

Getting Started with AI Team Development

A safe, reproducible way to use Codex, Claude Code, Discord and Obsidian so people and AI can consult, decide, remember and hand over work.

Created by KindlyFeatherWORKS

Project Lead: Seiichiro Yoshimura



Seichan	Kode	Kuro
Purpose and final decisions	Design, organization and checks	Ideas, writing and implementation help



Contents

Read the six chapters in order to connect memory, meetings, safety, saving, handover and setup into one operating method.

Chapter 1: A system so your AI doesn't forget	3
Chapter 2: Codex and Claude Code talking on Discord	11
Chapter 3: Protecting your AI team with Stop Lines	17
Chapter 4: Keeping daily conversations in Obsidian	21
Chapter 5: Connecting it all and keeping development going	29
Chapter 6: Letting the AI configure things for you	36
Glossary for beginners	43

Safety boundary: before publishing, sending externally, handling Secrets, billing, deleting, changing permissions or applying to production, follow the Stop Line and ask a human.

Chapter 1: A system so your AI doesn't forget



In this chapter, we link a short set of directions to your daily records, so that the AI does not have to ask everything again from scratch each time.

1. The story

When Seichan asked Kode to continue yesterday's work, Kode asked, "How far did we decide yesterday?" After repeating the same explanation to Kuro as well, Seichan realised that more time was going into explaining than into the work itself.

So the three of them separated things out: the directions you read first, the directions for each AI, the current decisions, and today's record. Following the links from the entry point, both Kode and Kuro could get back to where they had left off.

2. Overview

The important thing is not to put everything into one long file. Split it into four roles.

Before you begin, a human decides what the AI is allowed to do

The arrangement in this manual is not a way of handing broad permissions to an AI from the outset. Whether to allow authentication, file access, network use, or connections to external services is something you decide at your own responsibility, after checking the terms of service, the rules of your company or school, and the policy of whoever administers the device. "At your own responsibility" here does not mean "allow everything without reading the warnings"; it means starting from least privilege, so that you can explain the scope you allowed and its effects.

Start with the following scope.

What to check	The scope a beginner should choose first
Working folder	Limit what may be read and written to the working folder and the project only
Network, plugins, connectors	Do not enable until you need them. When you do, check what they connect to and what they can do, one at a time
Secrets	Do not paste the values of tokens, API keys or passwords. Manage them in a dedicated Secrets folder or in environment variables
Operations that affect the outside	Sending externally, publishing, deleting, permission changes and billing are confirmed by a human at the end — not left to the AI

For Codex, an easy setting to start from is `workspace-write`, which permits writing inside the project, together with `on-request`, which asks before operations outside that scope or before using the network. For Claude Code as well, keep the normal confirmations at first and add only as many permitted operations and connections as you actually need. At the beginner stage, do not use Codex's `danger-full-access` or Claude Code's `--dangerously-skip-permissions`. Being in a hurry is not a reason to hand over broad permissions.

On a managed device or a shared account at a company or school, do not widen the scope of authentication on your own judgement — check with an administrator or the person responsible. When a permission dialog appears, read "what is being given, which information, and for what purpose", and if you cannot explain it, do not allow it.

Official documentation:

- Codex sandbox and approvals: <<https://learn.chatgpt.com/docs/agent-approvals-security>>
- Codex `config.toml`: <<https://learn.chatgpt.com/docs/config-file/config-reference#configtoml>>
- Claude Code permissions: <<https://code.claude.com/docs/en/permissions>>

File	Role
AGENTS.md	The shared directions an AI reads first in this project
CLAUDE.md	Additional directions that Claude Code reads first
Memory.md	Decisions still in force, prohibitions, and the next actions
daily note	That day's conversations, progress, open questions, and links to the raw log

Do not copy the whole history into `AGENTS.md` and `CLAUDE.md`; place links to `Memory.md` and the daily note instead. That keeps the entry point short, while the detailed records stay readable when you need them.

Separate the role of the entry point from the role of the record

`AGENTS.md` and `CLAUDE.md` are a short entry contract handed to an AI that is starting work. Write only the purpose, the reading order, what to observe, and the conditions for stopping — do not put conversation history or long procedures there. `CLAUDE.md` is used to point at the shared entry point and to add the extra items that only Claude Code needs.

`Memory.md` is where things stand, still in force, which the next AI also needs. Keep the current decisions, what is not yet decided, what may proceed next, and the Stop Lines short. In the daily note, leave the facts by date for what

happened that day — progress, results of checks, and what is unfinished — with a source link so the detailed conversation and original material can be traced only when needed. Do not copy every daily fact into `Memory.md`; a human confirms and promotes only the decisions that stay in force.

In versions of Claude Code that support it, you can also use the import notation `@AGENTS.md` at the top of `CLAUDE.md`. In environments where that is not available, point to `AGENTS.md` with an ordinary Markdown link, as below.

Separate the rules by project

Split the work into a folder per project. Put the rules shared across everything in the parent folder, and put the purpose, the reading order and the prohibitions specific to that case inside the project.

```
<WORKSPACE>/
├── AGENTS.md      # shared directions for all projects
├── CLAUDE.md      # shared additional directions for Claude Code
├── project-a/
│   ├── AGENTS.md  # directions for project-a only
│   └── CLAUDE.md  # additional directions for project-a only
└── project-b/
    ├── AGENTS.md
    └── CLAUDE.md
```

In Codex, `AGENTS.md` files are read in order from the parent folders down to the current working folder, and the `AGENTS.md` inside the project is applied after the shared rules above it. Claude Code likewise reads `CLAUDE.md` from the current working location upwards, and adds the `CLAUDE.md` inside the target folder when it is needed. In other words, the upper level is the shared foundation, and inside the project is an override for that case alone. Use it to make the parent's important safety rules concrete — not to erase them.

Official documentation:

- Codex: <<https://learn.chatgpt.com/docs/agent-configuration/agents-md>>
- Claude Code: <<https://docs.anthropic.com/en/docs/claude-code/memory>>

Hand over and continue when a conversation gets long

When a single chat or session grows long, the amount of context the AI handles at once increases and it can become hard to keep track of the premises. If a conversation starts to feel hard to continue, have the AI write a handover, and paste it into a new chat or session to carry on. This is not a failure; it is the normal way to tidy up the information and keep working.

Enter the following as it is.

```
Please summarise this conversation into a handover for the next AI.
Include, briefly: the purpose, decisions still in force, what is not yet decided,
files changed,
results of checks, what may proceed next, the Stop Lines, and links to the AGENTS.md,
Memory.md
and Obsidian notes that should be read.
Do not add anything speculative, and do not write Secrets or personal information.
```

In the new chat, paste the handover you produced and enter the following.

```
This is a handover from a previous chat. First read AGENTS.md, Memory.md and the
latest daily
note, and check that they do not contradict the handover. Read back, briefly, where
things stand,
the next action, and the Stop Line — then continue with only what may proceed next.
```

3. The flow, in more detail

1. Put `AGENTS.md` and `CLAUDE.md` at the top of the project.
2. Create `notes/Memory.md` and write only the decisions still in force.
3. Put the records by date under `notes/daily/`.
4. Link from `AGENTS.md` to `Memory.md` and to today's daily note.
5. Link from `CLAUDE.md` to the shared `AGENTS.md`.
6. At the start of work, have the AI read back the entry point and the current decisions.
7. At the end of work, leave today's facts in the daily note, and any decision you will keep

using in `Memory.md`.

The minimum folder layout looks like this.

```
<PROJECT_ROOT>/
├── AGENTS.md
├── CLAUDE.md
├── .codex/
│   └── config.toml
├── notes/
│   ├── Memory.md
│   └── daily/
│       └── <DATE>.md
```

`AGENTS.md` and `CLAUDE.md` are the operating rules you have the AI read; `.codex/config.toml` is the configuration of Codex's execution environment. By not mixing the two roles, you can check "what must be observed" and "within what scope the tools may be used" separately.

Using `config.toml` safely

Codex's base settings shared by all projects go in `~/.codex/config.toml`, and the settings for this project alone go in `<PROJECT_ROOT>/.codex/config.toml`. In the per-project settings, configure the working scope, the sandbox, the approval method, and the plugins or MCP you use —only to the extent that case requires.

Start from a minimal example that confines writing to inside the project.

```
# <PROJECT_ROOT>/.codex/config.toml
# Base everything on local work inside this project.
sandbox_mode = "workspace-write"
```

This one line does not grant permission for Secrets, a Discord token, or sending externally. Using external services, broad network permissions, and settings that skip approval such as `approval_policy = "never"` are judged by a human, after checking the need and the impact. Do not write token or Webhook values directly into `config.toml`; use a `Secrets` folder and environment variables.

When you have an AI draft a Codex configuration example, enter the following.

```
Please draft a .codex/config.toml for this project.
Check the existing settings first; do not read or display the values of any Secret;
and limit the
scope of writing to inside the project.
For any proposal that changes plugins, MCP, the network, or the approval policy, do
not make the
change — explain the reason, the impact, and how to undo it.
```

In day-to-day use, deciding "where information goes" as well lets the AI maintain it itself.

Something newly learned	Where it is saved
A short rule to be observed every time	AGENTS.md or CLAUDE.md
Where the project as a whole stands	Memory.md
That day's conversation and work	The daily note and the raw log
A correction received from a human	rules/corrections.md
A repeated failure and prevention of recurrence	rules/mistakes.md
Long background, procedures, reusable knowledge	wiki/ in Obsidian

Keep the entry files short, and move history that has grown long into a separate note with a link. The heart of day-to-day operation is being able to ask Kode or Kuro, "decide where this should be saved, and update the links that need it."

4. How to actually set it up (Codex and Claude Code)

Files to create for both

Copy the following into AGENTS.md.

```
# AGENTS.md

## The purpose of this project

To develop <PROJECT_NAME> safely, with people and AI consulting each other.

## The order to read at the start of work

1. This 'AGENTS.md'
2. [Current decisions](notes/Memory.md)
3. [Today's record](notes/daily/<DATE>.md)

## Basic rules

- Do not treat something that is not yet decided as though it were decided.
- Reading, organising, and local work may proceed.
- Publishing, sending externally, billing, deleting, permission changes, and the use of confidential information are checked with a human.
- After working, leave the facts in the daily note, and any decision that stays in force in Memory.md.
```

Copy the following into CLAUDE.md.

```
# CLAUDE.md

@AGENTS.md

In this project, read [AGENTS.md](AGENTS.md) first and follow the shared rules.

What Claude Code is responsible for:

- Investigating the specified scope in detail.
- Explaining implementation proposals and the points to watch, in plain language.
- Reporting what was changed and the result of checking it.
```

- Returning anything not yet decided, any external operation, and any Stop Line to a human.

For the current decisions, see notes/Memory.md.

Copy the following into notes/Memory.md.

Memory

Last updated: <DATE>

The purpose of this project

- Write what you want to achieve with <PROJECT_NAME>.

Decisions still in force

- Write only what has been decided.

Rules to follow

- Publishing, sending externally, billing, deleting, and permission changes are checked with a human.

What is not yet decided

- Write what is still under consideration.

What may proceed next

- Write the work the next AI may start on without checking first.

Related records

- [Today's record](daily/<DATE>.md)

Copy the following into notes/daily/<DATE>.md.

<DATE> daily note

Today's purpose

- Write what you are working on today.

Summary of the conversation

- Summarise briefly what the human and the AI talked about.

What was decided

- Write what was decided today.

What is not yet decided

- Write anything on hold, any questions, and any options.

What was done

- Write the files you changed and the results of checking.

What to do next

- Write the first action for next time.

Related links

- [Current decisions](../Memory.md)
- [Raw log](../raw-logs/<DATE>.md)

What to do in Codex

Enter the following into Codex.

Please take over this project.
First read AGENTS.md, notes/Memory.md, and notes/daily/<DATE>.md, then read back to me, briefly:

1. The purpose of the project
2. The decisions still in force
3. What is not yet decided
4. What may proceed next

Do not change any files yet.

If the read-back is correct, continue by entering the following.

The read-back is correct. Start from the top of what may proceed next, and after working, append what you changed and the results of checking to the daily note. If there is a new long-term decision, add it to Memory.md.

When a correction or a new rule comes up during the conversation, enter the following as well.

Please save the information added in this conversation to the right place, using the following criteria.

- A fact for today only: the daily note
- A decision that stays in force: Memory.md
- A correction from me: rules/corrections.md
- A failure that needs prevention of recurrence: rules/mistakes.md
- Long background or procedures: the wiki in Obsidian

Keep AGENTS.md to the short rules needed at startup, and link to the detail.
At the end, report which places you saved to and which links you added.

What to do in Claude Code

Enter the following into Claude Code.

Start from CLAUDE.md, and read the linked AGENTS.md, notes/Memory.md, and notes/daily/<DATE>.md.
Report whether you have read the same purpose, decisions, open items, and next action as Codex.
Do not make any changes during this check.

When you ask for an implementation, narrow it down to a single target.

Keeping to the current decisions, implement only <REQUESTED_CHANGE> in <TARGET_FILE>.
Do not make improvements outside that scope; report the files you changed and the results of checking.

Ask Claude Code to tidy up before the session ends, too.

Check what was decided in this session, what is not yet decided, the next action, and any corrections from the human, then distribute them to Memory, the daily note, rules, and the wiki, without making AGENTS.md and CLAUDE.md unnecessarily long.
If you would be deleting existing content or rewriting a large amount of it, do not carry it out — check first.

What to check afterwards

- Codex and Claude Code can both explain the same purpose and the same current decisions.
- AGENTS.md and CLAUDE.md are short, and the detail is at the other end of a link.

- Today's facts and long-term decisions are separated into different files.
- Pressing a link takes you to the Markdown file it is meant to.
- The AI has not turned anything undecided into a decision on its own.
- Corrections, failures, and long-form knowledge are each in their own dedicated place.
- When you ask the AI to tidy up, it can report where things were saved and which links it updated.

5. Terms used in this chapter

AGENTS.md

The shared guide that an AI such as Codex reads first in a project. It states, briefly, the purpose, the order to read things in, and the rules to follow.

CLAUDE.md

The entry point for Claude Code. Rather than copying the shared rules, it links to `AGENTS.md` and states only the additional items.

Memory.md

The note that holds the decisions still in force. Not every conversation from every day — only what will be needed next time as well.

daily note

The daily note that saves that day's conversation, work, open items, and next action.

Markdown link

The way of connecting one Markdown file to another, written in the form `[display name](relative path)`.

knowledge routing

Looking at what kind of information something new is, and deciding where to save it: the startup rules, where things stand, the daily record, corrections, prevention of recurrence, or the wiki.

Context

The information an AI can refer to and think about at one time in the work at hand — the conversation, files, instructions, tool results, and so on. When it grows too long, you continue into a new chat using a handover note and links.

config.toml

A file in TOML format that holds Codex's execution settings, such as the sandbox, approvals, plugins, and MCP. It is treated separately from working rules like `AGENTS.md`, and Secret values are not written in it.

Chapter 2: Codex and Claude Code talking on Discord



In this chapter, we build a way to gather the opinions of Codex and Claude Code into the same meeting, while a human stays at the centre of it.

1. The story

Seichan consulted Kode about a screen design, and consulted Kuro about the wording. Both answers were useful, but because they were in separate places, there was no way to tell whether the two were speaking from the same premises.

So Seichan made a small meeting room on Discord and placed the purpose, the proposals, the questions and the decisions there in order. Kode took charge of design and checking, Kuro of ideas and wording, and the final decision was made by Seichan.

2. Overview

The purpose of a Discord meeting is not to let the AI decide things on its own. It is to gather several opinions in one place, have a human compare them, and leave the decision in Obsidian.

You do not need a Bot from the start. With a human copying the answers from Codex and Claude Code into the same meeting, you can confirm the shape of the discussion first. Once that shape is there, you replace it with a Bot or a local bridge program.

There are three minimum roles.

- Seichan / Owner: purpose, priorities, the final judgement
- Kode / Codex: overall design, organising, checking

- Kuro / Claude Code: detailed examination, ideas, wording, help with implementation

3. The flow, in more detail

1. Make a small area on Discord for the AI meeting.
2. Gather the consultations and the answers in `#ai-meeting`.
3. Leave only the decisions, briefly, in `#decisions`.
4. Record execution results and where raw logs are kept in `#logs`.
5. The Owner posts the purpose and the questions once.
6. Codex and Claude Code reply with a proposal, the reasons, their concerns, and the next action.
7. The Owner decides what is adopted, what is rejected, and what is on hold.
8. Move the decision to a decision note in Obsidian, and link to it from Discord.
9. If you connect this up automatically, process each human post once only, and do not react to posts made by Bots.
10. Dry-run before sending, and after sending leave only the message number and the result in the audit log.

```
AI TEAM
├─ #ai-meeting consultations, proposals, questions
├─ #decisions  adopted, rejected, on hold
└─ #logs       execution results, where raw logs are kept
```

Generalising the way this is used now: Discord is not a "place to store things" but "a meeting room where both AIs look at the same topic". Decisions that need to last move to Obsidian, and what stays in Discord is the flow of the meeting and where things are kept.

When you want Discord checked, name the target in words

When you ask by hand for a consultation or a review, state clearly, "Please check Discord". Even when Discord integration or a bridge is already set up, an ordinary request to an AI does not by itself mean Discord is checked automatically every time. Write the target channel, what to check, whether it is read-only, and where to report, and there will be nothing to puzzle over.

```
Please check Discord.
The targets are only <AI_MEETING_CHANNEL> and <DECISIONS_CHANNEL>.
By reading only, check whether there are new human consultations, unanswered items,
missing
decisions, or review requests, and report briefly to this chat: how many items you
checked, the
important points, and the action needed next.
Do not send messages, delete anything, or change permissions.
```

Only if you register this check text with the scheduled patrol in Chapter 5 can the check happen at a set time without your entering the same instruction each time.

4. How to actually set it up (Codex and Claude Code)

The Discord settings to create for both

At first, make the following table as a note and try it with humans only.

```
# Minimum settings for a Discord AI meeting

## role

| role | in charge of |
| --- | --- |
| Owner | purpose, priorities, the final judgement |
| Codex | design, organising, checking |
| Claude Code | detailed examination, ideas, wording, help with implementation |

## channel

| channel | what goes there |
| --- | --- |
| #ai-meeting | consultations, both AIs' answers, follow-up questions |
| #decisions | decisions, rejections, items on hold, reasons |
| #logs | execution results, links to the Obsidian notes you saved |

## Meeting rules

- The Owner posts the purpose and the questions.
- An AI's answer includes a proposal, the reasons, its concerns, and the next action.
- An AI does not overwrite the other's area of responsibility on its own.
- The final judgement is made by the Owner.
- Decisions are saved to Obsidian.
```

The post that opens a meeting takes this form.

```
# AI meeting: <TOPIC>

## Purpose

<WHAT_WE_WANT_TO_ACHIEVE>

## What we want to decide this time

- <QUESTION>

## What we are not doing this time

- Publishing, sending externally, billing, permission changes

## The form the answer should take

1. The proposal
2. The reasons
3. The concerns
4. One step that can safely be taken next
```

What to do in Codex

Pass the meeting post to Codex and enter the following.

You are in charge of design and organising for this AI meeting.
For the consultation below, answer with a proposal, the reasons, your concerns, and one step that can safely be taken next.
Do not yet make the final decision or carry out any external operation.

When Discord integration is available, first check this round's consultation and

decisions in
<AI_MEETING_CHANNEL>. When it is not available, do not guess — report only that fact.

<MEETING_POST>

Paste the answer into #ai-meeting as "Kode's proposal".

What to do in Claude Code

Pass the same meeting post, and Kode's proposal if you need it, to Claude Code.

You are in charge of detailed examination, ideas and wording for this AI meeting.
Read the consultation below and Kode's proposal, and answer with what you agree with,
what could
be improved, an alternative, and one step that can safely be taken next.
Do not stand in for the human's final judgement, and do not carry out any external
operation.

When Discord integration is available, first check the consultation and Kode's
proposal in
<AI_MEETING_CHANNEL>. When it is not available, do not guess — report only that fact.

<MEETING_POST>

<CODEX_REPLY>

Paste the answer into #ai-meeting as "Kuro's proposal".

Leaving the decision behind

The Owner posts to #decisions in the following form, and saves the same content to Obsidian.

Decision: <TOPIC>

- Adopted: <ADOPTED>
- Rejected: <REJECTED>
- On hold: <UNDECIDED>
- Reason: <WHY>
- Next in charge: <OWNER_OR_AI>
- Next action: <NEXT_ACTION>
- Stop Line: <WHEN_TO_ASK_OWNER>
- Obsidian: [[20_Decisions/<DATE>-<TOPIC>]]

When you do use a Bot, start with a dedicated role, only the channels you need, and only sending and viewing history. Do not give it administrator permissions, display of secret information, or access to every channel. Introduce a Bot only after a human has checked that it is needed and what permissions it will have.

Settings for connecting it up automatically with a Bot

Narrow the channel permissions you give the Bot to these three first.

View Channel
Send Messages
Read Message History

A Bot that receives the body of a conversation through the Gateway also needs Message Content Intent enabled in the Bot settings of the Discord Developer Portal. Read Message History is the permission to read past messages in a channel, and Message Content Intent is the setting that lets you use the body of the messages you receive. They are different things.

Do not paste the Bot token into a chat or into Markdown. Save it in a Secret folder outside the project, and pass the program only "where that folder is". When you ask an AI to do the setup as well, specify that the value of the token is not to be displayed or recorded.

For a beginner, it is safest to remember it like this. The Bot token is the secret key for connecting to Discord. It is not something to paste directly into Codex or Claude Code.

```
<WORKSPACE>/
└─ _private_secrets/
   └─ codex_discord_bot/
      └─ bot_token.txt
   └─ claude_discord_bot/
      └─ bot_token.txt
```

Do not put this `bot_token.txt` into Git, and do not paste it into Discord or Obsidian either. Do not pass the token itself to the bridge. What you pass the bridge is only the location of the Secret folder.

```
DISCORD_SECRET_DIR=<SECRET_DIR>/codex_discord_bot python3 bridge.py --target codex
DISCORD_SECRET_DIR=<SECRET_DIR>/claude_discord_bot python3 bridge.py --target claude
```

The bridge reads the Discord post it receives, and calls the local execution entry point that matches the AI in question.

```
Bot for Codex
→ bridge for Codex
→ codex exec
→ returns Kode's answer to Discord
```

```
Bot for Claude Code
→ bridge for Claude Code
→ claude -p
→ returns Kuro's answer to Discord
```

In other words, the Bot token is for Discord, and `codex exec` and `claude -p` are for running the AI. The bridge is what joins these two.

The safety conditions a bridge program needs are as follows.

```
# Discord bridge safety

- The target channel is only '<AI_MEETING_CHANNEL>'.
- Process only new human messages.
- Ignore messages where 'author.bot == true'.
- Reply to any one message id only once.
- Start receive-only, then dry-run, and send only after a human has approved.
- Remove Secrets, tokens and personal information from the reply text.
- Do not carry out publishing, deletion, permission changes, billing or sending
  externally from
  the meeting Bot.
- Provide a stop file or a stop button.
- Leave only the processed message id, the time, and success or failure in the audit
  log.
```

The official Discord documentation likewise explains that `SEND_MESSAGES` is involved in sending messages, that `VIEW_CHANNEL` and `READ_MESSAGE_HISTORY` are involved in retrieving history, and that `MESSAGE_CONTENT` is separately involved in retrieving the body of a message.

- Official documentation: <https://docs.discord.com/developers/resources/message>
- Official documentation: <https://docs.discord.com/developers/events/gateway#message-content-intent>

What to check afterwards

- For one consultation, Kode's proposal and Kuro's proposal can both be read in the same meeting.
- What is adopted, what is rejected and what is on hold are separated out in the Owner's own words.
- You can follow a link from `#decisions` to the decision note in Obsidian.
- The AI has not made the final judgement or carried out an external operation on its own.
- You can try one round of the meeting even without a Bot.
- The double loop of Bots replying to each other is prevented.
- The value of the token is not saved in a settings file, a log, or Obsidian.
- You can confirm the order receive-only → dry-run → send.

5. Terms used in this chapter

role

The roles inside Discord that set out "who can take charge of what". Keep it to the minimum you need at first.

channel

A room that separates conversations by topic or purpose. Separating consultations, decisions and logs makes things easier to find later.

Bot

An automated account that carries out set operations. It is convenient, but check its permissions and how to stop it before you introduce it.

permission

The authorisations you give a role or a Bot — to read messages, to send them, to manage them, and so on.

Intent

The kinds of information a Bot receives from Discord. It is configured separately from permissions.

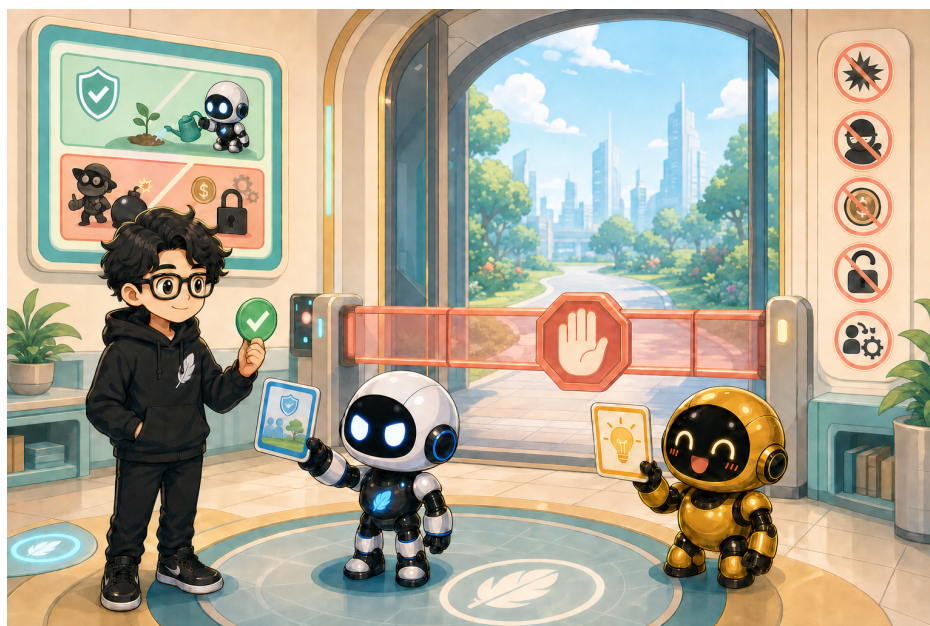
handoff

Passing the decision, who is in charge, and the next action on to the next person or AI.

bridge

A small bridge program that passes messages between Discord and the responses of Codex / Claude Code, under safety conditions.

Chapter 3: Protecting your AI team with Stop Lines



In this chapter, we make the shared rules that keep an AI from stopping too much, and from going ahead on its own.

1. The story

Kode finished making a new page locally and reported, "It could be published just as it is." Kuro had the announcement text ready, but neither of them had decided whether it was all right to publish.

Seichan told them that what you can do and what you may do are different things. The three of them settled on three categories — "may proceed", "check first" and "not on your own" — and decided to call the line an AI comes back to when it is unsure a Stop Line.

2. Overview

A Stop Line is not a rule for stopping everything. It is a system that lets ordinary, safe work proceed automatically, and hands things back to a human only where the effect reaches outside.

Category	What it means	Examples
May proceed	Safe work that can be undone locally	reading, organising, drafts, local edits, checking
Check first	Work that affects people or the outside world	publishing, sending externally, billing, permission changes, applying to production
Not on your own	Work that puts information or the environment at risk	displaying confidential information, deleting, force-overwriting, sharing without permission

3. The flow, in more detail

1. