v1:0'

## 애플리케이션 추론 프로필 ARN 사용
export ANTHROPIC_MODEL='arn:aws:bedrock:us-east-2:your-account-id:application-inference-profile/your-model-id'

## 선택 사항: 필요한 경우 프롬프트 캐싱 비활성화
export DISABLE_PROMPT_CACHING=1
```

> **Note:**
>
> 프롬프트 캐싱은 모든 지역에서 사용할 수 없을 수 있습니다.

각 모델 버전을 추론 프로필에 매핑

`ANTHROPIC_DEFAULT_*_MODEL` 환경 변수는 모델 제품군당 하나의 추론 프로필을 구성합니다. 조직이 `/model` 선택기에서 동일한 제품군의 여러 버전을 노출하고 각각 자신의 애플리케이션 추론 프로필 ARN으로 라우팅해야 하는 경우 설정 파일에서 `modelOverrides` 설정을 대신 사용하십시오.

이 예제는 세 개의 Opus 버전을 고유한 ARN에 매핑하므로 사용자는 조직의 추론 프로필을 우회하지 않고 버전 간에 전환할 수 있습니다:

```
{
  "modelOverrides": {
    "claude-opus-4-6": "arn:aws:bedrock:us-east-2:123456789012:application-inference-profile/opus-46-prod",
    "claude-opus-4-5-20251101": "arn:aws:bedrock:us-east-2:123456789012:application-inference-profile/opus-45-prod",
    "claude-opus-4-1-20250805": "arn:aws:bedrock:us-east-2:123456789012:application-inference-profile/opus-41-prod"
  }
}
```

사용자가 `/model`에서 이러한 버전 중 하나를 선택하면 Claude Code는 매핑된 ARN으로 Bedrock을 호출합니다. 재정의가 없는 버전은 기본 제공 Bedrock 모델 ID 또는 시작 시 발견된 일치하는 추론 프로필로 폴백됩니다. 재정의가 `availableModels` 및 기타 모델 설정과 상호 작용하는 방식에 대한 자세한 내용은 버전별 모델 ID 재정의를 참조하십시오.

IAM 구성

Claude Code에 필요한 권한이 있는 IAM 정책을 만드십시오:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowModelAndInferenceProfileAccess",
      "Effect": "Allow",
      "Action": [
        "bedrock:InvokeModel",
        "bedrock:InvokeModelWithResponseStream",
        "bedrock:ListInferenceProfiles"
      ],
      "Resource": [
        "arn:aws:bedrock:*:*:inference-profile/*",
        "arn:aws:bedrock:*:*:application-inference-profile/*",
        "arn:aws:bedrock:*:*:foundation-model/*"
      ]
    },
    {
      "Sid": "AllowMarketplaceSubscription",
      "Effect": "Allow",
      "Action": [
        "aws-marketplace:ViewSubscriptions",
        "aws-marketplace:Subscribe"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:CalledViaLast": "bedrock.amazonaws.com"
        }
      }
    }
  ]
}
```

더 제한적인 권한의 경우 리소스를 특정 추론 프로필 ARN으로 제한할 수 있습니다.

자세한 내용은 Bedrock IAM 설명서를 참조하십시오.

> **Note:**
> 비용 추적 및 액세스 제어를 단순화하기 위해 Claude Code용 전용 AWS 계정을 만드십시오.

AWS Guardrails

Amazon Bedrock Guardrails를 사용하면 Claude Code에 대한 콘텐츠 필터링을 구현할 수 있습니다. Amazon Bedrock 콘솔에서 Guardrail을 만들고 버전을 게시한 다음 Guardrail 헤더를 설정 파일에 추가하십시오. 교차 지역 추론 프로필을 사용하는 경우 Guardrail에서 교차 지역 추론을 활성화하십시오.

예제 구성:

```
{
  "env": {
    "ANTHROPIC_CUSTOM_HEADERS": "X-Amzn-Bedrock-GuardrailIdentifier: your-
guardrail-id\nX-Amzn-Bedrock-GuardrailVersion: 1"
  }
}
```

문제 해결

지역 문제가 발생하는 경우:

- 모델 가용성 확인: `aws bedrock list-inference-profiles --region your-region`
- 지원되는 지역으로 전환: `export AWS_REGION=us-east-1`
- 교차 지역 액세스를 위해 추론 프로필 사용 고려

“on-demand throughput isn’t supported” 오류가 발생하는 경우:

- 모델을 추론 프로필 ID로 지정하십시오

Claude Code는 Bedrock Invoke API를 사용하며 Converse API를 지원하지 않습니다.

추가 리소스

- Bedrock 설명서
- Bedrock 가격
- Bedrock 추론 프로필
- Claude Code on Amazon Bedrock: Quick Setup Guide
- Claude Code Monitoring Implementation (Bedrock)

Google Vertex AI에서 Claude Code 사용하기

> Google Vertex AI를 통한 Claude Code 구성, 설정, IAM 구성 및 문제 해결에 대해 알아봅니다.

필수 요구사항

Claude Code를 Vertex AI로 구성하기 전에 다음을 확인하십시오:

- 청구가 활성화된 Google Cloud Platform(GCP) 계정
- Vertex AI API가 활성화된 GCP 프로젝트
- 원하는 Claude 모델에 대한 액세스(예: Claude Sonnet 4.5)
- Google Cloud SDK(`gcloud`) 설치 및 구성
- 원하는 GCP 지역에 할당된 할당량

지역 구성

Claude Code는 Vertex AI 글로벌 및 지역 엔드포인트 모두에서 사용할 수 있습니다.

> **Note:**
>
> Vertex AI는 모든 지역에서 Claude Code 기본 모델을 지원하지 않을 수 있습니다. 지원되는 지역 또는 모델로 전환해야 할 수 있습니다.

> **Note:**
>
> Vertex AI는 글로벌 엔드포인트에서 Claude Code 기본 모델을 지원하지 않을 수 있습니다. 지역 엔드포인트 또는 지원되는 모델로 전환해야 할 수 있습니다.

설정

1. Vertex AI API 활성화

GCP 프로젝트에서 Vertex AI API를 활성화합니다:

```
## 프로젝트 ID 설정
gcloud config set project YOUR-PROJECT-ID

## Vertex AI API 활성화
gcloud services enable aiplatform.googleapis.com
```

2. 모델 액세스 요청

Vertex AI에서 Claude 모델에 대한 액세스를 요청합니다:

1. Vertex AI Model Garden으로 이동합니다
2. “Claude” 모델을 검색합니다
3. 원하는 Claude 모델에 대한 액세스를 요청합니다(예: Claude Sonnet 4.5)
4. 승인을 기다립니다(24-48시간이 소요될 수 있습니다)

3. GCP 자격증명 구성

Claude Code는 표준 Google Cloud 인증을 사용합니다.

자세한 내용은 Google Cloud 인증 설명서를 참조하십시오.

> **Note:**
>
> 인증할 때 Claude Code는 `ANTHROPIC_VERTEX_PROJECT_ID` 환경 변수에서 프로젝트 ID를 자동으로 사용합니다. 이를 재정의하려면 다음 환경 변수 중 하나를 설정하십시오: `GCLOUD_PROJECT`, `GOOGLE_CLOUD_PROJECT` 또는 `GOOGLE_APPLICATION_CREDENTIALS`.

4. Claude Code 구성

다음 환경 변수를 설정합니다:

```
## Vertex AI 통합 활성화
export CLAUDE_CODE_USE_VERTEX=1
export CLOUD_ML_REGION=global
export ANTHROPIC_VERTEX_PROJECT_ID=YOUR-PROJECT-ID

## 선택사항: 필요한 경우 prompt caching 비활성화
export DISABLE_PROMPT_CACHING=1

## CLOUD_ML_REGION=global일 때 지원되지 않는 모델의 지역 재정의
export VERTEX_REGION_CLAUDE_3_5_HAIKU=us-east5

## 선택사항: 다른 특정 모델의 지역 재정의
export VERTEX_REGION_CLAUDE_3_5_SONNET=us-east5
export VERTEX_REGION_CLAUDE_3_7_SONNET=us-east5
export VERTEX_REGION_CLAUDE_4_0_OPUS=europe-west1
export VERTEX_REGION_CLAUDE_4_0_SONNET=us-east5
export VERTEX_REGION_CLAUDE_4_1_OPUS=europe-west1
```

> **Note:**
>
> Prompt caching은 `cache_control` 임시 플래그를 지정할 때 자동으로 지원됩니다. 이를 비활성화하려면 `DISABLE_PROMPT_CACHING=1` 을 설정하십시오. 높은 속도 제한을 위해 Google Cloud 지원팀에 문의하십시오.

> **Note:**
>
> Vertex AI를 사용할 때 `/login` 및 `/logout` 명령은 Google Cloud 자격증명을 통해 인증이 처리되므로 비활성화됩니다.

5. 모델 구성

Claude Code는 Vertex AI에 대해 다음 기본 모델을 사용합니다:

모델 유형	기본값
주 모델	`claude-sonnet-4-5@20250929`
소형/빠른 모델	`claude-haiku-4-5@20251001`

> **Note:**
>
> Vertex AI 사용자의 경우 Claude Code는 Haiku 3.5에서 Haiku 4.5로 자동 업그레이드되지 않습니다. 최신 Haiku 모델로 수동으로 전환하려면 `ANTHROPIC_DEFAULT_HAIKU_MODEL` 환경 변수를 전체 모델 이름으로 설정하십시오(예: `claude-haiku-4-5@20251001`).

모델을 사용자 정의하려면:

```
export ANTHROPIC_MODEL='claude-opus-4-1@20250805'
export ANTHROPIC_SMALL_FAST_MODEL='claude-haiku-4-5@20251001'
```

IAM 구성

필요한 IAM 권한을 할당합니다:

`roles/aiplatform.user` 역할에는 필요한 권한이 포함됩니다:

- `aiplatform.endpoints.predict` - 모델 호출 및 토큰 계산에 필요

더 제한적인 권한의 경우 위의 권한만 포함하는 사용자 정의 역할을 만듭니다.

자세한 내용은 Vertex IAM 설명서를 참조하십시오.

> **Note:**
>
> 비용 추적 및 액세스 제어를 단순화하기 위해 Claude Code용 전용 GCP 프로젝트를 만드는 것을 권장합니다.

1M 토큰 context window

Claude Sonnet 4 및 Sonnet 4.5는 Vertex AI에서 1M 토큰 context window를 지원합니다.

> **Note:**
>
> 1M 토큰 context window는 현재 베타 버전입니다. 확장된 context window를 사용하려면 Vertex AI 요청에 `context-1m-2025-08-07` 베타 헤더를 포함하십시오.

문제 해결

할당량 문제가 발생하는 경우:

- Cloud Console을 통해 현재 할당량을 확인하거나 할당량 증가를 요청합니다

"모델을 찾을 수 없음" 404 오류가 발생하는 경우:

- Model Garden에서 모델이 활성화되어 있는지 확인합니다
- 지정된 지역에 액세스할 수 있는지 확인합니다
- `CLOUD_ML_REGION=global`을 사용하는 경우 Model Garden의 "Supported features"에서 모델이 글로벌 엔드포인트를 지원하는지 확인합니다. 글로벌 엔드포인트를 지원하지 않는 모델의 경우:
  - `ANTHROPIC_MODEL` 또는 `ANTHROPIC_SMALL_FAST_MODEL`을 통해 지원되는 모델을 지정하거나
  - `VERTEX_REGION_<MODEL_NAME>` 환경 변수를 사용하여 지역 엔드포인트를 설정합니다

429 오류가 발생하는 경우:

- 지역 엔드포인트의 경우 주 모델과 소형/빠른 모델이 선택한 지역에서 지원되는지 확인합니다
- 더 나은 가용성을 위해 `CLOUD_ML_REGION=global`로 전환하는 것을 고려합니다

추가 리소스

- Vertex AI 설명서
- Vertex AI 가격
- Vertex AI 할당량 및 제한

Microsoft Foundry의 Claude Code

> 설정, 구성 및 문제 해결을 포함하여 Microsoft Foundry를 통해 Claude Code를 구성하는 방법을 알아봅니다.

필수 조건

Microsoft Foundry로 Claude Code를 구성하기 전에 다음을 확인하세요:

- Microsoft Foundry에 액세스할 수 있는 Azure 구독
- Microsoft Foundry 리소스 및 배포를 만들 수 있는 RBAC 권한
- Azure CLI 설치 및 구성(선택 사항 - 자격 증명을 얻을 다른 메커니즘이 없는 경우에만 필요)

설정

1. Microsoft Foundry 리소스 프로비저닝

먼저 Azure에서 Claude 리소스를 만듭니다:

1. Microsoft Foundry 포털로 이동합니다
2. 새 리소스를 만들고 리소스 이름을 기록합니다
3. Claude 모델에 대한 배포를 만듭니다:
    - Claude Opus
    - Claude Sonnet
    - Claude Haiku

2. Azure 자격 증명 구성

Claude Code는 Microsoft Foundry에 대해 두 가지 인증 방법을 지원합니다. 보안 요구 사항에 가장 적합한 방법을 선택하세요.

옵션 A: API 키 인증

1. Microsoft Foundry 포털에서 리소스로 이동합니다
2. **엔드포인트 및 키** 섹션으로 이동합니다
3. **API 키** 복사합니다
4. 환경 변수를 설정합니다:

```
export ANTHROPIC_FOUNDRY_API_KEY=your-azure-api-key
```

옵션 B: Microsoft Entra ID 인증

`ANTHROPIC_FOUNDRY_API_KEY`가 설정되지 않으면 Claude Code는 자동으로 Azure SDK 기본 자격 증명 체인을 사용합니다. 이는 로컬 및 원격 워크로드를 인증하기 위한 다양한 방법을 지원합니다.

로컬 환경에서는 일반적으로 Azure CLI를 사용할 수 있습니다:

```
az login
```

> **Note:**
>
> Microsoft Foundry를 사용할 때 `/login` 및 `/logout` 명령은 Azure 자격 증명을 통해 인증이 처리되므로 비활성화됩니다.

3. Claude Code 구성

Microsoft Foundry를 활성화하려면 다음 환경 변수를 설정합니다. 배포의 이름은 Claude Code의 모델 식별자로 설정됩니다(제안된 배포 이름을 사용하는 경우 선택 사항일 수 있음).

```
## Microsoft Foundry 통합 활성화
export CLAUDE_CODE_USE_FOUNDRY=1

## Azure 리소스 이름 ({resource}를 리소스 이름으로 바꾸기)
export ANTHROPIC_FOUNDRY_RESOURCE={resource}
## 또는 전체 기본 URL 제공:
## export ANTHROPIC_FOUNDRY_BASE_URL=https://{resource}.services.ai.azure.com

## 모델을 리소스의 배포 이름으로 설정
export ANTHROPIC_DEFAULT_SONNET_MODEL='claude-sonnet-4-5'
export ANTHROPIC_DEFAULT_HAIKU_MODEL='claude-haiku-4-5'
export ANTHROPIC_DEFAULT_OPUS_MODEL='claude-opus-4-1'
```

모델 구성 옵션에 대한 자세한 내용은 모델 구성을 참조하세요.

