CLAUDE CODE DEEP DIVE WORKSHOP

Chapter 5 - CLI Reference

플래그 전집, Headless, 세션 제어, 자동화 파이프라인

Update : 2026.07 / Claude Code v2.1 최신 채널 기준, 공식 문서 code.claude.com/docs 반영

Nambong Ha
Solutions Architect
AWS Korea
nambong@amazon.com

Chapter 5 Agenda

9 Parts / 107 Slides

PART 01	claude 명령과 플래그	서브커맨드 지도, 플래그 6분류 전집
PART 02	Headless 심화	-p, JSON, 구조화 출력, 예산 상한
PART 03	세션 제어	continue, resume, from-pr, 웹 왕복
PART 04	스케줄과 자동 실행	/loop, /goal, Routines, deep links
PART 05	CI/CD 통합	GitHub Actions, GitLab, 비용 통제
PART 06	자동화 패턴	트리아지, 로그 분석, 보고서, 배치
PART 07	환경변수	분류별 카탈로그와 점검, 보안
PART 08	디버깅	debug 카테고리, safe-mode, 진단 흐름
PART 09	Recap & Labs	요약, FAQ, 실습 4종

Chapter 5 Agenda

9 Parts / 107 Slides

PART 01	claude 명령과 플래그	서브커맨드 지도, 플래그 6분류 전집
PART 02	Headless 심화	-p, JSON, 구조화 출력, 예산 상한
PART 03	세션 제어	continue, resume, from-pr, 웹 왕복
PART 04	스케줄과 자동 실행	/loop, /goal, Routines, deep links
PART 05	CI/CD 통합	GitHub Actions, GitLab, 비용 통제
PART 06	자동화 패턴	트리아지, 로그 분석, 보고서, 배치
PART 07	환경변수	분류별 카탈로그와 점검, 보안
PART 08	디버깅	debug 카테고리, safe-mode, 진단 흐름
PART 09	Recap & Labs	요약, FAQ, 실습 4종

Chapter 5 Learning Objectives

본 챕터에서 달성할 학습 목표

AT THE END OF THIS CHAPTER

플래그 체계를 자유자재로 조합하고, 헤드리스 출력으로 파이프라인을 만들며, 세션을 어디서든 이어가고, 예산과 턴 상한 아래 안전한 자동화를 운영할 수 있습니다.

OBJ 1	OBJ 2	OBJ 3	OBJ 4
플래그 정복	파이프라인	세션 자유	안전 자동화
6분류 체계로 어떤 조합도 스스로 구성.	-p와 JSON, 구조화 출력으로 스크립트 통합.	이어가기, 분기, 웹 왕복을 자유롭게.	예산, 턴, 도구 상한 아래 CI와 스케줄 운영.

9 Parts	4 Labs	60+ Flags	~3h + 1.5h
구조화된 학습 경로	Chapter 5 실습 랩	플래그 전집 범위	강의와 실습 시간

왜 CLI인가

대화형에서 무인 파이프라인까지의 스펙트럼

대화형의 세계 (Ch.1-4)

- 사람이 조타수, 확인과 반복
- 설정과 훅, MCP는 이미 완성
- 한 번에 한 세션, 한 저장소
- 성과가 개인 생산성에 머묾

CLI 자동화의 세계 (Ch.5)

- 스크립트가 조타수, exit code로 판정
- 같은 설정 자산이 무인에서도 동작
- cron, CI, Routines로 병렬과 반복
- 성과가 파이프라인 자산이 됨

-p 한 글자
두 세계의 스위치

재사용
Ch.4 자산 그대로

판정 가능
exit, JSON 계약

무인
예산, 턴 상한 전제

05

CLI Reference

명령, 플래그, 그리고 무인 자동화

CHAPTER 05 / PART 01

claude 명령과 플래그

60여 플래그를 6분류로 정복

- 명령 구조와 서브커맨드 지도
- 플래그 6분류 체계
- 동작, 모델, 권한, 구성 플래그
- 시스템 프롬프트 4종과 조합 관용구

명령 구조

claude [플래그] [프롬프트]

command anatomy

```
$ claude                        # 대화형 세션
$ claude "explain this project"   # 초기 프롬프트로 시작
$ claude -p "query"               # 실행 후 종료 (headless)
$ cat logs.txt | claude -p "explain"   # 파이프 입력
$ claude -c                     # 이 디렉토리 최근 대화 계속
$ claude -r "auth-refactor" "Finish this PR"

# 서브커맨드: claude <sub> (update, mcp, agents, ...)
# 오타 교정: claude udpate
#  -> Did you mean claude update?
```

3 형태
대화, 초기, -p

파이프
stdin이 입력

-c / -r
세션 복귀 (P3)

오타 제안
근접 서브커맨드

서브커맨드 지도 1 / 계정과 설치

세션 밖의 관리 명령

auth login / logout	--email, --sso, --console 옵션	구독 vs API 과금 선택
auth status	JSON 상태, --text, exit 0/1	스크립트 로그인 판정
setup-token	CI용 장기 OAuth 토큰 발급	출력만, 저장 안 함
update / install	갱신 / 버전 지정 재설치	install stable, 2.1.118
doctor	환경 진단과 자동 수정	Ch.4 진단 4도구
project purge [path]	프로젝트 로컬 상태 일괄 삭제	--dry-run, --all

서브커맨드 지도 2 / 운영

백그라운드와 확장의 관리 명령

agents / attach / logs	에이전트 뷰, 접속, 로그 (--json)	Ch.2 백그라운드 관리
stop / respawn / rm	중지, 재기동(--all), 목록 제거	바이너리 갱신 반영
daemon status / stop	슈퍼바이저 진단과 정지	--keep-workers
mcp login / logout	패널 없이 OAuth (--no-browser)	SSH 환경 인증
gateway --config	apps gateway 서버 기동	Ch.3 파트 3

플래그 6분류 지도

60여 개를 담는 여섯 박스

동작 모드	-p, --bg, --remote, --worktree, --bare, --safe-mode	이 파트
세션	-c, -r, --from-pr, --fork-session, -n	Part 3
모델과 사고	--model, --effort, --fallback-model, --advisor	이 파트
권한과 도구	--permission-mode, --tools, --allowed/disallowedTools	이 파트
구성과 확장	--settings, --agents, --mcp-config, --plugin-dir	이 파트
출력과 진단	--output-format, --json-schema, --verbose, --debug	Part 2, 8

동작 모드 플래그

세션이 어디서 어떻게 뜨는가

-p / --print	비대화 실행 후 종료, SDK 경유	Part 2
--bg (--exec)	백그라운드 기동, ID 반환 (-p 병용 불가)	--exec는 셸 잡
-w / --worktree	격리 워크트리, #PR번호로 분기 가능	--tmux 병용
--cloud / --teleport	웹 세션 생성 / 로컬로 회수	Part 3
--bare	훅, 스킬, MCP 미탐색 최소 기동	스크립트 가속
--init-only / --init	Setup 훅만 실행 / -p 전 준비	Ch.4 Setup

모델과 사고 플래그

지능 조절의 세션판

model flags

```
$ claude --model opus          # 별칭 또는 전체 ID
$ claude --effort high         # low..max (모델별 상이)

# 폴백 체인: 과부하, 미제공 시 순서대로
$ claude --fallback-model sonnet,haiku
#  설정 fallbackModel 키로 영구화 가능

$ claude --advisor opus        # 어드바이저 도구 활성
$ claude --betas <header>      # API 키 사용자 전용
 # 우선순위: 플래그 > ANTHROPIC_MODEL > 설정
```

별칭 4종
sonnet, opus, haiku, fable

폴백 체인
콤마 순차 시도

effort 5단
세션 한정 상향

플래그 우선
env, 설정보다

권한과 도구 플래그

--tools vs allowed vs disallowed

permission flags

```
$ claude --permission-mode plan     # 6모드 (Ch.4)
# 무확인 허용 (규칙 문법)
$ claude -p --allowed-tools "Bash(git log *)" "Read" ...
# 도구 자체를 목록에서 제거 vs 호출만 거부
$ claude --disallowedTools "Bash(rm *)"  # 해당 호출만 거부
$ claude --disallowedTools "mcp__*"      # 전 MCP 도구 제거
# 내장 도구 제한 (MCP 미영향)
$ claude --tools "Bash,Edit,Read"
```

베어 이름	스코프 규칙	--tools	--add-dir
도구 컨텍스트 제거	호출만 거부	내장 한정	범위 확장 (구성 미탐색)

구성과 확장 플래그

이 세션만의 설정 세계

config flags

```
# 세션 한정 설정 오버레이 (파일 또는 인라인)
$ claude --settings ./ci-settings.json
$ claude --settings '{"model":"haiku"}'

# 로드할 설정 소스 선별
$ claude --setting-sources user,project

# 확장 주입 (Ch.2, 4 복습)
$ claude --agents '{"reviewer":{...}}'
$ claude --mcp-config ./mcp.json --strict-mcp-config
$ claude --plugin-dir ./my-plugin --disable-slash-commands
```

--settings
세션 오버레이

sources 선별
CI 격리 재현

strict
지정 외 MCP 배제

managed 주의
사이드로드 차단 대상

시스템 프롬프트 4종

정체성 유지와 대체의 갈림

--append-system-prompt	기본 프롬프트 뒤에 텍스트 추가	정체성 유지 + 규칙
--append-system-prompt-file	파일 내용을 추가	긴 규칙, 버전 관리
--system-prompt	전체 대체 (안전 지침도 소멸)	비코딩 에이전트
--system-prompt-file	파일로 전체 대체	대체와 상호 배타
선택 기준	코딩 조수 유지면 append	대체는 책임 이전
대안	영구 페르소나는 output style	상시 규범은 CLAUDE.md

출력과 진단 플래그 개요

Part 2, 8의 예고편

--output-format	text, json, stream-json	Part 2 본진
--json-schema	스키마 검증된 구조화 출력	-p 전용, Part 2
--include-partial-messages	스트림에 부분 이벤트 포함	stream-json 전제
--include-hook-events	훅 수명주기 이벤트 포함	관측 강화
--verbose	턴 단위 전체 출력	viewMode 오버라이드
--debug / --debug-file	카테고리 필터 디버그, 파일 출력	Part 8

--bare vs --safe-mode

닮았지만 목적이 다른 두 스위치

TWO MINIMAL MODES

bare는 속도를 위한 최소 기동, safe-mode는 고장 원인 이분을 위한 무커스텀 기동입니다.
managed 정책은 safe-mode에서도 적용됩니다.

--bare (스크립트 가속)

- 훅, 스킬, 플러그인, MCP, CLAUDE.md 미탐색
- Bash, 읽기, 편집 도구는 사용 가능
- -p 반복 호출의 기동 시간 절약
- CLAUDE_CODE_SIMPLE 설정됨

--safe-mode (고장 진단)

- 전 커스터마이즈 비활성, 인증, 권한은 정상
- managed 정책 훅 등은 여전히 적용
- 커스텀이 원인인지 이분 판정
- Fable 5 자동 폴백 원인 추적에도

조합 관용구

현장에서 굳어진 다섯 줄

idioms

```
# 1. CI 리뷰: 예산과 도구를 잠근 헤드리스
claude -p --max-budget-usd 2 --allowed-tools "Read" "Grep" ...

# 2. 빠른 배치: 최소 기동 + 저비용 모델
claude --bare -p --model haiku "..."

# 3. 격리 실험: PR 분기 워크트리
claude -w '#123' --permission-mode plan
# 4. 세션 재현: 소스 고정 + 오버레이
claude --setting-sources project --settings ./ci.json -p "..."
# 5. 무인 야간: dontAsk + 폴백 체인
claude -p --permission-mode dontAsk --fallback-model sonnet,haiku "..."
```

5 관용구	상한 우선	재현성	Part 2, 5
복붙 출발선	무인의 규율	sources + settings	심화 연결

Part 1 Recap

명령과 플래그 핵심 정리

KEY TAKEAWAY

서브커맨드 두 지도와 플래그 여섯 서랍이면 전집이 정리됩니다.

무인에는 상한, 진단에는 safe-mode, 속도에는 bare입니다.

핵심 정리

- auth, project purge 등 서브커맨드 성숙
- disallowed 베어 이름 = 도구 제거
- --settings 세션 오버레이
- append vs 대체: 정체성 기준
- 관용구 5로 조합 시작

다음 파트

- Part 2 Headless 심화
- 입력 6경로와 exit code
- JSON, stream-json 스키마
- --json-schema 구조화 출력
- 예산, 턴 상한과 캐시 최적화

CHAPTER 05 / PART 02

Headless 심화

-p, 파이프라인의 심장

- 입력 6경로와 exit code
- JSON, stream-json 스키마
- --json-schema 구조화 출력
- 예산, 턴 상한과 캐시 최적화

-p의 본질

SDK를 경유하는 단발 실행

PRINT MODE

-p는 단순 출력 모드가 아니라 Agent SDK 경로를 타는 단발 에이전트 실행입니다. 도구와 훅, 설정이 모두 살아 있는 채로 결과만 표준출력에 남깁니다.

FULL

전 기능 유지

도구 실행, 훅, MCP, 권한이 대화형과 동일 동작.

CONTRACT

계약 3종

stdout 결과, stderr 진단, exit code 판정.

SDK

SDK 경유

동일 엔진의 CLI 창구, 심화는 Ch.6 SDK.

-p 전용

전용 플래그

max-turns, max-budget, json-schema 등은 -p 한정.

단발 에이전트
-p의 정체

3 계약
out, err, exit

-p 전용군
상한, 스키마

Ch.6
SDK 본편

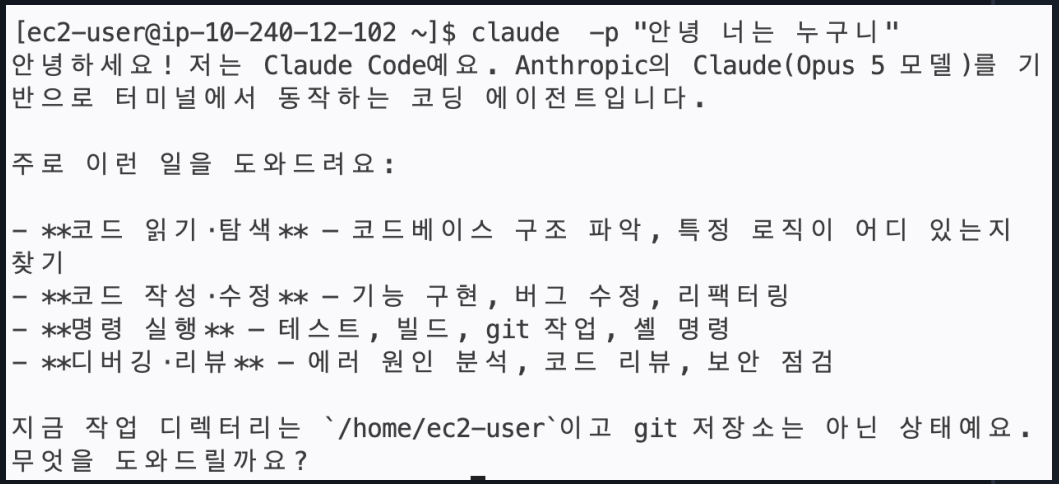
입력 6경로

프롬프트가 들어오는 모든 길

input paths

```
claude -p "직접 인자"                  # 1 인자
cat error.log | claude -p "원인 분석"      # 2 파이프
claude -p "$(cat prompt.txt)"          # 3 명령 치환
claude -p "요약해" < notes.md            # 4 리다이렉트
claude -p <<'EOF'                      # 5 히어독
여러 줄 지시문 ...
EOF
claude -c -p "이어서 리팩토링"           # 6 세션 이어받기

# 파일 참조는 프롬프트 안 @경로 도 유효 (Ch.4)
```



```
[ec2-user@ip-10-240-12-102 ~]$ claude  -p "안녕 너는 누구니"
안녕하세요! 저는 Claude Code예요. Anthropic의 Claude(Opus 5 모델)를 기
반으로 터미널에서 동작하는 코딩 에이전트입니다.

주로 이런 일을 도와드려요:

- **코드 읽기·탐색** — 코드베이스 구조 파악, 특정 로직이 어디 있는지
찾기
- **코드 작성·수정** — 기능 구현, 버그 수정, 리팩터링
- **명령 실행** — 테스트, 빌드, git 작업, 셸 명령
- **디버깅·리뷰** — 에러 원인 분석, 코드 리뷰, 보안 점검

지금 작업 디렉터리는 `/home/ec2-user`이고 git 저장소는 아닌 상태예요.
무엇을 도와드릴까요?
```

6 경로
인자, 파이프, 치환 등

파이프 = 데이터
인자 = 지시

-c -p
맥락 재사용

@경로
파일 참조 병행

Exit Code 계약

파이프라인 판정의 언어

exit codes

```
claude -p "테스트 실패 원인을 찾아 수정" \
  --max-turns 15
case $? in
  0) echo OK ;;
  *) echo "FAIL (code $?)" ; exit 1 ;;
esac

# 0 = 정상 완료 / 비0 = 오류, 상한 도달 등
# --max-turns 초과도 오류로 종료 -> 게이트 재료
# claude auth status: 로그인 0, 미로그인 1
# claude ultrareview: 통과 0, 발견 1
```

```
claude -p "Hello" \
  --max-turns 15
case $? in
  0) echo OK ;;
  *) echo "FAIL (code $?)" ; exit 1 ;;
esac
```

```
[ec2-user@ip-10-254-21-101 ch5]$ ./test.sh
Hello! I'm ready to help with your project in `claude-lab
/ch5`. What would you like to work on?
OK
```

0 / 비0
판정 이분법

상한 초과
오류 종료 활용

auth, ultrareview
판정형 서브커맨드

set -e 주의
의도적 분기 처리

--output-format json

결과 봉투의 스키마

json envelope

```
$ claude -p "고위험 파일 3개" --output-format json
{
  "type": "result",
  "subtype": "success",
  "result": "1. src/auth/... (본문)",
  "session_id": "...",
  "total_cost_usd": 0.0284,
  "num_turns": 4,
  "duration_ms": 21033,
  "usage": { "input_tokens": ..., "output_tokens": ... }
}
# result가 본문, 나머지는 관측 메타데이터
```

.result	cost, turns	session_id	is_error
본문 필드	관측 메타	재개 열쇠	오류 봉투 분기

stream-json 이벤트

진행을 실시간으로 읽기

system (init)	세션 시작, 모델과 도구 목록	첫 이벤트
assistant	모델 응답 메시지 단위	본문 스트림
user (tool_result)	도구 실행 결과 회신	진행 추적
result	최종 봉투 (json과 동일)	마지막 이벤트
--include-partial-messages	토큰 단위 부분 이벤트 추가	타자기 UI
--include-hook-events	훅 수명주기 이벤트 추가	심층 관측

--json-schema 구조화 출력

파싱이 아니라 계약

structured output

```
$ claude -p "이 diff의 위험도를 평가해" \
  --json-schema '{
    "type": "object",
    "properties": {
      "risk":    { "enum": ["low", "medium", "high"] },
      "reasons": { "type": "array", "items": {"type":"string"} },
      "block":   { "type": "boolean" }
    }, "required": ["risk", "block"] }'

# 워크플로 완료 후 스키마 검증된 JSO
# 정규식 긁기 대신 계약 기반 파이프라인
```

{"risk":"medium","block":false,"reasons":["결제 함수 charge()에 amount 입력 검증이 없음을 TOD
O 주석으로 명시한 채 방치 – 음수/비숫자 금액도 성공 처리됨","currency가 \"KRW\"로 하드코딩되
어 다중 통화 시나리오에서 오청구 가능성","변경 범위 자체는 1개 파일 2줄로 작고 기존 동작을 깨
뜨리지는 않음"]}

스키마 검증	-p 전용	enum, required	SDK 동형
출력 보증	적용 조건	게이트 친화 설계	Zod, Pydantic (Ch.6)

jq 파싱 기초

봉투에서 값 꺼내기

jq essentials

```
OUT=$(claude -p "..." --output-format json)

echo "$OUT" | jq -r '.result'         # 본문
echo "$OUT" | jq -r '.total_cost_usd'  # 비용
echo "$OUT" | jq -r '.session_id'      # 재개 키

# 구조화 출력 결합: 게이트 한 줄
RISK=$(claude -p "..." --json-schema "$SCHEMA" | jq -r '.risk')
[ "$RISK" = "high" ] && exit 1

# stream-json: 줄 단위 select
... | jq -c 'select(.type == "result")'
```

-r	.필드	select	게이트 1줄
raw 문자열 출력	봉투 접근	스트림 필터	구조화 결합 효과

예산과 턴 상한

무인 실행의 안전벨트

caps (-p only)

```
claude -p "의존성 취약점 정리해 패치 PR 초안까지" \
  --max-turns 20 \
  --max-budget-usd 3.00

# --max-turns: 에이전틱 턴 수 상한, 초과 시 오류 종료
#   기본 무제한 -> 무인에서는 반드시 지정
# --max-budget-usd: 달러 상한, 도달 시 중단

# 설계 요령
#  파일럿 실측 p95의 1.5배로 시작
#  초과 종료는 실패가 아니라 신호: 로그와 알람으로
```

무제한 기본
턴 상한 미지정 시

달러 캡
호출 단위 예산

p95 x1.5
상한 산정 요령

gateway 병행
Ch.3 개인 한도

에러 처리와 재시도 골격

일시 오류와 진짜 실패의 분리

retry skeleton

```
run_claude() {
  local attempt=1
  while [ $attempt -le 3 ]; do
    OUT=$(claude -p "$1" --output-format json \
        --max-turns 15 2>err.log) && {
      echo "$OUT"; return 0; }
    grep -qiE 'rate|overloaded|529' err.log || break
    sleep $(( attempt * 20 )); attempt=$((attempt+1))
  done
  return 1   # 진짜 실패: 재시도 무의미
}
# 폴백은 --fallback-model 이 더 우아한 1차 방어
```

3회 백오프
일시 오류 한정

stderr 판별
재시도 가치 판단

fallback 우선
모델 과부하 1차

함수화
전 스크립트 재사용

다중 호출 집계

배치 결과를 하나의 표로

aggregation

```
for f in src/services/*.ts; do
  claude --bare -p "이 파일의 복잡도를 평가" \
    --json-schema '{"type":"object","properties":{
      "file":{"type":"string"},"score":{"type":"number"},
      "top_issue":{"type":"string"}}}' < "$f"
done | jq -s 'sort_by(-.score)' > report.json

jq -r '.[] | [.file, .score, .top_issue] | @csv' \
  report.json > report.csv

# -s(slurp)로 배열화, @csv로 표 변환
```

jq -s
결과 배열화

@csv
표 변환 관용구

구조화 전제
집계 가능 조건

bare + haiku
배치 비용 통제

완성 스크립트

이 파트의 부품을 한 몸으로

nightly-deps-audit.sh

```
#!/usr/bin/env bash
set -uo pipefail
SCHEMA='{"type":"object","properties":{"critical":
 {"type":"integer"},"summary":{"type":"string"}},
 "required":["critical","summary"]}'
OUT=$(run_claude_json "npm audit 결과를 분석해 심각도별로 정리" \
 "$SCHEMA") || { notify "감사 실행 실패"; exit 1; }
CRIT=$(echo "$OUT" | jq -r '.critical')
echo "$OUT" | jq -r '.summary' > reports/deps-$(date +%F).md
[ "$CRIT" -gt 0 ] && notify "Critical $CRIT건" && exit 1
exit 0
```

```
run_claude_json() {
  local prompt="$1" schema="$2"
  claude -p "$prompt" --json-schema
"$schema" \
    --max-turns 10 --permission-mode
acceptEdits 2>/dev/null
}
```

부품 결합
재시도 + 스키마 + 게이트

리포트 축적
날짜별 산출

exit 게이트
cron, CI 겸용

Part 4, 5
스케줄, CI 탑재

Part 2 Recap

Headless 심화 핵심 정리

KEY TAKEAWAY

-p는 SDK 경유 단발 에이전트입니다.

구조화 출력으로 계약을 맺고, 턴과 예산 상한으로 무인의 규율을 세우면 파이프라인이 완성됩니다.

핵심 정리

- 계약 3종: stdout, stderr, exit
- json 봉투: result + 비용 메타
- --json-schema = 파싱에서 계약으로
- max-turns, max-budget 필수 상한
- exclude-dynamic으로 캐시 적중

다음 파트

- Part 3 세션 제어
- 저장 구조와 트랜스크립트
- continue, resume, from-pr
- fork와 이름 관리
- 웹 왕복과 Remote Control

CHAPTER 05 / PART 03

세션 제어

시간과 공간을 넘나드는 대화

- 저장 구조와 트랜스크립트
- continue, resume, from-pr
- fork, 이름, 내보내기
- 웹 왕복과 Remote Control

세션 저장 구조

대화는 어디에 사는가

SESSION STORAGE

세션 트랜스크립트는 홈의 projects 디렉토리에 프로젝트별 JSONL로 저장됩니다.

cleanupPeriodDays가 수명을, project purge가 일괄 정리를 담당합니다.

WHERE	LIFE	OFF	PURGE
저장 위치	수명	저장 끄기	일괄 정리
~/.claude/projects/ 아래 프로젝트별 JSONL.	cleanupPeriodDays 기본 30일 자동 정리 (Ch.4).	-p는 --no-session-persistence, 전 모드는 env 변수.	claude project purge, --dry-run으로 예행.

JSONL	30일	훅 입력	purge
저장 형식	기본 수명	transcript_path (Ch.4)	프로젝트 청소

continue vs resume

가장 최근 하나냐, 골라잡기냐

TWO WAYS BACK

-c는 이 디렉토리의 최근 대화 하나로 직행, -r은 ID나 이름으로 특정하거나 픽커에서 고릅니다.
백그라운드 세션도 픽커에 bg 표시로 등장합니다.

-c / --continue

- 현재 디렉토리 최근 대화 직행
- add-dir로 얹은 세션도 포함
- -c -p 로 헤드리스 이어받기
- 일상 복귀의 기본기

-r / --resume

- ID 또는 이름으로 특정 재개
- 인자 없이는 대화형 픽커
- 픽커에 bg 세션 표시 (v2.1.144+)
- ID 검색은 현 프로젝트와 워크트리 한정

--from-pr

PR이 세션의 주소가 된다

pr-linked sessions

```
# Claude가 만든 PR은 세션과 자동 링크됨
$ claude --from-pr 123
$ claude --from-pr https://github.com/org/repo/pull/123

# GitHub, GitHub Enterprise, GitLab MR,
# Bitbucket PR URL 모두 수용

# 리뷰 코멘트 대응 흐름
#  리뷰어 코멘트 확인 -> claude --from-pr 123
#  -> 그 PR을 만든 맥락 그대로 후속 수정

# 워크트리 결합: claude -w '#123' 은 코드 분기 (P1)
```

자동 링크
PR 생성 시

4 플랫폼
GH, GHE, GitLab, BB

맥락 복원
리뷰 대응 무재설명

-w '#N' 구분
코드 분기와 별개

fork와 session-id

타임라인 분기와 고정 좌표

fork / fixed id

```
# 원본 보존 분기: 재개하되 새 세션 ID로
$ claude --resume auth-refactor --fork-session
#  실험 방향 A, B를 같은 지점에서 각각 분기

# 고정 좌표: 스크립트가 세션 ID를 소유
$ SID=$(uuidgen)
$ claude -p --session-id "$SID" "1단계: 스캔"
$ claude -p --resume "$SID" "2단계: 스캔 결과로 수정"

# 대화형의 /fork (Ch.2)와 같은 철학, CLI 판
```

fork-session	A/B 실험	UUID 고정	다단 파이프
원본 보존 분기	같은 지점 두 갈래	스크립트 소유 세션	재개 체인 설계

이름과 내보내기

세션에 문패 달기

naming / export

```
# 시작부터 이름 부여
$ claude -n "payments-refactor"

# 세션 중 개명 (프롬프트 바에도 표시)
> /rename auth-hotfix

# 이름으로 복귀
$ claude -r "payments-refactor" "어제 이어서"

# 기록 내보내기
> /export          # 대화를 파일, 클립보드로
# 원본 JSONL은 projects 디렉토리에 그대로
```

-n / /rename
이름 두 경로

픽커 가독성
이름의 효용

/export
공유용 산출

팀 관례
티켓 번호 명명

체크포인트와 /rewind

세션 안의 시간 여행

CHECKPOINTING

Claude의 파일 편집은 체크포인트로 추적됩니다.
/rewind 로 코드와 대화를 이전 지점으로 되돌려, 잘못 든 길을 세션 안에서 무르는 안전망입니다.

TRACK
자동 추적

편집 시점마다 파일 상태 체크포인트 기록.

REWIND
/rewind

지점 선택 복원, 코드와 대화 범위 선택.

SCOPE
한계 인지

Bash 부수효과, 외부 시스템은 복원 밖.

VS GIT
git과 분업

세션 내 무르기는 rewind, 이력은 git.

자동
체크포인트 기록

지점 복원
코드, 대화 선택

부수효과 밖
복원 한계

git 병행
이력의 본진

웹 왕복 / --remote와 --teleport

터미널과 claude.ai 사이

STEP 1
위임
claude --remote "로그인 버그 수정"

›

STEP 2
클라우드 실행
claude.ai 샌드박스에서 진행

›

STEP 3
모니터
웹, 모바일 앱에서 진행 확인

›

STEP 4
회수
claude --teleport 로 로컬 재개

회의 전 위임, 자리 복귀 후 회수가 전형적 리듬

--remote
웹 세션 생성

--teleport
로컬 회수

샌드박스
클라우드 실행 환경

이동 근무
대표 시나리오

Remote Control

로컬 세션을 폰에서 조종

REMOTE CONTROL

웹 왕복과 반대 방향입니다.

내 기계에서 도는 세션을 claude.ai와 모바일 앱에서 이어서 조종합니다. 조직은 disableRemoteControl로 통제합니다.

MODE 1	MODE 2	USE	GOV
--rc 병행	서버 모드	시나리오	조직 통제
대화형 세션에 원격 접속 허용을 얹음.	claude remote-control, 로컬 화면 없이 대기.	퇴근길 승인, 소파에서 장기 작업 지시.	disableRemoteControl (Ch.3, 4 disable 패밀리).

--rc
세션 병행 모드

server
무화면 대기 모드

폰 승인
권한 프롬프트 응답

managed
차단 가능 키

Part 3 Recap

세션 제어 핵심 정리

KEY TAKEAWAY

세션은 JSONL로 살아 있고, -c와 -r, --from-pr로 언제든 돌아가며, fork로 분기하고, remote와 teleport로 공간까지 넘습니다.

핵심 정리

- projects JSONL + 30일 수명
- -c 직행, -r 픽커 (bg 포함)
- --from-pr: PR이 세션 주소
- fork-session, session-id 고정
- rewind 안전망, 웹 왕복

다음 파트

- Part 4 스케줄과 자동 실행
- 자동화 표면 지도
- /loop와 /goal
- Routines와 deep links
- cron 레시피

CHAPTER 05 / PART 04

스케줄과 자동 실행

손을 떼도 도는 Claude

- 자동화 표면 5종 지도
- /loop, /goal과 세션 내 예약
- Routines: 클라우드 자동 실행
- deep links, Slack, cron 레시피

자동화 표면 5종

어디서 반복시킬 것인가

세션 내	/loop, cron 도구, /goal	떠 있는 세션이 전제
Desktop 예약	Desktop 앱 scheduled tasks	내 기계, GUI 관리
Routines	Anthropic 관리 클라우드 실행	기계 불필요, 3종 트리거
CI 파이프라인	이벤트 구동 (push, PR)	Part 5 본진
cron + headless	OS 스케줄러 + -p 스크립트	완전 자가 통제

/loop와 cron 도구

세션 안의 반복과 예약

in-session scheduling

```
> /loop 테스트가 전부 통과할 때까지 실패를 고쳐
#  조건 충족까지 같은 작업을 반복

> 10분마다 배포 파이프라인 상태를 확인하고
  실패로 바뀌면 원인을 정리해줘
#  cron 스케줄링 도구로 반복, 폴링 예약

> 45분 뒤에 스탠드업 준비하라고 알려줘
#  1회성 리마인더도 같은 도구

# 세션 종료 시 함께 종료: 상주 요구는 다른 표면으로
```

/loop	cron 도구	1회 리마인더	세션 수명
조건 반복	반복, 폴링 예약	지연 실행	함께 종료 주의

/goal

완료 조건을 계약으로

goal-driven session

```
> /goal 전체 테스트 통과 + 린트 0 경고 상태 도달

# 동작
#  턴이 끝나도 조건 미충족이면 Claude가 계속 작업
#  조건 충족 시 목표 달성 보고 후 종료

# /loop와의 결
#  loop: 같은 작업의 반복 실행
#  goal: 상태 도달까지 수단을 바꿔가며 지속

# 무인화 짝: 예산, 턴 상한과 함께 (Part 2)
```

완료 조건	다턴 지속	loop와 구분	상한 병행
goal의 계약	수단 자율 변경	반복 vs 도달	무인 규율

Slack에서 위임

대화 채널이 디스패치 창구

HOW	WHERE	FIT	GOV
멘션 위임	실행 위치	맞는 일	운영
Slack에서 Claude를 멘션해 코딩 작업 위임.	클라우드 세션에서 수행, 결과는 스레드와 PR로.	이슈 조사, 소규모 수정, 상태 질의.	채널, 권한은 조직 설정과 연동 (Ch.3).

멘션
발화 방식

스레드 회신
결과 채널

경량 작업
적합 범위

slack 문서
설정 상세

--bg와 --exec

터미널을 돌려받는 백그라운드

background jobs

```
# Claude 세션을 백그라운드로
$ claude --bg "flaky 테스트 원인 조사"
#  세션 ID 반환, 터미널 즉시 복귀

# Agent 관리하면으로 접속
$ claude agents
```

즉시 복귀

ID 반환

Part 4 Recap

스케줄과 자동 실행 핵심 정리

KEY TAKEAWAY

요구를 다섯 표면에 매핑하세요.

세션 안은 loop와 goal, 기계 없는 정기 실행은 Routines, 완전 통제는 cron, 클릭 진입은 deep links입니다.

핵심 정리

- 표면 5종 지도가 선택의 전부
- loop 반복 vs goal 도달
- Routines 3 트리거, 무기계
- deep links로 런북 임베드
- cron은 PATH와 인증 지속성

다음 파트

- Part 5 CI/CD 통합
- 비대화 3원칙
- GitHub Actions, GitLab
- 시크릿과 비용 통제
- PR 자동 처리 완성

CHAPTER 05 / PART 05

CI/CD 통합

이벤트가 Claude를 부른다

- 비대화 3원칙과 CI 인증
- GitHub Actions, GitLab CI
- 시크릿과 비용 통제
- PR 자동 처리 완성 예시

비대화 3원칙

CI 속 Claude의 헌법

THREE PRINCIPLES

묻지 않고(-p와 허용 목록), 넘치지 않고(상한), 흔적을 남긴다(JSON과 아티팩트).
이 셋이 모든 CI 통합의 공통 골격입니다.

P1
묻지 않는다

-p + --allowed-tools 명시, 확인 프롬프트 원천 제거.

P2
넘치지 않는다

--max-turns, --max-budget-usd, 모델 하향 기본.

P3
흔적을 남긴다

--output-format json 저장, 아티팩트 업로드.

BASE
환경 전제

런너에 설치 + 인증, 그리고 체크아웃 깊이.

3 원칙
공통 골격

allowlist=능력
dontAsk 철학 (Ch.4)

상한 필수
무인 규율 (P2)

JSON 보존
감사, 디버깅 재료

CI 인증 전략

파이프라인에는 어떤 자격을

구독 조직	claude setup-token으로 장기 토큰 발급	시크릿 저장소 보관
API 조직	ANTHROPIC_API_KEY 시크릿	Console 발급 키
AWS 표준 (권장)	OIDC로 역할 인수 + Bedrock	장기 시크릿 0 (Ch.3)
게이트웨이 조직	BASE_URL + 서비스 자격	Ch.3 Part 3
공통 원칙	잡 권한 최소화, 키는 마스킹 로그	포크 PR 실행 주의

GitHub Actions 기본 골격

설치, 인증, 실행, 게이트

.github/workflows/audit.yml

```
jobs:
  deps-audit:
    runs-on: ubuntu-latest
    permissions: { id-token: write, contents: read }
    steps:
      - uses: actions/checkout@v4
      - uses: aws-actions/configure-aws-credentials@v4
        with: { role-to-assume: ${{ vars.CLAUDE_ROLE }},
                aws-region: ap-northeast-2 }
      - run: curl -fsSL https://claude.ai/install.sh | bash
      - run: CLAUDE_CODE_USE_BEDROCK=1 \
                claude –p ”심각도를 조사해서..”
```

OIDC 인수
id-token: write

설치 1줄
install.sh

스크립트 재사용
Part 2 산출물

exit 게이트
잡 성패 직결

GHA / PR 리뷰 잡

diff를 읽고 판정을 남긴다

pr-review job (steps 발췌)

```
on: { pull_request: { types: [opened, synchronize] } }
# ... checkout(fetch-depth: 0), 인증, 설치 생략 ...
- name: Review
  run: |
    git diff origin/${{ github.base_ref }}...HEAD > pr.diff
    claude -p "pr.diff를 리뷰해 심각도별로 정리" \
      --allowed-tools "Read" "Grep" "Bash(git diff *)" \
      --max-turns 12 --max-budget-usd 1.50 \
      --json-schema "$(cat .ci/review-schema.json)" \
      > review.json
    jq -e '.block == false' review.json  # 게이트
# 공식 claude-code-action, /code-review도 병행 선택지
```

diff 입력	읽기 전용 도구	schema 게이트	공식 액션
리뷰 재료	리뷰 잡 권한	jq -e 판정	관리형 대안

GitLab CI

같은 원칙, 다른 문법

.gitlab-ci.yml

```
claude-review:
  stage: test
  image: ubuntu:24.04
  rules:
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
  before_script:
    - apt-get update && apt-get install -y curl git jq
    - curl -fsSL https://claude.ai/install.sh | bash
    - export PATH="$HOME/.local/bin:$PATH"
  script:
    - ./scripts/mr-review.sh   # -p + 상한 + 스키마 동일
  artifacts: { paths: [review.json], when: always }
```

MR rules
이벤트 트리거

동일 스크립트
원칙 재사용

artifacts
흔적 보존

ID 토큰
OIDC 연동 가능

기타 CI 시스템

Jenkins와 그 밖의 요약

Jenkins	sh 스텝에서 동일 스크립트 호출	Credentials 바인딩
CircleCI, Buildkite	컨테이너 + install.sh 패턴 동일	잡 캐시로 설치 가속
표준 이미지	Ch.3의 조직 이미지에 사전 설치	설치 단계 자체 제거
자가 러너	인증 헬퍼, 프록시는 Ch.3 그대로	네트워크 파트 재활용
선택 기준	어디든 3원칙 + exit 게이트면 동작	플랫폼은 부차 변수

시크릿 관리

파이프라인 자격의 위생

SECRETS HYGIENE

1순위는 시크릿을 없애는 OIDC, 불가피한 토큰은 시크릿 저장소와 마스킹, 그리고 회전 대장입니다.
로그와 프롬프트에 값이 새지 않게 설계합니다.

RULE 1

OIDC 우선

역할 인수로 장기 시크릿 자체를 제거.

RULE 2

저장소와 마스킹

불가피한 토큰은 Secrets + 로그 마스킹 확인.

RULE 3

노출면 축소

프롬프트, 결과 JSON에 시크릿 미포함 설계.

RULE 4

포크 방어

포크 PR에는 시크릿 미주입 정책 유지.

0 secrets
OIDC 이상향

마스킹 검증
로그 점검 항목

프롬프트 위생
유출면 관리

Ch.3 연결
회전, 볼트 체계

CI 비용 통제

파이프라인 과금의 다이얼

호출 상한	--max-budget-usd, --max-turns 필수	Part 2 규율
모델 하향	리뷰, 분류는 sonnet, haiku 기본	opus는 선별 잡만
발동 조건	paths 필터, 라벨 조건으로 잡 자체 축소	docs 변경은 스킵
캐시	--exclude-dynamic... + 설치 캐시	다사용자 적중
관측	review.json의 cost 집계 대시보드	월간 파이프라인 회계
조직 안전망	gateway 한도, Budgets 알람	Ch.3 Part 3, 6

--init 준비 훅

CI 환경 셋업의 표준화

setup in CI

```
# .claude/settings.json (Ch.4 Setup 훅)
{ "hooks": { "Setup": [
  { "matcher": "init",
    "hooks": [{ "type": "command",
      "command": "npm ci && cp .env.ci .env" }] } ] } }

# CI 스텝
- run: claude -p --init "테스트 실패를 조사해 수정" ...
#  -p 실행 전에 init 매처 Setup 훅이 선행

# 준비만 따로: claude --init-only (검증 스텝 분리)
```

Setup + init
준비 훅 조합

레포에 동봉
셋업의 형상화

--init-only
준비 검증 분리

--maintenance
주기 정비 매처

실패 처리와 아티팩트

빨간 잡을 읽을 수 있게

SAVE

항상 보존

결과 JSON, stderr 로그를 성패 무관 아티팩트로.

CLASSIFY

실패 분류

게이트 실패 vs 실행 오류 vs 상한 도달 구분 출력.

SURFACE

표면화

요약을 PR 코멘트, 잡 서머리에 게시.

RERUN

재실행 설계

일시 오류는 재시도 함수가 이미 흡수 (P2).

when: always
보존 규칙

3 분류
실패 해석 축

코멘트 게시
개발자 접점

run_claude
재시도 재사용

완성 예시 / PR 자동 처리

리뷰, 게이트, 코멘트의 한 잡

pr-review.sh (CI 완성판)

```
#!/usr/bin/env bash
set -uo pipefail
git diff "origin/$BASE...HEAD" > pr.diff
OUT=$(run_claude "pr.diff 리뷰: 심각도, 사유, 차단 여부" \
  --json-schema "$(cat .ci/review-schema.json)") || exit 1
echo "$OUT" > review.json
jq -r '"### Claude Review\n" + .summary' review.json \
  > comment.md
gh pr comment "$PR" --body-file comment.md
jq -e '.block == false' review.json   # 최종 게이트
```

부품 총결합
P2 + P5 전부

gh comment
개발자 접점

jq -e
게이트 한 줄

Lab 3
실습 재료

Part 5 Recap

CI/CD 통합 핵심 정리

KEY TAKEAWAY

3원칙 위에 OIDC 인증과 비용 다이얼을 얹으면 어느 CI든 같은 스크립트가 돕니다.

흔적 보존과 코멘트 게시가 신뢰를 만듭니다.

핵심 정리

- 3원칙: 묻지 않고, 넘치지 않고, 남긴다
- 인증: OIDC > setup-token, 키
- 발동 조건 축소가 최대 절감
- --init으로 셋업 형상화
- 완성판: 리뷰 + 게이트 + 코멘트

다음 파트

- Part 6 자동화 패턴
- 이슈 트리아지
- 로그 분석과 일일 보고
- 문서 파이프
- 배치 마이그레이션

CHAPTER 05 / PART 06

자동화 패턴

현장 검증 5패턴

- 이슈 트리아지
- 로그 분석과 일일 보고서
- 문서 파이프라인
- 배치 마이그레이션과 운영 조합

Pattern 1 / 이슈 트리아지 설계

새 이슈를 사람 전에 분류

ISSUE TRIAGE

이슈 오픈 이벤트마다 본문을 분류해 라벨과 담당 팀, 우선순위를 제안합니다.
구조화 출력이 분류를, gh CLI가 액션을 담당합니다.

IN	SCHEMA	ACT	SAFE
입력	판정 스키마	액션	안전선
이슈 제목, 본문, 재현 정보 (gh issue view).	category enum, priority, team, duplicate_of.	라벨 부착, 팀 멘션 코멘트, 중복 링크.	닫기 금지, 제안까지만. 확정은 사람.

issues.opened	enum 분류	제안까지	haiku
트리거	흔들림 억제	권한 경계	비용 기본값

Pattern 1 / 스크립트

분류와 액션의 결합

triage.sh

```
#!/usr/bin/env bash
set -euo pipefail
N=$1
gh issue view "$N" --json title,body > issue.json
OUT=$(claude --bare -p "issue.json을 분류해" \
  --model haiku --max-turns 6 \
  --json-schema "$(cat .ci/triage-schema.json)") || exit 1
LABEL=$(echo "$OUT" | jq -r '.category')
TEAM=$(echo "$OUT" | jq -r '.team')
gh issue edit "$N" --add-label "$LABEL,$(echo "$OUT" | jq -r '.priority')"
echo "$OUT" | jq -r '.summary' | gh issue comment "$N" -F -
```

bare + haiku
경량 조합

gh 3연타
view, edit, comment

6턴 상한
분류 작업 한도

GHA on: issues
탑재 위치

Pattern 2 / 로그 분석 설계

대량 로그에서 사건을 추출

LOG ANALYSIS

수십만 줄 로그를 통째로 넣지 않습니다.

셸이 압축과 분할을, Claude가 해석과 상관을 맡는 분업이 비용과 품질을 동시에 지킵니다.

PRE	CHUNK	SCHEMA	MERGE
전처리	분할	판정	종합
grep, sort, uniq -c로 오류 시그니처 상위 추출.	시그니처, 시간창 단위로 나눠 개별 호출.	incident 여부, 근본 원인 가설, 심각도.	jq -s 집계 후 최종 1회 종합 호출.

셸 전처리	map-reduce	시그니처	2단 호출
토큰 절약 핵심	분할과 종합	분할 단위	개별 + 종합

Pattern 2 / 스크립트

전처리, 분할 판정, 종합

log-analyze.sh

```
grep -E 'ERROR|FATAL' app.log | \
  sed -E 's/[0-9a-f-]{36}/<id>/g' | sort | uniq -c | \
  sort -rn | head -20 > sigs.txt

while read -r line; do
  echo "$line" | claude --bare -p "이 오류 시그니처를 판정" \
    --model haiku --max-turns 4 \
    --json-schema "$(cat .ci/log-schema.json)"
done < sigs.txt | jq -s '.' > findings.json

claude -p "findings.json을 종합해 인시던트 보고 초안" \
  --max-turns 8 > incident-draft.md
```

uniq -c 상위
전처리 압축

행 단위 판정
개별 haiku 호출

slurp 종합
최종 1회 sonnet

초안 산출
사람 검토 전제

Pattern 3 / 일일 보고서 설계

아침마다 도착하는 저장소 브리핑

DAILY REPORT

어제의 커밋, PR, 이슈, CI 결과를 모아 팀 브리핑을 생성합니다.
수집은 gh와 git이, 서술은 Claude가, 배달은 Slack 웹훅이 맡습니다.

COLLECT 수집	WRITE 서술	DELIVER 배달	WHERE 표면
git log, gh pr list, gh issue list의 24시간 절단.	변경 요약, 리스크, 오늘 볼 것 3가지.	Markdown 저장 + Slack 웹훅 게시.	cron, Routines, Desktop 예약 어디든 (P4).

24h 절단	3 섹션	웹훅 배달	P4 표면
수집 범위	보고 규격	팀 접점	탑재 자유

Pattern 3 / 스크립트

수집, 서술, 배달 15줄

daily-report.sh

```
SINCE=$(date -d yesterday +%F)
{ git log --since="$SINCE" --oneline;
  gh pr list --state all --search "updated:>=$SINCE" \
    --json number,title,state;
  gh issue list --search "created:>=$SINCE" \
    --json number,title; } > digest.txt

claude -p "digest.txt로 팀 브리핑: 요약, 리스크, 오늘 볼 것 3" \
  --max-turns 8 --max-budget-usd 0.50 > report.md

curl -s -X POST "$SLACK_WEBHOOK" \
  -d "$(jq -n --rawfile t report.md '{text:$t}')"
```

digest 결합	0.5 달러 캡	웹훅 게시	07:30 cron
3소스 수집	일일 예산	배달 1줄	탑재 예시

Pattern 4 / 문서 파이프

생성과 번역의 배치화

docs-pipeline.sh

```
# 변경된 모듈만 문서 재생성
for m in $(git diff --name-only HEAD~1 | \
          grep '^src/' | cut -d/ -f2 | sort -u); do
  claude -p "src/$m 모듈의 API 문서를 docs/$m.md로 갱신" \
    --allowed-tools "Read" "Grep" "Write(./docs/**)" \
    --max-turns 10
done
# 한영 병행: 갱신분만 번역
for f in $(git diff --name-only -- docs/*.md); do
  claude --bare -p "기술 용어를 보존해 영어로 번역" \
    < "$f" > "docs/en/$(basename $f)"
done
```

변경분만
diff 기반 절약

Write 경로 제한
docs/** 한정

용어 보존
번역 지침 고정

CI post-merge
탑재 위치

Pattern 5 / 배치 마이그레이션 설계

수백 파일의 점진 변환

BATCH MIGRATION

파일 단위로 변환, 검증, 커밋을 원자화합니다.
실패 파일은 건너뛰고 목록에 남겨, 배치가 멈추지 않고 사람이 나중에 개입합니다.

UNIT
원자 단위

파일 1개 = 변환 + 테스트 + 커밋 한 묶음.

VERIFY
즉시 검증

관련 테스트 실행, 실패 시 git checkout 원복.

SKIP
실패 격리

failed.txt에 기록하고 다음 파일로 진행.

RESUME
재개 가능

완료 목록 대조로 중단 지점부터 재시작.

원자 커밋
롤백 단위 확보

테스트 게이트
품질 관문

failed.txt
사람 개입 큐

-w 격리
본 브랜치 보호

Pattern 5 / 스크립트

변환, 검증, 원복의 루프

migrate-batch.sh

```
for f in $(cat targets.txt); do
  grep -qx "$f" done.txt 2>/dev/null && continue
  claude -p "$f 를 신규 ORM API로 마이그레이션" \
    --allowed-tools "Read" "Edit" "Bash(npm run test *)" \
    --max-turns 12 --max-budget-usd 0.80
  if npm run test -- --findRelatedTests "$f" >/dev/null; then
    git add -A && git commit -m "migrate: $f"
    echo "$f" >> done.txt
  else
    git checkout -- . && echo "$f" >> failed.txt
  fi
done
```

done 대조
재개 메커니즘

관련 테스트
빠른 검증

checkout 원복
실패 원자성

야간 cron
탑재 예시 (P4)

패턴 조합과 운영

다섯 패턴을 한 체계로

공통 골격	run_claude + 스키마 + exit 게이트	Part 2 부품 재사용
표면 배치	이벤트형은 CI, 정기형은 cron/Routines	Part 4, 5 지도
비용 대장	결과 JSON cost 월간 집계	패턴별 단가 파악
실패 큐	failed 목록, 알림의 사람 개입 루프	무인과 유인의 경계
점진 확장	제안만 -> 게이트 -> 액션 순 신뢰 축적	권한도 단계 상향

Part 6 Recap

자동화 패턴 핵심 정리

KEY TAKEAWAY

전처리는 셸, 판정은 스키마, 액션은 CLI 도구, 실패는 큐로.

다섯 패턴 모두 이 한 문장의 변주입니다.

핵심 정리

- 트리아지: enum 분류 + 제안까지
- 로그: 셸 압축 + map-reduce
- 보고서: 수집, 서술, 배달 분업
- 문서: diff 기반 갱신분만
- 배치: 원자 커밋 + 재개 목록

다음 파트

- Part 7 환경변수
- 분류 지도와 카탈로그
- 인증, 네트워크, 기능 스위치
- 점검 원라이너
- 보안 원칙

CHAPTER 05 / PART 07

환경변수

실행을 감싸는 배경의 사전

- 분류 지도 7칸
- 인증, 네트워크, 모델, 기능 스위치
- 관측과 디렉토리, 기록
- 점검 원라이너와 보안 원칙

분류 지도

환경변수의 일곱 칸

인증, 공급자	USE_BEDROCK, ANTHROPIC_API_KEY, BASE_URL	Ch.3 복습
네트워크	HTTPS_PROXY, NODE_EXTRA_CA_CERTS	Ch.3 Part 4
모델, 사고	ANTHROPIC_MODEL, MAX_THINKING_TOKENS	이 파트
기능 스위치	DISABLE_* 패밀리, SIMPLE, SAFE_MODE	이 파트
관측	ENABLE_TELEMETRY, OTEL_*	Ch.3 Part 6
디렉토리, 기록	PROJECT_DIR, SKIP_PROMPT_HISTORY	이 파트

인증과 공급자

우선순위와 함께 복습

auth env

```
export CLAUDE_CODE_USE_BEDROCK=1     # 강제 스위치 (최상위)
export AWS_REGION=ap-northeast-2
# export CLAUDE_CODE_USE_VERTEX=1 / USE_FOUNDRY=1

export ANTHROPIC_API_KEY=...         # 직결 경로 키
export ANTHROPIC_BASE_URL=https://claude-gw.corp.example
#  게이트웨이 경유 (Ch.3 Part 3)

export CLAUDE_CODE_API_KEY_HELPER_TTL_MS=300000
#  helper 갱신 주기 (Ch.3 볼트 통합)

# 서열: 플래그 > env > 설정 (모델 계열 공통)
```

USE_* 3종
공급자 강제

BASE_URL
게이트웨이 창구

TTL_MS
helper 주기

managed env
조직 배포 채널

네트워크

Ch.3 Part 4의 사전판

HTTPS_PROXY, HTTP_PROXY	프록시 경유 강제	조직 프로파일 배포
NO_PROXY	사내 예외 목록	MCP, 게이트웨이 도메인
NODE_EXTRA_CA_CERTS	사내 루트 CA 신뢰	TLS 검사 장비 뒤
금지	NODE_TLS_REJECT_UNAUTHORIZED=0	검증 무력화 금지
진단	curl -v로 프록시, TLS 층 분리	Ch.3 진단표

모델과 사고

지능 칸의 대표들

model env

```
export ANTHROPIC_MODEL=apac.anthropic.claude-sonnet-5-v1:0
#  기본 모델 고정 (플래그가 있으면 플래그 우선)

export MAX_THINKING_TOKENS=0
#  확장 사고 끄기 (Anthropic API 기준)
#  Fable 5는 사고를 끌 수 없는 예외
#  서드파티는 thinking 파라미터 생략 방식

# 짝이 되는 설정 키 (Ch.4)
#  alwaysThinkingEnabled, availableModels,
#  fallbackModel, effortLevel
```

MODEL 고정
배치 표준화

THINKING=0
사고 차단

Fable 예외
끌 수 없음

설정 짝
Ch.4 키들

관측

Ch.3 계측 블록의 재확인

telemetry env

```
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=otlp
export OTEL_LOGS_EXPORTER=otlp
export OTEL_EXPORTER_OTLP_PROTOCOL=grpc
export OTEL_EXPORTER_OTLP_ENDPOINT=http://otel-collector:4317
export OTEL_RESOURCE_ATTRIBUTES=department=platform

# 파이프라인 관점 포인트
#  훅 입력의 prompt_id가 OTel 이벤트와 상관 연결
#  CI 잡 태그를 RESOURCE_ATTRIBUTES에 추가하면
#  파이프라인별 비용 절단 가능
```

6변수 세트
계측 표준

prompt_id
훅 상관키 (Ch.4)

잡 태그
파이프라인 절단

managed 배포
Ch.3 Part 6

디렉토리와 기록

경로와 흔적의 변수들

CLAUDE_PROJECT_DIR	훅, MCP에 노출되는 프로젝트 루트	Ch.4 플레이스홀더
CLAUDE_PLUGIN_ROOT / DATA	플러그인 설치, 영속 데이터 경로	플러그인 훅
CLAUDE_CODE_SKIP_PROMPT_HISTORY	트랜스크립트 기록 끄기 (전 모드)	P3 저장 구조
CLAUDE_CODE_REMOTE	웹 환경에서 true	환경 분기
CLAUDE_CODE_BRIDGE_SESSION_ID	Remote Control 연결 중 세션 ID	원격 인지
CLAUDE_CODE_DEBUG_LOGS_DIR	디버그 로그 디렉토리	P8 진단

점검 원라이너

지금 무엇이 걸려 있는가

inspection

```
# Claude 관련 변수 일람
$ env | grep -iE 'claude|anthropic|otel|aws_region' | sort

# 유효 구성의 최종 확인은 언제나
$ claude   ->   /status
#  공급자, 모델, managed 소스, 샌드박스

# 충돌 시 서열 복기
#  managed 설정 > CLI 플래그 > env > 파일 설정
#  (env는 managed의 env 블록으로 배포되면 강제층)
```

grep 일람
1차 점검

/status
유효 구성 확정

서열 복기
충돌 해석

managed env
강제층 승격

보안 원칙과 Part 7 Recap

배경에 비밀을 남기지 않는다

ENV HYGIENE

비밀은 helper와 OIDC로, 표준은 managed env로, 개인 실험은 셸 세션 한정으로.
일곱 칸 지도와 이 세 문장이 파트의 전부입니다.

보안 원칙

- 키 실값의 rc 파일 저장 금지
- helper, OIDC가 비밀의 제자리 (Ch.3)
- settings env 블록에도 비밀 금지 (Ch.4)
- 로그, CI 출력의 마스킹 확인

핵심 정리

- 7칸 지도: 인증, 네트워크, 모델, 스위치, 관측, 경로, 기록
- SIMPLE, SAFE_MODE는 모드 표식
- 점검: grep 일람 + /status
- 사전은 공식 env-vars 문서

CHAPTER 05 / PART 08

디버깅

안 될 때 가르는 순서

- 진단 흐름 6단
- --debug 카테고리와 --debug-file
- --safe-mode 이분법
- 헤드리스 전용 이슈와 errors 레퍼런스

진단 흐름 6단

재현에서 원인 격리까지

DIAGNOSTIC FLOW

재현 조건을 고정하고, 로그 해상도를 올리고, 커스텀을 이분하는 순서입니다.
각 단이 용의자 절반을 지웁니다.

1-2
재현 + verbose
최소 재현 명령 고정, 턴 출력 확대

›

3
--debug 카테고리
api, hooks, mcp 선별 로그

›

4
doctor
환경, 설정 유효성 판정

›

5
--safe-mode
커스텀 원인 여부 이분

›

6
--bare / 격리
최소 기동, 새 디렉토리 재현

--debug 카테고리

필요한 로그만 켠다

debug logging

```
$ claude --debug "api,hooks" -p "..."
#  선택 카테고리만: api, hooks, mcp 등

$ claude --debug "!statsig,!file"
# ! 접두로 특정 카테고리 제외

$ claude --debug-file /tmp/claude-debug.log -p "..."
#  파일로 기록 (debug 자동 활성)
#  CLAUDE_CODE_DEBUG_LOGS_DIR 보다 우선

# CI 요령: 실패 시에만 debug-file을 아티팩트로
```

카테고리 선별
잡음 없는 로그

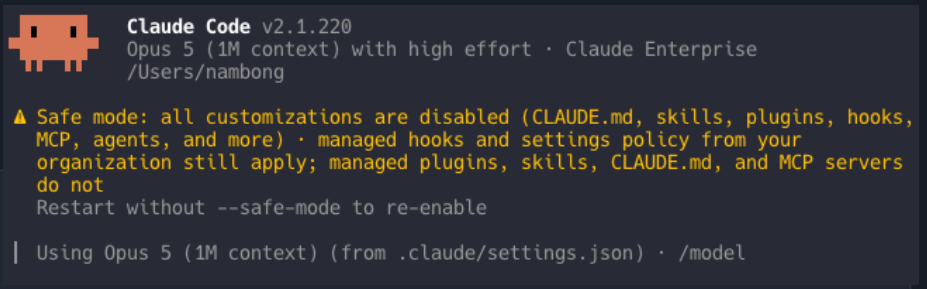
! 제외
부정 필터

debug-file
파일 기록

아티팩트화
CI 실패 첨부

--safe-mode 이분법

커스텀이 범인인가



SAFE-MODE BISECT

safe-mode에서 증상이 사라지면 범인은 커스텀입니다.

훅, 스킬, MCP, CLAUDE.md를 절반씩 복원하며 좁히면 몇 번 만에 특정됩니다.

safe-mode에서 정상

- 범인은 커스텀 계층
- disableAllHooks로 훅 먼저 가름
- --strict-mcp-config로 MCP 가름
- 커밋 이력으로 최근 변경 우선 의심

safe-mode에서도 재현

- 범인은 코어, 환경, 네트워크
- doctor와 네트워크 진단표 (Ch.3)
- 버전 회귀 의심 시 install stable
- errors 레퍼런스, 이슈 검색으로

doctor와 auth status

판정형 진단 명령

doctor / auth

```
$ claude doctor
#  설치, 설정 유효성, 무효 항목의 출처까지 표시
#  f 키로 자동 수정 제안 적용

$ claude auth status --text
#  로그인 상태, 계정, 조직
$ claude auth status; echo $?
#  0 로그인, 1 미로그인: 스크립트 사전 점검

# CI 첫 스텝 관용구
claude auth status || { echo 'auth missing'; exit 1; }
```

출처 표시
무효 항목 추적

자동 수정
f 키 제안

exit 0/1
인증 사전 점검

CI 관용구
첫 스텝 배치

헤드리스 전용 이슈

-p에서만 만나는 함정

조용히 오래 걸림	확인 대기 상태 (allow 누락)	allowed-tools 보강, dontAsk
결과가 잘림	stream 미수집, 파이프 버퍼	json 봉투로 수신
turns 초과 빈발	작업 대비 상한 과소	p95 재실측, 작업 분할
로컬 되고 CI 실패	설정 소스, env 차이	--setting-sources 고정
세션 못 찾음	디렉토리 다름 (ID는 프로젝트 한정)	같은 경로, 이름 재개
훅 미발화	bare 기동이 원인	bare 제거 또는 의도 확인

errors 레퍼런스와 도움 받기

혼자 넘긴 다음의 경로

GETTING HELP

런타임 오류 메시지는 공식 errors 레퍼런스에 의미와 처방이 정리되어 있습니다.
그래도 막히면 재현 번들과 함께 공식 채널로 갑니다.

ERRORS	BUNDLE	ISSUE	IN-APP
오류 사전	재현 번들	이슈 채널	/bug
docs/errors에서 메시지 검색, 의미와 수정.	버전, 명령, debug-file, doctor 출력 한 묶음.	github.com/anthropics/claude-code 이슈 검색, 등록.	세션 안에서 바로 리포트 제출.

errors 문서	번들 4종	이슈 검색 먼저	/bug
메시지 사전	재현 최소 세트	중복 확인	인앱 경로

CHAPTER 05 / PART 09

Recap & Labs

챕터 정리와 실습 4종

- Chapter 5 핵심 요약과 체크리스트
- FAQ
- 실습 랩 4종
- 실무 로드맵과 다음 챕터 예고

Chapter 5 핵심 요약

여덟 파트를 여섯 문장으로

FLAGS
플래그

지도 2 + 서랍 6, 무인은 상한, 진단은 safe-mode.

HEADLESS
-p

SDK 경유 단발 에이전트, 계약은 out, err, exit.

STRUCTURED
구조화

--json-schema로 파싱에서 계약으로.

SESSION
세션

-c, -r, --from-pr, fork, 웹 왕복.

SURFACE
표면 5

loop/goal, Desktop, Routines, CI, cron.

OPS
운영

3원칙, 비용 다이얼, 실패 큐, 진단 6단.

학습 목표 체크리스트

네 가지 목표 자가 점검

SELF CHECK

네 문장에 답할 수 있으면 목표 달성입니다.
막히면 표기된 파트로 복습하세요.

CHECK 1

플래그 정복

6서랍에서 조합 관용구 작성. (Part 1)

CHECK 2

파이프라인

스키마와 게이트로 스크립트 구성. (Part 2)

CHECK 3

세션 자유

from-pr, fork, teleport 사용. (Part 3)

CHECK 4

안전 자동화

표면 선택과 상한, 3원칙 적용. (Part 4-6)

4 Checks	Part 참조	Labs 4종	상한 습관
목표 점검	복습 경로	미완은 실습으로	무인 판정 기준

FAQ / 자주 나오는 질문

현장 질문 여섯 가지

-p에서도 훅이 도나요	예, 전 기능 동일 (bare만 예외)	Part 2, 8
왜 --max-turns가 필수인가요	기본 무제한, 무인 폭주 방지	Part 2
JSON을 프롬프트로 요청하면 안 되나요	--json-schema가 검증 보증	Part 2
resume이 세션을 못 찾아요	ID 검색은 프로젝트 한정	Part 3
CI 인증은 뭘로 하나요	OIDC 우선, 다음 setup-token	Part 5
--bare와 --safe-mode 차이는	가속 vs 진단, managed 적용 여부	Part 1

Lab 1 / 첫 Headless 호출

소요 15분

LAB 1 STEPS

```
cd claude-code-workshop/ch5/lab1

# 1. 세 출력 형식 비교
claude -p "이 저장소 구조를 세 줄로"
claude -p "..." --output-format json | jq .
claude -p "..." --output-format stream-json | head

# 2. 봉투에서 값 꺼내기
OUT=$(claude -p "..." --output-format json)
echo "$OUT" | jq -r '.result, .total_cost_usd'

# 3. exit code 체험: --max-turns 1로 초과 유발
```

15분	3 형식	jq 2필드	초과 체험
예상 소요	출력 비교	봉투 파싱	게이트 감각

Lab 2 / 자동화 스크립트

소요 25분

LAB 2 STEPS

```
# 1. 재시도 함수 완성 (제공 lib.sh 골격)
#    rate, overloaded 패턴만 3회 백오프

# 2. 구조화 판정
claude -p "git diff를 위험도 평가" \
  --json-schema "$(cat risk-schema.json)" > out.json

# 3. 게이트 연결
jq -e '.risk != "high"' out.json || exit 1

# 4. nightly-deps-audit.sh 완성 실행
#    reports/ 산출과 exit 게이트 확인
```

25분	lib.sh	jq -e	완성 실행
예상 소요	재시도 골격 제공	게이트 한 줄	P2 스크립트

Lab 3 / CI 통합

소요 25분

LAB 3 STEPS

```
# 1. 제공 워크플로 배치
cp ch5/gha-pr-review.yml .github/workflows/
#  OIDC 역할 변수 설정 (또는 setup-token 시크릿)

# 2. 테스트 PR 생성
git checkout -b lab3 && echo '// todo' >> src/app.ts
git commit -am 'lab3' && gh pr create --fill

# 3. 관찰 포인트
#  리뷰 코멘트 게시, review.json 아티팩트,
#  block=true 유발 후 잡 실패 게이트 확인
```

25분	제공 yml	코멘트 확인	게이트 실패
예상 소요	P5 완성판	접점 검증	의도 유발 체험

Lab 4 / 스케줄 표면 체험

소요 15분

LAB 4 STEPS

```
# 1. /loop 체험
claude   ->   /loop 린트 경고가 0이 될 때까지 수정

# 2. 세션 내 예약
> 5분 뒤에 진행 상황을 요약하라고 알려줘

# 3. 백그라운드와 회수
claude --bg "README 오탈자 정리"
claude logs <id> && claude attach <id>

# 4. (선택) crontab에 daily-report.sh 등록
```

15분	/loop	bg 왕복	cron 선택
예상 소요	조건 반복 체험	logs, attach	P4 레시피

실무 적용 로드맵

이번 분기의 자동화 여정

WEEK 1-2

개인 파이프

일일 보고서, 트리아지 1종 가동

›

WEEK 3-6

팀 CI

PR 리뷰 잡, 비용 대장 시작

›

WEEK 7-12

확장

배치 마이그레이션, Routines 이주

각 단계 관문: 상한 준수, 실패 큐 소화율, 월 비용 리뷰

1 패턴부터
작게 시작

비용 대장
신뢰의 근거

분기 1회
패턴 회고

Ch.3 연동
조직 관측 결합

Next / Chapter 6 예고

Agent SDK

CHAPTER 6 PREVIEW

-p가 경유하던 그 엔진을 라이브러리로 직접 씁니다.

TypeScript와 Python으로 커스텀 도구, 구조화 출력, 세션, 호스팅까지 프로덕션 에이전트를 만듭니다.

NEXT 1

SDK 기본

TS, Python 쿼리 루프와 스트리밍.

NEXT 2

커스텀 도구

인프로세스 MCP로 내 함수를 도구로.

NEXT 3

구조화, 세션

Zod, Pydantic 계약과 세션 영속.

NEXT 4

호스팅

컨테이너, 격리, 관측의 프로덕션 배포.

Ch.6	Day 3	라이브러리	To be continued
Agent SDK	워크샵 일정	시선 전환	다음 세션에서

References

본 챕터 공식 문서 출처 (code.claude.com)

CLI reference	docs/ko/cli-reference	Part 1 전집
Headless	docs/ko/headless, agent-sdk/structured-outputs	Part 2
Sessions	docs/ko/sessions, checkpointing, remote-control	Part 3
Scheduling	docs/ko/scheduled-tasks, goal, routines, deep-links	Part 4
CI	docs/ko/github-actions, gitlab-ci-cd	Part 5
Env, Debug	docs/ko/env-vars, errors, debug-your-config	Part 7, 8

Thank you!