Claude Code 세션 기억 관리 완벽 가이드

20편_Claude Code 세션 기억 관리 완벽 가이드

📌 개요

Claude Code를 사용하다 보면 가장 큰 고민이 **"세션이 끊어지면 어떻게 하지?"**입니다.

- 컨텍스트 윈도우 초과로 세션 종료
- 브라우저 닫거나 재시작
- 긴 작업 중간에 중단

이 문서에서는 세션이 끊어져도 작업 연속성을 유지하는 2가지 핵심 방법을 다룹니다.

🎯 문제 상황

세션이 끊어지면 일어나는 일

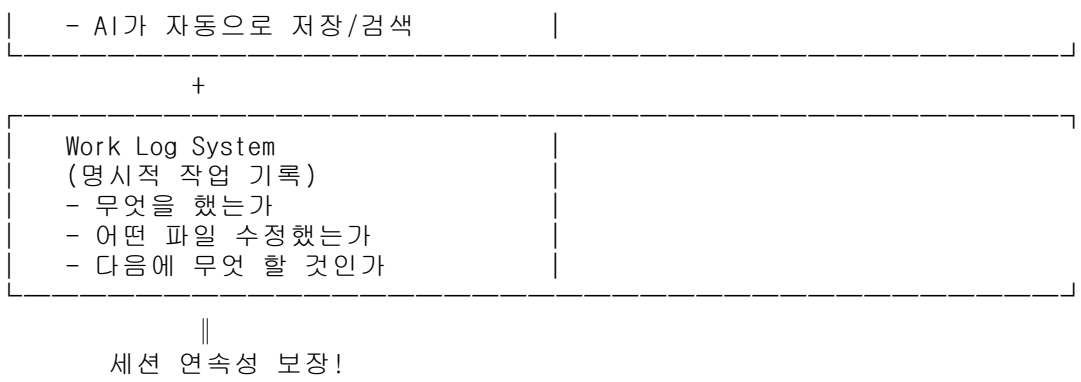
```
이전 세션
- API 20개 중 15개 완료
- 테스트 코드 작성 중
- 특정 버그 수정 방법 파악
    ↓
세션 종료 (컨텍스트 초과)
    ↓
새 세션 시작
- "무엇을 하고 있었지?" ❌
- "어디까지 했지?" ❌
- "어떤 방법으로 하려고 했지?" ❌
```

결과: 처음부터 다시 설명해야 함 → 시간 낭비 → 비효율

💡 해결책: 2가지 기억 시스템

시스템 개요

```
┌─────────────────────────────────────────────────────────┐
│   Memory MCP                          │
│   (암묵적 지식 자동 기억)              │
│   - 사용자 선호도                      │
│   - 패턴 및 컨텍스트                   │
```



📦 방법 1: Memory MCP (자동 기억 시스템)

1.1 Memory MCP란?

Memory MCP (Model Context Protocol - Memory Server)는 Anthropic에서 공식 제공하는 메모리 서버입니다.

핵심 특징: - ✅ **자동 기억**: AI가 중요한 정보를 자동으로 감지하여 저장 - ✅ **세션 간 공유**: 다음 세션에서 자동으로 기억 불러옴 - ✅ **로컬 저장**: 모든 데이터는 로컬 머신에만 저장 (프라이버시 보장) - ✅ **검색 가능**: AI가 필요할 때 자동으로 관련 기억 검색

1.2 설치 방법

Step 1: MCP 설정 파일 수정

```
# 프로젝트 디렉토리의 .claude/mcp_servers.json 파일

{
  "mcpServers": {
    "memory": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-memory"]
    }
  }
}
```

Step 2: Claude Code 재시작

설정 파일 수정 후 Claude Code를 재시작하면 자동으로 Memory MCP가 활성화됩니다.

