

혼자서 AI 팀 운영하기

Run a small AI team without dropping work

누락·중복·완료를 한 프로젝트로 관리하는 운영법
역할 카드 · 작업지시 · 반려 · 재개 · 완료 근거

메인

범위와 게이트

작업자

한 건의 산출물

검토자

누락과 근거

마이너스베타스튜디오 · 2026년 10월 무료 공개판

이 전자책은 kangdaejong.com 에서 무료로 공개한 판입니다. 개인 학습·업무에 자유롭게 읽고 활용하세요. 저작권은 강대중(마이너스베타스튜디오)에게 있습니다.

목차

숫자는 PDF의 실제 페이지입니다. 제목을 누르면 해당 본문으로 이동합니다.

서문	6
이 책을 읽는 방법	6
이 책에서 하지 않는 것	6
읽기 전 약속	7
학습용 가상 사례: 가을 워크숍 안내문	7
이 프로젝트에서 만드는 결과	7
역할 카드 3장	8
한 바퀴 순서	8
기대 결과	9
30분 빠른 시작	10
기록 방식	10
1장. 에이전트 1개의 한계	11
이 가상 프로젝트에서는	11
사고: 한 창에 모든 일을 몰아넣으면 똑똑한 도구도 흐려진다	11
그래서 생긴 룰: 한 에이전트에게 모든 역할을 주지 않는다	12
너 환경엔 이렇게 적용한다	12
작은 실험	13
이번 강의 핵심	14
2장. 일감 자동 분배	15
이 가상 프로젝트에서는	15
사고: 사람이 매번 배정하면 자동화가 다시 수동 일이 된다	15
그래서 생긴 룰: 일감은 큐에 넣고, 노는 에이전트가 집어간다	16
너 환경엔 이렇게 적용한다	16
큐에 넣기 좋은 일감과 나쁜 일감	17
작은 큐 운영 규칙	17
이번 강의 핵심	18
3장. 충돌 방지	18
이 가상 프로젝트에서는	18
사고: 둘이 같은 곳을 고치면 빠른 만큼 빨리 망가진다	19
그래서 생긴 룰: 영역, 브랜치, lease를 나눈다	19
너 환경엔 이렇게 적용한다	20
잠금은 자유를 줄이는 게 아니다	20
충돌이 났을 때의 복구 순서	21
이번 강의 핵심	21
4장. 중복·누락 막기	22
이 가상 프로젝트에서는	22
사고: 끝났는데 끝났다고 남기지 않으면 끝난 일이 아니다	22
그래서 생긴 룰: 완료 도장에는 근거를 붙인다	23
너 환경엔 이렇게 적용한다	23
완료와 관찰을 구분한다	24

스태일 작업을 줄이는 법	24
이번 장의 핵심	25
5장. 사람은 폰으로만 감독	26
이 가상 프로젝트에서는	26
사고: 모든 것을 승인하면 사람이 병목이고, 아무것도 승인하지 않으면 시스템이 위험하다	26
그래서 생긴 룰: 권한과 자문을 분리한다	27
너 환경엔 이렇게 적용한다	27
무응답은 승인 아니다	28
승인 게이트를 너무 넓히지 않는다	29
이번 장의 핵심	29
6장. 비용 폭주 막기	29
이 가상 프로젝트에서는	30
사고: 자동 루프는 성실해서 위험하다	30
그래서 생긴 룰: 비싼 판단과 싼 반복을 나눈다	30
너 환경엔 이렇게 적용한다	31
반복 제한을 둔다	31
프롬프트도 예산이다	31
비용 보고를 부끄러워하지 않는다	32
이번 장의 핵심	32
7장. 메인 ↔ 서브에이전트 구조 세팅	33
이 가상 프로젝트에서는	33
이 장에서 만들 것	34
작업표 만들기	34
도구에서 역할을 나누는 방법	35
왜 이름을 붙이나	35
메인과 서브를 어떻게 상상하면 좋을까	36
이 장에서 만든 구조	36
8장. 일감 큐 만들고 자동으로 집어가게 하기	37
이 가상 프로젝트에서는	37
작업지시 카드	37
자동분배는 거창하지 않다	37
좋은 큐의 조건	38
claim은 말로 하는 배경이 아니다	39
규칙을 에이전트 프롬프트로 바꾸기	39
이번 장의 핵심	40
9장. 충돌·중복 방지 최소 장치 붙이기	40
이 가상 프로젝트에서는	40
왜 최소 장치가 필요한가	41
완료 처리의 세 가지 경우	41
lease가 막는 것	42
중복 실행을 두려워하지 않는 구조	43
내 실전 작업에 붙이는 최소 규칙	44
실패 복구 체크리스트	44
이번 장의 핵심	44

10장. 결과 수합·검증·보고까지 한 바퀴 돌리기	45
이 가상 프로젝트에서는	45
자동분배는 보고까지 가야 끝난다	45
검토자가 보는 것	45
사람이 읽는 보고 만들기	46
남은 작업 닫기	47
종료와 보관	47
이 장에서 정리한 한 바퀴	47
“잘 됐다”는 말은 검증 뒤에만 한다	48
처음 운영할 때 지킬 기준	48
이번 장의 핵심	49
11장. 2대·구독 2개로 확장하면 뭐가 달라지나	50
이 가상 프로젝트에서는	50
확장은 속도보다 분리다	51
두 기기 구조의 기본 그림	51
계정이 두 개가 되면 비용도 두 개가 된다	52
작업 큐는 하나만 두는 것이 쉽다	52
파일 충돌은 역할보다 범위로 막는다	53
승인 게이트는 중앙에 둔다	54
알림은 많을수록 좋은 것이 아니다	54
장애 대응 규칙을 먼저 만든다	55
언제 확장해도 되는가	55
2대 확장 미니 청사진	56
예시: 두 기기에서 안내문 작업 나누기	56
마무리: 작은 운영 원칙이 큰 구조를 버틴다	57
확장 실패 복구 체크리스트	57
템플릿 01. 일감 큐 양식	58
빈 양식	58
상태값 기준	59
예시 작성본	59
학습용 가상 사례 작성본 — 가을 워크숍 안내문	59
작업 등록 프롬프트	60
완료 보고 프롬프트	60
사용 체크리스트	61
템플릿 02. 작업 분배 규칙표	61
빈 양식	61
처리 등급	62
예시 작성본	62
학습용 가상 사례 작성본 — 가을 워크숍 안내문	63
분배 결정 프롬프트	63
기본 규칙 문장	64
사용 체크리스트	64
템플릿 03. 승인 게이트 체크리스트	64
승인 필요 행동	64
빈 승인 요청 양식	65

예시 작성본	65
승인 판단 답변 양식	66
조건부 승인 예시	66
학습용 가상 사례 승인 요청 — 가을 워크숍 안내문	66
에이전트용 중지 문구	67
사용 체크리스트	67
템플릿 04. 사고 대응 룰 모음	67
기본 원칙	68
사고 기록 빈 양식	68
유형별 대응표	69
예시 작성본	69
학습용 가상 사례 사고 기록 — 잘못된 완료 시도	70
사고 발생 시 에이전트에게 보낼 프롬프트	71
복구 완료 체크리스트	71
짧은 중지 문구	71
맷는글	72
저자소개	72
무료 공개판 안내	72

서문

AI에게 일을 여러 개 맡기면 속도가 난 것처럼 보입니다. 한쪽에는 초안, 다른 쪽에는 확인, 또 다른 쪽에는 보고를 맡깁니다. 그런데 끝나 보니 같은 파일을 두 번 고쳤거나, 끝난 일이 다시 열려 있거나, “완료했습니다”만 있고 확인할 파일이 없는 경우가 있습니다. 문제는 모델이 부족해서가 아니라 **누락·중복·완료를 한곳에서 판단하지 않아서**입니다.

이 책은 그 판단 방법을 한 권에 모았습니다. 목표는 거창한 자동화 회사를 만드는 것이 아닙니다. 혼자 일하는 사람이 AI를 작은 팀처럼 굴릴 때, 누가 무엇을 맡았는지, 무엇이 반려됐는지, 무엇이 근거와 함께 끝났는지를 남기는 것입니다.

시리즈 이름은 AI WORKSHOP 02입니다. 1권이 반복 업무를 한 건씩 정리하는 작업장이라면, 2권은 **여러 일을 동시에 맡겼을 때의 운영**입니다. 3권의 원격 연결과 무관하게 이 책만으로 따라 할 수 있습니다.

이 책을 읽는 방법

처음부터 11개 장을 순서대로 읽지 않아도 됩니다. 다만 2부는 순서가 있습니다. 아래 경로가 이 책의 정보 읽기 루트입니다.

1. **가상 사례 한 장**을 읽는다. 「가을 워크숍 안내문」은 학습용 가상 사례다. 세 작업과 역할 카드 3장을 먼저 본다.
2. **1부**에서 그 세 작업을 한 창에 몰면 생기는 사고와 규칙을 표시한다. 각 장 앞의 “이 가상 프로젝트에서는”만 읽어도 다음 장으로 갈 수 있다.
3. **2부**에서 같은 세 작업을 작업표 한 장에 올린다. 시작 → 작업 맡기(claim) → 반려 → 재개 → 완료 근거 → 종료 순서로 따라간다.
4. **부록 템플릿**의 빈 양식과 작성 예시를 같은 가상 사례로 한 번 채운다.
5. 내 일의 작업 세 개로 역할 카드만 바꿔 한 주 운영해 본다.

이미 여러 에이전트를 쓰는 사람은 가상 사례를 읽은 뒤 3장, 4장, 9장을 먼저 봐도 됩니다. 대부분의 운영 사고는 도구 부족이 아니라 충돌, 완료 근거 누락, 승인 경계 혼동에서 생깁니다.

완료를 판단할 때는 에이전트의 말보다 결과 파일과 확인 기록을 기준으로 하세요.

이 책에서 하지 않는 것

이 책은 특정 서비스 계정, 개인 메시지 채널, 비밀값, 원격 접속 설정을 다루지 않습니다. 그런 것은 환경마다 다르고, 공개 문서에 그대로 쓰면 위험합니다. 대신 어떤 정보를 숨겨야 하는지, 어떤 행동에서 멈춰야 하는지, 어떤 근거를 남겨야 하는지를 다룹니다.

또한 이 책은 “AI가 사람을 완전히 대체한다”, “매출이 오른다”, “무제한 지원” 같은 약속을 하지 않습니다. 사람은 더 중요한 승인과 판단에 집중하고, 에이전트는 초안·검증·정리·보고를 맡습니다. 좋은 AI 팀 운영은 사람을 없애는 것이 아니라, 사람이 매번 같은 지점에서 다시 생각하지 않게 만드는 것입니다.

본문의 작업표와 수치는 학습용 가상 사례입니다. 도구 메뉴와 명령 이름은 버전에 따라 바뀔 수 있으므로 설치본의 `--help` 와 공식 문서를 확인하세요.

읽기 전 약속

- 가상 사례의 안내문을 실제로 발송하거나 게시하지 않는다.
- 완료는 기분으로 쓰지 않고 결과 파일, 작업표, 확인 기록으로 확인한다.
- 끝난 작업표는 지우지 않고 보관한다.
- 외부 발송, 결제, 배포, 삭제는 사람 승인 전에 실행하지 않는다.

학습용 가상 사례: 가을 워크숍 안내문

이 책의 모든 장은 하나의 가상 프로젝트를 따라갑니다. 실제 워크숍을 열거나 안내문을 발송하지 않습니다. 매출, 참석자 수, 절감 시간도 꾸미지 않습니다. 표시는 항상 학습용 가상 사례입니다.

이 프로젝트에서 만드는 결과

혼자 일하는 사람이 AI에게 일을 세 덩어리로 맡긴 상황입니다.

1. 안내문 목차 초안을 쓴다.
2. 안내문에 들어갈 일정·장소·준비물이 주최 메모와 맞는지 확인한다.
3. 사람이 읽을 한 줄 완료 보고를 남긴다.

작업 ID	작업	완료 조건	완료 근거(예시)
T-001	안내문 목차 초안	제목과 섹션 순서가 정해짐	<code>outline.md</code> , 섹션 6개
T-002	일정·장소 사실 확인	안내문의 날짜·장소·준비물이 주최 메모와 일치	확인표 3줄, 불일치 0건
T-003	한 줄 완료 보고	상태표와 확인 결과가 요약됨	보고 1줄, 남은 위험 표시

이 표가 이 책의 작업지시입니다. 1부는 이 세 일을 한 창에 몰지 않는 이유를 설명하고, 2부는 같은 세 일을 작업표에 올려 담당을 정하고, 반려하고, 재개하고, 근거와 함께 닫는 흐름을 보여 줍니다. 부록 템플릿도 같은 사례로 빈 칸과 작성 예시를 짚지었습니다.

역할 카드 3장

역할을 창 개수가 아니라 **입력과 산출물**로 나눕니다. 같은 AI 도구라도 역할 카드를 바꾸면 다른 운영 단위가 됩니다.

역할 카드 A. 메인

이름: main
하는 일: 목표를 세 작업으로 나누고, 작업표를 만들고, 전체 상태를 보고, 완료를 판정한다.
받지 않는 일: 안내문 문장을 직접 다듬거나, 다른 담당이 맡은 파일을 덮어쓰지 않는다.
입력: 프로젝트 목표 1줄, 금지 행동, 세 작업의 완료 조건.
산출물: 작업표 상태, 다음 행동, 사람 승인이 필요한 지점.
금지: 외부 발송, 결제, 공개 게시, 실제 참석자 명단 작성.

역할 카드 B. 작업자

이름: sub
하는 일: 작업표에서 비어 있는 작업 하나를 맡아 그 작업만 처리한다.
받지 않는 일: 다른 작업의 상태 변경, 전체 목차 재설계, 승인이 필요한 행동.
입력: 작업 ID, 작업 범위, 완료 조건, 건드리면 안 되는 것.
산출물: 그 작업의 결과 한 줄과 확인 방법.
금지: 이미 다른 담당이 맡은 작업을 다시 맡기, 다른 담당의 완료 처리.

역할 카드 C. 검토자

이름: reviewer (같은 사람이 프롬프트만 바뀌도 된다)
하는 일: 누락, 중복, 완료 근거 부족을 찾는다. 결과를 다시 쓰거나 발송하지 않는다.
입력: 원래 작업지시, 작업자가 남긴 결과, 작업표.
산출물: 통과/반려, 빠진 근거, 다시 열어야 할 작업 ID.
금지: 근거 없이 완료로 바꾸기, 범위 밖 파일 수정.

작업을 맡는 이름은 **main** 과 **sub** 두 개만 써도 됩니다. 검토자는 작업표와 완료 근거를 보는 사람(또는 같은 도구에 준 다른 지시)입니다. 이름이 두 개여도 역할은 셋입니다.

한 바퀴 순서

처음에는 이 순서만 기억하면 됩니다.

1. **시작**: 작업표에 세 작업을 **대기**로 적는다.

2. **작업 맡기(claim):** 비어 있는 작업의 담당 칸에 이름과 시작 시각을 적는다. 그 순간 그 작업지시는 그 역할의 것이 된다.
3. **반려:** 다른 담당의 작업을 맡거나 완료 처리하려 하면 검토자가 거부한다. 이 거부가 반려다.
4. **재개:** 멈춘 일은 담당이 반납(release)하거나 기한이 지난 뒤에만 다시 맡는다.
5. **완료 근거:** 결과 칸에 결과와 확인 방법을 남긴다.
6. **종료:** 세 작업이 모두 완료 인지 작업표로 본다.
7. **보관:** 다음 프로젝트를 시작할 때는 끝난 작업표를 지우지 말고 따로 보관한 뒤 새 표를 연다.

2부는 이 순서를 장마다 하나씩 풀어 설명합니다. 1부의 원칙은 같은 프로젝트에 대한 해설입니다.

기대 결과

이 책을 끝까지 읽고 따라 적으면 손에 남는 것은 추상 원칙이 아니라 아래 산출물입니다.

- 역할 카드 3장(위 양식을 내 일에 복사한 것)
- 세 줄짜리 작업표(문서나 스프레드시트)
- 반려와 재개를 기록한 메모 두 장
- 완료 근거가 채워진 결과 칸
- 빈 템플릿과 같은 사례의 작성 예시 짝

필요한 것은 문서 편집기나 스프레드시트 하나뿐입니다. 표와 양식은 모두 이 책 본문에 실려 있으니 그대로 옮겨 쓰면 됩니다.

30분 빠른 시작

책을 읽는 첫날, 가상 사례 세 작업을 시작·말기·검증까지 닫는 최소 루프입니다. 문서나 스프레드시트 한 장이면 충분하고, 실제 계정 연결, 외부 발송, 업로드는 하지 않습니다.

시간	할 일	통과 기준
0-5분	역할 카드 3장을 읽는다	메인·작업자·검토자의 입력과 산출물이 있다
5-12분	작업표에 세 작업을 적는다	T-001~T-003이 모두 대기다
12-20분	main과 sub가 서로 다른 작업을 맡는다	담당이 겹치지 않는다
20-26분	반려 기록과 재개 기록을 한 장씩 써 본다	누가, 왜, 다음에 무엇을 하는지 적혀 있다
26-30분	결과·근거 칸을 채우고 한 줄 보고로 닫는다	완료 3, 빈 근거 칸 0

기록 방식

성공/실패를 감으로 쓰지 말고, 작업 ID와 근거를 같이 남깁니다. 끝난 작업표는 지우지 않고 날짜를 붙여 보관합니다.

왜 에이전트 하나로도 부족한가

학습용 가상 사례 「가을 워크숍 안내문」에 필요한 운영 원칙.

01 · 병목

1장. 에이전트 1개의 한계

단일 에이전트 병목: 목표 → 실행 → 검토 → 보고 → 승인

작업보드 예시: T-001 접수 · 담당=main · 검증 전 외부 발송 금지 · 완료 근거만 보고

역할 분리 카드: 메인: 범위 결정 · 작업자: 산출물 작성 · 검토자: 조건 확인

이 가상 프로젝트에서는

학습용 가상 사례 「가을 워크숍 안내문」을 한 대화에 몰아넣으면 이렇게 됩니다. “목차 써줘, 일정도 확인해줘, 보고도 써줘.” 에이전트는 세 일을 한 맥락에서 붙잡습니다. 목차 초안을 쓰다가 일정 확인을 섞고, 한 줄 보고를 쓰면서 이미 정한 제목을 바꿉니다. 이 장의 규칙은 단순합니다. 메인·작업자·검토자를 나누고, 한 역할에는 한 종류의 입력만 줍니다.

기대 결과: 역할 카드 3장이 채워지고, 한 창에 세 일을 동시에 맡기지 않겠다는 표시가 남습니다. 다음 행동: 오늘 할 일을 조율·실행·검토 세 지시로 나눠 붙여 넣습니다.

사고: 한 창에 모든 일을 몰아넣으면 똑똑한 도구도 흐려진다

AI 에이전트 하나만으로도 많은 일을 처리할 수 있습니다. 초안을 쓰고, 코드를 고치고, 자료를 요약하고, 체크리스트를 만들 수 있습니다. 그래서 처음에는 자연스럽게 한 에이전트에게 모든 일을 맡기게 됩니다. “이것도 해줘”, “방금 한 일 이어서 저것도 해줘”, “아까 말한 파일 다시 보고 수정해줘” 같은 식입니다.

문제는 어느 순간부터 시작됩니다. 에이전트가 앞에서 정한 결정을 흐릿하게 기억하고, 작업의 우선순위가 뒤섞이고, 긴 대화 안에서 오래된 지시와 최신 지시를 동시에 붙잡으려 합니다. 사용자는 분명히 “이제 A는 끝났고 B만 해”라고 생각하지만, 에이전트는 A의 잔여 맥락을 계속 끌고 갑니다. 그러다 보면 새 작업을 시작했는데 이전 작업의 파일명을 언급하거나, 이미 끝난 검증을 다시 하거나, 반대로 꼭 해야 할 확인을 빠뜨립니다.

속도도 문제입니다. 한 에이전트가 긴 파일을 읽고, 고치고, 테스트하고, 보고하는 동안 다른 작은 일은 줄을 서서 기다립니다. 사람은 “AI가 일하고 있으니 괜찮다”고 생각하지만,

실제로는 모든 일이 한 줄 대기열에 갇힌 상태입니다. 사람이 직접 할 때와 다르게 보일 뿐 병목은 여전히 하나입니다.

더 위험한 문제는 검증입니다. 한 에이전트가 만든 결과를 같은 에이전트가 바로 “괜찮다”고 판단하면, 놓친 부분을 다시 놓칠 가능성이 큼니다. 특히 글쓰기, 코드 수정, 발송 전 문안처럼 주관과 실수가 섞이는 작업은 만든 사람과 검토하는 사람의 시야를 나누는 것만으로도 품질이 올라갑니다.

에이전트 하나의 한계는 지능 부족이 아니라 운영 구조 부족에서 온다.

그래서 생긴 룰: 한 에이전트에게 모든 역할을 주지 않는다

여러 에이전트를 쓴다는 말은 단순히 창을 많이 띄운다는 뜻이 아닙니다. 핵심은 역할을 나누는 것입니다. 한 에이전트는 조율자 역할을 하고, 다른 에이전트는 특정 작업을 맡고, 또 다른 에이전트는 검토만 맡습니다. 실제로 여러 구독이나 여러 컴퓨터가 없어도 이 사고방식은 쓸 수 있습니다. 같은 Claude Code 안에서도 “메인 역할”, “초안 작성 역할”, “검토 역할”을 분리해 지시하면 됩니다.

역할을 나누면 세 가지가 좋아집니다.

첫째, 맥락이 작아집니다. 서브 역할에는 필요한 배경만 줍니다. 전체 프로젝트 히스토리를 모두 넘기지 않고 “이 장의 초안만 써라”, “이 스크립트의 보안 문제만 보라”처럼 줍니다. 그러면 불필요한 기억이 줄어듭니다.

둘째, 기다림이 줄어듭니다. 서로 독립적인 작업은 동시에 준비할 수 있습니다. 예를 들어 메인 에이전트가 목차를 정리하는 동안, 서브 에이전트는 부록 템플릿 초안을 만들고, 다른 서브 에이전트는 예시 표의 사실 관계를 확인할 수 있습니다.

셋째, 검토가 쉬워집니다. 작성자와 검토자의 질문이 다릅니다. 작성자는 “내용을 채웠는가”를 보고, 검토자는 “요구사항을 지켰는가”, “위험한 정보가 들어갔는가”, “독자가 따라 할 수 있는가”를 봅니다. 이 차이가 결과물의 안전성을 만듭니다.

너 환경엔 이렇게 적용한다

처음부터 여러 계정, 여러 PC, 복잡한 자동화를 준비할 필요는 없습니다. 먼저 작업을 세 역할로 나눠보세요.

역할	하는 일	예시 지시
메인	목표를 쪼개고 순서를 정한다	“이 일을 5개 단계로 나눠줘. 각 단계의 완료 기준도 적어줘.”

역할	하는 일	예시 지시
작업자	한 단계만 맡아 결과물을 만든다	“아래 요구사항만 보고 2장 초안을 써줘. 다른 장은 건드리지 마.”
검토자	결과물이 요구사항을 지켰는지 확인한다	“이 초안에서 개인정보, 과장, 누락된 검증 조건을 찾아줘.”

이렇게 하면 에이전트 하나만 쓸 때도 운영 구조가 생깁니다. 실제로는 같은 AI에게 차례로 시킬 수 있습니다. 중요한 것은 “같은 대화에서 계속 이어붙이기”가 아니라, 역할마다 입력을 작게 주고 결과를 다시 메인이 모으는 습관입니다.

예를 들어 전자책을 쓴다면 이렇게 나눌 수 있습니다.

1. 메인: 전체 목차와 장별 메시지를 확정한다.
2. 작업자: 1장 초안만 쓴다.
3. 검토자: 1장이 설계와 어긋나지 않았는지 확인한다.
4. 메인: 검토 결과를 반영해 최종본에 넣는다.

코드 작업도 같습니다.

1. 메인: 고칠 파일과 테스트 기준을 정한다.
2. 작업자: 파일 하나를 고치고 테스트를 실행한다.
3. 검토자: 변경 범위가 과하지 않은지, 보안 문제가 없는지 본다.
4. 메인: 전체 테스트와 배포 여부를 판단한다.

작은 실험

오늘 할 일을 하나 고르세요. 그리고 같은 AI에게도 아래처럼 세 번 나눠 시켜보세요.

역할 1: 너는 조율자다. 아래 작업을 3단계로 나누고, 각 단계의 완료 기준을 적어줘.

작업: [내가 하려는 일]

역할 2: 너는 작업자다. 조율자가 정한 1단계만 처리해줘. 다른 단계는 건드리지 마.

1단계 요구사항:
[붙여넣기]

역할 3: 너는 검토자다. 아래 결과물이 요구사항을 지켰는지 확인해줘.
특히 빠진 조건, 위험한 표현, 사람이 승인해야 할 부분을 찾아줘.

요구사항:
[붙여넣기]

결과물:
[붙여넣기]

이 실험의 목적은 AI를 더 많이 쓰는 것이 아닙니다. 한 에이전트에게 모든 역할을 섞어 맡기지 않는 감각을 익히는 것입니다. 이 감각이 생기면 나중에 진짜 여러 에이전트, 여러 컴퓨터, 자동 큐로 확장해도 덜 꼬입니다.

이번 장의 핵심

에이전트 하나가 부족한 이유는 능력이 낮아서가 아닙니다. 한 대화에 목표, 실행, 검토, 보고, 승인까지 모두 넣으면 운영상 병목과 맥락 오염이 생기기 때문입니다. 역할을 나누면 같은 AI를 쓰더라도 결과가 안정됩니다. 멀티에이전트 운영은 창을 많이 띄우는 기술이 아니라, 역할과 책임을 작게 나누는 습관에서 시작합니다.

실제 운영에서는

실제 운영에서는 한 창에 실행·검토·보고를 모두 몰지 않습니다. 현재 운영 기준은 작업 1건마다 담당 1명, 검증 1회, 완료 보고 1회를 분리하는 방식입니다.

이번 장 복붙 프롬프트

아래 작업을 메인, 작업자, 검토자 역할로 나눠줘. 각 역할이 가져야 할 입력과 내야 할 산출물을 표로 정리해줘.

2장. 일감 자동 분배

work-stealing 결정: 대기열 → 빈 작업자 확인 → 맡기 → 작업 → 반환

작업보드 예시: 대기 3 · 진행 중 1 · 보류 0 · 완료 2

분배 체크: 독립성 · 공유 파일 · 순차 의존 · 검증 기준

이 가상 프로젝트에서는

안내문 목차(T-001), 사실 확인(T-002), 한 줄 보고(T-003)를 사람이 매번 “너는 이거, 너는 저거”라고 배정하면 세 일이어도 곧 지칩니다. 큐에 세 줄을 넣고 노는 역할이 집어가게 하면, 메인은 상태표만 보면 됩니다. 2부의 작업표에서 작업을 맡는 동작(claim)이 이 장을 실제로 옮긴 모습입니다.

기대 결과: 세 작업이 `todo` 로 한 표에 있고, 누가 집었는지를 사람이 기억하지 않아도 됩니다. 다음 행동: 내 일에서도 완료 조건이 있는 작은 작업만 큐에 넣습니다.

사고: 사람이 매번 배정하면 자동화가 다시 수동 일이 된다

여러 에이전트를 쓰기 시작하면 처음에는 일이 빨라지는 것처럼 보입니다. 한쪽에는 문서 초안, 다른 쪽에는 코드 수정, 또 다른 쪽에는 검토를 맡깁니다. 하지만 곧 새로운 일이 생깁니다. 사람이 계속 “너는 이것”, “너는 저것”, “이건 아직 하지 마”, “끝났으면 다음 것 해”라고 지시해야 합니다.

이 상태에서는 에이전트가 늘어날수록 사람이 더 바빠집니다. 에이전트의 작업 속도를 관리하고, 어느 에이전트가 놓고 있는지 확인하고, 중복으로 같은 일을 맡기지 않도록 조심해야 합니다. 자동화를 만들었는데 사람이 배차 담당자가 되어버리는 셈입니다.

작업이 세 개일 때는 괜찮습니다. 하지만 열 개가 넘어가면 문제가 커집니다. 어떤 작업은 작아서 금방 끝나고, 어떤 작업은 테스트 때문에 오래 걸립니다. 사람이 미리 “이 작업은 10분, 저 작업은 40분”이라고 정확히 예측하기 어렵습니다. 그래서 한 에이전트는 오래 붙잡혀 있고, 다른 에이전트는 할 일이 없는데도 기다립니다.

또 다른 사고는 우선순위 누락입니다. 사람이 급한 일만 눈앞에서 배정하다 보면 작지만 필요한 일, 예를 들어 검증 로그 정리나 문서 업데이트가 뒤로 밀립니다. 나중에 보면 결과물은 있는데 근거가 없고, 코드 변경은 있는데 설명이 없고, 작업은 끝났는데 완료 표시가 없습니다.

여러 에이전트의 진짜 병목은 에이전트 수가 아니라 일감이 흐르는 방식이다.

그래서 생긴 룰: 일감은 큐에 넣고, 노는 에이전트가 집어간다

이 문제를 해결하는 가장 단순한 방법은 일감 큐입니다. 큐는 할 일을 한 줄씩 적어두는 목록입니다. 각 일감에는 최소한 네 가지가 있어야 합니다.

필드	의미
id	작업을 구분하는 번호
status	todo, claimed, done 같은 상태
owner	지금 누가 맡았는지
result	완료 근거 또는 결과 파일

사람은 에이전트에게 직접 배정하지 않고 큐에 일을 넣습니다. 에이전트는 큐를 보고 **todo** 상태의 일을 하나 집어갑니다. 집어간 순간 상태를 **claimed** 로 바꾸고 **owner**를 적습니다. 끝나면 **done** 으로 바꾸고 결과를 남깁니다.

이 방식을 work-stealing이라고 부를 수 있습니다. 직역하면 “일감 훔치기”지만 실제 의미는 단순합니다. 놓고 있는 작업자가 남은 일을 스스로 가져가는 방식입니다. 사람이 모든 배정을 직접 하지 않아도 됩니다.

work-stealing이 잘 동작하려면 일감이 작아야 합니다. “전자책 완성”은 너무 큼니다. “1장 초안 작성”, “예시 표 사실 확인”, “PDF 1~5쪽 렌더 확인”처럼 쪼개야 합니다. 작은 일감은 집기 쉽고, 실패해도 다시 배정하기 쉽고, 완료 표시도 명확합니다.

너 환경엔 이렇게 적용한다

처음에는 스프레드시트 하나면 충분합니다. 아래 컬럼을 만드세요.

id	status	owner	task	done_when	result
1	todo	미배정	1장 초안 쓰기	800자 이상, 핵심 메시지 포함	대기
2	todo	미배정	1장 검토하기	누락/위험 표현 3개 이하 확인	대기
3	todo	미배정	PDF 렌더 확인	깨진 글자/겹침 없음	대기

Claude Code 한 개만 있어도 이 표를 쓸 수 있습니다. 메인 역할이 표를 만들고, 서브 역할에게 “todo 상태 중 하나를 골라 claimed로 바꾼 뒤 처리해”라고 시킵니다. 실제 자동화가 아니어도 괜찮습니다. 중요한 것은 사람이 매번 머리로 기억하지 않고 큐를 기준으로 대화하는 것입니다.

조금 더 익숙해지면 파일로 바꿀 수 있습니다. 예를 들어 `tasks.jsonl` 처럼 한 줄에 작업 하나를 저장합니다. 그러면 스크립트가 `todo` 작업을 찾아 claimed로 바꾸고, 결과를 기록할 수 있습니다. 이 책의 2부에서는 이 축소 모형을 작업표 한 장으로 따라가 봅니다.

큐에 넣기 좋은 일감과 나쁜 일감

큐에 넣기 좋은 일감은 완료 기준이 분명합니다.

- “3장 초안을 쓴다”보다 “3장에 사고 사례 1개, 룰 1개, 적용법 1개를 포함해 초안을 쓴다”가 좋습니다.
- “코드를 고친다”보다 “테스트 A가 통과하도록 파일 B만 수정한다”가 좋습니다.
- “검토한다”보다 “비밀값, 실명, 전화번호 노출 여부를 찾아 결과표로 남긴다”가 좋습니다.

반대로 큐에 넣기 나쁜 일감은 판단 범위가 너무 큼니다.

- “사업 전략을 정한다”
- “고객에게 보낼 최종 문안을 승인한다”
- “결제 금액을 입력한다”
- “운영 서버에 배포한다”

이런 일은 사람의 승인 게이트로 남겨야 합니다. 에이전트는 초안과 근거를 만들 수 있지만, 비가역 결정을 대신하면 안 됩니다.

작은 큐 운영 규칙

큐를 만들었다면 아래 규칙을 붙이세요.

1. 작업자는 `todo` 만 집을 수 있다.
2. 집는 순간 `claimed` 와 owner를 기록한다.
3. 이미 `claimed` 인 작업은 다른 작업자가 건드리지 않는다.
4. 오래 멈춘 `claimed` 는 메인이 확인한 뒤 다시 `todo` 로 되돌릴 수 있다.
5. 완료할 때는 결과 파일, 테스트 결과, 요약 중 하나를 반드시 남긴다.

이 다섯 줄만 있어도 중복 작업이 크게 줄어듭니다. 특히 5번이 중요합니다. “끝났어요”만 남기면 다음 사람이 확인할 수 없습니다. “끝났고, 결과는 이 파일이며, 이 명령으로 검증했다”까지 남겨야 큐가 기억장치가 됩니다.

이번 장의 핵심

멀티에이전트 운영의 첫 번째 구조는 일감 큐입니다. 사람이 모든 배정을 손으로 하면 에이전트가 늘수록 사람이 더 바빠집니다. 일을 작게 쪼개고, 큐에 넣고, 노는 에이전트가 집어가게 하세요. 처음에는 스프레드시트로 충분합니다. 중요한 것은 자동화 도구가 아니라 “일은 큐를 통해 흐른다”는 운영 원칙입니다.

실제 운영에서는

실제 운영에서는 쉬고 있는 작업자만 새 일을 맡습니다. 서로 독립된 일이 아니면 여러 에이전트로 나누지 않고, 작은 수정은 한 담당이 끝내는 규칙을 둡니다.

이번 장 복붙 프롬프트

아래 할 일 목록을 독립 작업과 순차 작업으로 나누고, work-stealing 해도 되는 항목만 골라줘.

03 · 충돌

3장. 충돌 방지

충돌 방지 흐름: task id → owner → lock → diff → report

작업 큐 예시: [~] T-001 담당=main · [] T-002 담당=sub · [x] T-003 근거=보고

검침 확인: 파일 · 상태 · 브랜치 · 검증

이 가상 프로젝트에서는

main 이 T-001 목차 초안을 쓰는 동안 sub 가 같은 T-001을 다시 집으면, 안내문 제목이 두 벌이 됩니다. 영역은 작업 ID입니다. T-001을 집은 역할만 그 결과를 남깁니다.

T-002는 다른 레일입니다. 이미 담당이 있는 작업을 다시 맡으려 하면 검토자가 거부합니다.

기대 결과: 같은 작업 ID를 두 역할이 동시에 소유하지 않습니다. 다음 행동: 작업지시문에 “수정 가능 범위 / 금지 범위”를 한 줄씩 적습니다.

사고: 둘이 같은 곳을 고치면 빠른 만큼 빨리 망가진다

멀티에이전트 운영에서 가장 자주 터지는 사고는 충돌입니다. 한 에이전트가 문서를 고치고 있는데 다른 에이전트도 같은 문서를 고칩니다. 한쪽은 목차를 바꾸고, 다른 쪽은 기존 목차를 기준으로 본문을 씁니다. 나중에 합치려고 보면 둘 다 맞는 말을 했지만 서로 기준이 다릅니다.

코드에서는 더 뚜렷합니다. 한 에이전트가 함수 이름을 바꾸고, 다른 에이전트가 예전 이름으로 테스트를 추가합니다. 한쪽은 설정 파일을 정리하고, 다른 쪽은 그 설정을 읽는 스크립트를 고칩니다. 각각의 작업만 보면 합리적이지만 동시에 진행되면 전체는 깨집니다.

문서 작업도 다르지 않습니다. 1장 담당자가 용어를 “작업자”라고 쓰고, 2장 담당자는 “워커”라고 쓰고, 부록 담당자는 “서브봇”이라고 쓰면 독자는 같은 개념인지 다른 개념인지 헷갈립니다. 병렬 작업은 빠르지만, 기준이 없으면 결과물이 조각납니다.

충돌은 항상 큰 사고로 보이지 않습니다. 더 위험한 것은 조용한 충돌입니다. 파일이 깨지지는 않았지만, 앞뒤 말이 달라지고, 완료 기준이 바뀌고, 같은 작업을 두 번 설명합니다. 이런 충돌은 마지막 검수에서야 드러나고, 그때는 고치는 비용이 큼니다.

병렬 작업은 같은 속도로 같은 곳을 향해 달릴 때가 아니라, 서로 다른 레일을 가질 때 안전하다.

그래서 생긴 룰: 영역, 브랜치, lease를 나눈다

충돌 방지의 핵심은 “누가 무엇을 건드릴 수 있는가”를 먼저 정하는 것입니다. 어려운 말로 하지 않아도 됩니다. 세 가지 질문이면 충분합니다.

1. 이 작업자는 어떤 파일 또는 범위만 수정하는가?
2. 이 작업자가 건드리면 안 되는 영역은 무엇인가?
3. 작업 중이라는 표시를 어디에 남기는가?

첫째는 영역 분리입니다. 전자책을 예로 들면 1부 담당자는 `chapters/01` 부터 `chapters/06` 만 고칩니다. 2부 담당자는 `chapters/07` 부터 `chapters/10` 만

고칩니다. 부록 담당자는 `templates/` 만 고칩니다. 이렇게 나누면 서로 발을 밟을 확률이 줄어듭니다.

둘째는 브랜치 또는 작업 사본입니다. 코드 프로젝트라면 각 작업자는 자기 브랜치에서 수정하고, 마지막에 메인에 합칩니다. 문서 프로젝트라면 폴더나 파일 단위로 소유권을 나눕니다. 중요한 것은 한 작업자가 다른 작업자의 변경을 지우지 않는 것입니다.

셋째는 lease입니다. lease는 “이 작업은 지금 내가 잠시 맡고 있다”는 임시 권한 표시입니다. 영구 소유가 아닙니다. 예를 들어 30분 동안 작업 3번을 맡겠다고 표시하고, 끝나면 반납합니다. 시간이 지나도 완료가 안 되면 메인이 확인해 다시 풀 수 있습니다.

너 환경엔 이렇게 적용한다

아래 표를 작업 시작 전에 만드세요.

작업자	맡은 범위	건드리지 말 것	완료 근거
메인	목차, 통합, 최종 검증	작업자 초안 직접 덮어쓰기	전체 체크리스트
작업자 A	1부 초안	2부, 부록, 빌드 스크립트	장별 파일
작업자 B	2부 초안	1부, 부록, 빌드 스크립트	장별 파일
검토자	개인정보·표현 검수	본문 수정 직접 반영	이슈 목록

이 표를 만들면 에이전트에게 지시하기 쉬워집니다.

```
너는 작업자 A다.
수정 가능 범위: chapters/01-*.md부터 chapters/06-*.md까지.
수정 금지 범위: chapters/07 이후, templates/, 빌드 스크립트.
목표: 각 장을 사고 -> 를 -> 적용 순서로 작성.
완료 보고: 수정한 파일 목록과 남은 우려를 적어라.
```

이렇게 적으면 에이전트가 “도움이 되려고” 다른 파일까지 손대는 일을 줄일 수 있습니다. AI는 친절해서 요청받지 않은 정리까지 하려는 경우가 있습니다. 멀티에이전트 운영에서는 그 친절이 충돌을 만듭니다.

잠금은 자유를 줄이는 게 아니다

처음에는 잠금과 lease가 답답하게 느껴질 수 있습니다. 하지만 잠금의 목적은 일을 못 하게 막는 것이 아닙니다. 동시에 같은 문을 지나가다 부딪히지 않게 하는 것입니다.

작은 프로젝트에서는 파일 이름 옆에 owner를 적는 것만으로도 충분합니다.

파일	owner	상태
chapters/01.md	A	working
chapters/02.md	A	todo
chapters/07.md	B	working
templates/t01.md	C	done

조금 더 자동화하면 큐에서 claim을 할 때 owner와 시간을 기록합니다. 같은 작업을 다른 에이전트가 집으려 하면 “이미 claimed”라고 알려줍니다. 2부의 작업표에서도 같은 구조를 씁니다.

충돌이 났을 때의 복구 순서

충돌은 완전히 없앨 수 없습니다. 그래서 복구 순서가 필요합니다.

1. 누가 어떤 파일을 고쳤는지 목록을 만든다.
2. 가장 최신 지시와 설계 문서를 기준으로 삼는다.
3. 두 변경 중 하나를 조용히 버리지 않는다.
4. 유지할 내용, 버릴 내용, 다시 써야 할 내용을 표로 나눈다.
5. 이후 같은 충돌이 나지 않도록 소유 범위나 lease 시간을 조정한다.

복구에서 가장 나쁜 행동은 “대충 합치기”입니다. 대충 합치면 파일은 저장되지만 기준이 사라집니다. 충돌은 단순한 텍스트 문제가 아니라 의사결정 충돌일 때가 많습니다.

이번 장의 핵심

멀티에이전트 운영에서 충돌 방지는 속도를 늦추는 절차가 아닙니다. 속도를 유지하기 위한 레일입니다. 작업 전에는 수정 범위, 금지 범위, 완료 근거를 정하세요. 작업 중에는 lease로 “누가 무엇을 맡았는지” 남기세요. 충돌이 나면 조용히 덮어쓰지 말고 기준 문서로 돌아가세요. 병렬성은 규칙이 있을 때만 이득이 됩니다.

실제 운영에서는

실제 운영에서는 같은 파일을 동시에 고치지 않도록 작업 ID, 담당, 잠금 표시를 같이 봅니다. 충돌 가능성이 보이면 먼저 반환합니다.

이번 장 복붙 프롬프트

두 에이전트가 동시에 건드리면 안 되는 파일과 상태를 찾아 충돌 방지 규칙으로 써줘.

04 · 원장

4장. 중복·누락 막기

완료 원장: 산출물 → 검증 → 증거 → 보고 → 마킹

완료 보고 카드: 제목 · 보낸 역할 · 근거 · 남은 위험

done 조건: 파일 존재 · 확인 통과 · 개인정보 점검 · 승인 여부

이 가상 프로젝트에서는

목차 초안을 말로만 “끝냈다”고 하면, 다음 역할은 T-001을 다시 시작합니다. 완료 도장은 작업표의 결과·근거 칸에 “목차 6개 섹션 초안, outline.md”처럼 결과와 파일이 남을 때 찍힙니다. 코드 작업이라면 파일 경로와 확인 명령을 같은 칸에 붙입니다.

기대 결과: 완료된 작업은 결과 칸이 비어 있지 않습니다. 다음 행동: “완료했습니다”만 있는 보고는 완료로 받지 않습니다.

사고: 끝났는데 끝났다고 남기지 않으면 끝난 일이 아니다

AI 에이전트는 일을 빠르게 처리하지만, 사람이 기대하는 방식으로 “기억”하지는 않습니다. 어떤 작업을 끝냈더라도 그 사실이 큐, 문서, 로그, 완료 표시 중 어딘가에 남아 있지 않으면 다음 작업자는 모릅니다. 그래서 같은 일을 다시 시작합니다.

예를 들어 한 에이전트가 PDF 빌드를 마쳤다고 합시다. 그런데 결과 파일 경로, 페이지 수, 검증 결과를 남기지 않았습니. 다음 에이전트는 작업 목록만 보고 “PDF 빌드 필요”라고 판단합니다. 다시 빌드합니다. 운이 좋으면 시간만 낭비합니다. 운이 나쁘면 이전에 통과한 결과를 덮어쓰고, 더 나쁜 파일을 새 최종본으로 착각합니다.

반대 사고도 있습니다. 실제로는 끝나지 않았는데 끝난 것처럼 보이는 경우입니다. 초안은 있지만 검수가 없고, 테스트는 돌렸지만 실패했으며, 파일은 만들어졌지만 열어보니 깨져 있습니다. 그런데 작업표에는 “완료”라고 적혀 있습니다. 이 경우 다음 작업자는 검증을 건너뜁니다.

중복과 누락은 모두 같은 뿌리에서 나옵니다. 결과와 근거가 분리되어 있기 때문입니다. “했다”는 말과 “무엇을 만들었고 어떻게 확인했는가”가 함께 남아야 합니다.

완료 표시는 기분이 아니라 증거다.

그래서 생긴 룰: 완료 도장에는 근거를 붙인다

작업을 완료할 때는 최소 세 가지를 남깁니다.

1. 무엇을 만들었는가
2. 어디에 있는가
3. 어떻게 확인했는가

예를 들어 이렇게 남깁니다.

```
완료: 2장 초안 작성
파일: chapters/02-work-stealing.md
확인: 장 핵심 메시지 포함, 사고/룰/적용 3단 구성 확인
남은 일: 전체 용어 통일은 최종 검수에서 확인
```

코드라면 이렇게 남깁니다.

```
완료: 일감 큐 말기(claim) 기능
파일: scripts/task_queue.py
확인: `python3 -m unittest` 실행, 테스트 3개 통과
남은 일: README에 사용 순서 반영 필요
```

이 정도만 남겨도 다음 에이전트가 상황을 훨씬 정확히 이해합니다.

너 환경엔 이렇게 적용한다

스프레드시트든 메모장이든 작업표에 아래 컬럼을 추가하세요.

컬럼	적는 내용
status	todo, claimed, done, blocked
result_path	결과 파일 또는 링크
verification	실행한 확인 명령 또는 육안 확인 내용

컬럼	적는 내용
next_action	남은 후속 작업

완료 보고 템플릿은 이렇게 쓰면 됩니다.

```
작업 ID:
결과:
결과 위치:
검증:
남은 위험:
다음 사람이 해야 할 일:
```

이 템플릿은 에이전트에게 그대로 줄 수 있습니다. “작업 끝나면 이 형식으로만 보고해” 라고 하면 보고 품질이 올라갑니다.

완료와 관찰을 구분한다

작업에는 세 가지 상태가 있습니다.

첫째, 완료입니다. 필요한 결과물이 있고, 검증도 끝났고, 더 이상 기다릴 외부 조건이 없습니다.

둘째, 진행 중입니다. 누군가 맡았지만 아직 결과가 없습니다.

셋째, 관찰 또는 대기입니다. 빌드는 끝났지만 하루 동안 오류가 없는지 봐야 하거나, 외부 승인을 기다려야 하거나, 사람이 확인해야 하는 상태입니다. 이 상태를 완료로 표시하면 사고가 납니다.

예를 들어 “상품 등록 직전 PDF 생성”은 완료일 수 있습니다. 하지만 “상품 공개”는 사람이 승인하기 전까지 대기입니다. PDF 파일이 있다고 해서 외부 발행까지 끝난 것은 아닙니다.

멀티에이전트 운영에서는 이 구분이 특히 중요합니다. 에이전트는 “파일이 생겼다”를 “전체 작업 완료”로 착각할 수 있습니다. 그래서 완료 조건을 작업마다 명확히 적어야 합니다.

스테일 작업을 줄이는 법

스테일 작업은 오래된 지시가 뒤늦게 실행되는 사고입니다. 이미 해결된 문제를 다시 고치거나, 더 최신 결정과 충돌하는 옛 계획을 실행하는 식입니다.

스테일을 줄이려면 작업 시작 전에 세 가지를 확인합니다.

1. 이 작업이 아직 열려 있는가?

2. 같은 결과물이 이미 만들어졌는가?

3. 더 최신 지시가 있는가?

이 확인을 어렵게 만들 필요는 없습니다. 작업표에서 status를 보고, 결과 폴더를 보고, 최근 보고 한 줄을 읽으면 됩니다. 중요한 것은 에이전트가 “받은 지시가 항상 최신”이라고 가정하지 않게 만드는 것입니다.

프롬프트에 이렇게 넣으세요.

작업 시작 전 확인:

1. 작업표에서 이 작업이 아직 `todo` 또는 `claimed`인지 확인해라.
2. 결과 파일이 이미 있으면 덮어쓰기 전에 보고해라.
3. 최신 설계 문서와 충돌하면 멈추고 질문해라.

이번 장의 핵심

멀티에이전트 운영에서 “완료”는 말이 아니라 증거입니다. 결과 파일, 확인 방법, 남은 위험을 같이 남겨야 중복과 누락이 줄어듭니다. 완료, 진행 중, 관찰/대기를 구분하세요. 오래된 지시가 뒤늦게 실행되지 않도록 시작 전 stale 확인을 습관으로 만드세요. AI 팀의 기억력은 에이전트 머릿속이 아니라 작업표와 완료 근거에서 나옵니다.

실제 운영에서는

실제 운영에서는 완료를 말로 단지 않습니다. 산출물 위치, 확인 결과, 접수번호 같은 근거가 있을 때만 완료로 넘깁니다.

이번 장 복붙 프롬프트

완료 보고에 필요한 근거 항목을 정하고, 완료로 표시하기 전 체크리스트를 만들어줘.

5장. 사람은 폰으로만 감독

사람 승인 게이트: 초안 → 검토 → 사람 확인 → 제출 → 접수

폰 승인 흐름: AI: 칸 안내 · 사람: 인증 · AI: 금액 대조 · 사람: 제출

사람 승인 행동: 외부 발송 · 배포 · 공식 제출 · 유료 전환

이 가상 프로젝트에서는

안내문 목차 초안, 사실 확인, 한 줄 보고는 내부에서 되돌릴 수 있습니다. 사람은 모든 줄을 승인하지 않습니다. 다만 이 가상 안내문을 실제 참석자에게 보내거나, 공개 페이지에 올리거나, 유료 신청 링크를 켜는 일은 사람 승인 전 멈춥니다. 학습용 가상 사례이므로 이 책은 그 버튼을 누르지 않습니다.

기대 결과: 자율/검토/승인 경계가 작업 종류마다 한 줄로 적힙니다. 다음 행동: 템플릿 03의 승인 체크리스트를 이 프로젝트 이름으로 한 번 채웁니다.

사고: 모든 것을 승인하면 사람이 병목이고, 아무것도 승인하지 않으면 시스템이 위험하다

AI 에이전트를 쓰다 보면 두 극단 사이에서 흔들립니다. 하나는 모든 행동을 사람이 확인하는 방식입니다. 초안 하나, 파일 하나, 테스트 하나마다 “해도 될까요?”라고 묻습니다. 처음에는 안전해 보이지만 곧 사람이 병목이 됩니다. 에이전트는 계속 멈추고, 사람은 작은 확인에 지칩니다.

다른 극단은 모든 것을 자율로 넘기는 방식입니다. 초안 작성, 파일 수정, 배포, 발송, 결제, 삭제까지 모두 에이전트가 판단합니다. 이 방식은 빠르지만 위험합니다. 로컬 초안이 틀리는 것은 고치면 됩니다. 하지만 고객에게 잘못된 메시지가 발송되거나, 유료 결제가 실행되거나, 공개 페이지가 배포되거나, 데이터가 삭제되면 복구 비용이 커집니다.

좋은 운영은 둘 사이에 있습니다. 사람은 모든 일을 승인하지 않습니다. 대신 되돌리기 어려운 행동만 승인합니다. 에이전트는 초안, 검증, 정리, 로컬 파일 작성처럼 되돌릴 수 있는 일을 자율로 처리합니다. 사람은 폰으로 핵심 질문만 받습니다.

사람의 역할은 마이크로매니저가 아니라 비가역 행동의 차단기다.

그래서 생긴 룰: 권한과 자문을 분리한다

AI가 “좋아 보인다”고 말하는 것과 “해도 된다”는 권한은 다릅니다. 이 둘을 섞으면 사고가 납니다.

예를 들어 에이전트가 어떤 발송 문안을 보고 “취지에 맞고 위험이 낮다”고 평가할 수 있습니다. 이것은 자문입니다. 하지만 그 평가가 사람 승인으로 바뀌면 안 됩니다. 자문은 품질 신호이고, 권한은 행동 허가입니다.

권한은 두 가지에서만 나와야 합니다.

1. 사람이 직접 승인했다.
2. 미리 정한 사전위임 범위 안이다.

사전위임은 예를 들어 이런 것입니다.

- 로컬 문서 초안 작성
- 테스트 실행
- 공개되지 않는 파일 정리
- 브랜치에 작업 커밋
- 검토용 PDF 생성

사람 승인이 필요한 것은 이런 것입니다.

- 고객이나 외부 채널로 발송
- 결제, 환불, 가격 입력
- 공개 배포
- 삭제나 덮어쓰기
- 개인정보 처리
- 계정, 토큰, 인증 정보 변경

AI 자문은 이 둘을 대신할 수 없습니다. “AI가 괜찮다고 했다”는 이유로 발송하거나 배포하지 않습니다.

너 환경엔 이렇게 적용한다

작업을 시작하기 전에 행동을 세 그룹으로 나누세요.

그룹	예시	처리
자율	초안, 요약, 로컬 파일, 테스트	에이전트가 진행

그룹	예시	처리
검토 필요	공개 전 문안, 중요한 코드 변경	에이전트가 근거를 만들고 사람이 확인
승인 필요	발송, 결제, 배포, 삭제	사람의 명시 승인 전 중단

이 표를 에이전트에게 보여주고, 아래 규칙을 붙입니다.

권한 규칙:

- 로컬 초안과 테스트는 진행해도 된다.
- 외부 발송, 결제, 배포, 삭제, 개인정보 처리는 사람 승인 전 멈춘다.
- AI의 품질 판단은 자문일 뿐 승인으로 해석하지 않는다.
- 애매하면 진행하지 말고 승인 요청 문안을 작성한다.

폰 감독의 핵심은 질문을 짧게 만드는 것입니다. 사람에게 긴 로그를 보내면 판단하기 어렵습니다. 승인 요청은 아래 네 줄이면 충분합니다.

하려는 행동:

되돌릴 수 없는 이유:

검증한 근거:

승인하면 실행할 명령 또는 다음 행동:

사람은 이 네 줄을 보고 approve 또는 hold만 결정합니다. 에이전트는 답이 없으면 진행하지 않습니다.

무응답은 승인 아니다

자동화에서 가장 위험한 착각 중 하나는 “답이 없으니 괜찮겠지”입니다. 사람에게 물었는데 답이 없으면 보류입니다. 특히 외부영향이 있는 행동은 무응답을 승인으로 해석하면 안 됩니다.

반대로 자율 범위 안의 작업은 사람 답을 기다리지 않아야 합니다. 로컬 초안 작성, 테스트, 문서 정리는 계속 진행합니다. 그래야 사람은 중요한 질문만 받습니다.

이 균형이 중요합니다.

- 되돌릴 수 있는 일: 자동 진행
- 되돌리기 어려운 일: 사람 승인
- 애매한 일: 보류하고 질문
- 무응답: 보류

승인 게이트를 너무 넓히지 않는다

처음에는 불안해서 모든 것을 승인 게이트에 넣고 싶습니다. 하지만 승인 게이트가 너무 넓으면 시스템은 멈춥니다. 사람이 바쁘거나 자고 있을 때 아무 일도 못 합니다. 그러면 에이전트를 여러 개 쓰는 의미가 줄어들습니다.

게이트는 좁고 단단해야 합니다. “외부영향, 돈, 데이터, 계정, 삭제” 정도로 시작하세요. 나머지는 자율로 두되, 결과 근거를 남기게 합니다. 문제가 생기면 그때 게이트를 조정합니다.

이번 장의 핵심

사람은 모든 일을 승인하는 관리자가 아니라 비가역 행동의 차단기입니다. AI의 자문과 사람의 권한을 분리하세요. 초안과 테스트는 자율로 보내고, 발송·결제·배포·삭제·개인정보 처리는 명시 승인 전 멈추세요. 무응답은 승인으로 해석하지 않습니다. 이 구조가 있어야 AI 팀이 빠르면서도 위험해지지 않습니다.

실제 운영에서는

실제 운영에서는 세금 신고 제출, 스토어 제출, 업로드처럼 법적·외부 효과가 있는 버튼은 사람이 직접 누릅니다. AI는 입력 보조와 체크리스트까지만 말합니다.

이번 장 복붙 프롬프트

아래 작업에서 사람이 직접 승인해야 하는 행동을 찾아 멈춤 지점을 표시해줘.

06 · 비용

6장. 비용 폭주 막기

비용 가드: 분해 판단 → 단일 경로 → 경량 요약 → 검증 → 보고

비용 기록 예시: 반복 2/5회 · 경과 48분 · 병렬 분배 없음 · 외부 호출 없음

사용 전 질문: 큰 작업인가 · 독립인가 · 원문이 큰가 · 요약 가능한가

이 가상 프로젝트에서는

세 작업은 작습니다. 목차 초안, 사실 확인, 보고 한 줄에 긴 추론 루프를 돌릴 필요가 없습니다. 개수 세기나 대조 같은 확인은 표와 간단한 도구가 더 싸고 정확합니다. AI에게 “잘 됐나 확인해줘”라고 하기 전에 작업표와 확인표를 먼저 봅니다.

기대 결과: 비싼 판단과 싼 확인이 표로 나뉩니다. 다음 행동: 같은 요청을 두 번 반복하기 전에 작업표와 결과부터 봅니다.

사고: 자동 루프는 성실해서 위험하다

AI 에이전트는 지치지 않습니다. 사람이 지루해질 반복 확인도 계속하고, 같은 파일을 다시 읽고, 더 좋은 답을 찾겠다며 여러 번 시도합니다. 이 성실함은 장점이지만 비용 관점에서는 위험합니다. 특히 여러 에이전트가 동시에 움직이면 작은 낭비가 빠르게 커집니다.

처음에는 “한 번 더 검토해줘”가 별일 아닌 것처럼 보입니다. 하지만 이 요청이 여러 장, 여러 파일, 여러 에이전트에 퍼지면 토큰이 쌓입니다. 자동 루프가 실패를 만나 재시도하고, 재시도 결과를 다시 요약하고, 다시 검토하면 비용은 조용히 늘어납니다.

더 위험한 것은 사람이 비용 증가를 늦게 알아차린다는 점입니다. 작업은 진행 중이고 결과도 조금씩 나오기 때문에 멈추기 어렵습니다. 나중에 보면 정말 필요한 판단에는 적은 비용을 쓰고, 반복 읽기와 불필요한 재생성에 많은 비용을 썼다는 사실을 알게 됩니다.

비용 폭주는 한 번의 큰 실수보다 작은 자동 반복이 쌓여 생긴다.

그래서 생긴 룰: 비싼 판단과 싼 반복을 나눈다

모든 작업에 가장 강한 모델이나 긴 추론을 쓸 필요는 없습니다. 작업을 세 종류로 나눠야 합니다.

첫째, 비싼 판단이 필요한 일입니다. 구조 설계, 안전성 판단, 공개 전 최종 문안, 복잡한 버그 원인 분석처럼 실수 비용이 큰 일입니다. 이런 일에는 좋은 모델과 충분한 시간을 써도 됩니다.

둘째, 싼 반복으로 충분한 일입니다. 파일 목록 확인, 형식 검사, 단순 요약, 표 변환, 맞춤법 후보 찾기 같은 일입니다. 이런 일에는 가벼운 모델, 짧은 프롬프트, 스크립트 검증을 씁니다.

셋째, AI가 아니라 프로그램이 해야 하는 일입니다. 줄 수 세기, 파일 존재 확인, JSON 파싱, PDF 페이지 수 확인, 금지어 검색 같은 일은 스크립트가 더 싸고 정확합니다.

에이전트에게 “확인해줘”라고 말하기 전에 명령으로 확인할 수 있는지 먼저 봐야 합니다.

너 환경엔 이렇게 적용한다

작업표에 비용 등급을 추가하세요.

비용 등급	예시	처리
low	파일 목록, 금지어 검색, 페이지 수 확인	스크립트 또는 짧은 요청
medium	장 초안, 요약, 표 정리	일반 모델, 범위 제한
high	구조 변경, 안전 판단, 외부 발행 전 검토	사전경고 후 실행

에이전트에게는 이렇게 지시합니다.

비용 규칙:

- 스크립트로 확인 가능한 것은 먼저 명령으로 확인한다.
- 같은 요청을 2회 이상 반복하기 전에 멈추고 원인을 보고한다.
- 긴 파일 전체를 다시 읽기 전에 필요한 범위를 먼저 찾는다.
- 비용이 큰 AI 호출이나 여러 에이전트에 장시간 나눠 맡기는 일(fan-out)은 사전경고 후 진행한다.

이 규칙은 창의성을 막는 것이 아닙니다. 오히려 비싼 판단에 예산을 남기는 방법입니다.

반복 제한을 둔다

자동화에는 반복 제한이 필요합니다. 실패했을 때 무한히 다시 시도하면 비용과 시간이 사라집니다.

예를 들어 PDF 빌드가 실패하면 이렇게 처리합니다.

1. 첫 실패: 오류 메시지를 읽고 명확한 원인을 고친다.
2. 두 번째 실패: 같은 원인인지 다른 원인인지 분리한다.
3. 세 번째 실패: 자동 반복을 멈추고 사람에게 현재 상태를 보고한다.

이 구조가 없으면 에이전트는 “조금만 더 하면 될 것 같다”는 식으로 계속 시도합니다. 사람도 작업이 진행 중이라고 착각합니다. 반복 제한은 포기가 아니라 관측 장치입니다.

프롬프트도 예산이다

긴 프롬프트는 항상 좋은 것이 아닙니다. 필요한 맥락만 주는 것이 중요합니다. 서브 작업자에게 전체 히스토리를 모두 넘기면 비용도 늘고 혼란도 늘어납니다. 대신 작업 단위별로 필요한 것만 줍니다.

나쁜 예:

이 프로젝트 전체를 읽고 알아서 필요한 걸 다 해줘.

좋은 예:

너는 3장 초안만 담당한다.
반드시 포함할 내용: 사고 1개, 룰 1개, 독자 적용법 1개.
수정 가능 파일: chapters/03-conflict-prevention.md.
수정 금지: 다른 장, 부록 템플릿, 빌드 스크립트.
완료 보고: 파일 경로와 남은 우려.

짧지만 경계가 분명한 프롬프트가 비용과 품질을 함께 잡습니다.

비용 보고를 부끄러워하지 않는다

에이전트 운영에서 비용 경고는 방해가 아닙니다. “이 작업은 긴 검토가 필요해서 토큰을 많이 쓸 수 있다”는 보고는 좋은 보고입니다. 사람은 그때 작업을 줄이거나, 나중에 미루거나, 더 싼 방법을 선택할 수 있습니다.

특히 아래 작업은 사전경고가 필요합니다.

- 긴 문서 여러 개 전수 검토
- 여러 에이전트에 나눠 맡기기
- LLM으로 대량 생성
- 반복 재작성
- 모호한 문제의 장시간 디버깅

사전경고는 허락을 받기 위한 형식이 아니라, 비용을 운영 대상으로 만드는 장치입니다.

이번 장의 핵심

AI 팀은 빠르지만 공짜가 아닙니다. 자동 루프와 병렬 작업은 작은 낭비를 크게 키울 수 있습니다. 비싼 판단, 싼 반복, 스크립트 검증을 나누세요. 반복 횟수에 제한을 두고, 긴 작업 전에는 비용을 경고하세요. 비용 가드는 AI를 덜 쓰자는 말이 아니라, 중요한 곳에 제대로 쓰자는 운영 원칙입니다.

실제 운영에서는

실제 운영에서는 2시간/5회 반복 같은 상한을 먼저 둡니다. 큰 작업만 나눠 맡기고, 파일 하나 수정은 한 담당이 끝냅니다.

이번 장 복붙 프롬프트

이 작업을 2시간/5회 반복 안에서 끝내려면 어떤 범위를 줄여야 하는지 제안해줘.

PART 2

한 프로젝트로 따라가는 운영 한 바퀴

시작 · 작업 맡기 · 반려 · 재개 · 완료 근거 · 종료.

07 · 구조

7장. 메인 ↔ 서브에이전트 구조 세팅

메인-서브 구조: 메인 → 브리프 → 작업자 → 검토 → 통합

브리프 카드: 목표 1줄 · 허용/금지 · 확인 방법 · 반환 조건

좋은 브리프: 자기완결 · 파일 범위 · 변경 분량 한도 · 완료 기준

이 가상 프로젝트에서는

지금부터 「가을 워크숍 안내문」 세 작업을 작업표에 올립니다. 이 장은 시작입니다. 맡기·반려·재개는 8장과 9장에서, 완료와 종료는 10장에서 다룹니다.

이 장에서 할 일은 세 가지뿐입니다.

1. 메인과 작업자에게 이름을 붙인다.
2. 작업표(`tasks.md` 또는 스프레드시트)를 하나 만든다.
3. 세 작업이 모두 대기 인지 확인한다.

이 장에서 만들 것

여기서 만드는 것은 회사용 자동화 시스템이 아닙니다. “메인 에이전트가 일감을 만들고, 서브에이전트가 하나씩 맡아 가는 구조”를 문서 한 장으로 흉내 내는 연습입니다.

처음부터 서버, 봇, 원격 접속, 여러 계정을 붙이면 배울 것이 너무 많아집니다. 처음에 중요한 것은 도구 종류가 아니라 운영 감각입니다. 누가 일을 만들고, 누가 맡고, 어디에 상태가 남고, 충돌이 나면 어떻게 막는지를 먼저 손으로 적어 봐야 합니다.

역할 카드	작업표에 쓰는 이름	이 프로젝트에서 하는 일	산출물
메인	main	세 작업을 작업표에 올리고 전체 상태를 본다	작업표, 다음 행동
작업자	sub	남은 작업 하나를 맡아 그 일만 처리한다	결과 한 줄
검토자	(사람 또는 다른 지시)	누락·중복·근거 부족을 찾는다	통과/반려

두 이름은 실제로 같은 AI 도구의 두 대화창이어도 되고, 한 대화창 안의 두 지시여도 됩니다. 이름을 따로 붙여 “두 에이전트가 같은 작업표를 보고 있다”는 감각을 익히는 것이 목적입니다. 나중에 실제 에이전트가 둘 이상이 되어도 원리는 같습니다.

작업표 만들기

준비물은 문서 편집기나 스프레드시트 하나입니다. 메모장의 `tasks.md` 파일도 좋고, 구글 시트 한 장도 좋습니다. 중요한 것은 **모든 역할이 같은 한 장을 본다**는 점입니다.

처음 작업표는 이렇게 시작합니다.

ID	상태	담당	시작 시각	기한	작업	결과·근거
T-001	대기	-	-	-	안내문 목차 초안	-
T-002	대기	-	-	-	일정·장소 사실 확인	-
T-003	대기	-	-	-	한 줄 완료 보고	-

1부에서 쓴 `todo`, `claimed`, `done` 은 이 표의 대기, 진행 중, 완료 와 같은 뜻입니다.

이 표가 이 작은 시스템의 **작업 원장**입니다. 데이터베이스도 아니고 서버도 아니지만, 지금 배우려는 핵심에는 충분합니다. 여러 에이전트가 같은 원장을 보고, 누가 무엇을 맡았는지 기록하는 것이 첫 번째 뼈대입니다.

여기서 봐야 할 것은 작업 이름보다 상태입니다. 지금은 세 작업 모두 대기입니다. 아직 아무도 맡지 않았다는 뜻입니다. 시작 시각과 기한 칸은 9장에서 쓰는 lease(임시 점유) 표시입니다. 지금은 비워 둡니다.

코드에 익숙하다면 같은 원장을 JSON 파일로 둘 수도 있습니다. 형식은 달라도 담은 정보는 같습니다.

```
{"id": "T-001", "status": "todo", "owner": null, "started_at": null, "result": null}
```

시작은 여기까지입니다. 다음 장에서 이 세 작업을 담당 이름과 함께 말합니다.

도구에서 역할을 나누는 방법

AI 코딩 도구는 대부분 “역할별 지시”를 따로 저장하는 기능이 있습니다. 예를 들어 Claude Code는 서브에이전트를 정의해 메인 대화와 별도 맥락에서 일을 맡길 수 있습니다. 메뉴 이름과 설정 방법은 버전에 따라 바뀌므로 공식 문서 [Create custom subagents](#)와 `claude --help`를 확인하세요.

도구 기능이 없어도 괜찮습니다. 대화창을 둘 열고 첫 메시지에 역할 카드를 붙여 넣으면 같은 효과를 낼 수 있습니다.

```
너는 sub 역할이다. 아래 역할 카드를 따른다.  
[역할 카드 B 내용 붙여 넣기]  
작업은 공유 작업표(tasks.md)에서만 받는다.  
작업표에 없는 일은 시작하지 말고 main에게 물어라.
```

왜 이름을 붙이나

혼자 쓰는 도구에서도 에이전트 이름은 중요합니다. 어떤 작업이 끝났을 때 “누가 했다”가 남아야 다음 판단을 할 수 있습니다. 이름이 없으면 완료 보고도, 충돌 조사도, 재시도 판단도 흐려집니다.

이 책에서는 `main`과 `sub`만 씁니다. 실제 환경에서는 이름이 더 많아질 수 있습니다. 하지만 원리는 같습니다. 이름은 멋을 내기 위한 별명이 아니라 작업 책임을 기록하기 위한 태그입니다.

여기서 가져갈 감각은 하나입니다.

에이전트는 말로만 나뉘는 것이 아니라, 작업 원장에 이름이 남을 때 운영 단위가

된다.

메인과 서브를 어떻게 상상하면 좋을까

메인은 사람과 가까운 역할입니다. 목표를 쪼개고, 작업표에 올리고, 전체 상태를 봅니다. 서브는 좁은 작업을 하나씩 가져가 처리합니다. 서브에게 전체 판단권을 주지 않습니다. 서브는 “작업표에서 하나를 맡고, 결과를 남긴다”에 집중합니다.

이 구조는 실제 팀과 비슷합니다. 팀장이 모든 일을 직접 하지 않고, 작업표를 만들고, 담당자가 하나씩 가져가고, 결과를 보고합니다. AI 팀도 다르지 않습니다. 다만 작업자가 사람이 아니라 에이전트일 뿐입니다.

이 장에서 만든 구조

지금 만든 구조는 아주 단순합니다.

- 작업표: 모든 역할이 함께 보는 작업 원장 한 장
- `main`, `sub`: 같은 작업표를 쓰는 두 에이전트 이름
- 역할 카드: 각 이름이 받는 입력과 내야 할 산출물
- 상태 확인: 메인이 작업표를 읽고 전체 상황을 보고하는 일

다음 장에서는 이 작업표에서 두 에이전트가 작업을 하나씩 맡아 가게 만듭니다. 아직 자동화라고 부르기엔 작지만, 운영의 핵심은 이미 들어 있습니다. 일감은 한곳에 있고, 상태는 기록되고, 담당자 이름이 남습니다.

실제 운영에서는

실제 운영에서는 메인과 서브의 역할을 텍스트로 먼저 고정합니다. 서브에게는 목표, 금지, 검증, 반환 조건만 넘깁니다.

이번 장 복붙 프롬프트

메인 에이전트가 서브 에이전트에게 넘길 자기완결 브리프를 작성해줘.

8장. 일감 큐 만들고 자동으로 집어가게 하기

큐 상태 전이: ready → claim → working → verify → done
 tasks.md 상태판: [] ready · [~] working · [?] blocked · [x] done
 큐 필드: task id · owner · status · evidence

이 가상 프로젝트에서는

이 장은 **작업 맡기(claim)**입니다. claim은 “이 일을 내가 맡겠다”는 서명이 원장에 남는 동작입니다. `main` 이 T-001 목차 초안을 맡으면, 그때부터 T-001의 작업지시는 `main` 의 것입니다. `sub` 는 T-002 사실 확인을 맡습니다. T-003 한 줄 보고는 아직 비어 있습니다.

기대 결과: 작업표에 담당이 서로 다른 두 작업이 **진행 중** 으로 남습니다. 다음 행동: 맡은 작업만 처리하고, 다른 ID는 건드리지 않습니다.

작업지시 카드

claim 전에 사람(또는 메인)이 작업자에게 주는 카드는 이렇게 씁니다. 실전에서는 이 네 줄을 그대로 붙입니다.

[작업지시 T-001]

역할: main

범위: 안내문 목차 초안만. T-002, T-003은 금지.

완료 조건: outline.md에 제목과 섹션 순서를 남긴다.

금지: 실제 발송, 공개 게시, 다른 작업 완료 처리.

[작업지시 T-002]

역할: sub

범위: 안내문의 날짜·장소·준비물을 주최 메모와 대조.

완료 조건: 확인표 3줄과 불일치 건수를 남긴다.

금지: T-001을 다시 맡거나 완료하지 말 것. 안내문 문장 수정 금지.

자동분배는 거창하지 않다

자동 분배의 핵심은 대단한 인공지능이 아닙니다. “아직 안 한 일은 무엇인가?”, “누가 맡았는가?”, “끝났는가?”를 한곳에 기록하는 것입니다. 이 장에서는 대기 → 진행 중 → 완료 흐름 중 앞의 두 단계, 즉 작업을 만들고 맡는 과정을 봅니다.

claim 규칙은 한 줄입니다. 위에서부터 읽어 내려가다가 대기인 첫 작업을 맡는다. 담당 칸이 이미 채워져 있으면 건너뛰다.

먼저 main 이 첫 번째 작업을 맡습니다. 담당 칸에 main , 시작 시각을 적고 상태를 진행 중 으로 바꿉니다. 이어서 sub 가 작업표를 읽습니다. T-001은 이미 담당이 있으니 건너뛰고, 다음으로 비어 있는 T-002를 맡습니다.

ID	상태	담당	시작 시각	기한	작업	결과·근거
T-001	진행 중	main	10:00	11:00	안내문 목차 초안	-
T-002	진행 중	sub	10:02	11:02	일정·장소 사실 확인	-
T-003	대기	-	-	-	한 줄 완료 보고	-

중요한 점은 sub 가 T-001을 또 맡지 않았다는 것입니다. 이미 main 이 맡은 작업은 건너뛰고, 다음으로 비어 있는 작업을 가져갔습니다. 이것이 작은 형태의 자동분배입니다.

표에서 봐야 할 것은 세 가지입니다.

- 상태: 작업이 대기 , 진행 중 , 완료 중 어디에 있는가
- 담당: 누가 맡았는가
- 요약: 전체에서 몇 개가 남았는가 (지금은 대기 1, 진행 중 2, 완료 0)

이 정도만 있어도 사람은 매번 “너 이거 해, 너는 저거 해”라고 말하지 않아도 됩니다. 에이전트는 작업표를 보고 남은 일을 맡으면 됩니다. 자동화는 여기서 시작합니다.

좋은 큐의 조건

처음에는 큐를 복잡하게 만들 필요가 없습니다. 오히려 단순해야 합니다.

조건	이유
작업 하나는 작아야 한다	오래 붙잡히지 않고 실패해도 다시 배정하기 쉽다
상태는 적어야 한다	대기 , 진행 중 , 완료 만으로도 처음에는 충분하다
담당자는 반드시 남아야 한다	충돌과 누락을 추적할 수 있다
완료 결과는 한 줄이라도 남아야 한다	다음 사람이 확인할 근거가 생긴다

작업이 너무 크면 에이전트가 오래 붙잡고 있게 됩니다. 그러면 다른 에이전트가 도와주기 어렵고, 실패했을 때 되돌리기도 어렵습니다. 작은 작업 여러 개가 큰 작업 하나보다 자동분배에 훨씬 잘 맞습니다.

예를 들어 “전자책 만들기”는 큐에 넣기에는 너무 큼니다. 대신 이렇게 쪼갭니다.

큰 작업	큐에 넣기 좋은 작은 작업
전자책 만들기	1장 초안 작성
전자책 만들기	1장 검토
전자책 만들기	예시 표 사실 확인
전자책 만들기	PDF 1~5쪽 렌더 확인

작업이 작으면 완료 기준도 분명해집니다. “1장 초안 작성”은 파일 하나로 확인할 수 있고, “예시 표 사실 확인”은 대조표로 확인할 수 있습니다.

claim은 말로 하는 배정이 아니다

중요한 차이가 있습니다. 사람이 `main`에게 “너 이거 해”라고 말하는 것은 배정입니다. 하지만 claim은 작업 원장에 기록되는 배정입니다. 이 차이가 운영 안정성을 만듭니다.

말로만 배정하면 다음 질문이 생깁니다.

- 방금 누가 말았지?
- 끝났나?
- 다른 에이전트가 같은 일을 하고 있나?
- 실패하면 누가 다시 잡아야 하지?

작업 원장에 claim이 남으면 이 질문에 답할 수 있습니다. 작업표를 보면 됩니다. 작은 프로젝트에서도 이 습관은 중요합니다. AI와 대화할 때 기억에 의존하지 않고 파일에 상태를 남기면, 오래된 맥락과 최신 상태가 덜 충돌합니다.

규칙을 에이전트 프롬프트로 바꾸기

이 규칙을 실제 AI 작업에 적용한다면, 서브 역할에게 이렇게 말할 수 있습니다.

```
너는 sub 역할이다.
공유 작업표 tasks.md를 위에서부터 읽고, 상태가 "대기"인 첫 작업 하나만 말아라.
말할 때 담당 칸에 sub, 시작 시각, 기한(시작 후 60분)을 적고 상태를 "진행 중"으로 바꿔라.
담당 칸이 이미 채워진 작업은 건너뛰어라.
말은 작업 하나만 처리하고, 끝나면 결과·근거 칸에 결과 한 줄과 확인 방법을 남겨라.
```

메인 역할에게는 이렇게 말합니다.

너는 main 역할이다.

tasks.md를 읽고 대기/진행 중/완료 개수, 기한이 지난 작업, 막힌 작업을 보고해라.
작업표의 내용은 직접 고치지 말고, 고칠 점은 다음 행동으로 제안만 해라.

이렇게 규칙과 역할을 함께 주면, AI가 “알아서 다 해보겠다”며 범위를 넓히는 일을 줄일 수 있습니다.

이번 장의 핵심

자동분배는 처음부터 복잡한 서버를 만드는 일이 아닙니다. 작업을 작게 쪼개고, 대기 상태로 두고, 에이전트가 맡으면서 담당 이름을 남기면 시작됩니다. 사람의 머릿속 배정표를 파일로 옮기는 것이 첫 단계입니다. 다음 장에서는 이렇게 맡은 작업을 어떻게 안전하게 완료하고, 중복 처리를 막을지 봅니다.

실제 운영에서는

실제 운영에서는 작업표가 단순 목록이 아니라 큐입니다. 대기, 진행 중, 막힘, 완료를 구분하면 여러 에이전트가 같은 기준으로 움직입니다.

이번 장 복붙 프롬프트

```
tasks.md 큐를 ready/working/blocked/done 상태로 운영하는 규칙을 만들어줘.
```

09 · 리스

9장. 충돌·중복 방지 최소 장치 붙이기

lease/idempotency: 맡기 → 진행 표시 → 기한 초과 → 재배정 → 완료

점유 점검 예시: 점유 표시 있음 · 담당 일치 · 기한 남음 · 보고 후 해제

중복 방지: 같은 결과는 한 번만 · 상태는 한 번에 변경 · 재시도해도 안전 · 근거 먼저

이 가상 프로젝트에서는

이 장은 완료 근거, 반려, 재개입니다.

- **main** 이 T-001을 “목차 6개 섹션 초안, outline.md”로 닫으면 목차 초안의 완료 근거가 작업표에 남습니다.
- **sub** 가 같은 T-001을 다른 결과로 닫으려 하면 검토자가 **반려**합니다.
- **sub** 가 T-002를 멈췄다가 이어 가려면 **재개**합니다. 반납(release) 뒤에 같은 담당이 다시 맡습니다.

검토자 역할은 작업표와 결과를 보고 “통과/반려”만 판정합니다. 결과를 덮어쓰지 않습니다.

왜 최소 장치가 필요한가

여러 에이전트가 같은 작업표를 보면 속도는 빨라질 수 있습니다. 동시에 위험도 생깁니다. 둘이 같은 일을 맡거나, 이미 끝난 일을 다시 끝내려 하거나, 다른 에이전트가 맡은 일을 대신 완료 처리할 수 있습니다.

그래서 최소한 두 장치가 필요합니다.

- lease: “이 작업은 지금 누가, 언제까지 잡고 있다”는 임시 점유 표시
- idempotency(멱등성): 같은 완료 요청이 반복되어도 결과가 망가지지 않게 하는 성질

말은 어렵지만 실제 동작은 단순합니다. lease는 작업표의 **진행 중** 상태와 담당, 시작 시각, 기한 칸입니다. idempotency는 “같은 요청이 두 번 와도 한 번 처리한 것과 같은 결과가 되어야 한다”는 뜻입니다.

자동화에서는 같은 일이 두 번 처리되는 경우가 생각보다 자주 생깁니다. 사용자가 같은 버튼을 다시 누를 수도 있고, 네트워크가 느려서 재시도할 수도 있고, 에이전트가 완료 보고를 다시 보낼 수도 있습니다. 이때 결과가 두 번 추가되거나, 이미 끝난 작업이 다른 결과로 덮이면 안 됩니다.

완료 처리의 세 가지 경우

main 이 T-001을 끝내면 작업표가 이렇게 바뀝니다.

ID	상태	담당	작업	결과·근거
T-001	완료	main	안내문 목차 초안	목차 6개 섹션 초안, outline.md
T-002	진행 중	sub	일정·장소 사실 확인	-
T-003	대기	-	한 줄 완료 보고	-

그다음 완료 요청이 또 들어오면 세 경우로 나눠 처리합니다.

들어온 요청	판정	작업표 처리
main 이 같은 결과로 T-001 완료를 다시 보고	이미 완료	아무것도 바꾸지 않는다. 행을 추가하지 않는다
sub 가 다른 결과로 T-001 완료를 보고	충돌 → 반려	기존 결과를 유지하고 반려 기록을 남긴다
sub 가 main 이 진행 중인 작업을 완료 처리	충돌 → 반려	담당을 바꾸지 않는다

첫 번째가 idempotency입니다. 이미 끝난 작업을 같은 담당이 같은 결과로 다시 완료하려 하면 “이미 끝났다”고 보고 상태를 유지합니다. 같은 요청이 두 번 들어와도 결과가 망가지지 않습니다.

두 번째와 세 번째는 막아야 합니다. 이 실패는 좋은 실패입니다. 안내문 목차는 이미 main 이 닫았으므로 sub 가 다른 결과로 덮을 수 없습니다. 조용히 덮어쓰는 것보다 훨씬 낫습니다. 자동화에서 가장 위험한 것은 실패가 아니라, 실패했는데 성공한 척 지나가는 것입니다.

반려 기록은 이렇게 남기면 됩니다.

```
[반려]
작업 ID: T-001
요청자: sub
이유: 이미 main이 "목차 6개 섹션 초안"으로 완료함
다음 행동: T-001을 다시 열지 않는다. T-002만 이어서 처리한다.
```

lease가 막는 것

lease는 “지금 누가, 언제까지 잡고 있는가”를 남깁니다. 작업을 맡으면 진행 중 상태가 되고 담당 이름과 기한이 적힙니다. 기한이 남아 있는 동안 다른 에이전트는 그 작업을 맡지 않습니다.

lease에 기한을 두는 이유는 에이전트가 작업을 잡은 채 멈춰 버려도 영원히 잠기면 안 되기 때문입니다. 이 책의 예시에서는 맡은 뒤 60분을 기한으로 둡니다. 기한이 지나면 메인이 상황을 확인한 뒤 다시 배정할 수 있습니다. 이때 두 가지를 지킵니다.

- 기한이 지난 작업은 기존 담당이라도 바로 완료 처리하지 않는다. 먼저 작업표를 보고 다시 맡은 뒤, 결과를 검토해 완료한다.
- 이미 다른 담당이 다시 맡았다면 그 담당의 결과와 조율한다. 두 결과를 조용히 합치지 않는다.

기다리지 않고 당장 되돌리려면 잡은 에이전트가 직접 반납합니다. `sub`가 T-002를 확인하다가 주최 메모를 기다려야 해서 멈췄다고 해 봅시다. `sub`는 상태를 대기 로 되돌리고 담당 칸을 비운 뒤, 메모가 오면 같은 `sub`가 다시 말합니다. 이것이 **재개**입니다.

순서	작업표 변화
1. 반납	T-002: 진행 중 → 대기, 담당 sub → 비움, 메모 칸에 “주최 메모 대기”
2. 다시 말기	T-002: 대기 → 진행 중, 담당 sub, 새 시작 시각과 기한

다른 에이전트가 남의 작업을 대신 반납하려 하면 이것도 충돌로 봅니다. 멈춘 작업을 조용히 가로채지 말고, 담당이 반납하거나 기한이 지난 뒤에만 다시 배정하세요.

재개 기록은 이렇게 남깁니다.

```
[재개]
작업 ID: T-002
이전 담당: sub
동작: 반납 후 같은 sub가 다시 말음
이유: 주최 메모를 받은 뒤 사실 확인을 이어서 하기 위함
금지: main이 T-002를 가로채지 않음
```

중복 실행을 두려워하지 않는 구조

자동화는 종종 같은 일을 두 번 시도합니다. 완전히 없애기 어렵습니다. 그래서 “절대 두 번 실행되지 않게 하겠다”보다 “두 번 실행되어도 안전하게 만들겠다”가 더 현실적입니다.

예를 들어 아래 세 상황을 비교해 보세요.

상황	나쁜 시스템	좋은 시스템
같은 완료 요청이 두 번 들어옴	결과가 두 번 쌓인다	이미 완료로 처리
다른 작업자가 완료하려 함	조용히 담당이 바뀐다	충돌로 반려
잡힌 작업을 다시 말으려 함	두 작업자가 같은 일을 한다	중복 말기 차단

좋은 시스템은 실패를 숨기지 않습니다. 실패를 사람이 볼 수 있는 기록으로 드러냅니다. 이 기록이 있어야 메인 역할이 “이 작업은 누가 잡고 있고, 왜 막혔는지” 판단할 수 있습니다.

작업표를 스크립트나 도구로 자동화할 때도 같은 규칙을 그대로 옮기면 됩니다. 같은 컴퓨터에서 여러 에이전트가 동시에 한 파일을 고친다면, 파일을 읽고 바꾸고 저장하는

동안 다른 쪽이 끼어들지 못하게 잠금을 거는 것이 기본입니다.

내 실전 작업에 붙이는 최소 규칙

코드가 없어도 이 규칙은 쓸 수 있습니다.

1. 작업을 시작하면 담당과 기한을 적는다.
2. 담당이 있는 작업은 다른 사람이 시작하지 않는다.
3. 완료할 때 결과를 함께 적는다.
4. 같은 결과로 다시 완료하면 “이미 완료”로 본다.
5. 다른 결과로 다시 완료하려 하면 충돌로 본다.
6. 작업자가 멈추면 반납하거나 기한이 지난 뒤에만 다시 맡는다.

이 여섯 줄을 작업표 위에 붙여 두세요. 작은 팀, 혼자 쓰는 AI, 여러 브라우저 창, 여러 에이전트 모두에 적용됩니다.

실패 복구 체크리스트

- [] 담당이 있는 작업을 다른 역할이 다시 맡지 않았다.
- [] 다른 담당의 완료 시도는 반려 기록으로 남겼다.
- [] 같은 완료 보고가 반복되어도 행이 늘지 않았다.
- [] 멈춘 작업은 반납 후 대기 가 되거나, 기한이 지난 뒤 다시 배정됐다.
- [] 원인을 추측해 상태를 덮어쓰지 않고, 작업표와 반려·재개 기록을 같이 본다.

이번 장의 핵심

멀티에이전트 시스템은 실패하지 않는 시스템이 아닙니다. 실패를 조용히 덮어쓰지 않는 시스템입니다. lease는 누가 언제까지 맡았는지 남기고, idempotency는 같은 요청이 반복되어도 안전하게 만듭니다. 충돌은 성공처럼 처리하지 말고, 눈에 보이게 반려해야 합니다. 이 작은 장치가 병렬 작업의 바닥 안전망입니다.

실제 운영에서는

실제 운영에서는 담당 표시를 영원히 믿지 않습니다. 작업자가 끊길 수 있으므로 소유권, 재시도, 중복 방지를 같이 설계합니다.

이번 장 복불 프롬프트

이 자동화에 `lease`, `timeout`, `idempotency`, `retry-safe` 조건을 붙여줘.

10 · 검증

10장. 결과 수합·검증·보고까지 한 바퀴 돌리기

수합·검증·보고: 수합 → 확인 → 개인정보 점검 → 화면 확인 → 보고

검증 요약 예시: 확인표 3줄 · 불일치 0건 · 개인정보 0건 · 화면 확인 완료

보고 근거: 확인 방법 · 결과 · 파일 · 남은 위험

이 가상 프로젝트에서는

이 장은 **완료 근거와 종료**입니다. 남은 T-002와 T-003을 결과와 근거를 붙여 달고, 검토자가 근거를 확인한 뒤, 사람이 읽을 보고 한 줄을 남깁니다. 세 칸이 모두 **완료** 이면 「가을 워크숍 안내문」 학습 루프가 끝난 것입니다.

자동분배는 보고까지 가야 끝난다

자동분배는 작업을 나눠 주는 것만으로 끝나지 않습니다. 마지막에는 결과를 모으고, 검증하고, 사람에게 보고해야 합니다. 이 과정이 없으면 에이전트가 바쁘게 움직였다는 사실만 남고, 실제로 믿어도 되는 결과인지 알 수 없습니다.

AI 운영에서 자주 생기는 착각이 있습니다. “에이전트가 완료라고 말했으니 완료”라고 믿는 것입니다. 하지만 완료 보고는 증거가 아닙니다. 결과 파일, 작업표, 확인 기록이 있어야 합니다.

검토자가 보는 것

검토자는 결과를 다시 쓰지 않습니다. 작업지시의 완료 조건과 결과·근거 칸을 나란히 놓고 대조만 합니다.

작업	완료 조건	검토자가 확인할 근거
T-001 목차 초안	제목과 섹션 순서가 정해짐	<code>outline.md</code> 가 있고 섹션이 6개인지, 다른 담당의 덮어쓰기가 반영됐는지

작업	완료 조건	검토자가 확인할 근거
T-002 사실 확인	날짜·장소·준비물이 주최 메모와 일치	확인표 3줄이 있고 불일치 건수가 적혀 있는지
T-003 한 줄 보고	작업표와 확인 결과가 요약됨	보고 숫자가 작업표와 같은지, 남은 위험이 적었는지

근거가 하나라도 비어 있으면 검토자는 통과 대신 반려를 적고, 다시 열어야 할 작업 ID를 남깁니다.

사람이 읽는 보고 만들기

에이전트에게는 작업표가, 사람에게는 짧은 보고가 필요합니다. 보고는 작지만 기본 요소를 갖고 있어야 합니다.

- 어떤 작업이 남았는가
- 어떤 작업을 누가 맡았는가
- 어떤 작업이 끝났고, 무엇으로 확인했는가

실전에서는 이 보고가 더 길어질 수 있습니다. 실패한 작업, 재시도할 작업, 사람 승인이 필요한 작업이 붙을 수 있습니다. 하지만 처음부터 복잡하게 만들 필요는 없습니다. “작업표 요약 + 확인 결과 + 다음 행동”이면 충분합니다.

T-001만 끝난 시점의 중간 보고는 이렇게 쓰면 됩니다.

현재 상태:

- 대기: 1개
- 진행 중: 1개
- 완료: 1개

확인:

- T-001 outline.md 섹션 6개 확인
- T-001에 대한 sub의 다른 결과 완료 시도는 반려
- T-002는 반납 후 재개, 진행 중

다음 행동:

- sub가 T-002 확인표를 채워 완료
- 남은 T-003을 sub가 맡아 한 줄 보고 작성

이런 보고가 있어야 사람이 폰으로도 감독할 수 있습니다. 긴 대화 기록 전체를 읽지 않아도 핵심 상태를 볼 수 있기 때문입니다.

남은 작업 닫기

sub가 T-002 확인표를 채우고, 남은 T-003을 맡아 보고를 쓰면 작업표가 이렇게 끝납니다.

ID	상태	담당	작업	결과·근거
T-001	완료	main	안내문 목차 초안	목차 6개 섹션 초안, outline.md
T-002	완료	sub	일정·장소 사실 확인	확인표 3줄, 불일치 0건 (재개 후 완료)
T-003	완료	sub	한 줄 완료 보고	“세 작업 완료, 발송은 승인 대기”

요약: 대기 0, 진행 중 0, 완료 3.

이것이 이 가상 프로젝트의 **완료 근거 묶음**입니다.

근거	어디에 있나	통과 기준
T-001 결과	outline.md	main이 완료했고 다른 담당의 덮어쓰기가 반려됨
T-002 결과	확인표 3줄	sub가 재개 후 완료, 불일치 0건
T-003 결과	보고 1줄	숫자가 작업표와 일치
검토 기록	반려·재개 메모	충돌이 조용히 덮이지 않음
사람용 보고	위 작업표 요약	대기 0, 진행 중 0, 완료 3

학습용 가상 사례이므로 안내문을 실제로 발송하지 않습니다. 발송이 필요해 보이는 다음 작업이 생기면 템플릿 03의 승인 게이트로 멈춥니다.

종료와 보관

프로젝트를 **종료**한 뒤에는 작업표를 지우지 말고 날짜를 붙여 보관합니다. 다음 프로젝트는 새 작업표로 시작합니다. 끝난 표에는 누가 무엇을 맡았고, 무엇이 반려됐고, 무엇으로 끝났는지가 남아 있어서 다음에 비슷한 일을 할 때 그대로 참고할 수 있습니다.

이 장에서 정리한 한 바퀴

지금까지의 흐름을 한 번에 정리하면 이렇습니다.

1. 작업표를 만든다.
2. 에이전트 이름을 붙여 작업을 맡는다.

3. 이미 맡은 작업은 다른 에이전트가 다시 맡지 못하게 한다.
4. 완료 요청은 결과와 함께 기록한다.
5. 같은 완료 요청이 반복되어도 안전하게 처리한다.
6. 검토자가 완료 조건과 근거를 대조한다.
7. 사람이 읽을 수 있는 한 줄 보고로 요약한다.

이 작은 흐름이 실제 AI 팀 운영의 축소판입니다. 큰 시스템은 여기에 원격 실행, 승인 게이트, 알림, 비용 제한, 배포 규칙이 붙습니다. 그러나 뼈대는 같습니다. 일감을 작게 만들고, 누가 맡았는지 기록하고, 끝났다는 근거를 남기고, 충돌은 조용히 덮어쓰지 않습니다.

“잘 됐다”는 말은 검증 뒤에만 한다

AI 에이전트와 일할 때 가장 중요한 습관 중 하나는 완료 선언을 늦추는 것입니다. 코드를 썼다고 끝난 것이 아닙니다. 문서를 만들었다고 끝난 것도 아닙니다. 확인이 끝나고, 결과를 사람이 볼 수 있는 형태로 남긴 뒤에야 완료라고 말할 수 있습니다.

작업 종류마다 “무엇을 확인해야 끝인가”를 미리 적어 두면 에이전트의 완료 보고를 그대로 믿지 않아도 됩니다.

작업 종류	확인 방법 예시
글 초안	파일이 있고, 요구한 섹션·분량이 들어 있는지
사실 확인	원본 자료와 대조한 표, 불일치 건수
코드 수정	테스트 실행 결과, 바뀐 파일 목록
화면·문서 결과물	실제로 열어 본 페이지, 스크린샷
보고	숫자가 작업표와 같은지

처음 운영할 때 지킬 기준

처음부터 완벽한 자동화 시스템을 만들 필요는 없습니다. 대신 아래 기준은 지키는 편이 좋습니다.

- 에이전트가 할 일을 한 줄 작업으로 쪼갬다.
- 작업 상태를 한곳에 기록한다.
- 담당자 이름을 반드시 남긴다.
- 완료 결과를 짧게라도 남긴다.

- 충돌은 실패로 드러나게 한다.
- “잘 됐다”는 말은 확인 뒤에만 한다.

이 기준이 있으면 작업표 한 장도 운영 연습이 됩니다. 다음 단계로 확장하더라도 같은 원칙을 그대로 가져갈 수 있습니다.

이번 강의 핵심

멀티에이전트 운영은 “일을 나눴다”에서 끝나지 않습니다. 결과를 모으고, 검증하고, 보고해야 끝납니다. 결과 파일과 확인 기록은 검토를 위한 근거이고, 작업표 요약은 사람을 위한 근거입니다. 둘 다 있어야 다음 행동을 안전하게 정할 수 있습니다.

실제 운영에서는

실제 운영에서는 '대충 봤다'를 완료 근거로 쓰지 않습니다. 확인 결과, 화면 캡처, 완성 페이지 점검처럼 다시 확인 가능한 증거를 남깁니다.

이번 장 복붙 프롬프트

산출물을 검증하고 보고하기 위한 확인 방법, 기대 결과, 근거 파일을 정리해줘.

더 나아가기

한 PC의 큐가 닫힌 뒤에만 기기를 나눕니다.

11 · 확장

11장. 2대·구독 2개로 확장하면 뭐가 달라지나

2기기 확장도: 오케스트레이터 → 실행자 A → 실행자 B → 검토 → 사람 게이트

멀티 역할 보드: 실행자 A 작업 중 · 실행자 B 대기 · 노트북 검토 · 폰 승인

확장 전 조건: 큐 · 락 · 검증 · 보고

이 가상 프로젝트에서는

「가을 워크숍 안내문」 세 작업이 한 PC의 큐·역할·완료 근거로 닫히기 전에는 기기를 늘리지 않습니다. 두 대가 되면 목차 파일 버전, 상태 파일 위치, 승인 위치가 갈라집니다. 이 장은 그 분리를 미리 그리는 청사진이지, 특정 장비를 복제하라는 지시가 아닙니다.

기대 결과: 확장 전에 큐 하나, 담당, 검증, 사람 게이트를 한 문장으로 말할 수 있는지 확인합니다. 다음 행동: 2부의 작업표가 완료 3, 빈 근거 칸 0으로 닫힌 뒤에만 확장 표를 채웁니다.

2부의 운영 한 바퀴는 일부러 컴퓨터 한 대와 에이전트 계정 하나를 기준으로 만들었습니다. 처음부터 여러 대의 컴퓨터, 여러 구독, 별도 알림 채널, 원격 접속을 붙이면 시스템을 이해하기 전에 운영 부담이 먼저 커집니다.

하지만 한 PC 안에서 메인 → 서브에이전트 → 작업 큐 → 검증 → 보고 흐름이 익숙해지면 다음 질문이 생깁니다.

“이 구조를 두 대의 컴퓨터로 나누면 무엇이 좋아지고, 무엇이 위험해질까?”

이 장의 목표는 특정 장비 구성이나 운영 방식을 복제하는 것이 아닙니다. 독자가 자기 환경에서 “확장해도 되는 시점”과 “확장할 때 반드시 분리해야 할 것”을 판단하도록 돕는 것입니다.

확장은 속도보다 분리다

컴퓨터가 두 대가 되면 가장 먼저 떠오르는 장점은 속도입니다. 한쪽이 긴 작업을 하는 동안 다른 쪽이 검토나 작은 수정을 맡을 수 있습니다. 구독이 두 개라면 사용량 제한에 걸렸을 때도 다른 계정으로 급한 일을 이어갈 수 있습니다.

하지만 실전에서 더 중요한 변화는 속도가 아니라 분리입니다.

한 PC에서 두 에이전트를 돌릴 때는 모든 것이 한 공간에 있습니다. 같은 파일을 보고, 같은 터미널을 쓰고, 같은 작업 큐를 읽습니다. 문제가 생기면 화면 하나만 보면 됩니다. 반면 두 대가 되면 작업 위치, 파일 버전, 승인 권한, 비용, 알림이 갈라집니다.

그래서 확장은 “더 많은 에이전트를 켜다”보다 “역할과 권한을 나눈다”에 가깝습니다.

좋은 확장 구조라면 아래 질문에 바로 답할 수 있어야 합니다.

- 어느 기기가 전체 작업 큐를 관리하는가
- 어느 기기가 실행만 담당하는가
- 어느 행동은 자동으로 해도 되는가
- 어느 행동은 사람 승인 전에는 멈춰야 하는가
- 작업이 끝났다는 증거는 어디에 남기는가
- 비용이나 오류가 커지면 누가 먼저 멈추는가

이 질문에 답하지 못한 채 기기만 늘리면, 한 PC에서 작게 나던 문제가 두 배로 빨라집니다.

두 기기 구조의 기본 그림

가장 단순한 확장 모델은 “오케스트레이터 1대 + 실행자 1대”입니다.

오케스트레이터는 지시하고, 나누고, 승인하고, 결과를 모읍니다. 실행자는 받은 일을 처리하고, 결과와 근거를 돌려줍니다. 두 기기가 모두 마음대로 큐에서 일을 꺼내고 서로에게 다시 지시하기 시작하면 구조가 빠르게 꼬입니다.

처음에는 아래처럼 나누는 것이 좋습니다.

역할	하는 일	하지 않는 일
오케스트레이터	작업 큐 관리, 우선순위 결정, 승인 게이트 판단, 최종 보고	작은 작업을 모두 직접 붙잡기
실행자	지정된 작업 수행, 로컬 검증, 결과 보고	새 작업을 마음대로 만들기, 외부 발송·배포·삭제 실행

여기서 중요한 원칙은 “실행자는 스스로 시작하지 않는다”입니다. 실행자는 큐에 있는 작업을 집거나, 오케스트레이터에게 받은 작업을 처리합니다. 자기 판단으로 새로운 큰 일을 벌이지 않습니다.

초기 구조는 느려 보여도 안정적입니다. 안정적인 구조 위에서만 나중에 자동 분배와 예외 처리를 늘릴 수 있습니다.

계정이 두 개가 되면 비용도 두 개가 된다

구독이 두 개가 되면 작업량을 나눌 수 있습니다. 예를 들어 한 계정은 긴 초안 작성에, 다른 계정은 검토와 확인에 쓸 수 있습니다. 사용량 제한이 한쪽에 몰리는 일도 줄어듭니다.

하지만 계정이 두 개라는 말은 비용 관리 지점도 두 개라는 뜻입니다.

한 계정만 쓸 때는 “오늘 너무 많이 썼나?”를 한 화면에서 확인하면 됩니다. 두 계정부터는 다음 규칙이 필요합니다.

- 계정별 하루 예산을 정합니다.
- 긴 작업은 시작 전에 예상 사용량과 시간을 적습니다.
- 반복 작업에는 최대 반복 횟수를 둡니다.
- 비용이 큰 작업은 자동으로 재시도하지 않습니다.
- 사용량 제한이 걸렸을 때 다른 계정으로 무조건 넘기지 않습니다.

특히 마지막 규칙이 중요합니다. 한쪽 계정이 막혔을 때 다른 계정으로 넘기면 편리하지만, 그 순간 비용 브레이크가 사라질 수 있습니다. 사용량 제한은 귀찮은 장애물이 아니라 “잠깐 멈추고 판단하라”는 신호로 다루는 편이 안전합니다.

작업 큐는 하나만 두는 것이 쉽다

두 기기에서 작업 목록을 따로 관리하면 처음에는 편해 보입니다. 하지만 곧 문제가 생깁니다.

한쪽에서는 완료로 보이는 일이 다른 쪽에서는 대기 중으로 남습니다. 한 기기가 이미 처리한 작업을 다른 기기가 다시 시작합니다. 오래 멈춘 작업인지, 아직 처리 중인 작업인지 판단하기 어렵습니다.

그래서 초반에는 작업 큐를 하나만 두는 것이 좋습니다.

큐는 복잡할 필요가 없습니다. 최소한 아래 항목만 있으면 됩니다.

항목	의미
작업 ID	사람이 읽을 수 있는 고유 번호
상태	대기, 진행 중, 검토 필요, 완료, 보류
담당	오케스트레이터, 실행자 A, 실행자 B 같은 일반 역할명
범위	건드려도 되는 파일·문서·기능
완료 근거	확인 결과, 생성 파일, 검토 메모
다음 행동	승인 요청, 재시도, 병합, 종료

두 기기가 같은 큐를 보더라도 쓰기 권한은 제한하는 것이 좋습니다. 예를 들어 실행자는 자기 작업의 상태와 결과만 바꾸고, 우선순위와 승인 상태는 오케스트레이터만 바꿉니다.

이렇게 하면 책임 경계가 선명해집니다.

파일 충돌은 역할보다 범위로 막는다

두 기기가 같은 프로젝트를 만질 때 가장 흔한 사고는 파일 충돌입니다. 한쪽이 문서를 고치는 동안 다른 쪽이 같은 문단을 다시 쓰거나, 한쪽이 설정을 바꾸는 동안 다른 쪽이 이전 상태를 기준으로 작업합니다.

“서로 조심하자”는 규칙으로는 부족합니다. 에이전트는 선의로도 충돌을 만듭니다.

충돌을 줄려면 작업마다 범위를 적어야 합니다.

나쁜 지시:

안내문 품질을 전체적으로 개선해줘.

좋은 지시:

작업 ID: B-014
범위: outline.md의 3번 섹션만 수정
하지 말 것: 다른 섹션, 확인표, 보고 문구 수정 금지
목표: 처음 오는 참가자가 이해할 수 있게 오시는 길 설명을 보강
검증: 수정 후 실제 연락처와 계정명이 있는지 확인

범위를 파일 단위로 정하는 것만으로 부족할 때도 있습니다. 같은 파일 안에서도 “표만 수정”, “예시 문장만 교체”, “일정 칸만 검토”처럼 더 좁힐 수 있습니다.

확장할수록 지시는 자세해져야 합니다. 사람이 귀찮아지는 것이 아니라, 충돌 복구 비용을 미리 줄이는 것입니다.

승인 게이트는 중앙에 둔다

두 기기 구조에서 가장 위험한 실수는 승인 권한까지 나눠 버리는 것입니다.

초안 작성, 로컬 확인, 문서 정리, 내부 보고는 실행자가 스스로 해도 됩니다. 하지만 외부로 나가는 행동은 한 곳에서 막아야 합니다.

사람 승인이 필요한 대표 행동은 다음과 같습니다.

- 고객, 독자, 동료에게 메시지 발송
- 결제, 구매, 구독 변경
- 공개 저장소나 배포 환경에 반영
- 파일 삭제, 데이터 초기화, 계정 설정 변경
- 개인정보, 비밀값, 고객 자료가 포함된 작업
- 반복 비용을 만들 수 있는 자동화

이런 행동은 실행자가 “진행해도 될까요?”라고 묻는 단계에서 멈춰야 합니다. 오케스트레이터도 바로 실행하지 말고 사람에게 판단 정보를 올립니다.

승인 요청에는 감상이 아니라 사실이 들어가야 합니다.

[승인 요청]

작업 ID:

하려는 행동:

외부 영향:

되돌릴 수 있는가:

비용 발생 여부:

개인정보·비밀값 포함 여부:

검증한 내용:

승인 후 실행할 정확한 한 줄:

이 양식은 귀찮아 보여도 사고를 줄입니다. “대충 괜찮겠지”가 들어갈 자리를 없애기 때문입니다.

알림은 많을수록 좋은 것이 아니다

두 기기로 확장하면 알림도 늘어납니다. 작업 시작, 작업 완료, 실패, 승인 요청, 비용 경고, 충돌 보고가 모두 알림이 될 수 있습니다.

처음에는 모든 것을 알려주는 편이 안심됩니다. 하지만 알림이 너무 많으면 사람은 중요한 승인 요청을 놓칩니다. 그래서 알림에는 등급이 필요합니다.

등급	예시	알림 방식
낮음	초안 작성 완료, 로컬 정리 완료	큐에 기록만 남김
보통	확인 실패, 검토 필요	작업 보고에 포함
높음	승인 필요, 비용 초과 예상, 비밀값 의심	즉시 사람에게 알림
긴급	외부 발송 오작동, 데이터 삭제 의심	전체 자동화 멈춤 후 보고

핵심은 “사람이 바로 봐야 하는 것”만 즉시 알리는 것입니다. 나머지는 큐와 결과표에 쌓아도 충분합니다.

장애 대응 규칙을 먼저 만든다

확장 전에 반드시 정해야 할 것은 정상 흐름보다 장애 흐름입니다.

정상 흐름은 대체로 잘 굴러갑니다. 문제는 중간에 멈췄을 때입니다. 실행자가 응답하지 않거나, 같은 작업을 두 번 처리했거나, 승인 없이 외부 행동을 했을 가능성이 보이면 어떻게 할까요?

최소 장애 규칙은 다음 네 가지입니다.

1. 의심되면 새 작업 배정을 멈춥니다.
2. 현재 진행 중인 작업 ID와 담당을 확인합니다.
3. 마지막으로 실제 변경된 파일·발송·배포·비용을 확인합니다.
4. 복구 후 큐에 원인과 재발 방지 메모를 남깁니다.

여기서 “의심되면 멈춘다”가 가장 중요합니다. 자동화 시스템에서 가장 비싼 사고는 멈춰야 할 때 계속 진행하는 것입니다.

언제 확장해도 되는가

두 번째 기기나 두 번째 구독은 문제를 해결해 주지만, 동시에 새 문제를 만듭니다. 따라서 아래 조건을 만족하기 전에는 확장을 늦춰도 됩니다.

- 한 PC에서 작업 큐를 꾸준히 쓰고 있다
- 작업 완료 근거를 남기는 습관이 있다
- 승인 게이트 기준이 문서로 정리되어 있다

- 에이전트가 건드려도 되는 범위를 지시문에 적고 있다
- 비용이 큰 작업을 시작하기 전에 멈춰서 판단한다
- 사고가 났을 때 복구 순서를 기록할 수 있다

반대로 이런 상태라면 아직 확장하지 않는 편이 낫습니다.

- 작업을 대부분 즉흥 채팅으로만 시킨다
- 완료 표시를 자주 빼먹는다
- 외부 발송과 로컬 초안을 같은 수준으로 다룬다
- 실패하면 원인을 확인하기보다 새 에이전트에게 다시 시킨다
- 어떤 계정이 어떤 비용을 쓰는지 모른다

기기를 늘리는 것은 운영 습관을 확대하는 일입니다. 좋은 습관도 빨라지고, 나쁜 습관도 빨라집니다.

2대 확장 미니 청사진

처음 확장한다면 아래 순서로 충분합니다.

1. 한 기기를 오케스트레이터로 정합니다.
2. 다른 기기를 실행자로 정합니다.
3. 작업 큐는 하나만 둡니다.
4. 실행자가 수정할 수 있는 범위를 작업마다 적습니다.
5. 외부 영향이 있는 작업은 무조건 승인 요청으로 올립니다.
6. 비용이 큰 작업은 시작 전에 예상 시간과 반복 횟수를 적습니다.
7. 완료 후에는 결과보다 먼저 “무엇을 바꿨는지”를 남깁니다.
8. 하루가 끝나면 완료, 보류, 실패, 승인 대기만 정리합니다.

이 정도만 해도 “여러 에이전트가 마구 움직이는 상태”가 아니라 “작은 팀처럼 역할이 나뉜 상태”가 됩니다.

예시: 두 기기에서 안내문 작업 나누기

「가을 워크숍 안내문」을 두 기기로 나눈다고 해 봅시다.

오케스트레이터는 전체 목차와 작업 큐를 관리합니다. 실행자는 특정 섹션 하나만 맡습니다. 실행자는 다른 섹션을 읽을 수는 있지만 수정하지 않습니다. 초안 작성과 문장 정리는 스스로 합니다. 하지만 공개 게시, 웹사이트 반영, 참가자에게 보내는 메시지는 반드시 승인 요청으로 올립니다.

작업 지시는 이렇게 쓸 수 있습니다.

작업 ID: E-021

담당: 실행자 A

범위: outline.md의 5번 섹션(준비물·문의)

목표: 처음 오는 참가자가 이해할 수 있는 준비물 안내 작성

금지: 실제 기기명, 계정명, 메신저 주소, 토큰, 개인 실명, 전화번호 사용 금지

검증: 초안 작성 후 실제 식별자가 남아 있는지 점검한 결과를 보고

완료 근거: 수정한 섹션, 주요 변경 요약, 개인정보·식별자 점검 결과

이 지시에는 무엇을 할지뿐 아니라 무엇을 하지 말아야 할지도 들어 있습니다. 여러 기기로 확장할수록 “하지 말 것”이 중요해집니다.

마무리: 작은 운영 원칙이 큰 구조를 버틴다

2대·구독 2개 구조는 멋진 자동화의 시작처럼 보이지만, 실제로는 운영 원칙의 시험대에 가깝습니다. 큐, 담당, 범위, 승인, 비용, 사고 대응이 한 PC에서 이미 익숙해야 두 기기에서도 버팁니다.

확장의 핵심은 에이전트 수가 아닙니다. 누가 판단하고, 누가 실행하고, 어디서 멈추고, 무엇을 근거로 끝났다고 말할지 정하는 것입니다.

한 PC에서 만든 작은 규칙을 버리지 말고 그대로 키우세요. 그러면 두 번째 기기와 두 번째 구독은 혼란을 늘리는 장치가 아니라, 혼자 일하는 사람에게 작은 팀의 리듬을 만들어 주는 도구가 됩니다.

확장 실패 복구 체크리스트

아래는 예시 점검표입니다. 통과한다고 수익이나 처리량이 늘어난다고 단정하지 마세요.

- [] 작업 큐가 두 벌로 갈라지지 않았는지 확인합니다.
- [] 실행자가 승인 없이 발송·결제·배포·삭제를 하지 않았는지 확인합니다.
- [] 같은 파일을 두 기기가 동시에 고치지 않았는지 범위부터 봅니다.
- [] 사용량 제한이 걸린 계정을 다른 계정으로 자동으로 넘기지 않습니다.
- [] 의심되면 새 배정을 멈추고, 마지막 실제 변경만 적습니다.
- [] Claude Code·Codex 화면 이름이 책과 다르면 공식 문서와 `--help` 를 따릅니다.
2026년 10월 확인 기준 Claude Code 서브에이전트 문서는
<https://code.claude.com/docs/en/subagents> , Codex 프로젝트 지침은
<https://developers.openai.com/codex/guides/agents-md> 입니다.

실제 운영에서는

실제 운영에서는 2대·구독 2개로 늘리기 전에 1대에서 담당·작업 큐·보고가 먼저 안정되어야 합니다. 구조가 없으면 기기가 늘수록 혼선도 늘어납니다.

이번 장 복붙 프롬프트

현재 1대 운영을 2대 이상으로 확장할 때 늘어나는 위험과 대응 규칙을 표로 만들어줘.

APPENDIX

부록: 바로 쓰는 템플릿

모든 양식은 이 책에 그대로 실려 있습니다. 문서나 스프레드시트에 복사해 쓰면 됩니다. 빈 양식과 작성 예시를 같은 가상 사례로 짰습니다.

템플릿 01. 일감 큐 양식

일감 큐는 여러 에이전트가 같은 작업을 중복해서 하거나, 끝난 일을 잊어버리는 문제를 줄이는 기본 장치다. 처음에는 복잡한 도구가 필요 없다. 아래 표를 문서나 스프레드시트에 그대로 옮겨 쓰면 된다.

빈 양식

작업 ID	상태	우선순위	담당	작업 범위	완료 조건	완료 근거	다음 행동
[작성]	대기 / 진행 중 / 검토 필요 / 완료 / 보류	높음 / 보통 / 낮음	[작성]	[작성]	[작성]	[검증 후 작성]	[작성]
[작성]	대기 / 진행 중 / 검토 필요 / 완료 / 보류	높음 / 보통 / 낮음	[작성]	[작성]	[작성]	[검증 후 작성]	[작성]

작업 ID	상태	우선순위	담당	작업 범위	완료 조건	완료 근거	다음 행동
[작성]	대기 / 진행 중 / 검토 필요 / 완료 / 보류	높음 / 보통 / 낮음	[작성]	[작성]	[작성]	[검증 후 작성]	[작성]

상태값 기준

상태	의미	다음에 할 일
대기	아직 아무도 시작하지 않음	담당을 정하거나 에이전트가 집을 수 있게 둔다
진행 중	누군가 작업 중	같은 범위를 다른 에이전트에게 주지 않는다
검토 필요	작업은 끝났지만 확인이 필요	사람 또는 메인 에이전트가 결과를 확인한다
완료	완료 조건과 근거가 남음	더 건드리지 않는다
보류	정보 부족, 승인 대기, 위험 발견	이유를 적고 새 작업 배정을 멈춘다

예시 작성본

작업 ID	상태	우선순위	담당	작업 범위	완료 조건	완료 근거	다음 행동
B-001	완료	높음	메인 에이전트	블로그 글 목차	제목 5개와 섹션 순서 제안	draft-outline.md 생성, 중복 제목 없음 확인	사람 검토
B-002	검토 필요	보통	서브 에이전트 A	2번 섹션 초안	초보자용 설명 800자 이상	초안 작성 완료, 외부 링크 없음	문장 톤 검토
B-003	대기	낮음	미정	참고 자료 정리	공개 자료 3개 요약	아직 없음	담당 배정

학습용 가상 사례 작성본 — 가을 워크숍 안내문

아래는 이 책 2부가 담은 상태입니다. 빈 양식에 이 세 줄을 옮겨 적어도 됩니다.

작업 ID	상태	우선순위	담당	작업 범위	완료 조건	완료 근거	다음 행동
T-001	완료	높음	main	안내문 목차 초안	제목과 섹션 순서 확정	outline.md 섹션 6개, 다른 담당자 완료는 반려됨	발송하지 않음
T-002	완료	보통	sub	일정· 장소 사실 확인	주최 메모와 일치	확인표 3줄·불일치 0건, 재개 후 완료	없음
T-003	완료	보통	sub	한 줄 완료 보고	작업표 완료 3	보고 1줄, 숫자 일치 확인	작업표 보관

작업 등록 프롬프트

아래 프롬프트를 에이전트에게 붙여 넣으면 새 작업을 큐에 넣기 쉽게 정리할 수 있다.

아래 요청을 일감 큐에 넣을 수 있는 작은 작업으로 쪼개줘.

요청:

[여기에 하고 싶은 일을 적는다]

조건:

- 작업 하나는 30~90분 안에 끝날 크기로 나눈다.
- 각 작업마다 작업 ID, 작업 범위, 완료 조건, 검증 방법을 적는다.
- 외부 발송, 결제, 배포, 삭제, 개인정보 처리가 필요한 작업은 "승인 필요"로 표시한다.
- 실제 계정명, 전화번호, 토큰, 개인 식별자는 쓰지 않는다.

출력 형식:

| 작업 ID | 상태 | 우선순위 | 담당 | 작업 범위 | 완료 조건 | 완료 근거 | 다음 행동 |

완료 보고 프롬프트

작업을 끝낸 에이전트에게는 결과만 묻지 말고 근거를 같이 요구한다.

작업 완료 보고를 작성해줘.

작업 ID:

작업 범위:

수정하거나 만든 것:

검증한 것:

완료 조건 충족 여부:

남은 위험:

다음 행동:

주의:

- "완료했습니다"만 쓰지 말고 확인 근거를 적는다.
- 작업 범위 밖의 파일이나 내용은 건드렸다면 반드시 보고한다.
- 외부 발송, 배포, 삭제, 결제는 실행하지 말고 승인 필요로 표시한다.

사용 체크리스트

- [] 작업 ID가 겹치지 않는다.
- [] 작업 범위가 파일, 문서, 기능 단위로 좁혀져 있다.
- [] 완료 조건이 “잘게 만들기”처럼 모호하지 않다.
- [] 완료 근거가 비어 있는 작업은 완료로 바꾸지 않는다.
- [] 승인 필요한 작업은 실행 전에 멈춘다.
- [] 하루가 끝나면 완료, 보류, 검토 필요 상태만 다시 확인한다.

템플릿 02. 작업 분배 규칙표

작업 분배 규칙표는 “어떤 일은 에이전트가 알아서 해도 되고, 어떤 일은 사람에게 물어봐야 하는가”를 정하는 표다. 이 표가 없으면 사람은 사소한 일까지 계속 승인하게 되고, 에이전트는 위험한 일까지 자율 처리하려고 한다.

빈 양식

작업 종류	예시	기본 처리	담당 후보	승인 필요 조건	금지 사항
초안 작성	[작성]	자율 / 검토 / 승인	[작성]	[작성]	[작성]
문장 정리	[작성]	자율 / 검토 / 승인	[작성]	[작성]	[작성]
코드·자동화 수정	[작성]	자율 / 검토 / 승인	[작성]	[작성]	[작성]
공개 발송	[작성]	자율 / 검토 / 승인	[작성]	[작성]	[작성]

작업 종류	예시	기본 처리	담당 후보	승인 필요 조건	금지 사항
결제·구독 변경	[작성]	자율 / 검토 / 승인	[작성]	[작성]	[작성]
삭제·초기화	[작성]	자율 / 검토 / 승인	[작성]	[작성]	[작성]
개인정보·비밀값 포함 작업	[작성]	자율 / 검토 / 승인	[작성]	[작성]	[작성]

처리 등급

등급	의미	예시
자율	에이전트가 바로 처리해도 됨	초안, 내부 요약, 로컬 파일 정리
검토	에이전트가 처리한 뒤 사람 또는 메인이 확인	긴 문서, 코드 수정, 중요한 문구
승인	실행 전 사람 승인이 필요	외부 발송, 결제, 배포, 삭제, 개인정보 처리

예시 작성본

작업 종류	예시	기본 처리	담당 후보	승인 필요 조건	금지 사항
초안 작성	블로그 글 1차 초안	자율	서브 에이전트	외부 공개 문구로 바로 쓰려는 경우	실제 인물 정보 추정 금지
문장 정리	전자책 문단 다듬기	검토	서브 에이전트	제목·가격·상품 설명 변경	범위 밖 장 수정 금지
코드·자동화 수정	로컬 스크립트 오류 수정	검토	실행자	배포, 데이터 삭제, 계정 설정 변경	테스트 없이 완료 처리 금지
공개 발송	뉴스레터 발송	승인	오케스트레이터	항상 승인 필요	승인 전 발송 금지
결제·구독 변경	유료 도구 업그레이드	승인	사람	항상 승인 필요	자동 결제 실행 금지

작업 종류	예시	기본 처리	담당 후보	승인 필요 조건	금지 사항
삭제·초기화	오래된 파일 삭제	승인	사람	되돌리기 어렵거나 원본이 하나뿐일 때	백업 확인 전 삭제 금지
개인정보·비밀값 포함 작업	고객 자료 요약	승인	사람	항상 승인 필요	원문을 불필요하게 복사 금지

학습용 가상 사례 작성본 — 가을 워크숍 안내문

작업 종류	예시	기본 처리	담당 후보	승인 필요 조건	금지 사항
초안 작성	T-001 안내문 목차	자율	main	실제 발송 문구로 바로 쓰려는 경우	참석자 실명 추정 금지
사실 확인	T-002 일정·장소 대조	자율	sub	없음. 내부 확인만	T-001 완료 처리 금지
한 줄 보고	T-003 상태 요약	검토	sub → 검토자	보고를 외부 채널에 올리려는 경우	근거 없이 완료 선언 금지
공개 발송	안내문 실제 발송	승인	사람	항상 승인 필요	이 책은 발송하지 않음

분배 결정 프롬프트

작업을 누구에게 맡길지 애매할 때 아래 프롬프트를 쓴다.

아래 작업을 자율, 검토, 승인 중 어디에 들지 판단해줘.

작업 설명:

[작업 설명]

판단 기준:

- 외부로 공개되거나 발송되는가?
- 결제, 구독, 비용 증가가 생기는가?
- 삭제, 초기화, 되돌리기 어려운 변경인가?
- 개인정보, 비밀값, 고객 자료가 포함되는가?
- 같은 파일을 다른 에이전트가 작업 중인가?
- 실패해도 로컬에서 쉽게 되돌릴 수 있는가?

출력:

1. 처리 등급:
2. 추천 담당:
3. 승인 필요 여부:
4. 작업 범위:
5. 금지 사항:
6. 완료 근거:

기본 규칙 문장

아래 문장을 프로젝트 운영 규칙 맨 위에 붙여 둘 수 있다.

에이전트는 초안 작성, 내부 정리, 로컬 검증은 자율로 처리할 수 있다.
외부 발송, 결제, 배포, 삭제, 개인정보·비밀값 처리, 되돌리기 어려운 변경은 사람 승인 전까지 실행하지 않는다.
작업 범위가 명확하지 않으면 시작하지 말고 범위를 먼저 질문한다.
완료 보고에는 결과, 변경 범위, 검증 근거, 남은 위험을 포함한다.

사용 체크리스트

- [] 자율 처리 가능한 작업과 승인 필요한 작업이 분리되어 있다.
- [] “검토”와 “승인”을 구분했다.
- [] 외부 영향 작업은 항상 승인으로 표시했다.
- [] 개인정보와 비밀값이 포함될 가능성을 별도 항목으로 뒀다.
- [] 담당 후보가 역할명으로 적혀 있고 실제 계정명은 쓰지 않았다.
- [] 금지 사항이 비어 있지 않다.

템플릿 03. 승인 게이트 체크리스트

승인 게이트는 에이전트가 위험한 행동을 하기 직전에 멈추게 만드는 문이다. 모든 일을 승인받게 만들면 자동화가 느려진다. 반대로 아무 일도 승인받지 않게 만들면 사고가 난다. 핵심은 승인 대상을 명확히 정하는 것이다.

승인 필요 행동

아래 중 하나라도 해당하면 실행 전에 멈추고 사람 승인을 받는다.

- [] 외부 사람에게 메시지, 이메일, 댓글, 게시물을 보낸다.
- [] 공개 웹사이트, 스토어, 저장소, 문서에 반영한다.

- [] 결제, 구매, 환불, 구독 변경, 유료 API 사용량 증가가 생긴다.
- [] 파일 삭제, 데이터 초기화, 계정 설정 변경처럼 되돌리기 어렵다.
- [] 개인정보, 고객 자료, 비밀값, 인증 정보가 포함된다.
- [] 자동화가 반복 실행되어 비용이나 발송이 누적될 수 있다.
- [] 작업 범위가 불명확하거나 다른 에이전트와 충돌할 가능성이 있다.

빈 승인 요청 양식

[승인 요청]

작업 ID:

요청자:

하려는 행동:

대상:

외부 영향:

되돌릴 수 있는가:

비용 발생 여부:

개인정보·비밀값 포함 여부:

검증한 내용:

실패 시 영향:

승인 후 실행할 정확한 내용:

승인 없이는 하지 않을 것:

예시 작성본

[승인 요청]

작업 ID: P-014

요청자: 메인 에이전트

하려는 행동: 완성된 글을 공개 블로그 임시글로 등록

대상: 프로젝트 블로그

외부 영향: 공개 전 임시글이지만 웹 관리자 화면에 저장됨

되돌릴 수 있는가: 삭제 가능하지만 게시 설정 실수 시 공개될 수 있음

비용 발생 여부: 없음

개인정보·비밀값 포함 여부: 없음으로 확인

검증한 내용: 글 안에 실제 전화번호, 계정명, 토큰 형식 문자열이 없는지 확인

실패 시 영향: 비공개 초안이 잘못 공개될 수 있음

승인 후 실행할 정확한 내용: 임시글로만 저장하고 공개 버튼은 누르지 않음

승인 없이는 하지 않을 것: 공개 발행, 외부 공유, 알림 발송

승인 판단 답변 양식

사람이 승인할 때도 답을 짧고 명확하게 남긴다.

[승인 답변]

작업 ID:

판단: 승인 / 조건부 승인 / 거절

조건:

승인 범위:

기한:

추가 확인:

조건부 승인 예시

[승인 답변]

작업 ID: P-014

판단: 조건부 승인

조건: 임시글 저장까지만 허용

승인 범위: 블로그 관리자 화면에 비공개 초안 저장

기한: 오늘 작업 종료 전까지

추가 확인: 저장 후 공개 상태가 "비공개"인지 보고

학습용 가상 사례 승인 요청 — 가을 워크숍 안내문

이 책은 아래 요청을 실행하지 않습니다. 같은 양식을 내 일에 쓸 때의 작성 예시입니다.

[승인 요청]

작업 ID: T-004 (작업표에는 아직 없음. 가상 다음 단계)

요청자: 메인

하려는 행동: 안내문 초안을 실제 참석자에게 이메일로 보냄

대상: 외부 수신자

외부 영향: 있음

되돌릴 수 있는가: 발송 후 회수 어려움

비용 발생 여부: 메일 서비스 사용량

개인정보·비밀값 포함 여부: 수신자 주소가 개인정보에 해당할 수 있음

검증한 내용: T-001~T-003 완료와 근거 확인. 발송은 아직 안 함

실패 시 영향: 미완성 안내문이 외부로 나감

승인 후 실행할 정확한 내용: 없음. 학습용 가상 사례이므로 발송하지 않음

승인 없이는 하지 않을 것: 이메일 발송, 공개 게시, 신청 링크 활성화

[승인 답변]

작업 ID: T-004

판단: 거절

조건: 없음

승인 범위: 없음

기한: 없음

추가 확인: 내부 초안 단계만 유지. 발송이 필요하면 별도 사람 작업으로 새로 연다.

에이전트용 중지 문구

승인 게이트를 시스템 지시문이나 작업 지시문에 넣을 때는 아래처럼 쓸 수 있다.

다음 행동은 사람 승인 전까지 실행하지 마라:

- 외부 발송
- 공개 게시
- 결제 또는 구독 변경
- 배포
- 삭제 또는 초기화
- 개인정보, 비밀번호, 고객 자료 처리
- 되돌리기 어려운 변경

해당 행동이 필요하면 실행하지 말고 승인 요청 양식으로 보고하라.

승인 요청에는 작업 ID, 하려는 행동, 외부 영향, 비용, 되돌릴 수 있는지, 검증한 내용을 포함하라.

사용 체크리스트

- [] 승인 요청에 “승인 후 실행할 정확한 내용”이 들어 있다.
- [] 승인 없이는 하지 않을 행동이 명시되어 있다.
- [] 비용 발생 여부가 확인되었다.
- [] 개인정보·비밀값 포함 여부가 확인되었다.
- [] 되돌릴 수 있는지 적혀 있다.
- [] 조건부 승인일 때 허용 범위가 좁게 적혀 있다.
- [] 승인 기록 없이 외부 영향 작업을 완료로 표시하지 않는다.

템플릿 04. 사고 대응 룰 모음

사고 대응 룰은 문제가 생긴 뒤에 만들면 늦다. 에이전트 운영에서 사고는 보통 큰 사건처럼 시작하지 않는다. 같은 작업을 두 번 하거나, 완료 표시를 빼먹거나, 비용 큰 작업이 반복되거나, 작업 범위를 넘어선 수정이 생기는 식으로 작게 시작한다.

아래 양식은 “일단 멈추고, 사실을 확인하고, 복구하고, 재발 방지를 남기는” 순서로 쓰면 된다.

기본 원칙

의심되면 새 작업 배정을 멈춘다.
추측으로 원인을 단정하지 않는다.
마지막으로 실제 바뀐 것을 먼저 확인한다.
복구가 끝나기 전에는 같은 자동화를 다시 돌리지 않는다.
사고 기록에는 원인, 영향, 복구, 재발 방지를 남긴다.

사고 기록 빈 양식

[사고 기록]

사고 ID:
발견 시각:
발견자:
관련 작업 ID:
사고 유형:
현재 상태: 진행 중 / 멈춤 / 복구 완료 / 보류

무슨 일이 있었나:

영향 범위:

즉시 멈춘 것:

확인한 사실:

복구한 내용:

남은 위험:

재발 방지 규칙:

완료 근거:

유형별 대응표

사고 유형	즉시 할 일	확인할 것	재발 방지
같은 작업 중복 실행	새 작업 배정 중지	두 결과 중 최신·정확한 결과	작업 ID와 상태값 필수화
같은 파일 충돌	해당 파일 작업 중지	누가 어떤 범위를 수정했는지	작업 범위와 담당 분리
완료 마킹 누락	큐 상태 점검	실제 완료 근거가 있는지	완료 근거 없이는 완료 금지
비용 폭주 의심	반복 실행 중지	실행 횟수, 사용량, 남은 작업	반복 횟수 제한과 사전 경고
비밀값 노출 의심	공유·발송 중지	노출 위치와 접근 범위	비밀값 스캔과 승인 게이트
외부 발송 실수	추가 발송 중지	누구에게 무엇이 나갔는지	발송 전 승인 양식 강제
범위 밖 수정	해당 작업 중지	원래 범위와 실제 변경 범위	지시문에 금지 범위 추가

예시 작성본

[사고 기록]

사고 ID: I-003

발견 시각: 2026-01-15 16:20

발견자: 메인 에이전트

관련 작업 ID: D-008

사고 유형: 같은 작업 중복 실행

현재 상태: 복구 완료

무슨 일이 있었나:

문서 3번 섹션 요약 작업을 두 에이전트가 각각 처리했다.

영향 범위:

초안 파일 1개에 서로 다른 요약본이 생겼다. 외부 발송이나 공개 반영은 없었다.

즉시 멈춘 것:

문서 요약 관련 새 작업 배정을 중지했다.

확인한 사실:

작업 큐에서 D-008 상태가 "진행 중"으로 바뀌지 않아 두 번째 에이전트가 같은 작업을 집었다.

복구한 내용:

두 요약본을 비교해 더 정확한 문장만 남기고, 나머지는 삭제하지 않고 별도 메모로 옮겼다.

남은 위험:

다른 대기 작업에서도 상태값 누락 가능성이 있다.

재발 방지 규칙:

작업을 시작하기 전 상태를 반드시 "진행 중"으로 바꾼다. 상태 변경에 실패하면 작업을 시작하지 않는다.

완료 근거:

큐에서 D-008이 완료로 표시되었고, 완료 근거에 최종 파일명과 검토 메모를 남겼다.

학습용 가상 사례 사고 기록 — 잘못된 완료 시도

9장에서 `sub`가 이미 끝난 T-001을 닫으려 한 일을 이 양식에 옮긴 예시입니다.

[사고 기록]

사고 ID: I-W01

발견 시각: 작업 중 (학습용 가상 사례)

발견자: 검토자

관련 작업 ID: T-001

사고 유형: 다른 담당자의 완료 시도

현재 상태: 복구 완료

무슨 일이 있었나:

`sub`가 이미 `main`이 완료한 T-001을 다른 결과로 닫으려 했다.

영향 범위:

작업표 상태는 바뀌지 않았다. 외부 발송은 없었다.

즉시 멈춘 것:

T-001에 대한 추가 완료 처리.

확인한 사실:

작업표에 T-001 완료, 담당 `main`, 결과 "목차 6개 섹션 초안"이 이미 있음

복구한 내용:

복구 불필요. 검토자가 반려했다. T-002만 이어서 처리했다.

남은 위험:

반려를 성공으로 오해하면 다음 작업을 덮을 수 있다.

재발 방지 규칙:

완료는 작업지시의 담당자만 한다. 반려 기록을 사고 기록에 남긴다.

완료 근거:

작업표에서 T-001이 완료, 담당 main, 결과 "목차 6개 섹션 초안, outline.md".

사고 발생 시 에이전트에게 보낼 프롬프트

현재 사고 대응 모드다. 새 작업을 시작하지 말고 아래 순서로 보고해줘.

1. 관련 작업 ID
2. 마지막으로 실행한 행동
3. 실제로 바뀐 파일, 문서, 설정, 발송, 비용
4. 외부 영향 여부
5. 되돌릴 수 있는지
6. 지금 즉시 멈춰야 할 자동화
7. 복구를 위해 사람 승인이 필요한 행동

주의:

- 원인을 추측해서 단정하지 마라.
- 확인하지 않은 것은 "미확인"이라고 적어라.
- 삭제, 발송, 배포, 결제, 계정 설정 변경은 승인 전 실행하지 마라.

복구 완료 체크리스트

- [] 새 작업 배정을 멈췄다.
- [] 관련 작업 ID와 담당을 확인했다.
- [] 실제 변경된 파일, 문서, 설정, 발송, 비용을 확인했다.
- [] 외부 영향이 있는지 확인했다.
- [] 복구 과정에서 추가 삭제나 발송을 하지 않았다.
- [] 필요한 경우 사람 승인을 받았다.
- [] 작업 큐 상태를 최신으로 바꿨다.
- [] 사고 기록에 원인, 영향, 복구, 재발 방지를 남겼다.
- [] 같은 유형의 사고를 막는 규칙을 분배표나 승인 게이트에 반영했다.

짧은 중지 문구

급할 때는 아래 한 문장만 먼저 보내도 된다.

사고 대응 모드로 전환한다. 모든 새 작업 배정을 멈추고, 관련 작업 ID와 마지막 실제 변경 사항만 보고하라. 외부 발송, 배포, 삭제, 결제, 계정 변경은 승인 전 실행하지 마라.

맺는글

AI 에이전트를 여러 개 쓴다는 말은 화려하게 들립니다. 이 책이 남긴 것은 화려한 구조가 아닙니다. 작업을 작게 나누고, 역할 카드를 붙이고, 큐에 넣고, 누가 맡았는지 적고, 잘못된 완료는 반려하고, 멈춘 일은 재개하고, 끝난 일에는 근거를 남기는 것입니다.

「가을 워크숍 안내문」은 학습용 가상 사례입니다. 세 작업이면 충분합니다. 목차 초안, 사실 확인, 한 줄 보고. 이 세 칸이 내 일의 세 칸으로 바뀌면 책이 끝난 것입니다.

첫 주에 할 일은 이렇습니다.

- 역할 카드 3장을 내 프로젝트 이름으로 복사한다.
- 작업 세 개를 큐에 넣는다.
- 담당 이름을 남긴다.
- 완료 조건과 확인 방법을 적는다.
- 외부 영향이 있으면 멈추고 승인 요청을 만든다.

이 다섯 가지가 반복되면 AI는 채팅 상대가 아니라 작은 팀처럼 움직이기 시작합니다. 그때부터 에이전트를 하나 더 쓰는 것도, 다른 컴퓨터로 나누는 것도 의미가 생깁니다.

오늘 한 일을 내일의 에이전트가 알아볼 수 있게 남기세요. 그 기록이 운영입니다.

저자소개

마이너스베타스튜디오는 1인 사업자와 작은 팀을 위한 AI 자동화, 콘텐츠 운영, 업무 시스템 설계를 실험하고 정리합니다. 복잡한 기술을 그대로 복제하게 하기보다, 비개발자도 자기 환경에 맞게 가져갈 수 있는 작은 운영 규칙과 실전 템플릿으로 바꾸는 데 집중합니다.

이번 책은 AI에게 여러 일을 맡겼을 때 생기는 누락, 중복, 완료 판단 문제를 공개 가능한 형태로 정리한 AI WORKSHOP 두 번째 실전 PDF입니다. 핵심은 단순합니다. AI가 더 많은 일을 하게 만들수록, 사람은 더 명확한 기준과 멈춤 장치를 가져야 합니다.

무료 공개판 안내

이 전자책은 kangdaejong.com 에서 무료로 공개한 판입니다. 개인 학습·업무에 자유롭게 읽고 활용하세요. 템플릿과 예시는 자신의 작업에 맞게 수정해 쓸 수 있습니다. 저작권은 강대중(마이너스베타스튜디오)에게 있습니다.