Azure RBAC 구성

`Azure AI User` 및 `Cognitive Services User` 기본 역할에는 Claude 모델을 호출하는 데 필요한 모든 권한이 포함됩니다.

더 제한적인 권한의 경우 다음을 포함하는 사용자 지정 역할을 만듭니다:

```
{
  "permissions": [
    {
      "dataActions": [
        "Microsoft.CognitiveServices/accounts/providers/*"
      ]
    }
  ]
}
```

자세한 내용은 Microsoft Foundry RBAC 설명서를 참조하세요.

문제 해결

"Failed to get token from azureADTokenProvider: ChainedTokenCredential authentication failed" 오류가 발생하면:

- 환경에서 Entra ID를 구성하거나 `ANTHROPIC_FOUNDRY_API_KEY` 를 설정합니다.

추가 리소스

- Microsoft Foundry 설명서
- Microsoft Foundry 모델
- Microsoft Foundry 가격

LLM gateway 구성

> Claude Code를 LLM gateway 솔루션과 함께 작동하도록 구성하는 방법을 알아봅니다. Gateway 요구사항, 인증 구성, 모델 선택 및 공급자별 엔드포인트 설정을 다룹니다.

LLM gateway는 Claude Code와 모델 공급자 간의 중앙 집중식 프록시 계층을 제공하며, 종종 다음을 제공합니다:

- **중앙 집중식 인증** - API 키 관리를 위한 단일 지점
- **사용량 추적** - 팀 및 프로젝트 전체의 사용량 모니터링
- **비용 제어** - 예산 및 속도 제한 구현
- **감사 로깅** - 규정 준수를 위한 모든 모델 상호작용 추적
- **모델 라우팅** - 코드 변경 없이 공급자 간 전환

Gateway 요구사항

LLM gateway가 Claude Code와 함께 작동하려면 다음 요구사항을 충족해야 합니다:

API 형식

Gateway는 클라이언트에 다음 API 형식 중 최소 하나를 노출해야 합니다:

1. **Anthropic Messages**: `/v1/messages`, `/v1/messages/count_tokens`
   - 요청 헤더를 전달해야 함: `anthropic-beta`, `anthropic-version`
2. **Bedrock InvokeModel**: `/invoke`, `/invoke-with-response-stream`
   - 요청 본문 필드를 보존해야 함: `anthropic_beta`, `anthropic_version`
3. **Vertex rawPredict**: `:rawPredict`, `:streamRawPredict`, `/count-tokens:rawPredict`
   - 요청 헤더를 전달해야 함: `anthropic-beta`, `anthropic-version`

헤더를 전달하지 않거나 본문 필드를 보존하지 않으면 기능이 감소하거나 Claude Code 기능을 사용할 수 없을 수 있습니다.

> **Note:**
>
> Claude Code는 API 형식을 기반으로 활성화할 기능을 결정합니다. Bedrock 또는 Vertex와 함께 Anthropic Messages 형식을 사용할 때 환경 변수 `CLAUDE_CODE_DISABLE_EXPERIMENTAL_BETAS=1`을 설정해야 할 수 있습니다.

구성

모델 선택

기본적으로 Claude Code는 선택한 API 형식에 대해 표준 모델 이름을 사용합니다.

Gateway에서 사용자 정의 모델 이름을 구성한 경우 모델 구성에 문서화된 환경 변수를 사용하여 사용자 정의 이름과 일치시킵니다.

LiteLLM 구성

> **Note:**
>
> LiteLLM은 제3자 프록시 서비스입니다. Anthropic은 LiteLLM의 보안 또는 기능을 보증, 유지 관리 또는 감사하지 않습니다. 이 가이드는 정보 제공 목적으로 제공되며 오래될 수 있습니다. 자신의 판단에 따라 사용하십시오.

필수 조건

- 최신 버전으로 업데이트된 Claude Code
- 배포되고 액세스 가능한 LiteLLM Proxy Server
- 선택한 공급자를 통한 Claude 모델 액세스

기본 LiteLLM 설정

Claude Code 구성:

인증 방법

정적 API 키

고정 API 키를 사용한 가장 간단한 방법:

```
## 환경에서 설정
export ANTHROPIC_AUTH_TOKEN=sk-litellm-static-key

## 또는 Claude Code 설정에서
{
  "env": {
    "ANTHROPIC_AUTH_TOKEN": "sk-litellm-static-key"
  }
}
```

이 값은 `Authorization` 헤더로 전송됩니다.

헬퍼를 사용한 동적 API 키

회전하는 키 또는 사용자별 인증의 경우:

1. API 키 헬퍼 스크립트를 만듭니다:

```
#!/bin/bash
## ~/bin/get-litellm-key.sh

## 예: 자격 증명 모음에서 키 가져오기
vault kv get -field=api_key secret/litellm/claude-code

## 예: JWT 토큰 생성
jwt encode \
  --secret="${JWT_SECRET}" \
  --exp="+1h" \
  '{"user":"'${USER}'","team":"engineering"}'
```

1. 헬퍼를 사용하도록 Claude Code 설정을 구성합니다:

```
{
  "apiKeyHelper": "~/bin/get-litellm-key.sh"
}
```

1. 토큰 새로고침 간격을 설정합니다:

```
## 1시간마다 새로고침 (3600000 ms)
export CLAUDE_CODE_API_KEY_HELPER_TTL_MS=3600000
```

이 값은 `Authorization` 및 `X-Api-Key` 헤더로 전송됩니다. `apiKeyHelper`는 `ANTHROPIC_AUTH_TOKEN` 또는 `ANTHROPIC_API_KEY`보다 우선순위가 낮습니다.

통합 엔드포인트 (권장)

LiteLLM의 Anthropic 형식 엔드포인트 사용:

```
export ANTHROPIC_BASE_URL=https://litellm-server:4000
```

통합 엔드포인트의 통과 엔드포인트 대비 이점:

- 로드 밸런싱
- 폴백
- 비용 추적 및 최종 사용자 추적에 대한 일관된 지원

공급자별 통과 엔드포인트 (대안)

LiteLLM을 통한 Claude API

통과 엔드포인트 사용:

```
export ANTHROPIC_BASE_URL=https://litellm-server:4000/anthropic
```

LiteLLM을 통한 Amazon Bedrock

통과 엔드포인트 사용:

```
export ANTHROPIC_BEDROCK_BASE_URL=https://litellm-server:4000/bedrock
export CLAUDE_CODE_SKIP_BEDROCK_AUTH=1
export CLAUDE_CODE_USE_BEDROCK=1
```

LiteLLM을 통한 Google Vertex AI

통과 엔드포인트 사용:

```
export ANTHROPIC_VERTEX_BASE_URL=https://litellm-server:4000/vertex_ai/v1
export ANTHROPIC_VERTEX_PROJECT_ID=your-gcp-project-id
export CLAUDE_CODE_SKIP_VERTEX_AUTH=1
export CLAUDE_CODE_USE_VERTEX=1
export CLOUD_ML_REGION=us-east5
```

더 자세한 정보는 LiteLLM 문서를 참조하십시오.

추가 리소스

- LiteLLM 문서
- Claude Code 설정

- 엔터프라이즈 네트워크 구성
- 제3자 통합 개요

엔터프라이즈 네트워크 구성

> 프록시 서버, 사용자 정의 인증 기관(CA), 상호 전송 계층 보안(mTLS) 인증을 통해 엔터프라이즈 환경에서 Claude Code를 구성합니다.

Claude Code는 환경 변수를 통해 다양한 엔터프라이즈 네트워크 및 보안 구성을 지원합니다. 여기에는 기업 프록시 서버를 통한 트래픽 라우팅, 사용자 정의 인증 기관(CA) 신뢰, 향상된 보안을 위한 상호 전송 계층 보안(mTLS) 인증서를 사용한 인증이 포함됩니다.

> **Note:**
>
> 이 페이지에 표시된 모든 환경 변수는 `settings.json` 에서도 구성할 수 있습니다.

프록시 구성

환경 변수

Claude Code는 표준 프록시 환경 변수를 준수합니다:

```
## HTTPS 프록시 (권장)
export HTTPS_PROXY=https://proxy.example.com:8080

## HTTP 프록시 (HTTPS를 사용할 수 없는 경우)
export HTTP_PROXY=http://proxy.example.com:8080

## 특정 요청에 대해 프록시 우회 - 공백으로 구분된 형식
export NO_PROXY="localhost 192.168.1.1 example.com .example.com"
## 특정 요청에 대해 프록시 우회 - 쉼표로 구분된 형식
export NO_PROXY="localhost,192.168.1.1,example.com,.example.com"
## 모든 요청에 대해 프록시 우회
export NO_PROXY="*"
```

> **Note:**
>
> Claude Code는 SOCKS 프록시를 지원하지 않습니다.

기본 인증

프록시에 기본 인증이 필요한 경우 프록시 URL에 자격 증명을 포함합니다:

```
export HTTPS_PROXY=http://username:password@proxy.example.com:8080
```

> **Warning:**
>
> 스크립트에 암호를 하드코딩하지 마십시오. 대신 환경 변수 또는 보안 자격 증명 저장소를 사용하십시오.

> **Tip:**
>
> 고급 인증(NTLM, Kerberos 등)이 필요한 프록시의 경우 인증 방법을 지원하는 LLM Gateway 서비스 사용을 고려하십시오.

사용자 정의 CA 인증서

엔터프라이즈 환경에서 HTTPS 연결을 위해 사용자 정의 CA를 사용하는 경우(프록시를 통하든 직접 API 액세스를 통하든) Claude Code를 구성하여 이를 신뢰하도록 합니다:

```
export NODE_EXTRA_CA_CERTS=/path/to/ca-cert.pem
```

mTLS 인증

클라이언트 인증서 인증이 필요한 엔터프라이즈 환경의 경우:

```
## 인증용 클라이언트 인증서
export CLAUDE_CODE_CLIENT_CERT=/path/to/client-cert.pem

## 클라이언트 개인 키
export CLAUDE_CODE_CLIENT_KEY=/path/to/client-key.pem

## 선택 사항: 암호화된 개인 키의 암호
export CLAUDE_CODE_CLIENT_KEY_PASSPHRASE="your-passphrase"
```

네트워크 액세스 요구 사항

Claude Code는 다음 URL에 대한 액세스가 필요합니다:

- `api.anthropic.com` : Claude API 엔드포인트
- `claude.ai` : claude.ai 계정 인증
- `platform.claude.com` : Anthropic Console 계정 인증

프록시 구성 및 방화벽 규칙에서 이러한 URL이 허용 목록에 있는지 확인하십시오. 이는 특히 컨테이너화되거나 제한된 네트워크 환경에서 Claude Code를 사용할 때 중요합니다.

추가 리소스

- Claude Code 설정
- 환경 변수 참조
- 문제 해결 가이드

Part 12: Environment Setup

개발 컨테이너

> 일관된 보안 환경이 필요한 팀을 위한 Claude Code 개발 컨테이너에 대해 알아보세요.

참조 devcontainer 설정과 관련 Dockerfile은 그대로 사용하거나 필요에 맞게 사용자 정의할 수 있는 사전 구성된 개발 컨테이너를 제공합니다. 이 devcontainer는 Visual Studio Code Dev Containers 확장과 유사한 도구와 함께 작동합니다.

컨테이너의 향상된 보안 조치(격리 및 방화벽 규칙)를 통해 `claude --dangerously-skip-permissions` 을 실행하여 무인 작동을 위한 권한 프롬프트를 우회할 수 있습니다.

> **Warning:**
>
> devcontainer가 상당한 보호를 제공하지만, 모든 공격에 완전히 면역인 시스템은 없습니다. `--dangerously-skip-permissions` 으로 실행할 때, devcontainer는 Claude Code 자격 증명을 포함하여 devcontainer에서 접근 가능한 모든 것을 악의적인 프로젝트가 유출하는 것을 방지하지 않습니다. 신뢰할 수 있는 저장소로 개발할 때만 devcontainer를 사용하는 것을 권장합니다. 항상 좋은 보안 관행을 유지하고 Claude의 활동을 모니터링하세요.

주요 기능

- **프로덕션 준비 완료 Node.js**: 필수 개발 종속성이 포함된 Node.js 20을 기반으로 구축
- **설계상 보안**: 필요한 서비스로만 네트워크 접근을 제한하는 사용자 정의 방화벽
- **개발자 친화적 도구**: git, 생산성 향상 기능이 있는 ZSH, fzf 등 포함
- **원활한 VS Code 통합**: 사전 구성된 확장 및 최적화된 설정
- **세션 지속성**: 컨테이너 재시작 간 명령 기록 및 구성 보존
- **어디서나 작동**: macOS, Windows 및 Linux 개발 환경과 호환

4단계로 시작하기

1. VS Code 및 Remote - Containers 확장 설치
2. Claude Code 참조 구현 저장소 복제
3. VS Code에서 저장소 열기

4. 메시지가 표시되면 “Reopen in Container” 클릭(또는 Command Palette 사용: Cmd+Shift+P → “Remote-Containers: Reopen in Container”)

구성 분석

devcontainer 설정은 세 가지 주요 구성 요소로 구성됩니다:

- **devcontainer.json**: 컨테이너 설정, 확장 및 볼륨 마운트 제어
- **Dockerfile**: 컨테이너 이미지 및 설치된 도구 정의
- **init-firewall.sh**: 네트워크 보안 규칙 설정

보안 기능

컨테이너는 방화벽 구성을 통해 다층 보안 접근 방식을 구현합니다:

- **정확한 접근 제어**: 아웃바운드 연결을 화이트리스트된 도메인으로만 제한(npm 레지스트리, GitHub, Claude API 등)
- **허용된 아웃바운드 연결**: 방화벽은 아웃바운드 DNS 및 SSH 연결을 허용합니다
- **기본 거부 정책**: 다른 모든 외부 네트워크 접근 차단
- **시작 확인**: 컨테이너 초기화 시 방화벽 규칙 검증
- **격리**: 주 시스템과 분리된 보안 개발 환경 생성

사용자 정의 옵션

devcontainer 구성은 필요에 맞게 조정할 수 있도록 설계되었습니다:

- 워크플로우에 따라 VS Code 확장 추가 또는 제거
- 다양한 하드웨어 환경을 위한 리소스 할당 수정
- 네트워크 접근 권한 조정
- 셸 구성 및 개발자 도구 사용자 정의

사용 사례 예시

보안 클라이언트 작업

devcontainer를 사용하여 다양한 클라이언트 프로젝트를 격리하여 코드와 자격 증명이 환경 간에 혼합되지 않도록 합니다.

팀 온보딩

새 팀 원들은 모든 필요한 도구와 설정이 사전 설치된 완전히 구성된 개발 환경을 몇 분 내에 얻을 수 있습니다.

일관된 CI/CD 환경

devcontainer 구성을 CI/CD 파이프라인에 반영하여 개발 및 프로덕션 환경이 일치하도록 합니다.

관련 리소스

- VS Code devcontainers 문서
- Claude Code 보안 모범 사례
- 엔터프라이즈 네트워크 구성

터미널 설정 최적화

> Claude Code는 터미널이 제대로 구성되었을 때 최적으로 작동합니다. 이 지침을 따라 환경을 최적화하세요.

테마 및 모양

Claude는 터미널의 테마를 제어할 수 없습니다. 이는 터미널 애플리케이션에서 처리됩니다. `/config` 명령을 통해 언제든지 Claude Code의 테마를 터미널과 일치시킬 수 있습니다.

Claude Code 인터페이스 자체를 추가로 사용자 정의하려면 사용자 정의 상태 표시줄을 구성하여 현재 모델, 작업 디렉토리 또는 git 분기와 같은 상황별 정보를 터미널 하단에 표시할 수 있습니다.

줄 바꿈

Claude Code에 줄 바꿈을 입력하는 여러 옵션이 있습니다:

- **빠른 이스케이프**: `\` 를 입력한 후 Enter를 눌러 새 줄을 만듭니다
- **Shift+Enter**: iTerm2, WezTerm, Ghostty 및 Kitty에서 기본적으로 작동합니다
- **키보드 단축키**: 다른 터미널에서 새 줄을 삽입하도록 키 바인딩을 설정합니다

다른 터미널에서 Shift+Enter 설정

Claude Code 내에서 `/terminal-setup` 을 실행하여 VS Code, Alacritty, Zed 및 Warp에 대해 Shift+Enter를 자동으로 구성합니다.

> **Note:**
>
> `/terminal-setup` 명령은 수동 구성이 필요한 터미널에서만 표시됩니다. iTerm2, WezTerm, Ghostty 또는 Kitty를 사용 중인 경우 Shift+Enter가 이미 기본적으로 작동하므로 이 명령이 표시되지 않습니다.

Option+Enter 설정 (VS Code, iTerm2 또는 macOS Terminal.app)

Mac Terminal.app의 경우:

1. 설정 → 프로필 → 키보드 열기
2. “Option을 Meta 키로 사용” 확인

iTerm2 및 VS Code 터미널의 경우:

1. 설정 → 프로필 → 키 열기

2. 일반에서 왼쪽/오른쪽 Option 키를 “Esc+”로 설정

알림 설정

적절한 알림 구성으로 Claude가 작업을 완료할 때 놓치지 마세요:

iTerm 2 시스템 알림

작업 완료 시 iTerm 2 알림의 경우:

1. iTerm 2 환경설정 열기
2. 프로필 → 터미널로 이동
3. “Silence bell” 활성화 및 필터 알림 → “이스케이프 시퀀스 생성 알림 전송”
4. 선호하는 알림 지연 설정

이러한 알림은 iTerm 2에만 해당되며 기본 macOS 터미널에서는 사용할 수 없습니다.

사용자 정의 알림 훅

고급 알림 처리의 경우 알림 훅을 만들어 자신의 로직을 실행할 수 있습니다.

큰 입력 처리

광범위한 코드 또는 긴 지침으로 작업할 때:

- **직접 붙여넣기 피하기**: Claude Code는 매우 긴 붙여넣은 콘텐츠로 어려움을 겪을 수 있습니다
- **파일 기반 워크플로우 사용**: 파일에 콘텐츠를 작성하고 Claude에 읽도록 요청합니다
- **VS Code 제한 사항 인식**: VS Code 터미널은 특히 긴 붙여넣기를 자르는 경향이 있습니다

Vim 모드

Claude Code는 `/vim` 으로 활성화하거나 `/config` 를 통해 구성할 수 있는 Vim 키 바인딩의 부분 집합을 지원합니다.

지원되는 부분 집합에는 다음이 포함됩니다:

- 모드 전환: `Esc` (NORMAL로), `i` / `I` , `a` / `A` , `o` / `O` (INSERT로)
- 네비게이션: `h` / `j` / `k` / `l` , `w` / `e` / `b` , `0` / `$` / `^` , `gg` / `G` , `f` / `F` / `t` / `T` (`;` / `,` 반복 포함)
- 편집: `x` , `dw` / `de` / `db` / `dd` / `D` , `cw` / `ce` / `cb` / `cc` / `C` , `.` (반복)
- 복사/붙여넣기: `yy` / `Y` , `yw` / `ye` / `yb` , `p` / `P`
- 텍스트 객체: `iw` / `aw` , `iW` / `aW` , `i"` / `a"` , `i'` / `a'` , `i(` / `a(` , `i[` / `a[` , `i{` / `a{`
- 들여쓰기: `>>` / `<<`
- 줄 작업: `J` (줄 결합)

완전한 참조는 대화형 모드를 참조하세요.

상태 표시줄 사용자 정의

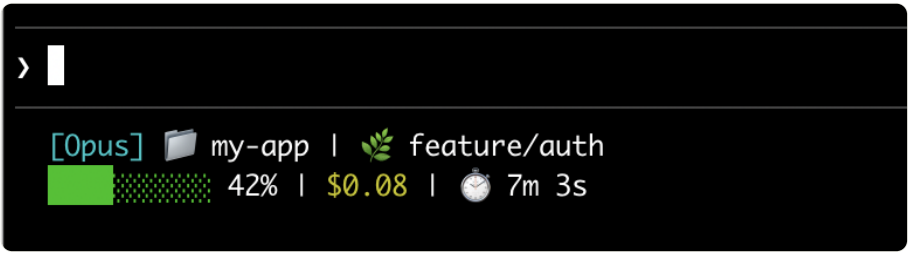
> Claude Code에서 컨텍스트 윈도우 사용량, 비용 및 git 상태를 모니터링하기 위해 사용자 정의 상태 표시줄 구성

상태 표시줄은 Claude Code 하단의 사용자 정의 가능한 표시줄로, 구성한 모든 셸 스크립트를 실행합니다. stdin을 통해 JSON 세션 데이터를 수신하고 스크립트가 출력하는 모든 내용을 표시하여 컨텍스트 사용량, 비용, git 상태 또는 추적하려는 다른 항목을 한눈에 볼 수 있는 지속적인 보기를 제공합니다.

상태 표시줄은 다음과 같은 경우에 유용합니다:

- 작업 중 컨텍스트 윈도우 사용량을 모니터링하려는 경우
- 세션 비용을 추적해야 하는 경우
- 여러 세션에서 작업하고 이들을 구분해야 하는 경우
- git 브랜치 및 상태를 항상 표시하려는 경우

다음은 첫 번째 줄에 git 정보를 표시하고 두 번째 줄에 색상으로 구분된 컨텍스트 표시줄을 표시하는 다중 줄 상태 표시줄의 예입니다.



모델 이름, 디렉토리, git 브랜치를 첫 번째 줄에 표시하고 컨텍스트 사용량 진행률 표시줄, 비용 및 기간을 두 번째 줄에 표시하는 다중 줄 상태 표시줄

이 페이지는 기본 상태 표시줄 설정을 안내하고, 데이터 흐름이 Claude Code에서 스크립트로 어떻게 흐르는지 설명하며, 표시할 수 있는 모든 필드를 나열하고, git 상태, 비용 추적 및 진행률 표시줄과 같은 일반적인 패턴에 대한 즉시 사용 가능한 예제를 제공합니다.

상태 표시줄 설정

`/statusline` 명령을 사용하여 Claude Code가 스크립트를 생성하도록 하거나, 수동으로 스크립트를 생성하여 설정에 추가합니다.

/statusline 명령 사용

`/statusline` 명령은 표시하려는 내용을 설명하는 자연어 지시사항을 허용합니다. Claude Code는 `~/.claude/` 디렉토리에 스크립트 파일을 생성하고 설정을 자동으로 업데이트합니다:

```
/statusline show model name and context percentage with a progress bar
```

상태 표시줄 수동 구성

사용자 설정(`~/.claude/settings.json`, 여기서 `~`는 홈 디렉토리) 또는 프로젝트 설정에 `statusLine` 필드를 추가합니다. `type`을 `"command"`로 설정하고 `command`를 스크립트 경로 또는 인라인 셸 명령으로 지정합니다. 스크립트 생성에 대한 전체 설명은 단계별로 상태 표시줄 빌드를 참조하세요.

```
{
  "statusLine": {
    "type": "command",
    "command": "~/.claude/statusline.sh",
    "padding": 2
  }
}
```

`command` 필드는 셸에서 실행되므로 스크립트 파일 대신 인라인 명령을 사용할 수도 있습니다. 이 예제는 `jq`를 사용하여 JSON 입력을 구문 분석하고 모델 이름 및 컨텍스트 백분율을 표시합니다:

```
{
  "statusLine": {
    "type": "command",
    "command": "jq -r '\"[\\(.model.display_name)] \\(.context_window.used_percent
age // 0)% context\"'"
  }
}
```

선택적 `padding` 필드는 상태 표시줄 콘텐츠에 추가 수평 간격(문자 단위)을 추가합니다. 기본값은 `0`입니다. 이 패딩은 인터페이스의 기본 제공 간격에 추가되므로 터미널 가장자리로부터의 절대 거리가 아닌 상대 들여쓰기를 제어합니다.

상태 표시줄 비활성화

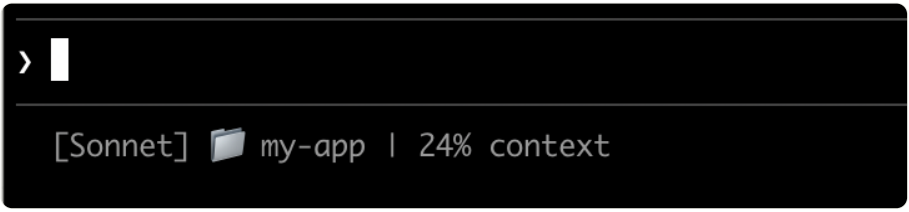
`/statusline` 을 실행하고 상태 표시줄을 제거하거나 지우도록 요청합니다(예: `/statusline delete`, `/statusline clear`, `/statusline remove it`). settings.json에서 `statusLine` 필드를 수동으로 삭제할 수도 있습니다.

단계별로 상태 표시줄 빌드

이 설명은 현재 모델, 작업 디렉토리 및 컨텍스트 윈도우 사용량 백분율을 표시하는 상태 표시줄을 수동으로 생성하여 내부에서 무엇이 일어나는지 보여줍니다.

> **Note:**
>
> `/statusline` 을 원하는 내용 설명과 함께 실행하면 이 모든 것이 자동으로 구성됩니다.

이 예제는 macOS 및 Linux에서 작동하는 Bash 스크립트를 사용합니다. Windows에서는 Windows 구성을 참조하여 PowerShell 및 Git Bash 예제를 확인하세요.



모델 이름, 디렉토리 및 컨텍스트 백분율을 표시하는 상태 표시줄

Step 1: JSON을 읽고 출력을 인쇄하는 스크립트 생성

Claude Code는 stdin을 통해 JSON 데이터를 스크립트로 보냅니다. 이 스크립트는 `jq` (설치해야 할 수 있는 명령줄 JSON 파서)를 사용하여 모델 이름, 디렉토리 및 컨텍스트 백분율을 추출한 다음 형식이 지정된 줄을 인쇄합니다.

이를 `~/.claude/statusline.sh` 에 저장합니다(여기서 `~` 는 홈 디렉토리이며, macOS에서는 `/Users/username`, Linux에서는 `/home/username`):

```
#!/bin/bash
## Claude Code가 stdin으로 보내는 JSON 데이터 읽기
input=$(cat)

## jq를 사용하여 필드 추출
MODEL=$(echo "$input" | jq -r '.model.display_name')
DIR=$(echo "$input" | jq -r '.workspace.current_dir')
## "// 0"은 필드가 null인 경우 폴백을 제공합니다
PCT=$(echo "$input" | jq -r '.context_window.used_percentage // 0' | cut -d. -f1)

## 상태 표시줄 출력 - ${DIR##*/}는 폴더 이름만 추출합니다
echo "[$MODEL] 📁 ${DIR##*/} | ${PCT}% context"
```

Step 2: 실행 가능하게 만들기

셸이 실행할 수 있도록 스크립트를 실행 가능하게 표시합니다:

```
chmod +x ~/.claude/statusline.sh
```

Step 3: 설정에 추가

Claude Code에 스크립트를 상태 표시줄로 실행하도록 지시합니다. 이 구성을 `~/.claude/settings.json`에 추가합니다. 이는 `type`을 `"command"`로 설정하고(의미: "이 셸 명령 실행") `command`를 스크립트로 지정합니다:

```
{
  "statusLine": {
    "type": "command",
    "command": "~/.claude/statusline.sh"
  }
}
```

상태 표시줄이 인터페이스 하단에 나타납니다. 설정은 자동으로 다시 로드되지만 Claude Code와의 다음 상호 작용까지 변경 사항이 나타나지 않습니다.

상태 표시줄 작동 방식

Claude Code는 스크립트를 실행하고 stdin을 통해 JSON 세션 데이터를 파이프합니다. 스크립트는 JSON을 읽고 필요한 것을 추출한 다음 stdout에 텍스트를 인쇄합니다. Claude Code는 스크립트가 인쇄하는 모든 것을 표시합니다.

업데이트 시기

스크립트는 새로운 어시스턴트 메시지 후, 권한 모드가 변경될 때 또는 vim 모드가 전환될 때 실행됩니다. 업데이트는 300ms에서 디바운스되므로 빠른 변경이 함께 배치되고 스크립트는 상황이 안정될 때 한 번 실행됩니다. 스크립트가 여전히 실행 중인 동안 새 업데이트가 트리거되면 진행 중인 실행이 취소됩니다. 스크립트를 편집하면 Claude Code와의 다음 상호 작용이 업데이트를 트리거할 때까지 변경 사항이 나타나지 않습니다.

스크립트가 출력할 수 있는 것

- **여러 줄**: 각 `echo` 또는 `print` 문은 별도의 행으로 표시됩니다. 다중 줄 예제를 참조하세요.
- **색상**: `\033[32m` (녹색)과 같은 ANSI 이스케이프 코드를 사용합니다(터미널이 지원해야 함). git 상태 예제를 참조하세요.
- **링크**: OSC 8 이스케이프 시퀀스를 사용하여 텍스트를 클릭 가능하게 만듭니다(macOS에서는 Cmd+클릭, Windows/Linux에서는 Ctrl+클릭). iTerm2, Kitty 또는 WezTerm과 같이 하이퍼링크를 지원하는 터미널이 필요합니다. 클릭 가능한 링크 예제를 참조하세요.

> **Note:**
>
> 상태 표시줄은 로컬에서 실행되며 API 토큰을 소비하지 않습니다. 자동 완성 제안, 도움말 메뉴 및 권한 프롬프트를 포함한 특정 UI 상호 작용 중에 일시적으로 숨겨집니다.

사용 가능한 데이터

Claude Code는 stdin을 통해 스크립트에 다음 JSON 필드를 보냅니다:

필드	설명
`model.id`, `model.display_name`	현재 모델 식별자 및 표시 이름
`cwd`, `workspace.current_dir`	현재 작업 디렉토리. 두 필드 모두 동일한 값을 포함합니다. `workspace.current_dir`은 `workspace.project_dir`과의 일관성을 위해 선호됩니다.
`workspace.project_dir`	Claude Code가 시작된 디렉토리로, 세션 중에 작업 디렉토리가 변경되면 `cwd`와 다를 수 있습니다

필드	설명
`cost.total_cost_usd`	USD 단위의 총 세션 비용
`cost.total_duration_ms`	세션 시작 이후의 총 벽시계 시간(밀리초)
`cost.total_api_duration_ms`	API 응답 대기에 소비된 총 시간(밀리초)
`cost.total_lines_added`, `cost.total_lines_removed`	변경된 코드 줄
`context_window.total_input_tokens`, `context_window.total_output_tokens`	세션 전체의 누적 토큰 수
`context_window.context_window_size`	최대 컨텍스트 윈도우 크기(토큰). 기본값은 200000이거나 확장된 컨텍스트가 있는 모델의 경우 1000000입니다.
`context_window.used_percentage`	사용된 컨텍스트 윈도우의 사전 계산된 백분율
`context_window.remaining_percentage`	남은 컨텍스트 윈도우의 사전 계산된 백분율
`context_window.current_usage`	마지막 API 호출의 토큰 수(컨텍스트 윈도우 필드에 설명됨)
`exceeds_200k_tokens`	가장 최근 API 응답의 총 토큰 수(입력, 캐시 및 출력 토큰 결합)가 200k를 초과하는지 여부. 이는 실제 컨텍스트 윈도우 크기와 관계없이 고정된 임계값입니다.
`session_id`	고유 세션 식별자
`transcript_path`	대화 기록 파일의 경로
`version`	Claude Code 버전
`output_style.name`	현재 출력 스타일의 이름
`vim.mode`	vim 모드가 활성화되어 있을 때 현재 vim 모드 (`NORMAL` 또는 `INSERT`)
`agent.name`	`--agent` 플래그 또는 에이전트 설정이 구성되어 있을 때 에이전트 이름
`worktree.name`	활성 worktree의 이름. `--worktree` 세션 중에만 표시됩니다

필드	설명
`worktree.path`	worktree 디렉토리의 절대 경로
`worktree.branch`	worktree의 Git 브랜치 이름(예: `"worktree-my-feature"`). 훅 기반 worktree의 경우 없음
`worktree.original_cwd`	worktree에 들어가기 전에 Claude가 있던 디렉토리
`worktree.original_branch`	worktree에 들어가기 전에 체크아웃된 Git 브랜치. 훅 기반 worktree의 경우 없음

상태 표시줄 명령은 stdin을 통해 이 JSON 구조를 수신합니다:

```
{
  "cwd": "/current/working/directory",
  "session_id": "abc123...",
  "transcript_path": "/path/to/transcript.jsonl",
  "model": {
    "id": "claude-opus-4-6",
    "display_name": "Opus"
  },
  "workspace": {
    "current_dir": "/current/working/directory",
    "project_dir": "/original/project/directory"
  },
  "version": "1.0.80",
  "output_style": {
    "name": "default"
  },
  "cost": {
    "total_cost_usd": 0.01234,
    "total_duration_ms": 45000,
    "total_api_duration_ms": 2300,
    "total_lines_added": 156,
    "total_lines_removed": 23
  },
  "context_window": {
    "total_input_tokens": 15234,
    "total_output_tokens": 4521,
    "context_window_size": 200000,
    "used_percentage": 8,
    "remaining_percentage": 92,
    "current_usage": {
      "input_tokens": 8500,
      "output_tokens": 1200,
      "cache_creation_input_tokens": 5000,
      "cache_read_input_tokens": 2000
    }
  },
  "exceeds_200k_tokens": false,
  "vim": {
```

```
    "mode": "NORMAL"
  },
  "agent": {
    "name": "security-reviewer"
  },
  "worktree": {
    "name": "my-feature",
    "path": "/path/to/.claude/worktrees/my-feature",
    "branch": "worktree-my-feature",
    "original_cwd": "/path/to/project",
    "original_branch": "main"
  }
}
```

없을 수 있는 필드 (JSON에 없음):

- `vim` : vim 모드가 활성화되어 있을 때만 나타남
- `agent` : `--agent` 플래그 또는 에이전트 설정이 구성되어 있을 때만 나타남
- `worktree` : `--worktree` 세션 중에만 나타남. 존재할 때 `branch` 및 `original_branch` 도 훅 기반 worktree의 경우 없을 수 있습니다

`null` 일 수 있는 필드:

- `context_window.current_usage` : 세션의 첫 번째 API 호출 전에 `null`
- `context_window.used_percentage` , `context_window.remaining_percentage` : 세션 초기에 `null` 일 수 있음

스크립트에서 조건부 액세스로 누락된 필드를 처리하고 null 값을 폴백 기본값으로 처리합니다.

컨텍스트 윈도우 필드

`context_window` 객체는 컨텍스트 사용량을 추적하는 두 가지 방법을 제공합니다:

- **누적 합계** (`total_input_tokens` , `total_output_tokens`): 전체 세션 전체의 모든 토큰의 합계로, 총 소비량을 추적하는 데 유용합니다
- **현재 사용량** (`current_usage`): 가장 최근 API 호출의 토큰 수로, 실제 컨텍스트 상태를 반영하므로 정확한 컨텍스트 백분율에 사용합니다

`current_usage` 객체에는 다음이 포함됩니다:

- `input_tokens` : 현재 컨텍스트의 입력 토큰
- `output_tokens` : 생성된 출력 토큰

- `cache_creation_input_tokens`: 캐시에 기록된 토큰
- `cache_read_input_tokens`: 캐시에서 읽은 토큰

`used_percentage` 필드는 입력 토큰만으로 계산됩니다: `input_tokens + cache_creation_input_tokens + cache_read_input_tokens`. `output_tokens`는 포함하지 않습니다.

`current_usage`에서 컨텍스트 백분율을 수동으로 계산하는 경우 동일한 입력 전용 공식을 사용하여 `used_percentage`와 일치시킵니다.

`current_usage` 객체는 세션의 첫 번째 API 호출 전에 `null`입니다.

예제

이 예제는 일반적인 상태 표시줄 패턴을 보여줍니다. 예제를 사용하려면:

1. 스크립트를 `~/.claude/statusline.sh`(또는 `.py` / `.js`)와 같은 파일에 저장합니다
2. 실행 가능하게 만듭니다: `chmod +x ~/.claude/statusline.sh`
3. 설정에 경로를 추가합니다

Bash 예제는 `jq`를 사용하여 JSON을 구문 분석합니다. Python 및 Node.js는 기본 제공 JSON 구문 분석을 가집니다.

컨텍스트 윈도우 사용량

현재 모델 및 컨텍스트 윈도우 사용량을 시각적 진행률 표시줄과 함께 표시합니다. 각 스크립트는 stdin에서 JSON을 읽고, `used_percentage` 필드를 추출하고, 채워진 블록(▓)이 사용량을 나타내는 10자 표시줄을 빌드합니다:

모델 이름과 백분율이 있는 진행률 표시줄을 표시하는 상태 표시줄

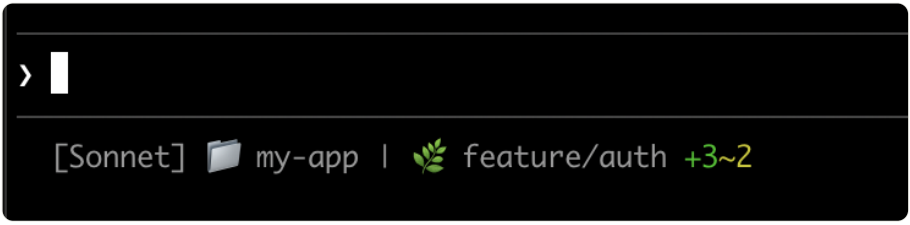
```bash
#!/bin/bash
## stdin의 모든 내용을 변수로 읽기
input=$(cat)

## jq로 필드 추출, "// 0"은 null에 대한 폴백 제공
MODEL=$(echo "$input" | jq -r '.model.display_name')
PCT=$(echo "$input" | jq -r '.context_window.used_percentage // 0' | cut -d. -f1)

## 진행률 표시줄 빌드: printf는 공백을 만들고, tr은 블록으로 바꿈
BAR_WIDTH=10
FILLED=$((PCT * BAR_WIDTH / 100))
EMPTY=$((BAR_WIDTH - FILLED))
BAR=""
[ "$FILLED" -gt 0 ] && BAR=$(printf "%${FILLED}s" | tr ' ' '▓')
[ "$EMPTY" -gt 0 ] && BAR="${BAR}$(printf "%${EMPTY}s" | tr ' ' '░')"

echo "[$MODEL] $BAR $PCT%"
```

```python
#!/usr/bin/env python3
import json, sys

## json.load는 한 단계에서 stdin을 읽고 구문 분석합니다
data = json.load(sys.stdin)
model = data['model']['display_name']
## "or 0"은 null 값을 처리합니다
pct = int(data.get('context_window', {}).get('used_percentage', 0) or 0)

## 문자열 곱셈이 표시줄을 빌드합니다
filled = pct * 10 // 100
bar = '▓' * filled + '░' * (10 - filled)

print(f"[{model}] {bar} {pct}%")
```

```
#!/usr/bin/env node
// Node.js는 이벤트로 stdin을 비동기적으로 읽습니다
let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
    const data = JSON.parse(input);
    const model = data.model.display_name;
    // 선택적 체이닝(?.)은 null 필드를 안전하게 처리합니다
    const pct = Math.floor(data.context_window?.used_percentage || 0);

    // String.repeat()이 표시줄을 빌드합니다
    const filled = Math.floor(pct * 10 / 100);
    const bar = '▓'.repeat(filled) + '░'.repeat(10 - filled);

    console.log(`[${model}] ${bar} ${pct}%`);
});
```

색상이 있는 git 상태

색상으로 구분된 스테이징 및 수정된 파일 표시기가 있는 git 브랜치를 표시합니다. 이 스크립트는 터미널 색상에 ANSI 이스케이프 코드를 사용합니다: `\033[32m`은 녹색, `\033[33m`은 노란색, `\033[0m`은 기본값으로 재설정합니다.



모델, 디렉토리, git 브랜치 및 스테이징 및 수정된 파일에 대한 색상 표시기를 표시하는 상태 표시줄

각 스크립트는 현재 디렉토리가 git 저장소인지 확인하고, 스테이징 및 수정된 파일을 계산하고, 색상으로 구분된 표시기를 표시합니다:

```
#!/bin/bash
input=$(cat)

MODEL=$(echo "$input" | jq -r '.model.display_name')
DIR=$(echo "$input" | jq -r '.workspace.current_dir')

GREEN='\033[32m'
YELLOW='\033[33m'
RESET='\033[0m'

if git rev-parse --git-dir > /dev/null 2>&1; then
    BRANCH=$(git branch --show-current 2>/dev/null)
    STAGED=$(git diff --cached --numstat 2>/dev/null | wc -l | tr -d ' ')
    MODIFIED=$(git diff --numstat 2>/dev/null | wc -l | tr -d ' ')

    GIT_STATUS=""
    [ "$STAGED" -gt 0 ] && GIT_STATUS="${GREEN}+${STAGED}${RESET}"
    [ "$MODIFIED" -gt 0 ] && GIT_STATUS="${GIT_STATUS}${YELLOW}~${MODIFIED}$
{RESET}"

    echo -e "[$MODEL] 📁 ${DIR##*/} | 🌿 $BRANCH $GIT_STATUS"
else
    echo "[$MODEL] 📁 ${DIR##*/}"
fi
```

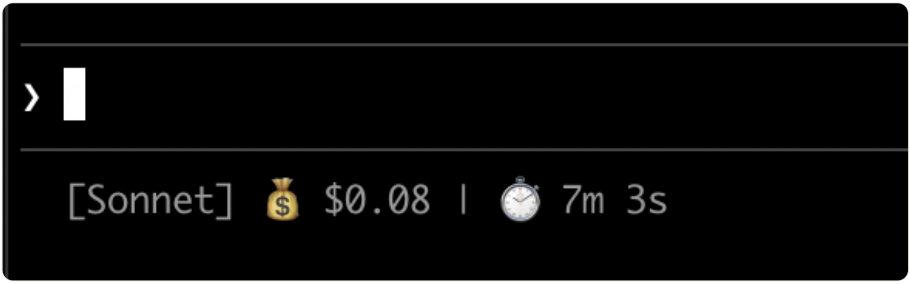
```
#!/usr/bin/env python3
import json, sys, subprocess, os

data = json.load(sys.stdin)
model = data['model']['display_name']
directory = os.path.basename(data['workspace']['current_dir'])

GREEN, YELLOW, RESET = '\033[32m', '\033[33m', '\033[0m'

try:
    subprocess.check_output(['git', 'rev-parse', '--git-dir'], stderr=subprocess.D
EVNULL)
    branch = subprocess.check_output(['git', 'branch', '--show-current'], text=Tru
e).strip()
    staged_output = subprocess.check_output(['git', 'diff', '--cached', '--
numstat'], text=True).strip()
    modified_output = subprocess.check_output(['git', 'diff', '--numstat'], text=T
rue).strip()
    staged = len(staged_output.split('\n')) if staged_output else 0
    modified = len(modified_output.split('\n')) if modified_output else 0

    git_status = f"{GREEN}+{staged}{RESET}" if staged else ""
    git_status += f"{YELLOW}~{modified}{RESET}" if modified else ""

    print(f"[{model}] 📁 {directory} | 🌿 {branch} {git_status}")
except:
    print(f"[{model}] 📁 {directory}")
```

```
#!/usr/bin/env node
const { execSync } = require('child_process');
const path = require('path');

let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
    const data = JSON.parse(input);
    const model = data.model.display_name;
    const dir = path.basename(data.workspace.current_dir);

    const GREEN = '\x1b[32m', YELLOW = '\x1b[33m', RESET = '\x1b[0m';

    try {
        execSync('git rev-parse --git-dir', { stdio: 'ignore' });
        const branch = execSync('git branch --show-current', { encoding:
'utf8' }).trim();
        const staged = execSync('git diff --cached --numstat', { encoding: 'utf8'
}).trim().split('\n').filter(Boolean).length;
        const modified = execSync('git diff --numstat', { encoding:
'utf8' }).trim().split('\n').filter(Boolean).length;

        let gitStatus = staged ? `${GREEN}+${staged}${RESET}` : '';
        gitStatus += modified ? `${YELLOW}~${modified}${RESET}` : '';

        console.log(`[${model}] 📁 ${dir} | 🌿 ${branch} ${gitStatus}`);
    } catch {
        console.log(`[${model}] 📁 ${dir}`);
    }
});
```

비용 및 기간 추적

세션의 API 비용 및 경과 시간을 추적합니다. `cost.total_cost_usd` 필드는 현재 세션의 모든 API 호출 비용을 누적합니다. `cost.total_duration_ms` 필드는 세션 시작 이후의 총 경과 시간을 측정하는 반면, `cost.total_api_duration_ms` 는 API 응답 대기에 소비된 시간만 추적합니다.

각 스크립트는 비용을 통화로 형식화하고 밀리초를 분과 초로 변환합니다:



모델 이름, 세션 비용 및 기간을 표시하는 상태 표시줄

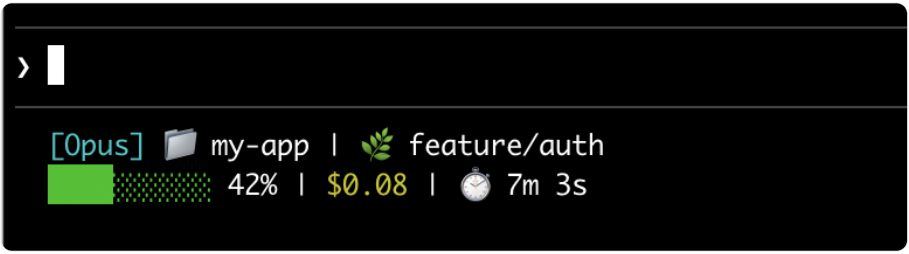
```bash
#!/bin/bash
input=$(cat)

MODEL=$(echo "$input" | jq -r '.model.display_name')
COST=$(echo "$input" | jq -r '.cost.total_cost_usd // 0')
DURATION_MS=$(echo "$input" | jq -r '.cost.total_duration_ms // 0')

COST_FMT=$(printf '$%.2f' "$COST")
DURATION_SEC=$((DURATION_MS / 1000))
MINS=$((DURATION_SEC / 60))
SECS=$((DURATION_SEC % 60))

echo "[$MODEL] 💰 $COST_FMT | ⏱️ ${MINS}m ${SECS}s"
```

```python
#!/usr/bin/env python3
import json, sys

data = json.load(sys.stdin)
model = data['model']['display_name']
cost = data.get('cost', {}).get('total_cost_usd', 0) or 0
duration_ms = data.get('cost', {}).get('total_duration_ms', 0) or 0

duration_sec = duration_ms // 1000
mins, secs = duration_sec // 60, duration_sec % 60

print(f"[{model}] 💰 ${cost:.2f} | ⏱️ {mins}m {secs}s")
```

```
#!/usr/bin/env node
let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
    const data = JSON.parse(input);
    const model = data.model.display_name;
    const cost = data.cost?.total_cost_usd || 0;
    const durationMs = data.cost?.total_duration_ms || 0;

    const durationSec = Math.floor(durationMs / 1000);
    const mins = Math.floor(durationSec / 60);
    const secs = durationSec % 60;

    console.log(`[${model}] 💰 $${cost.toFixed(2)} | ⏱️ ${mins}m ${secs}s`);
});
```

여러 줄 표시

스크립트는 여러 줄을 출력하여 더 풍부한 디스플레이를 만들 수 있습니다. 각 `echo` 문은 상태 영역에서 별도의 행을 생성합니다.



첫 번째 줄에 모델 이름, 디렉토리, git 브랜치를 표시하고 두 번째 줄에 컨텍스트 사용량 진행률 표시줄, 비용 및 기간을 표시하는 다중 줄 상태 표시줄

이 예제는 여러 기법을 결합합니다: 임계값 기반 색상(70% 미만 녹색, 70-89% 노란색, 90%+ 빨간색), 진행률 표시줄 및 git 브랜치 정보. 각 `print` 또는 `echo` 문은 별도의 행을 만듭니다:

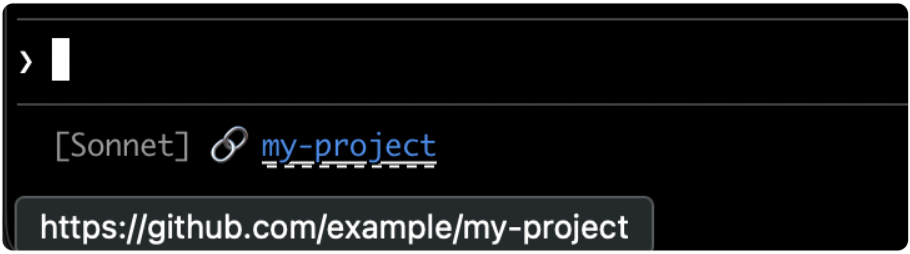
```
#!/bin/bash
input=$(cat)

MODEL=$(echo "$input" | jq -r '.model.display_name')
DIR=$(echo "$input" | jq -r '.workspace.current_dir')
COST=$(echo "$input" | jq -r '.cost.total_cost_usd // 0')
PCT=$(echo "$input" | jq -r '.context_window.used_percentage // 0' | cut -d. -f1)
DURATION_MS=$(echo "$input" | jq -r '.cost.total_duration_ms // 0')

CYAN='\033[36m'; GREEN='\033[32m'; YELLOW='\033[33m'; RED='\033[31m'; RESET='\033[
0m'

## 컨텍스트 사용량에 따라 표시줄 색상 선택
if [ "$PCT" -ge 90 ]; then BAR_COLOR="$RED"
elif [ "$PCT" -ge 70 ]; then BAR_COLOR="$YELLOW"
else BAR_COLOR="$GREEN"; fi

FILLED=$((PCT / 10)); EMPTY=$((10 - FILLED))
BAR=$(printf "%${FILLED}s" | tr ' ' '▓')$(printf "%${EMPTY}s" | tr ' ' '░')

MINS=$((DURATION_MS / 60000)); SECS=$(((DURATION_MS % 60000) / 1000))

BRANCH=""
git rev-parse --git-dir > /dev/null 2>&1 && BRANCH=" | 🌿 $(git branch --show-
current 2>/dev/null)"

echo -e "${CYAN}[$MODEL]${RESET} 📁 ${DIR##*/}$BRANCH"
COST_FMT=$(printf '$%.2f' "$COST")
echo -e "${BAR_COLOR}${BAR}${RESET} ${PCT}% | ${YELLOW}${COST_FMT}${RESET} | ⏱️ $
{MINS}m ${SECS}s"
```

```
#!/usr/bin/env python3
import json, sys, subprocess, os

data = json.load(sys.stdin)
model = data['model']['display_name']
directory = os.path.basename(data['workspace']['current_dir'])
cost = data.get('cost', {}).get('total_cost_usd', 0) or 0
pct = int(data.get('context_window', {}).get('used_percentage', 0) or 0)
duration_ms = data.get('cost', {}).get('total_duration_ms', 0) or 0

CYAN, GREEN, YELLOW, RED, RESET = '\033[36m', '\033[32m', '\033[33m', '\033[31m',
'\033[0m'

bar_color = RED if pct ≥ 90 else YELLOW if pct ≥ 70 else GREEN
filled = pct // 10
bar = '█' * filled + '░' * (10 - filled)

mins, secs = duration_ms // 60000, (duration_ms % 60000) // 1000

try:
    branch = subprocess.check_output(['git', 'branch', '--show-current'], text=Tru
e, stderr=subprocess.DEVNULL).strip()
    branch = f" | 🌿 {branch}" if branch else ""
except:
    branch = ""

print(f"{CYAN}[{model}]{RESET} 📁 {directory}{branch}")
print(f"{bar_color}{bar}{RESET} {pct}% | {YELLOW}${cost:.2f}{RESET} | ⏱️ {mins}m
{secs}s")
```

```
#!/usr/bin/env node
const { execSync } = require('child_process');
const path = require('path');

let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
    const data = JSON.parse(input);
    const model = data.model.display_name;
    const dir = path.basename(data.workspace.current_dir);
    const cost = data.cost?.total_cost_usd || 0;
    const pct = Math.floor(data.context_window?.used_percentage || 0);
    const durationMs = data.cost?.total_duration_ms || 0;

    const CYAN = '\x1b[36m', GREEN = '\x1b[32m', YELLOW = '\x1b[33m', RED = '\x1b[
31m', RESET = '\x1b[0m';

    const barColor = pct >= 90 ? RED : pct >= 70 ? YELLOW : GREEN;
    const filled = Math.floor(pct / 10);
    const bar = '▓'.repeat(filled) + '░'.repeat(10 - filled);

    const mins = Math.floor(durationMs / 60000);
    const secs = Math.floor((durationMs % 60000) / 1000);

    let branch = '';
    try {
        branch = execSync('git branch --show-current', { encoding: 'utf8', stdio:
['pipe', 'pipe', 'ignore'] }).trim();
        branch = branch ? ` | 🌿 ${branch}` : '';
    } catch {}

    console.log(`${CYAN}[${model}]${RESET} 📁 ${dir}${branch}`);
    console.log(`${barColor}${bar}${RESET} ${pct}% | ${YELLOW}$${cost.toFixed(2)}$
{RESET} | ⏱️ ${mins}m ${secs}s`);
});
```

클릭 가능한 링크

이 예제는 GitHub 저장소에 대한 클릭 가능한 링크를 만듭니다. git 원격 URL을 읽고, `sed` 를 사용하여 SSH 형식을 HTTPS로 변환하고, 저장소 이름을 OSC 8 이스케이프 코드로 래핑합니다. Cmd(macOS) 또는 Ctrl(Windows/Linux)을 누르고 클릭하여 브라우저에서 링크를 엽니다.



GitHub 저장소에 대한 클릭 가능한 링크를 표시하는 상태 표시줄

각 스크립트는 git 원격 URL을 가져오고, SSH 형식을 HTTPS로 변환하고, 저장소 이름을 OSC 8 이스케이프 코드로 래핑합니다. Bash 버전은 `printf '%b'` 를 사용하여 다양한 셸에서 백슬래시 이스케이프를 더 안정적으로 해석합니다:

```
#!/bin/bash
input=$(cat)

MODEL=$(echo "$input" | jq -r '.model.display_name')

## git SSH URL을 HTTPS로 변환
REMOTE=$(git remote get-url origin 2>/dev/null | sed 's/git@github.com:/https:\/\/
github.com\//' | sed 's/\.git$//')

if [ -n "$REMOTE" ]; then
    REPO_NAME=$(basename "$REMOTE")
    # OSC 8 형식: \e]8;;URL\a then TEXT then \e]8;;\a
    # printf %b는 셸 전체에서 이스케이프 시퀀스를 안정적으로 해석합니다
    printf '%b' "[$MODEL] 🔗 \e]8;;${REMOTE}\a${REPO_NAME}\e]8;;\a\n"
else
    echo "[$MODEL]"
fi
```

```
#!/usr/bin/env python3
import json, sys, subprocess, re, os

data = json.load(sys.stdin)
model = data['model']['display_name']

## git 원격 URL 가져오기
try:
    remote = subprocess.check_output(
        ['git', 'remote', 'get-url', 'origin'],
        stderr=subprocess.DEVNULL, text=True
    ).strip()
    # SSH를 HTTPS 형식으로 변환
    remote = re.sub(r'^git@github\.com:', 'https://github.com/', remote)
    remote = re.sub(r'\.git$', '', remote)
    repo_name = os.path.basename(remote)
    # OSC 8 이스케이프 시퀀스
    link = f"\033]8;;{remote}\a{repo_name}\033]8;;\a"
    print(f"[{model}] 🔗 {link}")
except:
    print(f"[{model}]")
```

```
#!/usr/bin/env node
const { execSync } = require('child_process');
const path = require('path');

let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
    const data = JSON.parse(input);
    const model = data.model.display_name;

    try {
        let remote = execSync('git remote get-url origin', { encoding: 'utf8', std
io: ['pipe', 'pipe', 'ignore'] }).trim();
        // SSH를 HTTPS 형식으로 변환
        remote = remote.replace(/^git@github\.com:/, 'https://github.com/').replac
e(/\.git$/, '');
        const repoName = path.basename(remote);
        // OSC 8 이스케이프 시퀀스
        const link = `\x1b]8;;${remote}\x07${repoName}\x1b]8;;\x07`;
        console.log(`[${model}] 🔗 ${link}`);
    } catch {
        console.log(`[${model}]`);
    }
});
```

비용이 많이 드는 작업 캐싱

상태 표시줄 스크립트는 활성 세션 중에 자주 실행됩니다. `git status` 또는 `git diff` 와 같은 명령은 특히 큰 저장소에서 느릴 수 있습니다. 이 예제는 git 정보를 임시 파일에 캐싱하고 5초마다만 새로 고칩니다.

`/tmp/statusline-git-cache` 와 같은 안정적인 고정 파일 이름을 캐시 파일에 사용합니다. 각 상태 표시줄 호출은 새 프로세스로 실행되므로 `$$` , `os.getpid()` 또는 `process.pid` 와 같은 프로세스 기반 식별자는 매번 다른 값을 생성하고 캐시는 재사용되지 않습니다.

각 스크립트는 git 명령을 실행하기 전에 캐시 파일이 누락되었거나 5초보다 오래되었는지 확인합니다:

```
#!/bin/bash
input=$(cat)

MODEL=$(echo "$input" | jq -r '.model.display_name')
DIR=$(echo "$input" | jq -r '.workspace.current_dir')

CACHE_FILE="/tmp/statusline-git-cache"
CACHE_MAX_AGE=5  # seconds

cache_is_stale() {
    [ ! -f "$CACHE_FILE" ] || \
    # stat -f %m은 macOS, stat -c %Y는 Linux
    [ $(($(date +%s) - $(stat -f %m "$CACHE_FILE" 2>/dev/null || stat -c %Y "$CACH
E_FILE" 2>/dev/null || echo 0))) -gt $CACHE_MAX_AGE ]
}

if cache_is_stale; then
    if git rev-parse --git-dir > /dev/null 2>&1; then
        BRANCH=$(git branch --show-current 2>/dev/null)
        STAGED=$(git diff --cached --numstat 2>/dev/null | wc -l | tr -d ' ')
        MODIFIED=$(git diff --numstat 2>/dev/null | wc -l | tr -d ' ')
        echo "$BRANCH|$STAGED|$MODIFIED" > "$CACHE_FILE"
    else
        echo "||" > "$CACHE_FILE"
    fi
fi

IFS='|' read -r BRANCH STAGED MODIFIED < "$CACHE_FILE"

if [ -n "$BRANCH" ]; then
    echo "[$MODEL] 📁 ${DIR##*/} | 🌿 $BRANCH +$STAGED ~$MODIFIED"
else
    echo "[$MODEL] 📁 ${DIR##*/}"
fi
```

```
#!/usr/bin/env python3
import json, sys, subprocess, os, time

data = json.load(sys.stdin)
model = data['model']['display_name']
directory = os.path.basename(data['workspace']['current_dir'])

CACHE_FILE = "/tmp/statusline-git-cache"
CACHE_MAX_AGE = 5  # seconds

def cache_is_stale():
    if not os.path.exists(CACHE_FILE):
        return True
    return time.time() - os.path.getmtime(CACHE_FILE) > CACHE_MAX_AGE

if cache_is_stale():
    try:
        subprocess.check_output(['git', 'rev-parse', '--git-dir'], stderr=subproce
ss.DEVNULL)
        branch = subprocess.check_output(['git', 'branch', '--show-current'],
text=True).strip()
        staged = subprocess.check_output(['git', 'diff', '--cached', '--
numstat'], text=True).strip()
        modified = subprocess.check_output(['git', 'diff', '--numstat'],
text=True).strip()
        staged_count = len(staged.split('\n')) if staged else 0
        modified_count = len(modified.split('\n')) if modified else 0
        with open(CACHE_FILE, 'w') as f:
            f.write(f"{branch}|{staged_count}|{modified_count}")
    except:
        with open(CACHE_FILE, 'w') as f:
            f.write("||")

with open(CACHE_FILE) as f:
    branch, staged, modified = f.read().strip().split('|')

if branch:
```

```
    print(f"[{model}] 📁 {directory} | 🌿 {branch} +{staged} ~{modified}")
else:
    print(f"[{model}] 📁 {directory}")
```

```
#!/usr/bin/env node
const { execSync } = require('child_process');
const fs = require('fs');
const path = require('path');

let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
    const data = JSON.parse(input);
    const model = data.model.display_name;
    const dir = path.basename(data.workspace.current_dir);

    const CACHE_FILE = '/tmp/statusline-git-cache';
    const CACHE_MAX_AGE = 5; // seconds

    const cacheIsStale = () => {
        if (!fs.existsSync(CACHE_FILE)) return true;
        return (Date.now() / 1000) - fs.statSync(CACHE_FILE).mtimeMs / 1000 >
CACHE_MAX_AGE;
    };

    if (cacheIsStale()) {
        try {
            execSync('git rev-parse --git-dir', { stdio: 'ignore' });
            const branch = execSync('git branch --show-current', { encoding: 'utf8
' }).trim();
            const staged = execSync('git diff --cached --numstat', { encoding: 'ut
f8' }).trim().split('\n').filter(Boolean).length;
            const modified = execSync('git diff --numstat', { encoding:
'utf8' }).trim().split('\n').filter(Boolean).length;
            fs.writeFileSync(CACHE_FILE, `${branch}|${staged}|${modified}`);
        } catch {
            fs.writeFileSync(CACHE_FILE, '||');
        }
    }

    const [branch, staged, modified] = fs.readFileSync(CACHE_FILE,
'utf8').trim().split('|');
```

```
    if (branch) {
        console.log(`[${model}] 📁 ${dir} | 🌿 ${branch} +${staged} ~${modified}
`);
    } else {
        console.log(`[${model}] 📁 ${dir}`);
    }
});
```

Windows 구성

Windows에서 Claude Code는 Git Bash를 통해 상태 표시줄 명령을 실행합니다. 해당 셸에서 PowerShell을 호출할 수 있습니다:

```
{
  "statusLine": {
    "type": "command",
    "command": "powershell -NoProfile -File C:/Users/username/.claude/
statusline.ps1"
  }
}
```

```
$input_json = $input | Out-String | ConvertFrom-Json
$cwd = $input_json.cwd
$model = $input_json.model.display_name
$used = $input_json.context_window.used_percentage
$dirname = Split-Path $cwd -Leaf

if ($used) {
    Write-Host "$dirname [$model] ctx: $used%"
} else {
    Write-Host "$dirname [$model]"
}
```

또는 Bash 스크립트를 직접 실행합니다:

```
{
  "statusLine": {
    "type": "command",
    "command": "~/.claude/statusline.sh"
  }
}
```

```
#!/usr/bin/env bash
input=$(cat)
cwd=$(echo "$input" | grep -o '"cwd":"[^"]*"' | cut -d'"' -f4)
model=$(echo "$input" | grep -o '"display_name":"[^"]*"' | cut -d'"' -f4)
dirname="${cwd##*[/\\]}"
echo "$dirname [$model]"
```

팁

- **모의 입력으로 테스트**: `echo '{"model":{"display_name":"Opus"},"context_window":{"used_percentage":25}}' | ./statusline.sh`
- **출력을 짧게 유지**: 상태 표시줄의 너비가 제한되어 있으므로 긴 출력이 잘리거나 어색하게 줄 바꿈될 수 있습니다
- **느린 작업 캐싱**: 스크립트는 활성 세션 중에 자주 실행되므로 `git status` 와 같은 명령이 지연을 유발할 수 있습니다. 이를 처리하는 방법은 캐싱 예제를 참조하세요.

ccstatusline 및 starship-claude와 같은 커뮤니티 프로젝트는 테마 및 추가 기능이 있는 사전 구축된 구성을 제공합니다.

문제 해결

상태 표시줄이 나타나지 않음

- 스크립트가 실행 가능한지 확인합니다: `chmod +x ~/.claude/statusline.sh`
- 스크립트가 stderr가 아닌 stdout으로 출력하는지 확인합니다
- 스크립트를 수동으로 실행하여 출력을 생성하는지 확인합니다
- 설정에서 `disableAllHooks` 가 `true` 로 설정되어 있으면 상태 표시줄도 비활성화됩니다. 이 설정을 제거하거나 `false` 로 설정하여 다시 활성화합니다.
- `claude --debug` 를 실행하여 세션의 첫 번째 상태 표시줄 호출에서 종료 코드 및 stderr를 기록합니다

- Claude에 설정 파일을 읽고 `statusLine` 명령을 직접 실행하도록 요청하여 오류를 표시합니다

상태 표시줄이 `--` 또는 빈 값을 표시함

- 필드는 첫 번째 API 응답이 완료되기 전에 `null` 일 수 있습니다
- jq의 `// 0` 과 같은 폴백으로 스크립트에서 null 값을 처리합니다
- 여러 메시지 후에도 값이 비어 있으면 Claude Code를 다시 시작합니다

컨텍스트 백분율이 예상치 못한 값을 표시함

- 누적 합계 대신 정확한 컨텍스트 상태를 위해 `used_percentage` 를 사용합니다
- `total_input_tokens` 및 `total_output_tokens` 는 세션 전체에 누적되며 컨텍스트 윈도우 크기를 초과할 수 있습니다
- 각각이 계산되는 시기로 인해 컨텍스트 백분율이 `/context` 출력과 다를 수 있습니다

OSC 8 링크를 클릭할 수 없음

- 터미널이 OSC 8 하이퍼링크를 지원하는지 확인합니다(iTerm2, Kitty, WezTerm)
- Terminal.app은 클릭 가능한 링크를 지원하지 않습니다
- SSH 및 tmux 세션은 구성에 따라 OSC 시퀀스를 제거할 수 있습니다
- `\e]8;;` 과 같은 리터럴 텍스트로 이스케이프 시퀀스가 나타나면 `echo -e` 대신 `printf '%b'` 를 사용하여 더 안정적인 이스케이프 처리를 합니다

이스케이프 시퀀스로 인한 디스플레이 결함

- 복잡한 이스케이프 시퀀스(ANSI 색상, OSC 8 링크)는 다른 UI 업데이트와 겹치면 가끔 손상된 출력을 유발할 수 있습니다
- 손상된 텍스트가 보이면 스크립트를 일반 텍스트 출력으로 단순화해 봅니다
- 이스케이프 코드가 있는 다중 줄 상태 표시줄은 일반 텍스트 단일 줄보다 렌더링 문제가 더 발생하기 쉽습니다

스크립트 오류 또는 중단

- 0이 아닌 코드로 종료되거나 출력을 생성하지 않는 스크립트는 상태 표시줄을 공백으로 만듭니다
- 느린 스크립트는 완료될 때까지 상태 표시줄이 업데이트되지 않도록 차단합니다. 오래된 출력을 피하려면 스크립트를 빠르게 유지합니다.
- 느린 스크립트가 실행 중인 동안 새 업데이트가 트리거되면 진행 중인 스크립트가 취소됩니다
- 구성하기 전에 모의 입력으로 스크립트를 독립적으로 테스트합니다

알림이 상태 표시줄 행을 공유함

- MCP 서버 오류, 자동 업데이트 및 토큰 경고와 같은 시스템 알림은 상태 표시줄과 동일한 행의 오른쪽에 표시됩니다
- 자세한 모드를 활성화하면 이 영역에 토큰 카운터가 추가됩니다
- 좁은 터미널에서 이러한 알림이 상태 표시줄 출력을 자를 수 있습니다

Part 13: Troubleshooting & Changelog

문제 해결

> Claude Code 설치 및 사용 중 발생하는 일반적인 문제에 대한 해결책을 알아봅니다.

설치 문제 해결

> **Tip:**
>
> 터미널을 완전히 건너뛰고 싶다면, Claude Code Desktop 앱을 사용하여 그래픽 인터페이스를 통해 Claude Code를 설치하고 사용할 수 있습니다. macOS 또는 Windows용으로 다운로드하고 명령줄 설정 없이 코딩을 시작하세요.

표시되는 오류 메시지 또는 증상을 찾으세요:

표시되는 내용	해결책
`command not found: claude` 또는 `'claude' is not recognized`	PATH 수정
`syntax error near unexpected token '<'`	설치 스크립트가 HTML 반환
`curl: (56) Failure writing output to destination`	스크립트를 먼저 다운로드한 후 실행
Linux에서 설치 중 `Killed`	저메모리 서버에 스왑 공간 추가
`TLS connect error` 또는 `SSL/TLS secure channel`	CA 인증서 업데이트
`Failed to fetch version` 또는 다운로드 서버에 연결할 수 없음	네트워크 및 프록시 설정 확인
`irm is not recognized` 또는 `&& is not valid`	셸에 맞는 명령 사용

표시되는 내용	해결책
`Claude Code on Windows requires git-bash`	Git Bash 설치 또는 구성
`Error loading shared library`	시스템에 맞지 않는 바이너리 변형
Linux에서 `Illegal instruction`	아키텍처 불일치
macOS에서 `dyld: cannot load` 또는 `Abort trap`	바이너리 호환성 문제
`Invoke-Expression: Missing argument in parameter list`	설치 스크립트가 HTML 반환
`App unavailable in region`	Claude Code는 귀국에서 사용할 수 없습니다. 지원되는 국가를 참조하세요.
`unable to get local issuer certificate`	기업 CA 인증서 구성
`OAuth error` 또는 `403 Forbidden`	인증 수정

문제가 나열되지 않은 경우 다음 진단 단계를 진행하세요.

설치 문제 디버깅

네트워크 연결 확인

설치 프로그램은 `storage.googleapis.com`에서 다운로드합니다. 연결할 수 있는지 확인하세요:

```
curl -sI https://storage.googleapis.com
```

이것이 실패하면 네트워크가 연결을 차단할 수 있습니다. 일반적인 원인:

- Google Cloud Storage를 차단하는 기업 방화벽 또는 프록시
- 지역 네트워크 제한: VPN 또는 대체 네트워크 시도
- TLS/SSL 문제: 시스템의 CA 인증서를 업데이트하거나 `HTTPS_PROXY`가 구성되어 있는지 확인

기업 프록시 뒤에 있는 경우 설치하기 전에 `HTTPS_PROXY` 및 `HTTP_PROXY`를 프록시 주소로 설정하세요. 프록시 URL을 모르면 IT 팀에 문의하거나 브라우저의 프록시 설정을 확인하세요.

이 예제는 두 프록시 변수를 설정한 후 프록시를 통해 설치 프로그램을 실행합니다:

```
export HTTP_PROXY=http://proxy.example.com:8080
export HTTPS_PROXY=http://proxy.example.com:8080
curl -fsSL https://claude.ai/install.sh | bash
```

PATH 확인

설치가 성공했지만 `claude`를 실행할 때 `command not found` 또는 `not recognized` 오류가 나타나면 설치 디렉토리가 PATH에 없습니다. 셸은 PATH에 나열된 디렉토리에서 프로그램을 검색하며, 설치 프로그램은 macOS/Linux에서 `~/.local/bin/claude`에, Windows에서 `%USERPROFILE%\.local\bin\claude.exe`에 `claude`를 배치합니다.

PATH 항목을 나열하고 `local/bin`으로 필터링하여 설치 디렉토리가 PATH에 있는지 확인하세요:

macOS/Linux

```
echo $PATH | tr ':' '\n' | grep local/bin
```

출력이 없으면 디렉토리가 없습니다. 셸 구성에 추가하세요:

```
## Zsh (macOS 기본값)
echo 'export PATH="$HOME/.local/bin:$PATH"' >> ~/.zshrc
source ~/.zshrc

## Bash (Linux 기본값)
echo 'export PATH="$HOME/.local/bin:$PATH"' >> ~/.bashrc
source ~/.bashrc
```

또는 터미널을 닫았다가 다시 여세요.

수정이 작동했는지 확인하세요:

```
claude --version
```

Windows PowerShell

```
$env:PATH -split ';' | Select-String 'local\\bin'
```

출력이 없으면 설치 디렉토리를 사용자 PATH에 추가하세요:

```
$currentPath = [Environment]::GetEnvironmentVariable('PATH', 'User')
[Environment]::SetEnvironmentVariable('PATH', "$currentPath;$env:USERPROFILE\.local\bin", 'User')
```

변경 사항이 적용되려면 터미널을 다시 시작하세요.

수정이 작동했는지 확인하세요:

```
claude --version
```

Windows CMD

```
echo %PATH% | findstr /i "local\bin"
```

출력이 없으면 시스템 설정을 열고 환경 변수로 이동한 후 `%USERPROFILE%\.local\bin` 을 사용자 PATH 변수에 추가하세요. 터미널을 다시 시작하세요.

수정이 작동했는지 확인하세요:

```
claude --version
```

충돌하는 설치 확인

여러 Claude Code 설치는 버전 불일치 또는 예기치 않은 동작을 유발할 수 있습니다. 설치된 항목을 확인하세요:

macOS/Linux

PATH에서 찾은 모든 `claude` 바이너리를 나열하세요:

```
which -a claude
```

네이티브 설치 프로그램과 npm 버전이 있는지 확인하세요:

```
ls -la ~/.local/bin/claude
```

```
ls -la ~/.claude/local/
```

```
npm -g ls @anthropic-ai/claude-code 2>/dev/null
```

Windows PowerShell

```
where.exe claude
Test-Path "$env:LOCALAPPDATA\Claude Code\claude.exe"
```

여러 설치를 찾으면 하나만 유지하세요. `~/.local/bin/claude` 의 네이티브 설치가 권장됩니다. 추가 설치를 제거하세요:

npm 전역 설치 제거:

```
npm uninstall -g @anthropic-ai/claude-code
```

macOS에서 Homebrew 설치 제거:

```
brew uninstall --cask claude-code
```

디렉토리 권한 확인

설치 프로그램은 `~/.local/bin/` 및 `~/.claude/` 에 대한 쓰기 액세스가 필요합니다. 설치가 권한 오류로 실패하면 이 디렉토리가 쓰기 가능한지 확인하세요:

```
test -w ~/.local/bin && echo "writable" || echo "not writable"
test -w ~/.claude && echo "writable" || echo "not writable"
```

디렉토리가 쓰기 가능하지 않으면 설치 디렉토리를 만들고 사용자를 소유자로 설정하세요:

```
sudo mkdir -p ~/.local/bin
sudo chown -R $(whoami) ~/.local
```

바이너리 작동 확인

`claude`가 설치되었지만 시작 시 충돌하거나 중단되면 다음 검사를 실행하여 원인을 좁혀보세요.

바이너리가 존재하고 실행 가능한지 확인하세요:

```
ls -la $(which claude)
```

Linux에서 누락된 공유 라이브러리를 확인하세요. `ldd`가 누락된 라이브러리를 표시하면 시스템 패키지를 설치해야 할 수 있습니다. Alpine Linux 및 기타 musl 기반 배포판의 경우 Alpine Linux 설정을 참조하세요.

```
ldd $(which claude) | grep "not found"
```

바이너리가 실행될 수 있는지 빠른 건전성 검사를 실행하세요:

```
claude --version
```

일반적인 설치 문제

가장 자주 발생하는 설치 문제와 해결책입니다.

설치 스크립트가 셸 스크립트 대신 HTML 반환

설치 명령을 실행할 때 다음 오류 중 하나가 표시될 수 있습니다:

```
bash: line 1: syntax error near unexpected token `<'
bash: line 1: `<!DOCTYPE html>'
```

PowerShell에서 동일한 문제는 다음과 같이 나타납니다:

```
Invoke-Expression: Missing argument in parameter list.
```

이는 설치 URL이 설치 스크립트 대신 HTML 페이지를 반환했음을 의미합니다. HTML 페이지에 "App unavailable in region"이 표시되면 Claude Code는 귀국에서 사용할 수 없습니다. 지원되는 국가를 참조하세요.

그렇지 않으면 네트워크 문제, 지역 라우팅 또는 일시적인 서비스 중단으로 인해 발생할 수 있습니다.

해결책:

1. **대체 설치 방법 사용**:
   macOS 또는 Linux에서 Homebrew를 통해 설치하세요:

```
brew install --cask claude-code
```

Windows에서 WinGet을 통해 설치하세요:

```
winget install Anthropic.ClaudeCode
```

1. **몇 분 후 다시 시도하세요**: 문제는 종종 일시적입니다. 기다렸다가 원래 명령을 다시 시도하세요.

설치 후 `command not found: claude`

설치가 완료되었지만 `claude`가 작동하지 않습니다. 정확한 오류는 플랫폼에 따라 다릅니다:

플랫폼	오류 메시지
macOS	`zsh: command not found: claude`
Linux	`bash: claude: command not found`
Windows CMD	`'claude' is not recognized as an internal or external command`
PowerShell	`claude : The term 'claude' is not recognized as the name of a cmdlet`

이는 설치 디렉토리가 셸의 검색 경로에 없음을 의미합니다. 각 플랫폼에서 수정하려면 PATH 확인을 참조하세요.

`curl: (56) Failure writing output to destination`

`curl ... | bash` 명령은 스크립트를 다운로드하고 파이프(`|`)를 사용하여 Bash에 직접 전달하여 실행합니다. 이 오류는 스크립트 다운로드가 완료되기 전에 연결이 끊어졌음을 의미합니다. 일반적인 원인은 네트워크 중단, 다운로드가 중간에 차단되거나 시스템 리소스 제한입니다.

해결책:

1. **네트워크 안정성 확인**: Claude Code 바이너리는 Google Cloud Storage에서 호스팅됩니다. 연결할 수 있는지 테스트하세요:

```
curl -fsSL https://storage.googleapis.com -o /dev/null
```

명령이 조용히 완료되면 연결이 정상이고 문제는 일시적일 가능성이 높습니다. 설치 명령을 다시 시도하세요. 오류가 표시되면 네트워크가 다운로드를 차단할 수 있습니다.

1. **대체 설치 방법 시도**:
   macOS 또는 Linux에서:

```
brew install --cask claude-code
```

Windows에서:

```
winget install Anthropic.ClaudeCode
```

TLS 또는 SSL 연결 오류

`curl: (35) TLS connect error`, `schannel: next InitializeSecurityContext failed` 또는 PowerShell의 `Could not establish trust relationship for the SSL/TLS secure channel`과 같은 오류는 TLS 핸드셰이크 실패를 나타냅니다.

해결책:

1. **시스템 CA 인증서 업데이트**:
   Ubuntu/Debian에서:

```
sudo apt-get update && sudo apt-get install ca-certificates
```

Homebrew를 통한 macOS에서:

```
brew install ca-certificates
```

1. **Windows에서 설치 프로그램을 실행하기 전에 PowerShell에서 TLS 1.2 활성화**:

```
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12
irm https://claude.ai/install.ps1 | iex
```

1. **프록시 또는 방화벽 간섭 확인**: TLS 검사를 수행하는 기업 프록시는 `unable to get local issuer certificate`를 포함한 이러한 오류를 유발할 수 있습니다. `NODE_EXTRA_CA_CERTS`를 기업 CA 인증서 번들로 설정하세요:

```
export NODE_EXTRA_CA_CERTS=/path/to/corporate-ca.pem
```

인증서 파일이 없으면 IT 팀에 문의하세요. 프록시가 원인인지 확인하기 위해 직접 연결에서도 시도할 수 있습니다.

`Failed to fetch version from storage.googleapis.com`

설치 프로그램이 다운로드 서버에 연결할 수 없습니다. 이는 일반적으로 `storage.googleapis.com`이 네트워크에서 차단되었음을 의미합니다.

해결책:

1. **연결 직접 테스트**:

```
curl -sI https://storage.googleapis.com
```

1. **프록시 뒤에 있는 경우** `HTTPS_PROXY`를 설정하여 설치 프로그램이 프록시를 통해 라우팅할 수 있도록 하세요. 자세한 내용은 프록시 구성을 참조하세요.

```
export HTTPS_PROXY=http://proxy.example.com:8080
curl -fsSL https://claude.ai/install.sh | bash
```

1. **제한된 네트워크에 있는 경우** 다른 네트워크 또는 VPN을 시도하거나 대체 설치 방법을 사용하세요:

   macOS 또는 Linux에서:

```
brew install --cask claude-code
```

Windows에서:

```
winget install Anthropic.ClaudeCode
```

Windows: `irm` 또는 `&&` 인식 안 됨

`'irm' is not recognized` 또는 `The token '&&' is not valid`가 표시되면 셸에 맞지 않는 명령을 실행하고 있습니다.

- **`irm` 인식 안 됨**: PowerShell이 아닌 CMD에 있습니다. 두 가지 옵션이 있습니다:
  시작 메뉴에서 "PowerShell"을 검색하여 PowerShell을 열고 원래 설치 명령을 실행하세요:

```
irm https://claude.ai/install.ps1 | iex
```

또는 CMD에 머물러 있고 CMD 설치 프로그램을 대신 사용하세요:

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd && del
install.cmd
```

- **`&&` 유효하지 않음**: PowerShell에 있지만 CMD 설치 프로그램 명령을 실행했습니다.
  PowerShell 설치 프로그램을 사용하세요:

```
irm https://claude.ai/install.ps1 | iex
```

저메모리 Linux 서버에서 설치 중단됨

VPS 또는 클라우드 인스턴스에서 설치 중에 `Killed`가 표시되면:

```
Setting up Claude Code...
Installing Claude Code native build latest...
bash: line 142: 34803 Killed    "$binary_path" install ${TARGET:+"$TARGET"}
```

Linux OOM killer가 시스템 메모리 부족으로 인해 프로세스를 종료했습니다. Claude Code는 최소 4GB의 사용 가능한 RAM이 필요합니다.

해결책:

1. **서버의 RAM이 제한된 경우 스왑 공간 추가**. 스왑은 디스크 공간을 오버플로우 메모리로 사용하여 물리적 RAM이 낮아도 설치를 완료할 수 있습니다.
   2GB 스왑 파일을 만들고 활성화하세요:

```
sudo fallocate -l 2G /swapfile
sudo chmod 600 /swapfile
sudo mkswap /swapfile
sudo swapon /swapfile
```

그런 다음 설치를 다시 시도하세요:

```
curl -fsSL https://claude.ai/install.sh | bash
```

1. **다른 프로세스 종료** 설치 전에 메모리를 확보하세요.
2. **더 큰 인스턴스 사용** 가능한 경우. Claude Code는 최소 4GB의 RAM이 필요합니다.

Docker에서 설치 중단

Docker 컨테이너에서 Claude Code를 설치할 때 root로 `/`에 설치하면 중단될 수 있습니다.

해결책:

1. **설치 프로그램을 실행하기 전에 작업 디렉토리 설정**. `/`에서 실행하면 설치 프로그램이 전체 파일 시스템을 스캔하여 과도한 메모리 사용을 유발합니다. `WORKDIR`을 설정하면 스캔이 작은 디렉토리로 제한됩니다:

```
WORKDIR /tmp
RUN curl -fsSL https://claude.ai/install.sh | bash
```

1. **Docker 메모리 제한 증가** Docker Desktop을 사용하는 경우:

```
docker build --memory=4g .
```

Windows: Claude Desktop이 `claude` CLI 명령 재정의

Claude Desktop의 이전 버전을 설치한 경우 `WindowsApps` 디렉토리에 `Claude.exe`를 등록할 수 있으며, 이는 Claude Code CLI보다 PATH 우선순위를 가집니다. `claude`를 실행하면 CLI 대신 Desktop 앱이 열립니다.

Claude Desktop을 최신 버전으로 업데이트하여 이 문제를 해결하세요.

Windows: "Claude Code on Windows requires git-bash"

Windows의 네이티브 Claude Code는 Git Bash를 포함하는 Git for Windows가 필요합니다.

Git이 설치되지 않은 경우 git-scm.com/downloads/win에서 다운로드하여 설치하세요. 설정 중에 "Add to PATH"를 선택하세요. 설치 후 터미널을 다시 시작하세요.

Git이 이미 설치되어 있지만 Claude Code가 여전히 찾을 수 없으면 settings.json 파일에서 경로를 설정하세요:

```
{
  "env": {
    "CLAUDE_CODE_GIT_BASH_PATH": "C:\\Program Files\\Git\\bin\\bash.exe"
  }
}
```

Git이 다른 곳에 설치된 경우 PowerShell에서 `where.exe git`을 실행하여 경로를 찾고 해당 디렉토리의 `bin\bash.exe` 경로를 사용하세요.

Linux: 잘못된 바이너리 변형 설치됨 (musl/glibc 불일치)

설치 후 `libstdc++.so.6` 또는 `libgcc_s.so.1`과 같은 누락된 공유 라이브러리에 대한 오류가 표시되면 설치 프로그램이 시스템에 맞지 않는 바이너리 변형을 다운로드했을 수 있습니다.

```
Error loading shared library libstdc++.so.6: No such file or directory
```

이는 musl 교차 컴파일 패키지가 설치된 glibc 기반 시스템에서 발생할 수 있으며, 설치 프로그램이 시스템을 musl로 잘못 감지하게 합니다.

해결책:

1. **시스템이 어떤 libc를 사용하는지 확인**:

```
ldd /bin/ls | head -1
```

`linux-vdso.so` 또는 `/lib/x86_64-linux-gnu/`에 대한 참조가 표시되면 glibc를 사용하고 있습니다. `musl`이 표시되면 musl을 사용하고 있습니다.

1. **glibc에 있지만 musl 바이너리를 받은 경우** 설치를 제거하고 다시 설치하세요. `https://storage.googleapis.com/claude-code-dist-86c565f3-f756-42ad-8dfa-d59b1c096819/`

`claude-code-releases/{VERSION}/manifest.json`의 GCS 버킷에서 올바른 바이너리를 수동으로 다운로드할 수도 있습니다. `ldd /bin/ls` 및 `ls /lib/libc.musl*`의 출력과 함께 GitHub 이슈를 제출하세요.

2. **실제로 musl에 있는 경우** (Alpine Linux) 필요한 패키지를 설치하세요:

```
apk add libgcc libstdc++ ripgrep
```

Linux에서 `Illegal instruction`

설치 프로그램이 OOM `Killed` 메시지 대신 `Illegal instruction`을 인쇄하면 다운로드된 바이너리가 CPU 아키텍처와 일치하지 않습니다. 이는 ARM 서버가 x86 바이너리를 받거나 필요한 명령 세트가 없는 이전 CPU에서 일반적으로 발생합니다.

```
bash: line 142: 2238232 Illegal instruction     "$binary_path" install ${TARGET:
+"$TARGET"}
```

해결책:

1. **아키텍처 확인**:

```
uname -m
```

`x86_64`는 64비트 Intel/AMD를 의미하고 `aarch64`는 ARM64를 의미합니다. 바이너리가 일치하지 않으면 출력과 함께 GitHub 이슈를 제출하세요.

1. **아키텍처 문제가 해결되는 동안 대체 설치 방법 시도**:

```
brew install --cask claude-code
```

macOS에서 `dyld: cannot load`

설치 중에 `dyld: cannot load` 또는 `Abort trap: 6`이 표시되면 바이너리가 macOS 버전 또는 하드웨어와 호환되지 않습니다.

```
dyld: cannot load 'claude-2.1.42-darwin-x64' (load command 0x80000034 is unknown)
Abort trap: 6
```

해결책:

1. **macOS 버전 확인**: Claude Code는 macOS 13.0 이상이 필요합니다. Apple 메뉴를 열고 "이 Mac에 관하여"를 선택하여 버전을 확인하세요.
2. **이전 버전을 사용 중인 경우 macOS 업데이트**. 바이너리는 이전 macOS 버전이 지원하지 않는 로드 명령을 사용합니다.
3. **대체 설치 방법으로 Homebrew 시도**:

```
brew install --cask claude-code
```

Windows 설치 문제: WSL의 오류

WSL에서 다음 문제가 발생할 수 있습니다:

OS/플랫폼 감지 문제: 설치 중에 오류가 발생하면 WSL이 Windows `npm`을 사용할 수 있습니다. 다음을 시도하세요:

- 설치 전에 `npm config set os linux` 실행
- `npm install -g @anthropic-ai/claude-code --force --no-os-check`로 설치하세요. `sudo`를 사용하지 마세요.

Node를 찾을 수 없는 오류: `claude`를 실행할 때 `exec: node: not found`가 표시되면 WSL 환경이 Windows Node.js 설치를 사용할 수 있습니다. `which npm` 및 `which node`로 확인할 수 있으며, `/usr/`로 시작하는 Linux 경로가 아닌 `/mnt/c/`로 시작하는 경로를 가리켜야 합니다. 이를 해결하려면 Linux 배포판의 패키지 관리자 또는 `nvm`을 통해 Node를 설치해 보세요.

nvm 버전 충돌: WSL과 Windows 모두에 nvm이 설치되어 있으면 WSL에서 Node 버전을 전환할 때 버전 충돌이 발생할 수 있습니다. 이는 WSL이 기본적으로 Windows PATH를 가져오기 때문에 WSL 설치보다 Windows nvm/npm이 우선순위를 가집니다.

다음을 실행하여 이 문제를 식별할 수 있습니다:

- `which npm` 및 `which node` 실행 - `/mnt/c/`로 시작하는 경로를 가리키면 Windows 버전이 사용 중입니다
- WSL에서 nvm으로 Node 버전을 전환한 후 기능이 손상됨

이 문제를 해결하려면 Linux PATH를 수정하여 Linux node/npm 버전이 우선순위를 가지도록 하세요:

기본 해결책: nvm이 셸에 제대로 로드되는지 확인

가장 일반적인 원인은 nvm이 비대화형 셸에 로드되지 않는 것입니다. 셸 구성 파일(`~/.bashrc`, `~/.zshrc` 등)에 다음을 추가하세요:

```
## nvm이 있으면 로드
export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion"
```

또는 현재 세션에서 직접 실행하세요:

```
source ~/.nvm/nvm.sh
```

대체: PATH 순서 조정

nvm이 제대로 로드되었지만 Windows 경로가 여전히 우선순위를 가지면 셸 구성에서 Linux 경로를 PATH 앞에 명시적으로 추가할 수 있습니다:

```
export PATH="$HOME/.nvm/versions/node/$(node -v)/bin:$PATH"
```

> **Warning:**
>
> `appendWindowsPath = false`를 통해 Windows PATH 가져오기를 비활성화하지 마세요. WSL에서 Windows 실행 파일을 호출할 수 있는 기능이 손상됩니다. 마찬가지로 Windows 개발에 사용하는 경우 Windows에서 Node.js를 제거하지 마세요.

WSL2 샌드박스 설정

샌드박싱은 WSL2에서 지원되지만 추가 패키지를 설치해야 합니다. `/sandbox`를 실행할 때 “Sandbox requires socat and bubblewrap” 과 같은 오류가 표시되면 종속성을 설치하세요:

Ubuntu/Debian

```
sudo apt-get install bubblewrap socat
```

Fedora

```
sudo dnf install bubblewrap socat
```

WSL1은 샌드박싱을 지원하지 않습니다. “Sandboxing requires WSL2”가 표시되면 WSL2로 업그레이드하거나 샌드박싱 없이 Claude Code를 실행해야 합니다.

설치 중 권한 오류

네이티브 설치 프로그램이 권한 오류로 실패하면 대상 디렉토리가 쓰기 가능하지 않을 수 있습니다. 디렉토리 권한 확인을 참조하세요.

이전에 npm으로 설치했고 npm 관련 권한 오류가 발생하면 네이티브 설치 프로그램으로 전환하세요:

```
curl -fsSL https://claude.ai/install.sh | bash
```

권한 및 인증

이 섹션에서는 로그인 실패, 토큰 문제 및 권한 프롬프트 동작을 다룹니다.

반복되는 권한 프롬프트

특정 명령을 반복적으로 승인해야 하는 경우 `/permissions` 명령을 사용하여 특정 도구가 승인 없이 실행되도록 허용할 수 있습니다. 권한 문서를 참조하세요.

인증 문제

인증 문제가 발생하는 경우:

1. `/logout` 을 실행하여 완전히 로그아웃하세요
2. Claude Code 종료
3. `claude` 로 다시 시작하고 인증 프로세스를 완료하세요

로그인 중에 브라우저가 자동으로 열리지 않으면 `c` 를 눌러 OAuth URL을 클립보드에 복사한 후 브라우저에 수동으로 붙여넣으세요.

OAuth 오류: 유효하지 않은 코드

`OAuth error: Invalid code. Please make sure the full code was copied` 가 표시되면 로그인 코드가 만료되었거나 복사-붙여넣기 중에 잘렸습니다.

해결책:

- Enter를 눌러 다시 시도하고 브라우저가 열린 후 빠르게 로그인을 완료하세요
- 브라우저가 자동으로 열리지 않으면 `c` 를 입력하여 전체 URL을 복사하세요
- 원격/SSH 세션을 사용하는 경우 브라우저가 잘못된 머신에서 열릴 수 있습니다. 터미널에 표시된 URL을 복사하여 로컬 브라우저에서 열어보세요.

로그인 후 403 Forbidden

로그인 후 `API Error: 403 {"error":{"type":"forbidden","message":"Request not allowed"}}` 가 표시되면:

- **Claude Pro/Max 사용자**: claude.ai/settings에서 구독이 활성화되어 있는지 확인하세요
- **Console 사용자**: 관리자가 계정에 "Claude Code" 또는 "Developer" 역할을 할당했는지 확인하세요
- **프록시 뒤에 있음**: 기업 프록시가 API 요청을 방해할 수 있습니다. 프록시 설정은 네트워크 구성을 참조하세요.

WSL2에서 OAuth 로그인 실패

WSL2의 브라우저 기반 로그인은 WSL이 Windows 브라우저를 열 수 없으면 실패할 수 있습니다. `BROWSER` 환경 변수를 설정하세요:

```
export BROWSER="/mnt/c/Program Files/Google/Chrome/Application/chrome.exe"
claude
```

또는 수동으로 URL을 복사하세요: 로그인 프롬프트가 나타나면 `c`를 눌러 OAuth URL을 복사한 후 Windows 브라우저에 붙여넣으세요.

"Not logged in" 또는 토큰 만료됨

Claude Code가 세션 후 다시 로그인하도록 요청하면 OAuth 토큰이 만료되었을 수 있습니다.

`/login`을 실행하여 다시 인증하세요. 이것이 자주 발생하면 시스템 시계가 정확한지 확인하세요. 토큰 검증은 올바른 타임스탬프에 따라 달라집니다.

구성 파일 위치

Claude Code는 여러 위치에 구성을 저장합니다:

파일	목적
`~/.claude/settings.json`	사용자 설정 (권한, hooks, 모델 재정의)
`.claude/settings.json`	프로젝트 설정 (소스 제어에 체크인됨)
`.claude/settings.local.json`	로컬 프로젝트 설정 (커밋되지 않음)

파일	목적
`~/.claude.json`	전역 상태 (테마, OAuth, MCP 서버)
`.mcp.json`	프로젝트 MCP 서버 (소스 제어에 체크인됨)
`managed-mcp.json`	관리되는 MCP 서버
관리되는 설정	관리되는 설정 (서버 관리, MDM/OS 수준 정책 또는 파일 기반)

Windows에서 `~`는 `C:\Users\YourName`과 같은 사용자 홈 디렉토리를 나타냅니다.

이 파일 구성에 대한 자세한 내용은 설정 및 MCP를 참조하세요.

구성 재설정

Claude Code를 기본 설정으로 재설정하려면 구성 파일을 제거할 수 있습니다:

```
## 모든 사용자 설정 및 상태 재설정
rm ~/.claude.json
rm -rf ~/.claude/

## 프로젝트별 설정 재설정
rm -rf .claude/
rm .mcp.json
```

> **Warning:**
> 이렇게 하면 모든 설정, MCP 서버 구성 및 세션 기록이 제거됩니다.

성능 및 안정성

이 섹션에서는 리소스 사용, 응답성 및 검색 동작과 관련된 문제를 다룹니다.

높은 CPU 또는 메모리 사용

Claude Code는 대부분의 개발 환경에서 작동하도록 설계되었지만 대규모 코드베이스를 처리할 때 상당한 리소스를 소비할 수 있습니다. 성능 문제가 발생하는 경우:

1. `/compact`를 정기적으로 사용하여 컨텍스트 크기 감소
2. 주요 작업 사이에 Claude Code 종료 및 다시 시작
3. 큰 빌드 디렉토리를 `.gitignore` 파일에 추가하는 것을 고려하세요

명령 중단 또는 정지

Claude Code가 응답하지 않는 것처럼 보이면:

1. Ctrl+C를 눌러 현재 작업을 취소하세요
2. 응답하지 않으면 터미널을 닫고 다시 시작해야 할 수 있습니다

검색 및 발견 문제

Search 도구, `@file` 언급, 사용자 정의 에이전트 및 사용자 정의 skills가 작동하지 않으면 시스템 `ripgrep` 을 설치하세요:

```
## macOS (Homebrew)
brew install ripgrep

## Windows (winget)
winget install BurntSushi.ripgrep.MSVC

## Ubuntu/Debian
sudo apt install ripgrep

## Alpine Linux
apk add ripgrep

## Arch Linux
pacman -S ripgrep
```

그런 다음 환경에서 `USE_BUILTIN_RIPGREP=0` 을 설정하세요.

WSL에서 느리거나 불완전한 검색 결과

WSL에서 파일 시스템 간 작업할 때 디스크 읽기 성능 저하로 인해 WSL에서 Claude Code를 사용할 때 예상보다 적은 일치 항목이 발생할 수 있습니다. 검색은 여전히 작동하지만 네이티브 파일 시스템보다 적은 결과를 반환합니다.

> **Note:**
>
> 이 경우 `/doctor` 는 검색을 OK로 표시합니다.

해결책:

1. **더 구체적인 검색 제출**: 디렉토리 또는 파일 유형을 지정하여 검색되는 파일 수를 줄이세요: "auth-service 패키지에서 JWT 검증 로직 검색" 또는 "JS 파일에서 md5 해시 사용 찾기".
2. **프로젝트를 Linux 파일 시스템으로 이동**: 가능하면 프로젝트가 Windows 파일 시스템(`/mnt/c/`)이 아닌 Linux 파일 시스템(`/home/`)에 있는지 확인하세요.
3. **네이티브 Windows 사용**: 더 나은 파일 시스템 성능을 위해 WSL 대신 Windows에서 기본적으로 Claude Code를 실행하는 것을 고려하세요.

IDE 통합 문제

Claude Code가 IDE에 연결되지 않거나 IDE 터미널 내에서 예기치 않게 동작하면 아래 해결책을 시도하세요.

WSL2에서 JetBrains IDE 감지 안 됨

WSL2에서 Claude Code를 사용하고 JetBrains IDE를 사용하며 "No available IDEs detected" 오류가 발생하면 WSL2의 네트워킹 구성 또는 Windows Firewall이 연결을 차단할 가능성이 높습니다.

WSL2 네트워킹 모드

WSL2는 기본적으로 NAT 네트워킹을 사용하며, 이는 IDE 감지를 방지할 수 있습니다. 두 가지 옵션이 있습니다:

옵션 1: Windows Firewall 구성 (권장)

1. WSL2 IP 주소 찾기:

```
wsl hostname -I
## 예제 출력: 172.21.123.45
```

1. PowerShell을 관리자로 열고 방화벽 규칙을 만드세요:

```
New-NetFirewallRule -DisplayName "Allow WSL2 Internal Traffic" -Direction Inbound
-Protocol TCP -Action Allow -RemoteAddress 172.21.0.0/16 -LocalAddress 172.21.0.0/
16
```

1단계의 WSL2 서브넷을 기반으로 IP 범위를 조정하세요.

1. IDE와 Claude Code를 모두 다시 시작하세요

옵션 2: 미러링된 네트워킹으로 전환

Windows 사용자 디렉토리의 `.wslconfig` 에 추가하세요:

```
[wsl2]
networkingMode=mirrored
```

그런 다음 PowerShell에서 `wsl --shutdown` 으로 WSL을 다시 시작하세요.

> **Note:**
>
> 이러한 네트워킹 문제는 WSL2에만 영향을 미칩니다. WSL1은 호스트의 네트워크를 직접 사용하며 이러한 구성이 필요하지 않습니다.

추가 JetBrains 구성 팁은 JetBrains IDE 가이드를 참조하세요.

Windows IDE 통합 문제 보고

Windows에서 IDE 통합 문제가 발생하는 경우 다음 정보와 함께 이슈를 생성하세요:

- 환경 유형: 네이티브 Windows (Git Bash) 또는 WSL1/WSL2
- WSL 네트워킹 모드 (해당하는 경우): NAT 또는 미러링됨
- IDE 이름 및 버전
- Claude Code 확장/플러그인 버전
- 셸 유형: Bash, Zsh, PowerShell 등

JetBrains IDE 터미널에서 Escape 키가 작동하지 않음

JetBrains 터미널에서 Claude Code를 사용하고 `Esc` 키가 예상대로 에이전트를 중단하지 않으면 JetBrains의 기본 단축키와 키 바인딩이 충돌할 가능성이 높습니다.

이 문제를 해결하려면:

1. 설정 → 도구 → 터미널로 이동하세요
2. 다음 중 하나를 수행하세요:
   - “Move focus to the editor with Escape” 선택 해제, 또는
   - “Configure terminal keybindings” 을 클릭하고 “Switch focus to Editor” 단축키 삭제
3. 변경 사항 적용

이렇게 하면 `Esc` 키가 Claude Code 작업을 제대로 중단할 수 있습니다.

Markdown 형식 문제

Claude Code는 때때로 코드 펜스에 언어 태그가 누락된 markdown 파일을 생성하며, 이는 GitHub, 편집기 및 문서 도구의 구문 강조 및 가독성에 영향을 미칠 수 있습니다.

코드 블록의 누락된 언어 태그

생성된 markdown에서 다음과 같은 코드 블록을 발견하면:

````
```
function example() {
  return "hello";
}
```text
````

다음과 같이 적절히 태그된 블록 대신:

````
```javascript
function example() {
  return "hello";
}
```text
````

해결책:

1. **Claude에게 언어 태그 추가 요청**: "이 markdown 파일의 모든 코드 블록에 적절한 언어 태그를 추가하세요."라고 요청하세요.
2. **사후 처리 hooks 사용**: 누락된 언어 태그를 감지하고 추가하는 자동 형식 지정 hooks를 설정하세요. 편집 후 자동 형식 지정에서 PostToolUse 형식 지정 hook의 예를 참조하세요.
3. **수동 확인**: markdown 파일을 생성한 후 적절한 코드 블록 형식을 검토하고 필요하면 수정을 요청하세요.

일관성 없는 간격 및 형식

생성된 markdown에 과도한 빈 줄이나 일관성 없는 간격이 있으면:

해결책:

1. **형식 수정 요청**: Claude에게 "이 markdown 파일의 간격 및 형식 문제를 수정하세요."라고 요청하세요.

2. **형식 지정 도구 사용**: `prettier` 또는 사용자 정의 형식 지정 스크립트와 같은 markdown 포매터를 실행하는 hooks를 설정하세요.
3. **형식 지정 기본 설정 지정**: 프롬프트 또는 프로젝트 메모리 파일에 형식 지정 요구 사항을 포함하세요.

Markdown 형식 문제 감소

형식 지정 문제를 최소화하려면:

- **요청에서 명시적으로**: "언어 태그가 있는 적절히 형식된 markdown"을 요청하세요
- **프로젝트 규칙 사용**: 선호하는 markdown 스타일을 `CLAUDE.md` 에 문서화하세요
- **검증 hooks 설정**: 사후 처리 hooks를 사용하여 일반적인 형식 지정 문제를 자동으로 확인하고 수정하세요

추가 도움 받기

여기에 다루지 않은 문제가 발생하는 경우:

1. Claude Code 내에서 `/bug` 명령을 사용하여 Anthropic에 문제를 직접 보고하세요
2. GitHub 저장소에서 알려진 문제를 확인하세요
3. `/doctor` 를 실행하여 문제를 진단하세요. 다음을 확인합니다:
   - 설치 유형, 버전 및 검색 기능
   - 자동 업데이트 상태 및 사용 가능한 버전
   - 잘못된 설정 파일 (형식이 잘못된 JSON, 잘못된 유형)
   - MCP 서버 구성 오류
   - 키 바인딩 구성 문제
   - 컨텍스트 사용 경고 (큰 CLAUDE.md 파일, 높은 MCP 토큰 사용, 도달할 수 없는 권한 규칙)
   - 플러그인 및 에이전트 로딩 오류
4. Claude에게 직접 기능 및 특징에 대해 물어보세요 - Claude는 문서에 대한 기본 제공 액세스 권한이 있습니다