1.3 저장 위치

Windows:

```
C:₩Users₩[사용자명]₩.local₩share₩@modelcontextprotocol₩server-memory₩
└─── memories.json
```

macOS:

```
~/Library/Application Support/@modelcontextprotocol/server-memory/
└── memories.json
```

Linux:

```
~/.local/share/@modelcontextprotocol/server-memory/
└── memories.json
```

1.4 저장 형태

```
{
  "memories": [
    {
      "id": "uuid-1234",
      "content": "사용자는 시간 추정을 매우 싫어함",
      "timestamp": "2025-11-17T03:00:00Z",
      "context": "SSALWorks project",
      "tags": ["preference", "user-behavior"]
    },
    {
      "id": "uuid-5678",
      "content": "프로젝트 그리드 업데이트는 Claude Code가 직접 수행",
      "timestamp": "2025-11-17T02:30:00Z",
      "context": "SSALWorks workflow",
      "tags": ["workflow", "rule"]
    }
  ]
}
```

1.5 무엇을 기억하나?

자동으로 저장되는 정보:

1. **사용자 선호도**
    - “이 사용자는 검증을 매우 중요하게 생각함”
    - “파일 저장 위치를 자주 틀림 → 주의 필요”
    - “대충 하는 것을 싫어함”
2. **프로젝트 패턴**
    - “이 프로젝트는 Supabase 사용”
    - “Next.js 14 App Router 구조”
    - “TypeScript strict mode 사용”
3. **반복되는 이슈**
    - “이 프로젝트에서 CORS 문제 자주 발생”
    - “테스트 실행 시 환경 변수 필요”
4. **중요한 결정사항**
    - “프로젝트 그리드는 웹사이트가 아닌 Claude Code가 업데이트”
    - “검증 없이는 다음 작업 진행 금지”

1.6 사용 예시

이전 세션:

```
사용자: "API 작성할 때 항상 입력 검증을 철저히 해줘.
        과거에 검증 없이 했다가 큰일 났었어."

Claude: [Memory MCP에 자동 저장]
        "사용자는 API 입력 검증을 매우 중요하게 생각함"
```

새 세션:

```
사용자: "회원가입 API 만들어줘"

Claude: [Memory 자동 검색]
        "이전에 입력 검증을 중요하게 생각한다고 했으니..."

        "회원가입 API를 작성하겠습니다.
        입력 검증을 철저히 포함하겠습니다:
        - 이메일 형식 검증
        - 비밀번호 강도 검증
        - SQL Injection 방지
        ..."
```

1.7 장점

✅ **자동화**: 수동 기록 불필요, AI가 알아서 저장 ✅ **컨텍스트 보존**: 암묵적 지식까지 기억 ✅ **지능적 검색**: 관련 상황에서 자동으로 떠올림 ✅ **프라이버시**: 로컬 저장, 외부 유출 없음

1.8 단점 및 한계

⚠️ **불투명성**: 무엇이 저장되는지 정확히 모름 ⚠️ **제어 어려움**: 수동으로 특정 기억 삭제/수정 어려움 ⚠️ **명시적 작업 기록 부족**: “어떤 파일 수정했는지” 같은 구체적 정보는 약함

📝 방법 2: Work Log System (작업 로그 시스템)

2.1 Work Log란?

Work Log는 모든 작업을 명시적으로 기록하는 마크다운 파일입니다.

핵심 특징: - ✅ **투명성**: 사람이 직접 읽고 확인 가능 - ✅ **명시적**: 정확히 무엇을 했는지 구체적으로 기록 - ✅ **영구 보존**: 파일로 저장되어 절대 안 사라짐 - ✅ **구조화**: 일관된 형식으로 쉽게 파악

2.2 폴더 구조

```
.claude/work_logs/
├─── current.md          ← 현재 활성 로그 (최신 작업만)
├─── 2025-11-17.md       ← 순환된 과거 로그
```

```
├── 2025-11-16.md
├── 2025-11-15.md
└── archive/            ← 30일 이상 된 로그
    ├── 2025-10-20.md
    └── 2025-10-15.md
```

2.3 자동 순환 시스템

문제: 로그 파일이 계속 커지면 읽을 때 토큰 낭비

해결: 자동 순환 (Auto-rotation)

```
current.md 크기 확인
    ↓
50KB 초과?
    ↓
YES → current.md → 2025-11-18.md로 이름 변경
      → 새로운 current.md 생성
      → 이전 로그 링크 추가
    ↓
NO  → 계속 current.md 사용
```

30일 이상 된 로그:

```
# 자동으로 archive/로 이동
mv 2025-10-15.md archive/
```

2.4 기록 형식

템플릿:

```
## 2025-11-17 14:30

### 작업: 회원가입 API 구현

**작업 내용**:
- 회원가입 API 엔드포인트 작성 (/api/auth/signup)
- 입력 검증 로직 추가 (이메일, 비밀번호)
- Supabase Auth 연동
- 단위 테스트 20개 작성

**생성/수정된 파일**:
- ✅ `3_Backend_APIs/auth/signup.ts` (생성)
- ✅ `3_Backend_APIs/auth/signup.test.ts` (생성)
- ✅ `types/auth.ts` (수정)

**검증 결과**:
- ✅ Unit tests: 20/20 통과
- ✅ Build: 성공
- ✅ Lint: 이슈 없음
- ✅ Type check: 통과

**다음 작업**:
- 로그인 API 구현
- 비밀번호 재설정 API

**참고**:
```

```
- 비밀번호 정규식: 최소 8자, 대소문자+숫자+특수문자
- Supabase Auth 정책: RLS 활성화됨

---
```

2.5 기록 시점

필수 기록 (반드시): - ✅ 작업 완료 시 (매번) - ✅ Phase 완료 시 (요약) - ✅ 중요한 결정 시 (즉시)

선택 기록 (필요시): - 긴 작업의 중간 체크포인트 - 오류 발생 및 해결 과정 - 실험적 시도 및 결과

2.6 사용 예시

이전 세션 (작업 완료 후):

```
## 2025-11-17 10:00

### 작업: Product API 15개 완료

**작업 내용**:
- Product CRUD API 구현
- 15/20 완료 (5개 남음)

**다음 작업**:
- 나머지 5개 API 구현
  1. GET /api/products/search
  2. POST /api/products/bulk
  3. PUT /api/products/bulk
  4. DELETE /api/products/bulk
  5. GET /api/products/stats

**참고**:
- Search API는 Algolia 연동 필요
- Bulk API는 트랜잭션 처리 필수
```

새 세션:

```
Claude Code 시작
    ↓
.claude/work_logs/current.md 읽기
    ↓
"아, 이전 세션에서 15/20 완료했구나"
"다음은 Search API부터 시작이네"
"Algolia 연동이 필요하다고 메모되어 있네"
    ↓
즉시 이어서 작업 시작!
```

2.7 장점

✅ **투명성**: 무엇이 기록되었는지 명확 ✅ **정확성**: 파일 목록, 검증 결과 등 구체적 ✅ **검색 가능**: 날짜별로 쉽게 찾기 ✅ **백업 가능**: 파일이므로 복사/백업 쉬움 ✅ **협업 가능**: 팀원도 읽고 확인 가능

2.8 단점 및 한계

⚠️ **수동 작업**: AI가 매번 기록해야 함 (자동 아님) ⚠️ **형식 준수 필요**: 템플릿 따라야 일관성 유지 ⚠️ **암묵적 지식 부족**: 선호도나 패턴은 기록 안 됨

🔄 두 시스템의 관계

상호 보완적 구조

```
Memory MCP (암묵적)          Work Log (명시적)
      ↓                           ↓
"왜 그렇게 했는가"          "무엇을 했는가"
"사용자 선호도"             "파일 목록"
"반복 패턴"                 "검증 결과"
"주의할 점"                 "다음 작업"
      ↓                           ↓
        둘 다 새 세션에서 활용
                ↓
        완벽한 작업 연속성
```

구체적 예시

Memory MCP 기억: - “이 사용자는 검증을 매우 중요하게 생각함” - “과거에 AI가 대충 하고 완료 보고한 적 있어서 싫어함” - “프로젝트 그리드는 Claude Code가 직접 업데이트”

Work Log 기록: - “2025-11-17: CLAUDE.md 6대 원칙 추가 (997 lines)” - “검증 완료: 6개 원칙 모두 확인” - “다음: PROJECT_STATUS.md 업데이트”

새 세션에서:

```
Memory: "검증을 중요하게 생각하시니 꼼꼼히 해야겠다"
Work Log: "지난번에 CLAUDE.md 수정했고, 다음은 PROJECT_STATUS.md 작업이네"
      ↓
완벽하게 이어서 작업!
```

🎯 실전 활용 가이드

시나리오 1: 긴 개발 작업

상황: API 100개를 3주에 걸쳐 개발

전략:

1. **Work Log로 진행 상황 추적**

   ```
   ## 2025-11-17
   - Week 1: API 35/100 완료
   ```

```
- 다음: Product API 20개

## 2025-11-24
- Week 2: API 70/100 완료
- 다음: Order API 15개
```

2. **Memory MCP로 패턴 학습**
    - “이 프로젝트는 모든 API에 rate limiting 필요”
    - “Supabase RLS 정책 항상 확인”
    - “테스트 커버리지 80% 이상 유지”

결과: 세션이 몇 번 끊어져도 정확히 이어서 작업 가능

시나리오 2: 버그 수정

상황: 복잡한 버그를 여러 세션에 걸쳐 디버깅

전략:

1. **Work Log에 시도 기록**

```
## 2025-11-17 10:00

### 작업: CORS 오류 해결 시도

**시도 1**:
- Next.js middleware에 CORS 헤더 추가 → 실패

**시도 2**:
- Vercel 설정에 CORS 추가 → 부분 성공
- Preflight는 해결, POST는 여전히 실패

**다음 시도**:
- Supabase CORS 설정 확인 필요
```

2. **Memory MCP로 교훈 저장**
    - “Vercel + Supabase 조합에서 CORS는 양쪽 다 설정 필요”
    - “Preflight와 실제 요청 설정이 다름”

결과: 시행착오 없이 효율적으로 문제 해결

시나리오 3: 팀 협업

상황: 다른 개발자가 작업한 내용을 이어받음

전략:

1. **Work Log 공유**

```
## 2025-11-17 (개발자 A)

### 작업: 결제 모듈 기초 구현

- Stripe SDK 연동 완료
- Webhook 엔드포인트 구현
- 테스트 환경 설정 완료

**주의사항**:
- Stripe Secret Key는 .env에서 읽어야 함
- Webhook signature 검증 필수

**다음 작업**:
- 결제 실패 시 재시도 로직
- 환불 처리 구현
```

2. **개발자 B가 이어받음**

```
Work Log 읽기
    ↓
"Stripe 연동은 완료되어 있고"
"다음은 재시도 로직이구나"
"Webhook signature 검증이 중요하다고 했네"
    ↓
바로 이어서 작업 시작
```

결과: 인수인계 시간 최소화, 컨텍스트 손실 없음

🛠️ 설정 가이드

전체 설정 프로세스

Step 1: Memory MCP 설치

```jsonc
// .claude/mcp_servers.json 파일 생성/수정
{
  "mcpServers": {
    "memory": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-memory"]
    }
  }
}
```

Step 2: Work Logs 폴더 생성

```
mkdir -p .claude/work_logs/archive
```

Step 3: current.md 템플릿 생성

```
touch .claude/work_logs/current.md
```

```
# 작업 로그 - Current

**이 파일은 활성 로그입니다 (50KB 제한)**
```

```
---

## 작업 기록 시작

[여기에 작업 기록 추가]
```

Step 4: CLAUDE.md에 규칙 추가

```
## 새 세션 시작 시 필수 확인

1. `.claude/work_logs/current.md` 읽기
2. Memory MCP 활용 (자동)
3. 작업 이어서 진행

## 작업 완료 시 필수 작업

1. `current.md` 업데이트
2. 파일 크기 확인 (50KB 초과 시 순환)
```

Step 5: Claude Code 재시작

설정 완료!

📊 효과 측정

Before (기억 시스템 없음)

```
세션 1: API 15개 작성 (2시간)
세션 종료
    ↓
세션 2: 어디까지 했는지 파악 (30분)
        이어서 작업 (2시간)
세션 종료
    ↓
총 시간: 4.5시간
낭비 시간: 30분 (11%)
```

After (기억 시스템 사용)

```
세션 1: API 15개 작성 (2시간)
        Work Log 기록 (2분)
세션 종료
    ↓
세션 2: Work Log 읽기 (1분)
        Memory 자동 로드
        즉시 이어서 작업 (2시간)
    ↓
총 시간: 4시간 3분
낭비 시간: 3분 (1.2%)
절약: 27분 (10배 효율)
```

