

AUNOVA Obsidian 전문가 이론 및 지식 그물망 아키텍처

여러 바이브코딩 도구와 CLI를 연결하는 프로젝트 지식 운영체제

AUNOVA Harness Architecture

2026-07-21

Contents

AUNOVA Obsidian 전문가 이론 및 지식 그물망 아키텍처	4
목차	4
1. 핵심 결론	5
2. Obsidian을 선택하는 이유	6
2.1 로컬 폴더와 Markdown 기반	6
2.2 내부 링크와 역링크	6
2.3 구조화된 속성	6
2.4 Graph와 Canvas의 올바른 위치	6
3. 전문가 이론의 계보	8
3.1 Vannevar Bush: 연상 경로와 Memex	8
3.2 Douglas Engelbart: 인간·도구·언어·방법의 통합	8
3.3 Niklas Luhmann: 노트 저장소가 아니라 대화 상대	8
3.4 Andy Matuschak: Evergreen Notes	8
3.5 Tiago Forte: 실행 가능성 중심의 PARA와 점진적 압축	9
3.6 Nick Milo: MOC와 이질적 연결	9
3.7 인지적 오프로딩과 분산 인지	9
3.8 Pirolli와 Card: 정보 탐색과 Sensemaking의 두 고리	10
3.9 Personal Knowledge Graph	10
이론 계보 개념도	10
4. 폴더와 그물망은 서로 대체하지 않는다	11
4.1 폴더가 잘하는 것	11
4.2 링크가 잘하는 것	11
4.3 속성이 잘하는 것	11
4.4 MOC가 잘하는 것	11
4.5 AUNOVA 결론	11
5. AUNOVA 7계층 지식 그물망	12
계층 1. 물리 저장망	12
계층 2. 의미 연결망	12
계층 3. 탐색망	12
계층 4. 업무 상태망	12
계층 5. 근거·검증망	12
계층 6. 실행·에이전트망	12
계층 7. 동기화·보존망	13
7계층 그물망 개념도	13
6. Source of Truth 분리	14
7. Project-first와 Knowledge Vault의 결합	15
7.1 프로젝트는 실제 작업의 중심	15
7.2 Vault는 프로젝트 파일의 복제본이 아니다	15
7.3 지식은 프로젝트에서 추출된다	15

7.4 프로젝트와 지식의 양방향 관계	15
7.5 Canonical Project와 RAW Workspace Instance	15
7.6 여러 RAW 폴더의 통합은 Registry가 담당한다	16
7.7 PC별 Obsidian 경로와 논리 Root	16
7.8 NAS는 Cross-device Sync Hub다	16
7.9 업무시작·업무마무리가 시스템의 연결점이다	17
7.10 다중 Workspace의 쓰기 권한	17
8. JIT Context와 Context Compiler	18
8.1 모든 문서를 읽지 않는 이유	18
8.2 Context Compiler의 역할	18
입력	18
검색 우선순위	18
출력	18
8.3 점진적 문맥 압축	18
8.4 문맥 압축 시 반드시 보존할 것	18
9. 바이브코딩·Obsidian·CLI·Harness 역할 분리	20
9.1 바이브코딩 툴	20
9.2 Obsidian	20
9.3 CLI	20
9.4 Harness	21
10. 다중 CLI 협업과 토론 구조	22
10.1 Single Executor Mode	22
10.2 Planner-Executor-Reviewer Mode	22
10.3 Parallel Proposal Mode	22
10.4 Debate and Consensus Mode	22
10.5 원칙	22
다중 도구 실행 흐름	23
11. E·T·C·S·L·V 이론 모델	24
E — Environment	24
E의 경로 추상화 원칙	24
T — Tools	24
C — Context	24
S — State	24
L — Logs	25
V — Validation	25
12. 노트 유형과 메타데이터	26
12.1 핵심 노트 유형	26
12.2 공통 속성 예시	26
12.3 ID 원칙	26
13. 링크 문법과 관계 어휘	27
13.1 권장 관계	27
13.2 링크는 문맥 안에서 작성	27
13.3 속성 링크와 본문 링크의 구분	27
14. MOC와 탐색 구조	28
14.1 MOC의 세 종류	28
Home MOC	28
Project MOC	28
Workflow MOC	28
14.2 MOC는 자동 목록과 설명을 결합	28
14.3 MOC가 비대해지는 것을 방지	28
15. Inbox에서 재사용 지식까지	29
15.1 Inbox는 입출력 게이트	29
15.2 처리 상태	29
15.3 지식 생명주기	29

15.4 승격 기준	29
16. 세션·Handoff·충돌 제어	30
16.1 세션은 도구별로 분리	30
16.2 Handoff에 남길 내용	30
16.3 충돌 단위	30
16.4 동시 작업 예시	30
17. 안전·보안·동기화 원칙	31
17.1 최소 권한	31
17.2 민감정보	31
17.3 저장소 선택은 독립적	31
17.4 파괴적 작업	31
17.5 채널 장애 독립성과 상태 판정	31
18. 실패 패턴과 예방 규칙	32
실패 1. Vault를 프로젝트 백업 폴더로 사용	32
실패 2. Graph를 지식 품질로 착각	32
실패 3. 모든 노트를 원자화	32
실패 4. 모든 Skill과 모든 문서를 상시 로딩	32
실패 5. 세 CLI를 매번 동시에 호출	32
실패 6. 대화가 상태 저장소가 됨	32
실패 7. 경로 하드코딩	32
실패 8. 파일을 여러 저장소에서 동시 편집	32
19. 운영 품질 지표	33
19.1 문맥 효율	33
19.2 인계 품질	33
19.3 지식 재사용	33
19.4 충돌·안전	33
19.5 결과 품질	33
20. 단계적 도입 로드맵	34
단계 1. 관찰 가능한 최소 구조	34
단계 2. JIT Context	34
단계 3. CLI 연결	34
단계 4. 오케스트레이션	34
단계 5. 저장소 연동	34
단계 6. 학습과 개선	34
21. 최종 설계 원칙	35
원칙 1. 프로젝트가 먼저다	35
원칙 2. Vault는 창고가 아니라 네트워크다	35
원칙 3. 모든 기록을 지식으로 만들지 않는다	35
원칙 4. Context는 컴파일한다	35
원칙 5. 도구의 역할을 분리한다	35
원칙 6. 다중 CLI는 토론과 검증에 사용한다	35
원칙 7. 상태는 대화가 아니라 파일에 남긴다	35
원칙 8. 저장소마다 역할을 하나씩 부여한다	35
원칙 9. 자동화보다 안전 경계가 먼저다	35
원칙 10. 반복 오류는 프롬프트가 아니라 시스템을 고친다	35
22. 참고문헌 및 전문가 출처	36
부록 A. AUNOVA 노트 품질 체크리스트	36
프로젝트 노트	36
지식 노트	36
Workflow 노트	36
부록 B. 그물망 건강도 체크	36
부록 C. 본 문서와 후속 파일의 관계	36
부록 D. Multi-PC · Multi-RAW 핵심 용어	36

AUNOVA Obsidian 전문가 이론 및 지식 그물망 아키텍처

문서 목적

이 문서는 Obsidian을 단순 메모 애플리케이션이 아니라, 여러 프로젝트·바이브코딩 도구·CLI·저장소를 연결하는 **지식 및 작업 상태 운영체제**로 설계하기 위한 이론적 기준을 정리한다.

파일과 폴더의 실제 배치, 자동 설치, 스킴 구현, Codex·Claude Code·Antigravity용 하네스는 후속 문서에서 이 기준을 구현한다.

목차

1. 핵심 결론
2. Obsidian을 선택하는 이유
3. 전문가 이론의 계보
4. 폴더와 그물망은 서로 대체하지 않는다
5. AUNOVA 7계층 지식 그물망
6. Source of Truth 분리
7. Project-first와 Knowledge Vault의 결합
8. JIT Context와 Context Compiler
9. 바이브코딩·Obsidian·CLI·Harness 역할 분리
10. 다중 CLI 협업과 토론 구조
11. E·T·C·S·L·V 이론 모델
12. 노트 유형과 메타데이터
13. 링크 문법과 관계 어휘
14. MOC와 탐색 구조
15. Inbox에서 재사용 지식까지
16. 세션·Handoff·충돌 제어
17. 안전·보안·동기화 원칙
18. 실패 패턴과 예방 규칙
19. 운영 품질 지표
20. 단계적 도입 로드맵
21. 최종 설계 원칙
22. 참고문헌 및 전문가 출처

1. 핵심 결론

AUNOVA의 작업 환경에서는 한 프로젝트 안에서 사업계획서, 앱·웹 개발, 이미지·영상 제작, 발표자료, 자동화, 데이터 정리 등이 동시에 진행된다. 또한 Codex, Claude Code, Antigravity 등 여러 바이브코딩·CLI 도구를 교차 사용하고, 프로젝트 파일은 NAS·로컬 디스크·GitHub·Google Drive 등 서로 다른 위치에 존재할 수 있다.

따라서 Obsidian의 역할을 다음과 같이 제한하고 강화해야 한다.

Obsidian은 모든 프로젝트 파일을 복제하는 저장소가 아니다.
Obsidian은 프로젝트의 의미, 결정, 상태, 연결, 인계 정보를 관리한다.

AUNOVA 구조의 핵심은 다음 다섯 가지다.

1. 현재 활성 RAW Workspace가 이번 세션 작업 파일의 기준점이다.
2. Obsidian은 지식과 작업 상태의 기준점이다.
3. 바이브코딩 툴은 사용자와 대화하고 전체 작업을 지휘한다.
4. CLI는 좁은 범위의 실행을 빠르고 저렴하게 수행한다.
5. Harness는 필요한 맥락만 조립하고 도구 간 작업을 통제한다.

이를 한 문장으로 표현하면 다음과 같다.

프로젝트는 일을 담고, Obsidian은 일을 이해하며, CLI는 일을 실행하고, 바이브코딩 툴은 일을 지휘한다.

2. Obsidian을 선택하는 이유

2.1 로컬 폴더와 Markdown 기반

Obsidian의 Vault는 로컬 파일 시스템의 폴더이며, 기존 폴더를 Vault로 열 수도 있다.¹ 이는 프로젝트 경로가 NAS인지 로컬 디스크인지와 무관하게, 사용자가 자신의 저장 위치를 통제할 수 있다는 뜻이다.

AUNOVA 관점에서 이 특성은 다음 장점을 갖는다.

- 특정 SaaS의 데이터베이스 구조에 종속되지 않는다.
- Markdown 파일을 CLI·스크립트·Git이 직접 읽고 수정할 수 있다.
- Obsidian이 실행되지 않아도 파일 자체는 사용할 수 있다.
- 프로젝트별 절대경로를 설정 파일로 관리할 수 있다.
- 여러 바이브코딩 도구가 동일한 지식 파일을 참조할 수 있다.

2.2 내부 링크와 역링크

Obsidian은 내부 링크를 통해 노트와 파일을 연결하여 지식 네트워크를 만들 수 있다.² Backlinks는 현재 노트를 참조하는 다른 노트를 역방향으로 보여준다.³

이 기능의 실무적 의미는 단순한 “관련 문서 링크”보다 크다.

예를 들어 로그인 기능 요구사항 노트가 다음 노트에 연결될 수 있다.

```
[[사용자 페르소나]]
[[제품 요구사항]]
[[앱 개발 결정 기록]]
[[로그인 화면 디자인]]
[[보안 검증 체크리스트]]
[[IR 데모 시나리오]]
```

이때 로그인 기능은 개발 폴더 하나에 갇히지 않고, 제품·디자인·보안·발표라는 여러 맥락에서 다시 사용된다.

2.3 구조화된 속성

Obsidian Properties는 텍스트, 링크, 날짜, 체크박스, 숫자 같은 구조화 데이터를 YAML로 저장한다.⁴ 속성은 사람이 읽는 문서와 기계가 읽는 상태값을 한 파일 안에 공존시키는 핵심 장치다.

```
type: decision
project: mind-weather
workstream: development
status: accepted
owner: codex
created: 2026-07-21
related:
  - "[로그인 기능 요구사항]"
  - "[보안 검증 체크리스트]"
```

이렇게 하면 Obsidian은 메모장에 머물지 않고 하네스가 읽을 수 있는 **상태 저장소**가 된다.

2.4 Graph와 Canvas의 올바른 위치

Graph View는 노트를 노드로, 내부 링크를 연결선으로 시각화한다.⁵ 그러나 그래프가 예쁘게 보이는 것과 지식 구조가 잘 작동하는 것은 다르다.

Graph는 다음 용도로 제한해야 한다.

- 고립된 노트 발견
- 특정 프로젝트의 연결 밀도 확인
- 중요한 허브 노트 식별
- 예상하지 못한 연결 탐색

¹Obsidian Help, **Create a vault**. <https://obsidian.md/help/Getting%20started/Create%20a%20vault>

²Obsidian Help, **Internal links**. <https://obsidian.md/help/links>

³Obsidian Help, **Backlinks**. <https://obsidian.md/help/plugins/backlinks>

⁴Obsidian Help, **Properties**. <https://obsidian.md/help/properties>

⁵Obsidian Help, **Graph view**. <https://obsidian.md/help/plugins/graph>

Canvas는 노트·첨부파일·웹페이지를 2차원 공간에 배치하고 선으로 연결하는 시각적 사고 도구다.⁶ AUNOVA에서는 다음에 적합하다.

- 서비스 구조도
- 사용자 여정
- 사업모델 관계도
- 영상 스토리보드
- 앱 화면 흐름
- 다중 에이전트 작업 분해

Graph는 전체 관계의 진단 도구이고, Canvas는 특정 문제의 시각적 작업판이다.

⁶Obsidian Help, **Canvas**. <https://obsidian.md/help/plugins/canvas>

3. 전문가 이론의 계보

3.1 Vannevar Bush: 연상 경로와 Memex

Vannevar Bush는 1945년 「As We May Think」에서 정보를 단순 분류표가 아니라 **연상 경로(associative trails)**로 연결하는 Memex를 제안했다.⁷

핵심은 정보를 많이 저장하는 것이 아니라, 두 항목을 연결하고 그 연결 경로를 다시 따라갈 수 있게 하는 것이다.

AUNOVA에 적용하면 다음과 같다.

공고문

- 평가항목
- 사업계획서 근거
- 기능 요구사항
- 데모 화면
- 발표 대본
- 심사 피드백

이 연결은 폴더 경로만으로는 표현하기 어렵다. 하나의 공고문이 사업계획서, 앱, 영상, 발표자료에 동시에 영향을 주기 때문이다.

3.2 Douglas Engelbart: 인간·도구·언어·방법의 통합

Douglas Engelbart는 1962년 「Augmenting Human Intellect」에서 복잡한 문제 해결 능력을 높이기 위해 인간, 인공물, 언어, 방법론을 하나의 상호작용 시스템으로 보았다.⁸

AUNOVA 하네스도 같은 관점이 필요하다.

사람의 판단

- + 바이브코딩 인터페이스
- + Obsidian 지식 구조
- + CLI 실행 능력
- + Harness 운영 방법
- = 증강된 프로젝트 수행 능력

중요한 점은 최신 AI 모델 하나를 추가하는 것만으로 성능이 완성되지 않는다는 것이다. 폴더 구조, 문맥 전달, 검증, 역할 분담이 함께 개선되어야 복합 효과가 발생한다.

3.3 Niklas Luhmann: 노트 저장소가 아니라 대화 상대

Niklas Luhmann은 자신의 카드함을 단순 기록 보관소가 아니라 자신과 상호작용하는 별도의 체계로 설명했다.⁹ 이후 연구는 Luhmann의 카드 인덱스를 사고 도구, 의사소통 상대, 출판 생산 장치로 분석했다.¹⁰

이 관점에서 좋은 Obsidian Vault는 다음 조건을 갖는다.

- 과거 생각을 그대로 쌓기만 하지 않는다.
- 새 정보가 기존 개념과 충돌하거나 연결된다.
- 예상하지 못한 관련 노트가 다시 나타난다.
- 프로젝트가 끝난 뒤에도 핵심 지식이 재사용된다.
- 노트가 최종 산출물의 부품으로 작동한다.

3.4 Andy Matuschak: Evergreen Notes

Andy Matuschak은 Evergreen Notes를 프로젝트를 넘어 장기간 발전하고 축적되는 노트로 설명한다. 핵심 원칙은 원자성, 개념 중심, 높은 연결 밀도, 계층보다 연상 관계를 선호하는 것이다.¹¹

AUNOVA에서는 모든 기록을 Evergreen으로 만들지 않는다.

⁷Bush, V. (1945). **As We May Think**. The Atlantic. <https://www.theatlantic.com/magazine/archive/1945/07/as-we-may-think/303881/>

⁸Engelbart, D. C. (1962). **Augmenting Human Intellect: A Conceptual Framework**. Stanford Research Institute. https://www.dougelbart.org/pubs/papers/scanned/Doug_Engelbart-AugmentingHumanIntellect.pdf

⁹Luhmann, N. (1981). **Communicating with Slip Boxes: An Empirical Account**. English translation. <https://luhmann.ir/wp-content/uploads/2021/07/Communicating-with-Slip-Boxes.pdf>

¹⁰Schmidt, J. F. K. (2018). **Niklas Luhmann's Card Index: The Fabrication of Serendipity**. Sociologica, 12(1), 53-60. https://docdrop.org/download_annotation_doc/Niklas-Luhmanns-Card-Index_-Th---Schmidt-Johannes-F.K_-0rcv9.pdf

¹¹Matuschak, A. **Evergreen notes**. <https://notes.andymatuschak.org/z5E5QawiXCMbtNtupvxexoEX>

일시적 정보 → Inbox 또는 Session Log
프로젝트 특화 정보 → Project Context
반복 사용되는 개념 → Evergreen Knowledge
탐색 경로 → MOC

이 부분이 없으면 Vault가 다시 거대한 자료 창고가 된다.

3.5 Tiago Forte: 실행 가능성 중심의 PARA와 점진적 압축

Tiago Forte의 PARA는 정보를 Projects, Areas, Resources, Archives로 나누고, 주제보다 현재의 실행 가능성을 중심으로 구성한다.¹² Progressive Summarization은 자료를 처음부터 완벽히 요약하기보다, 반복 사용하면서 중요한 부분을 단계적으로 드러내는 방식이다.¹³

AUNOVA에는 PARA를 그대로 복제하지 않고 다음처럼 변형한다.

Projects → 실제 프로젝트 인덱스와 현재 상태
Areas → 지속 운영 영역과 공통 역량
Resources → 재사용 지식·워크플로·참고자료
Archives → 종료 프로젝트와 폐기된 결정

점진적 압축은 Context Compiler와 직접 연결된다.

원문 전체
→ 관련 구간
→ 핵심 근거
→ 현재 작업용 요약
→ CLI Task Packet

3.6 Nick Milo: MOC와 이질적 연결

MOC(Map of Content)는 노트를 단순 목록으로 모으는 것이 아니라, 특정 관점에서 연결하고 탐색하는 지도다.¹⁴ 같은 노트가 여러 MOC에 속할 수 있으므로 단일 계층 구조보다 복합 프로젝트에 적합하다.

예를 들어 브랜드 캐릭터 노트는 다음 세 지도에 동시에 포함될 수 있다.

[[MindWeather Product MOC]]
[[Media Production MOC]]
[[IR Storytelling MOC]]

이는 한 파일을 여러 폴더에 복사하지 않고도 여러 업무 맥락에서 사용할 수 있게 한다.

3.7 인지적 오프로딩과 분산 인지

Risko와 Gilbert는 인지적 오프로딩을 과업의 정보처리 요구를 줄이기 위해 외부 행동이나 도구를 사용하는 것으로 설명한다.¹⁵

Obsidian의 목적은 기억을 대신하는 데 그치지 않는다. 다음을 외부화해야 한다.

- 어떤 결정을 왜 내렸는가
- 현재 무엇이 완료되었는가
- 어떤 위험이 남아 있는가
- 어떤 자료가 근거인가
- 다음 도구가 무엇을 이어서 해야 하는가

여러 CLI가 협업할 때도 개별 모델의 기억에 의존하지 않고, 공통 상태·결정·검증 결과를 외부 문서로 유지해야 한다.

¹²Forte, T. (updated 2026). **The PARA Method: The Simple System for Organizing Your Digital Life in Seconds**. <https://fortelabs.com/blog/para/>

¹³Forte, T. **Progressive Summarization: A Practical Technique for Designing Discoverable Notes**. <https://fortelabs.com/blog/progressive-summarization-a-practical-technique-for-designing-discoverable-notes/>

¹⁴Milo, N. **MOCs Overview / Linking Your Thinking**. <https://notes.linkingyourthinking.com/Cards/MOCs%2BOverview>

¹⁵Risko, E. F., & Gilbert, S. J. (2016). **Cognitive Offloading**. Trends in Cognitive Sciences, 20(9), 676-688. <https://pubmed.ncbi.nlm.nih.gov/27542527/>

3.8 Pirolli와 Card: 정보 탐색과 Sensemaking의 두 고리

Pirolli와 Card는 분석 작업을 자료를 찾고 좁히는 **Foraging Loop**와, 자료를 재구성해 해석과 결과물을 만드는 **Sense-making Loop**로 설명한다.¹⁶

AUNOVA 작업 흐름에 대응시키면 다음과 같다.

Foraging Loop

외부 자료 확인 → Inbox 수집 → 검색 → 필터 → 근거 추출

Sensemaking Loop

연결 → 구조화 → 가설 → 결정 → 제작 → 검증 → 결과물

이 구분은 바이브코딩과 CLI의 역할 배분에도 유용하다.

- 파일 목록, 검색, 비교, 추출은 스크립트와 CLI에 맡긴다.
- 목표 해석, 구조 결정, 상충 의견 조정은 바이브코딩 오케스트레이터가 맡는다.

3.9 Personal Knowledge Graph

개인 지식 그래프는 개인이 소유하는 엔터티·속성·관계의 구조화된 자원으로 볼 수 있다.¹⁷ Obsidian은 완전한 데이터베이스형 Knowledge Graph는 아니지만, 다음 요소를 결합하면 실용적인 개인 지식 그래프 역할을 수행할 수 있다.

노트	= Node
내부 링크	= Edge
속성	= Node Metadata
MOC	= Curated Subgraph
검색·Bases	= Query Layer
Harness	= Retrieval and Action Layer

이론 계보 개념도

flowchart LR

```
B[1945 Vannevar Bush<br/>Associative Trails]
E[1962 Douglas Engelbart<br/>Human-Tool-Method System]
L[1981 Niklas Luhmann<br/>Slip-box as Communication Partner]
M[Evergreen Notes<br/>Atomic & Densely Linked]
F[PARA & Progressive Summarization<br/>Actionability & Distillation]
O[MOC / Heterarchy<br/>Multiple Navigation Paths]
A[AUNOVA Obsidian Harness<br/>Knowledge + State + Orchestration]
```

```
B --> E
E --> L
L --> M
L --> F
M --> O
F --> O
O --> A
```

¹⁶Pirolli, P., & Card, S. (2005). **The Sensemaking Process and Leverage Points for Analyst Technology as Identified Through Cognitive Task Analysis**. <https://andymatuschak.org/files/papers/Pirolli%2C%20Card%20-%202005%20-%20The%20sensemaking%20process%20and%20leverage%20points%20for%20analyst%20technology%20as.pdf>

¹⁷Skjæveland, M. G., Balog, K., Bernard, N., Łajewska, W., & Linjordet, T. (2023). **An Ecosystem for Personal Knowledge Graphs: A Survey and Research Roadmap**. <https://arxiv.org/abs/2304.09572>

4. 폴더와 그물망은 서로 대체하지 않는다

4.1 폴더가 잘하는 것

폴더는 다음에 강하다.

- 실제 파일 위치
- 접근 권한
- 백업 범위
- 삭제와 보관
- 프로그램의 상대경로
- 대용량 자산 구분

4.2 링크가 잘하는 것

링크는 다음에 강하다.

- 여러 맥락에 동시에 포함
- 원인과 결과 연결
- 근거와 결정 연결
- 요구사항과 구현 연결
- 프로젝트를 넘어서는 재사용

4.3 속성이 잘하는 것

속성은 다음에 강하다.

- 상태 필터링
- 자동 분류
- 날짜·도구·프로젝트 식별
- 하네스가 읽는 기계 친화적 정보
- Bases·검색·스크립트의 입력

4.4 MOC가 잘하는 것

MOC는 다음에 강하다.

- 사람이 이해할 수 있는 탐색 경로
- 우선순위와 설명이 있는 큐레이션
- 여러 폴더를 가로지르는 프로젝트 지도
- 신규 사용자·신규 에이전트 온보딩

4.5 AUNOVA 결론

폴더만 사용하면 관계가 사라진다.

링크만 사용하면 위치와 운영 경계가 흐려진다.

속성만 사용하면 의미와 설명이 빈약해진다.

그래프만 사용하면 시각화가 목적이 된다.

따라서 다음 조합을 사용한다.

폴더 = 물리적 경계

링크 = 의미적 관계

속성 = 기계 판독 상태

MOC = 사람과 에이전트의 탐색 지도

Harness = 필요한 부분만 실행으로 연결

5. AUNOVA 7계층 지식 그물망

AUNOVA의 “그물망”은 단순한 Obsidian Graph가 아니라, 파일·지식·업무·에이전트·저장소를 연결하는 7계층 구조다.

계층 1. 물리 저장망

프로젝트 로컬 폴더
NAS
GitHub 저장소
Google Drive
Obsidian Vault

목적은 파일의 실제 위치와 권한을 명확하게 하는 것이다.

계층 2. 의미 연결망

문제 ↔ 고객
고객 ↔ 요구사항
요구사항 ↔ 구현
구현 ↔ 검증
검증 ↔ 결과

Obsidian 내부 링크와 관계 속성으로 표현한다.

계층 3. 탐색망

Home MOC
Project MOC
Workstream MOC
Knowledge MOC
Workflow MOC

사람과 에이전트가 “어디서부터 읽을지” 결정한다.

계층 4. 업무 상태망

Task
Decision
Risk
Dependency
Blocker
Next Action

현재 프로젝트가 어디까지 진행되었는지 나타낸다.

계층 5. 근거·검증망

Source
Evidence
Assumption
Experiment
Validation
Outcome

결과와 산출물이 무엇을 근거로 만들어졌는지 추적한다.

계층 6. 실행·에이전트망

바이브코딩 오케스트레이터
Codex CLI
Claude Code CLI
Antigravity CLI
스크립트
MCP

Skill

작업 유형에 따라 실행자를 배분하고 결과를 비교한다.

계층 7. 동기화·보존망

Git Commit
Drive Deliverable
NAS Backup
Archive
Session Handoff

결과가 사라지거나 서로 덮어쓰지 않게 한다.

7계층 그물망 개념도

flowchart TB

L7[7. 동기화·보존망
GitHub · Drive · NAS · Archive]
L6[6. 실행·에이전트망
Vibe Tool · Codex · Claude · Antigravity]
L5[5. 근거·검증망
Source · Evidence · Validation]
L4[4. 업무 상태망
Task · Decision · Risk · Handoff]
L3[3. 탐색망
Home · Project MOC · Workflow MOC]
L2[2. 의미 연결망
Internal Links · Relations]
L1[1. 물리 저장망
Vault · Project Folder · NAS · Git]

L7 --> L6 --> L5 --> L4 --> L3 --> L2 --> L1
L1 --> L2 --> L3 --> L4 --> L5 --> L6 --> L7

이 구조는 위에서 아래로 명령이 전달되고, 아래에서 위로 파일과 근거가 회수되는 양방향 시스템이다.

6. Source of Truth 분리

한 파일을 GitHub, Drive, NAS, Obsidian에 모두 복사하면 어느 것이 최신인지 알기 어렵다. 각 저장소의 기준 역할을 명확하게 분리해야 한다.

대상	Source of Truth	주요 내용
프로젝트 작업 파일	선택한 실제 프로젝트 폴더	코드, 원본 문서, 이미지, 영상, 설정
코드 이력	GitHub 또는 로컬 Git	커밋, 브랜치, PR, Issue
프로젝트 의미·결정	Obsidian	개요, 결정, 상태, 위험, 인계
공유 문서·제출물	Google Drive	PPT, PDF, DOCX, XLSX, 검토본
대용량 자산·백업	NAS 또는 외장 저장소	영상, 3D, 모델, 렌더, 전체 백업
현재 실행 상태	Session State	현재 도구, 범위, 잠금, 결과

핵심 규칙은 다음과 같다.

동일한 파일을 여러 저장소에서 동시에 편집하지 않는다.
각 저장소는 역할에 맞는 복사본 또는 기록만 갖는다.

7. Project-first와 Knowledge Vault의 결합

7.1 프로젝트는 실제 작업의 중심

각 프로젝트는 자체 구조와 파일을 유지한다. 이미 존재하는 프로젝트의 폴더를 Obsidian 기준에 맞추어 강제로 재배치하지 않는다.

Z:\HDD1\yongusb\ aunova\MindWeather
D:\Projects\ClientApp
C:\Users\User\Documents\VideoProject

하네스는 사용자가 선택한 폴더를 등록하고 역할을 매핑한다.

7.2 Vault는 프로젝트 파일의 복제본이 아니다

Vault에는 다음만 저장한다.

Project Overview
Current State
Decisions
Risks
Tasks
Handoffs
Workflow References
Reusable Knowledge

7.3 지식은 프로젝트에서 추출된다

프로젝트 자료
→ 작업 및 검증
→ 반복 사용 가능성 확인
→ 핵심 개념 추출
→ Evergreen Note
→ 관련 MOC 연결

이는 Vault가 처음부터 완벽한 백과사전이 되는 것을 방지한다.

7.4 프로젝트와 지식의 양방향 관계

Project → 검증된 지식 → Knowledge Vault
Knowledge Vault → 필요한 규칙 → 새 Project

프로젝트가 지식을 생산하고, 지식이 다음 프로젝트의 시작 비용을 낮춘다.

7.5 Canonical Project와 RAW Workspace Instance

AUNOVA에서는 하나의 프로젝트를 물리적 폴더 하나와 동일시하지 않는다.

Canonical Project
= 이름·목표·결정·상태를 공유하는 논리적 프로젝트

RAW Workspace Instance
= 특정 PC 또는 특정 바이브코딩 툴이 실제로 여는 작업 폴더

예를 들어 Mind Weather라는 하나의 Canonical Project가 다음 RAW Workspace를 가질 수 있다.

D:\CodexProjects\MindWeather
D:\AntigravityProjects\MindWeather
C:\ClaudeProjects\MindWeather
Z:\HDD1\yongusb\ aunova\MindWeather ← NAS Sync Hub

이 폴더들을 물리적으로 합치는 대신 다음 식별자로 논리 통합한다.

project_id / project_uuid
workspace_id
device_id
tool_id
local_path

모든 RAW Workspace에는 가능한 경우 다음 최소 식별 파일을 둔다.

```
.aunova/  
├─ project.yaml  
├─ workspace.yaml  
├─ sync-manifest.json  
├─ pending-sync.yaml  
├─ handoff/  
└─ validation/
```

7.6 여러 RAW 폴더의 통합은 Registry가 담당한다

여러 RAW 폴더를 한 상위 폴더로 옮기는 방식은 기존 바이브코딩 툴의 프로젝트 구조를 깨뜨릴 수 있다. 따라서 통합은 중앙 등록부가 담당한다.

Project Registry
→ Canonical Project의 논리 정보

Workspace Registry
→ 현재 PC에서 사용 가능한 RAW Workspace의 실제 경로

Device Registry
→ PC별 Vault 위치·환경·사용 가능한 저장소

프로젝트 공통 정보에는 절대경로를 저장하지 않고 project_id, workspace_id, 논리 참조만 저장한다. PC별 실제 경로는 로컬 장치 설정에서 해석한다.

7.7 PC별 Obsidian 경로와 논리 Root

Obsidian Vault의 실제 드라이브 문자는 PC마다 달라도 된다.

메인 데스크톱: D:\AUNOVA_AI_OS
다른 PC: C:\AUNOVA_AI_OS
하네스 표현: \${AUNOVA_AI_OS_ROOT}

Vault 내부 링크는 상대경로로 작성한다.

[\[\[10_Project_Index/MindWeather/PROJECT_OVERVIEW\]\]](#)

다음과 같은 PC 고정 절대경로 링크는 지식 노트에 직접 기록하지 않는다.

D:\AUNOVA_AI_OS\10_Project_Index\MindWeather

PC별 절대경로는 %USERPROFILE%\aunova\device.local.yaml 또는 동등한 로컬 설정에만 둔다. 이 파일은 Git이나 Vault 동기화의 기준 파일로 사용하지 않는다.

7.8 NAS는 Cross-device Sync Hub다

NAS는 작업 중인 모든 폴더를 실시간으로 공동 편집하는 위치가 아니라, PC 간 검증된 변경을 전달하는 교환 허브다.

업무시작
NAS Manifest·최근 Handoff 확인
→ 로컬 RAW와 비교
→ 사용자 승인
→ NAS 변경을 로컬 RAW로 Pull

업무마무리
로컬 RAW Validation
→ Obsidian 상태·Handoff 갱신
→ 검증된 변경을 NAS로 Push
→ NAS Manifest 갱신

한 채널의 장애가 전체 작업을 멈추게 하지 않는다. NAS가 오프라인이면 로컬 작업은 계속할 수 있고 pending-sync.yaml에 보류한다. 다만 이전 PC의 최신 변경이 아직 NAS에 반영되지 않았다는 신호가 있으면 다른 PC에서의 쓰기 작업은 충돌 위험을 사용자에게 명확히 알려야 한다.

7.9 업무시작·업무마무리가 시스템의 연결점이다

AUNOVA에서 업무시작과 업무마무리는 단순 인사 명령이 아니라 다음 시스템 경계를 실행하는 Skill이다.

업무시작

현재 열린 RAW 발견

- Project/Workspace 식별
- PC별 Vault 경로 해석
- Obsidian 최근 상태 확인
- NAS 비교·Pull
- Lock 생성
- Context Pack 생성

업무마무리

Validation

- CURRENT_STATE·DECISIONS·Handoff 갱신
- NAS Push·Manifest 갱신
- GitHub·Drive 독립 동기화
- 실패 채널 pending 기록
- Lock 해제

7.10 다중 Workspace의 쓰기 권한

동일 Canonical Project에 RAW Workspace가 여러 개 있어도 한 파일을 여러 Workspace가 동시에 쓰는 것은 허용하지 않는다.

- 기본 잠금 단위는 project 또는 workstream이다.
- 같은 프로젝트의 개발과 사업계획서처럼 범위가 완전히 다르면 Workstream Lock으로 병행할 수 있다.
- 동일 파일의 병행 수정은 Git Worktree, 별도 Draft 또는 읽기 전용 검토로 격리한다.
- 이미지·영상·3D 원본처럼 병합이 어려운 파일은 자동 병합하지 않는다.

8. JIT Context와 Context Compiler

8.1 모든 문서를 읽지 않는 이유

여러 프로젝트와 워크플로가 쌓이면 전체 Vault를 매번 읽는 것은 다음 문제를 만든다.

- 토큰 사용 증가
- 응답 지연
- 오래된 규칙과 현재 규칙의 충돌
- 관련 없는 정보로 인한 추론 품질 저하
- 민감한 자료의 불필요한 노출

8.2 Context Compiler의 역할

Context Compiler는 현재 요청에 필요한 정보만 조립한다.

입력

프로젝트 ID
작업 유형
사용자 목표
현재 상태
최근 결정
관련 파일 범위
선택한 CLI

검색 우선순위

1. PROJECT_OVERVIEW
2. CURRENT_STATE
3. DECISIONS
4. 최근 HANDOFF
5. 작업영역 CONTEXT
6. 관련 WORKFLOW
7. 필요한 근거 자료

출력

SESSION_CONTEXT.md
TASK_PACKET.yaml
FILE_SCOPE.txt
VALIDATION_PLAN.md

CLI는 이 네 파일과 지정된 프로젝트 파일만 읽는다.

8.3 점진적 문맥 압축

원본 100%
→ 검색 결과 20%
→ 관련 구간 8%
→ 작업 문맥 3%
→ CLI 지시 1%

비율은 절대 기준이 아니라 설계 목표다. 핵심은 “많이 읽는 것”이 아니라 “필요한 근거를 읽지 않고 작게 전달하는 것”이다.

8.4 문맥 압축 시 반드시 보존할 것

목표
제약
사용자 승인사항
결정 근거
파일 범위
완료 조건
검증 방법

삭제 가능한 것은 다음과 같다.

반복 대화
이미 폐기된 초안
전체 디버그 로그
관련 없는 프로젝트 배경
동일 내용의 중복 설명

9. 바이브코딩 · Obsidian · CLI · Harness 역할 분리

9.1 바이브코딩 툴

바이브코딩 툴은 사용자가 보는 오케스트레이션 인터페이스다.

담당:

- 사용자의 목적과 우선순위 확인
- 단계별 질문
- 작업 분해
- CLI 배정
- 여러 CLI 결과 비교
- 상충 의견 조정
- 사용자에게 계획과 결과 표시
- 승인 요청

하지 않을 일:

- 모든 프로젝트 파일을 무차별 로딩
- 단순 파일 검색에 고가의 추론 사용
- 사용자 승인 없는 삭제 · 배포 · Push

9.2 Obsidian

Obsidian은 공통 기억과 의미 구조다.

담당:

- 프로젝트 개요
- 현재 상태
- 결정과 근거
- 위험과 미해결 문제
- 재사용 지식
- 세부 워크플로
- 세션 인계

하지 않을 일:

- 프로젝트 원본 전체 복제
- 대용량 영상 · 모델 저장
- 코드 빌드와 테스트 직접 실행

9.3 CLI

CLI는 좁은 범위의 실행자다.

담당:

- 파일 탐색
- 코드 · 문서 수정
- 일괄 변환
- 테스트 · 빌드 · 검증
- Git 상태 확인
- 결과 요약

하지 않을 일:

- 전체 프로젝트 방향을 임의로 변경
- 다른 CLI의 작업영역 침범
- 승인 없이 원격 저장소에 강제 반영

9.4 Harness

Harness는 **작업 운영 규칙**이다.

담당:

- E·T·C·S·L·V 상태 확인
- 현재 Stage와 Route 결정
- 필요한 Skill만 로딩
- Context Compiler 실행
- CLI 호출과 결과 회수
- Validation 강제
- Handoff 기록

10. 다중 CLI 협업과 토론 구조

AUNOVA는 Codex CLI, Claude Code CLI, Antigravity CLI를 모두 설치하고, 작업 중인 바이브코딩 툴이 세 CLI를 오케스트레이션하는 구조를 목표로 한다.

모든 작업에서 세 CLI를 동시에 호출하는 것은 비효율적이다. 작업 난이도와 위험도에 따라 다음 모드를 선택한다.

10.1 Single Executor Mode

낮은 위험의 명확한 작업에 사용한다.

Codex 또는 다른 CLI 1개

- 수정
- 자동 검증
- 바이브코딩 툴이 결과 표시

예:

- 오타자 수정
- 파일명 정리
- 명확한 UI 버그
- 단순 테스트 추가

10.2 Planner-Executor-Reviewer Mode

중요한 작업의 기본 모드다.

CLI A: 계획 제안

CLI B: 실행

CLI C: 독립 검토

바이브코딩 툴: 차이 조정 및 사용자 보고

역할은 프로젝트와 도구 성능에 따라 바꿀 수 있다.

10.3 Parallel Proposal Mode

정답이 하나가 아닌 설계·기획 문제에 사용한다.

Codex → 구현 중심 안

Claude → 구조·문서 중심 안

Antigravity → 통합·시각·실행 중심 안

바이브코딩 오케스트레이터 → 비교표와 통합안

10.4 Debate and Consensus Mode

고위험 변경에 사용한다.

1. 세 CLI가 독립 분석
2. 각 CLI가 다른 안의 약점을 검토
3. 공통 합의와 이견을 분리
4. Critic이 완료 조건과 위험을 판정
5. 바이브코딩 툴이 사용자에게 선택지 제시
6. 승인 후 한 CLI가 실행
7. 다른 CLI가 결과 검증

10.5 원칙

여러 CLI는 다수결을 위한 것이 아니다.

서로 다른 관점의 오류를 발견하기 위한 것이다.

최종 결정권은 바이브코딩 오케스트레이터와 사용자에게 있다.

다중 도구 실행 흐름

flowchart TD

U[사용자 요청]
V[바이트코딩 오케스트레이터]
H[AUNOVA Harness]
O[Obsidian Context Compiler]
C1[Codex CLI]
C2[Claude Code CLI]
C3[Antigravity CLI]
R[비교·토론·Critic Review]
X[선택된 실행자]
T[테스트·Validation]
D[사용자에게 결과 표시]
S[Obsidian 상태·Handoff 갱신]

U --> V --> H --> O
O --> C1
O --> C2
O --> C3
C1 --> R
C2 --> R
C3 --> R
R --> X --> T --> D --> S
S --> V

11. E·T·C·S·L·V 이론 모델

기존 AUNOVA Harness의 구조를 Obsidian 운영체제와 직접 연결한다.

E — Environment

Vault 위치
프로젝트 실제 경로
로컬·NAS·클라우드 상태
읽기·쓰기 권한
운영체제
현재 작업 디렉터리

E는 단순한 “로컬인가 클라우드인가”가 아니라 **실제 파일 경계와 접근 가능성**을 뜻한다.

E의 경로 추상화 원칙

E는 절대경로 목록이 아니라 현재 장치에서 논리 경로를 실제 경로로 해석하는 계층이다.

`${AUNOVA_AI_OS_ROOT}`
→ 현재 PC의 Obsidian Vault Root

`workspace_id`
→ 현재 PC의 실제 RAW Workspace 경로

`${AUNOVA_NAS_ROOT}`
→ 현재 PC에 연결된 NAS Root

RAW Root 하나를 전역으로 가정하지 않는다. 프로젝트별·도구별 RAW 폴더가 여러 개일 수 있기 때문이다.

T — Tools

바이브코딩 툴
Codex CLI
Claude Code CLI
Antigravity CLI
MCP
스크립트
Git
빌드·테스트 도구

T는 설치 목록이 아니라 현재 Task에 사용할 수 있는 실행 능력의 목록이다.

C — Context

Project Overview
Current State
Decision
Handoff
Workflow
관련 Source
작업 파일

C는 Context Compiler가 JIT로 생성한다.

S — State

현재 Stage
현재 Task
담당 도구
작업 범위
진행률
Blocker
사용자 승인 상태

S는 대화 기억이 아니라 파일로 외부화한다.

L — Logs

Session Summary

변경 파일

CLI 실행 결과

오류 요약

Handoff

동기화 기록

전체 원시 로그를 영구 보존하지 않고, 재현과 인계에 필요한 요약을 유지한다.

V — Validation

완료 조건

테스트

빌드

문서 검수

Gate 판정

위험 검토

사용자 확인

V는 작업 종료에 추가하는 선택 항목이 아니라, 완료의 정의 자체다.

12. 노트 유형과 메타데이터

AUNOVA Vault에서 모든 노트를 동일하게 취급하지 않는다.

12.1 핵심 노트 유형

유형	목적	보존 기간
project	프로젝트 진입점	프로젝트 수명 전체
context	작업영역 배경	지속 갱신
decision	결정과 근거	영구
task	실행 단위	완료 후 요약
risk	위험·제약	해소까지
source	외부 근거	필요 기간
workflow	반복 실행 절차	영구·버전 관리
knowledge	재사용 가능한 개념	영구·발전
session	현재 세션 기록	요약 보존
handoff	도구 간 인계	최근본 우선
validation	검증 결과	산출물과 함께 보존

12.2 공통 속성 예시

```
id: MW-DEC-024
type: decision
project: mind-weather
workstream: development
status: accepted
created: 2026-07-21
updated: 2026-07-21
owner: human
agent: claude-code
confidence: high
supersedes:
related:
  - "[MW-REQ-012 로그인 요구사항]"
source:
  - "[사용자 인터뷰 2026-07-18]"
```

12.3 ID 원칙

파일명만으로 식별하면 이름 변경 시 자동화가 깨질 수 있다. 중요한 객체에는 안정적인 ID를 둔다.

```
PROJECT-WORKSTREAM-TYPE-NUMBER
MW-DEV-TASK-024
TB-BIZ-DEC-007
AR-MEDIA-RISK-003
```

13. 링크 문법과 관계 어휘

단순히 관련 있음만 사용하면 네트워크의 의미가 약해진다. 자주 쓰는 관계를 표준화한다.

13.1 권장 관계

supports	근거로 지원한다
contradicts	반대 근거를 제시한다
derived-from	여기서 파생되었다
implements	요구사항을 구현한다
validates	결과를 검증한다
blocks	진행을 막는다
depends-on	선행 조건이다
supersedes	이전 결정을 대체한다
reuses	기존 자산을 재사용한다
produces	산출물을 만든다

13.2 링크는 문맥 안에서 작성

좋지 않은 예:

관련: [[고객 문제]] [[로그인 기능]] [[데모]]

권장 예:

이 결정은 [[고객 문제]]에서 확인된 가입 이탈을 줄이기 위해
[[로그인 기능]]을 단순화하며, 결과는 [[IR 데모 시나리오]]에서 검증한다.

링크 주변의 문장이 관계의 의미를 설명해야 한다.

13.3 속성 링크와 본문 링크의 구분

속성 링크 = 자동 검색과 상태 관리

본문 링크 = 사람이 이해하는 논리와 맥락

두 가지를 함께 사용한다.

14. MOC와 탐색 구조

14.1 MOC의 세 종류

Home MOC

Vault 전체의 진입점이다.

활성 프로젝트
현재 위험
최근 결정
주요 워크플로
Inbox 상태

Project MOC

한 프로젝트의 모든 작업영역을 연결한다.

Overview
Business
Product
Development
Media
Presentation
Tasks
Decisions
Handoffs

Workflow MOC

작업 방법을 연결한다.

App Development Workflow
Video Production Workflow
Business Plan Workflow
Research Workflow
Validation Workflow

14.2 MOC는 자동 목록과 설명을 결합

단순 자동 쿼리만 사용하면 우선순위와 이유가 사라진다. MOC에는 사람이 작성한 설명과 자동 목록을 함께 둔다.

현재 가장 중요한 결정
- [[MW-DEC-024 로그인 단순화]] — 데모 전 완료 필요

자동 목록
- status가 `open`인 development task

14.3 MOC가 비대해지는 것을 방지

- 하나의 MOC는 하나의 탐색 목적을 가진다.
- 모든 노트를 한 MOC에 넣지 않는다.
- 하위 MOC로 분리한다.
- 종료된 항목은 Archive MOC로 이동한다.

15. Inbox에서 재사용 지식까지

15.1 Inbox는 입출력 게이트

Inbox는 임시 저장소가 아니라 외부 정보가 프로젝트 본문에 바로 섞이는 것을 막는 안전 구역이다.

GitHub 변경
Google Drive 신규 문서
NAS 신규 자산
사용자 제공 파일
다른 CLI의 Handoff

15.2 처리 상태

pending
in-progress
blocked
conflicts
processed

15.3 지식 생명주기

flowchart LR
 I[Inbox]
 S[Source Note]
 P[Project Context]
 D[Decision / Task]
 V[Validation]
 K[Reusable Knowledge]
 M[MOC]
 A[Archive]

 I --> S --> P --> D --> V
 V --> K --> M
 V --> A
 M --> P

15.4 승격 기준

다음 조건 중 둘 이상을 만족하면 재사용 지식으로 승격한다.

- 두 개 이상의 프로젝트에서 사용 가능
- 반복 오류를 줄이는 규칙
- 검증된 실행 방법
- 중요한 의사결정 기준
- 향후 검색 가치가 높은 설명

단순 세션 기록과 임시 프롬프트는 지식으로 승격하지 않는다.

16. 세션·Handoff·충돌 제어

16.1 세션은 도구별로 분리

여러 바이브코딩 툴과 CLI가 동일한 프로젝트를 사용할 수 있으므로 전역 active-project 하나만 두지 않는다.

```
session_id  
vibe_tool  
project_id  
workstream  
read_scope  
write_scope  
cli_assignments  
started_at  
status
```

16.2 Handoff에 남길 내용

목표
완료한 작업
변경 파일
검증 결과
남은 문제
중요 결정
다음 권장 작업

긴 대화 전체를 넘기지 않는다.

16.3 충돌 단위

프로젝트 전체 잠금은 병렬 작업을 막으므로 다음 순서로 좁게 잠근다.

파일
→ 폴더
→ Workstream
→ 프로젝트 전체

16.4 동시 작업 예시

Codex CLI → app/src/login 수정
Claude Code CLI → 사업계획서 시장분석 검토
Antigravity CLI → 데모 UI와 실행 결과 확인

허용된다.

Codex CLI와 Claude Code CLI가 동시에 app/src/login/page.tsx 수정

충돌 경고와 사용자 승인이 필요하다.

17. 안전·보안·동기화 원칙

17.1 최소 권한

탐색 = Read Only
수정 = 제한된 Workspace Write
배포·삭제 = 사용자 승인 필수

17.2 민감정보

다음 파일은 Obsidian과 일반 동기화 대상에서 제외한다.

.env
API Key
OAuth Token
Credential
개인정보 원본
비밀 인증서

17.3 저장소 선택은 독립적

GitHub 켜기/끄기
Google Drive 켜기/끄기
NAS 켜기/끄기

프로젝트 설정과 현재 세션 설정을 분리한다.

프로젝트에는 GitHub가 연결되어 있음
현재 세션에서는 GitHub 확인을 건너뛴다

둘 다 가능해야 한다.

17.4 파괴적 작업

다음은 자동 실행하지 않는다.

- 강제 Push
- 브랜치 삭제
- NAS 원본 삭제
- 루트 폴더 일괄 삭제
- 충돌 자동 병합
- Vault 전체 재구성
- 공개 링크 생성

17.5 채널 장애 독립성과 상태 판정

외부 채널은 독립적으로 검사하고 처리한다.

READY_ONLINE

→ 필요한 로컬 환경과 모든 선택 채널 사용 가능

READY_DEGRADED

→ 로컬 RAW와 Obsidian은 사용 가능하지만 일부 채널 오프라인

READY_OFFLINE

→ 외부 채널 없이 로컬 작업·Validation·Handoff 가능

BLOCKED

→ 현재 RAW Workspace 또는 필수 Obsidian 상태를 열 수 없음

GitHub, NAS, Google Drive 중 하나의 실패가 다른 채널의 성공을 취소하지 않는다. 실패한 채널만 pending-sync로 남긴다.

18. 실패 패턴과 예방 규칙

실패 1. Vault를 프로젝트 백업 폴더로 사용

증상: 코드, 영상, PDF, 중간 렌더가 모두 들어가 Vault가 비대해진다.

예방: Vault에는 지식·상태·인덱스만 두고 대용량 원본은 프로젝트/NAS에 유지한다.

실패 2. Graph를 지식 품질로 착각

증상: 링크 수는 많지만 관계의 의미가 없다.

예방: 링크 문맥, 관계 어휘, MOC, 속성을 함께 사용한다.

실패 3. 모든 노트를 원자화

증상: 조각난 노트가 너무 많아 읽기 어렵다.

예방: 재사용 가치가 있는 개념만 원자화하고, 프로젝트 문맥은 적절한 길이로 유지한다.

실패 4. 모든 Skill과 모든 문서를 상시 로딩

증상: 토큰 증가, 규칙 충돌, 응답 지연.

예방: Stage·Route·Task에 따라 JIT 로딩한다.

실패 5. 세 CLI를 매번 동시에 호출

증상: 비용과 시간이 증가하고 같은 답이 반복된다.

예방: 위험도 기반으로 Single, Review, Parallel, Debate 모드를 선택한다.

실패 6. 대화가 상태 저장소가 됨

증상: 다른 톨에서 이어서 작업할 수 없다.

예방: Current State, Decision, Handoff를 파일로 기록한다.

실패 7. 경로 하드코딩

증상: NAS 드라이브 문자나 사용자 PC가 바뀌면 시스템이 깨진다.

예방: 저장소 ID와 사용자별 로컬 경로 설정을 분리한다.

실패 8. 파일을 여러 저장소에서 동시 편집

증상: 최신본 불명확, 덮어쓰기, 충돌.

예방: 저장소별 Source of Truth와 업로드 방향을 정의한다.

19. 운영 품질 지표

Obsidian 시스템은 노트 수가 아니라 작업 효율로 평가한다.

19.1 문맥 효율

전체 Vault 대비 CLI 전달 문맥 비율
관련 없는 파일 로딩 횟수
작업 시작까지 걸린 시간

19.2 인계 품질

다른 도구가 추가 질문 없이 이어서 작업한 비율
Handoff 누락으로 재작업한 횟수
결정 근거를 찾는 데 걸린 시간

19.3 지식 재사용

두 개 이상 프로젝트에서 사용된 Knowledge Note 수
반복 오류를 방지한 Workflow 수
새 프로젝트 초기 설정 시간 감소율

19.4 충돌·안전

동시 수정 충돌 수
잘못된 원격 업로드 수
파괴적 명령 차단 수
복구 가능한 백업 비율

19.5 결과 품질

Validation 통과율
CLI 간 검토에서 발견된 오류 수
사용자 수정 요청 횟수
최종 산출물 재작업률

20. 단계적 도입 로드맵

단계 1. 관찰 가능한 최소 구조

- Obsidian Vault 생성
- 프로젝트 등록부
- Project Overview
- Current State
- Decision
- Handoff

단계 2. JIT Context

- 작업 유형별 Context 문서
- Context Compiler
- Task Packet
- Validation Plan

단계 3. CLI 연결

- Codex CLI
- Claude Code CLI
- Antigravity CLI
- 공통 Adapter

단계 4. 오케스트레이션

- Single Executor
- Planner-Executor-Reviewer
- Parallel Proposal
- Debate and Consensus

단계 5. 저장소 연동

- GitHub 선택 동기화
- Google Drive 산출물 업로드
- NAS 대용량 백업
- 세션별 건너뛰기

단계 6. 학습과 개선

- 반복 오류를 Workflow로 승격
- 검증 규칙 자동화
- 사용량·시간·충돌 지표 분석
- 하네스 규칙 개정

21. 최종 설계 원칙

원칙 1. 프로젝트가 먼저다

지식 관리가 실제 작업을 방해해서는 안 된다.

원칙 2. Vault는 창고가 아니라 네트워크다

정보의 양보다 연결의 의미와 재사용성이 중요하다.

원칙 3. 모든 기록을 지식으로 만들지 않는다

Inbox, Source, Project Context, Evergreen Knowledge를 구분한다.

원칙 4. Context는 컴파일한다

전체 Vault를 전달하지 않고 현재 작업에 필요한 최소 문맥만 만든다.

원칙 5. 도구의 역할을 분리한다

바이브코딩 = 지휘와 대화
Obsidian = 기억과 의미
CLI = 실행과 검증
Harness = 흐름과 통제

원칙 6. 다중 CLI는 토론과 검증에 사용한다

세 CLI를 항상 병렬 실행하지 않고, 중요한 작업에서 서로 다른 관점의 오류를 찾게 한다.

원칙 7. 상태는 대화가 아니라 파일에 남긴다

다른 도구가 언제든지 인계받을 수 있어야 한다.

원칙 8. 저장소마다 역할을 하나씩 부여한다

중복 편집과 최신본 혼란을 방지한다.

원칙 9. 자동화보다 안전 경계가 먼저다

삭제, Push, 배포, 공개는 명시적 승인 후 수행한다.

원칙 10. 반복 오류는 프롬프트가 아니라 시스템을 고친다

Workflow, Skill, Validation, Folder Map, Adapter를 개선한다.

22. 참고문헌 및 전문가 출처

부록 A. AUNOVA 노트 품질 체크리스트

프로젝트 노트

- 이 노트가 어느 프로젝트에 속하는가?
- 현재 상태가 속성으로 표시되어 있는가?
- 관련 결정과 작업이 연결되어 있는가?
- 원본 파일의 실제 경로가 명확한가?
- 최신 Handoff로 이동할 수 있는가?

지식 노트

- 하나의 핵심 개념을 설명하는가?
- 특정 프로젝트에만 종속되지 않는가?
- 근거 출처가 있는가?
- 적어도 하나의 MOC에 연결되어 있는가?
- 비슷한 노트와 차이가 설명되어 있는가?

Workflow 노트

- 시작 조건이 명확한가?
- 입력과 출력이 명확한가?
- 실행 도구가 명확한가?
- 사용자 승인 지점이 있는가?
- 검증과 실패 처리 방법이 있는가?

부록 B. 그물망 건강도 체크

고립 노트가 지나치게 많지 않은가?
하나의 거대 허브에 모든 링크가 몰리지 않았는가?
Project MOC에서 현재 상태를 1분 안에 파악할 수 있는가?
결정에서 근거와 구현으로 이동할 수 있는가?
구현에서 검증 결과로 이동할 수 있는가?
최근 Handoff가 실제 변경 파일과 일치하는가?
종료 프로젝트의 지식이 활성 프로젝트에 불필요하게 섞이지 않는가?
CLI가 전체 Vault 없이도 Task Packet으로 작업할 수 있는가?

부록 C. 본 문서와 후속 파일의 관계

파일 1 전문가 이론과 그물망 아키텍처
↓
파일 4 실제 Obsidian 폴더·설정·Skill 구조
↓
파일 5 바이브코딩 오케스트레이션과 다중 CLI 실행 규격
↓
파일 6 직접 설치·환경 설정·다운로드 가이드
↓
파일 2 Obsidian 자동 설치용 지시 파일
파일 3 3개 CLI 자동 설치용 지시 파일
↓
파일 7 Codex용 하네스 4파일 개편본
↓
Claude Code·Antigravity용 하네스 확장

부록 D. Multi-PC · Multi-RAW 핵심 용어

용어	의미
Canonical Project RAW Workspace	여러 Workspace가 공유하는 논리적 프로젝트 특정 PC·도구가 실제로 여는 작업 폴더

용어	의미
Workspace Instance	Canonical Project에 등록된 RAW Workspace 한 개
Device Profile	PC별 Vault·NAS·CLI 경로 설정
NAS Sync Hub	PC 간 검증된 파일과 Manifest를 교환하는 위치
Project Registry	Canonical Project 목록과 공통 정보
Workspace Registry	장치별 RAW Workspace 실제 경로 매핑
Path Resolver	논리 ID를 현재 PC의 실제 경로로 변환하는 기능
Manifest	파일 경로·크기·수정시각·해시·동기화 상태 목록
Pending Sync	연결 실패로 아직 반영되지 않은 동기화 작업