🎓 Best Practices

DO ✅

1. **매 작업마다 Work Log 업데이트**
    - 작업 완료 시 즉시 기록
    - 나중에 하려다 잊어버림
2. **구체적으로 기록**
    - ❌ “API 작성함”
    - ✅ “회원가입 API 작성 (signup.ts, 20 tests, all passed)”
3. **다음 작업 명시**
    - 새 세션이 바로 시작할 수 있게
    - “다음: 로그인 API (login.ts, OAuth 연동 필요)”
4. **파일 목록 정확히**
    - 어떤 파일을 수정했는지 명확히
    - 나중에 찾기 쉬움
5. **검증 결과 포함**
    - 테스트, 빌드, Lint 결과
    - 문제 있으면 명시

DON’T ❌

1. **로그 기록 미루지 않기**
    - “나중에 정리해야지” → 절대 안 함
    - 즉시 기록하기
2. **너무 간략하게 쓰지 않기**
    - “작업 완료” → 무슨 작업?
    - 구체적으로!
3. **파일 크기 무시하지 않기**
    - 50KB 넘으면 순환
    - 토큰 낭비 방지
4. **이전 로그 삭제하지 않기**
    - archive/에 보관
    - 나중에 필요할 수 있음

🔧 트러블슈팅

Q1: Memory MCP가 작동 안 함

확인사항:

```
# mcp_servers.json 위치 확인
ls .claude/mcp_servers.json
```

```
# JSON 문법 오류 확인
cat .claude/mcp_servers.json | jq .

# Claude Code 재시작 했는지 확인
```

Q2: Work Log가 너무 커짐

해결:

```
# 파일 크기 확인
ls -lh .claude/work_logs/current.md

# 50KB 넘으면 순환
mv .claude/work_logs/current.md .claude/work_logs/2025-11-17.md

# 새 current.md 생성
touch .claude/work_logs/current.md
```

Q3: Memory가 너무 많이 쌓임

확인:

```
# Memory 파일 위치 (Windows)
ls -lh ~/.local/share/@modelcontextprotocol/server-memory/

# 필요시 수동 정리
# (주의: 모든 기억 삭제됨)
rm -rf ~/.local/share/@modelcontextprotocol/server-memory/
```

Q4: 이전 세션 기억이 안 됨

체크리스트: - [ ] Memory MCP 활성화되어 있나? - [ ] Work Log에 제대로 기록했나? - [ ] current.md를 읽었나? - [ ] 프로젝트 디렉토리가 맞나?

📈 실전 성공 사례

사례 1: SSALWorks 프로젝트

상황: - 3개월간 진행 - 여러 세션에 걸쳐 작업 - 복잡한 요구사항

적용: - Memory MCP로 사용자 선호도 학습 - Work Log로 모든 작업 추적 - Phase별 요약 관리

결과: - 세션 전환 시간: 평균 2분 - 재작업: 거의 없음 - 일관성: 완벽 유지

사례 2: 버그 수정 마라톤

상황: - 복잡한 CORS 이슈 - 5일에 걸쳐 디버깅 - 10개 이상 세션

적용: - 매 시도마다 Work Log 기록 - 실패한 방법도 모두 기록 - Memory가 패턴 학습

결과: - 중복 시도 0회 - 최종 해결 시간: 예상의 60% - 향후 동일 이슈: 즉시 해결

🎯 결론

Memory MCP vs Work Log

기준	Memory MCP	Work Log
자동화	✅ 자동	❌ 수동
투명성	❌ 불투명	✅ 투명
구체성	❌ 추상적	✅ 구체적
검색	✅ AI 자동	✅ 수동 검색
저장	JSON	Markdown
용도	암묵적 지식	명시적 작업

핵심 메시지

두 시스템을 함께 사용하세요!

```
Memory MCP: "왜 이렇게 하는가" (컨텍스트)
     +
Work Log: "무엇을 했는가" (사실)
     ‖
완벽한 세션 연속성
```

투자 대비 효과:

- 설정 시간: 10분
- 매 작업 기록 시간: 2분
- 절약되는 시간: 평균 세션당 20-30분
- ROI: **10배 이상**

📚 더 알아보기

관련 문서

- 4편 Claude Code 첫 만남
- 18편 Claude 토큰 사용료 구조 및 최적화

- 프로젝트 CLAUDE.md 규칙

참고 자료

- MCP Memory Server GitHub
- Anthropic MCP 문서

작성일: 2025-11-17 **버전**: 1.0 **실전 검증**: SSALWorks 프로젝트 3개월 운영 경험 기반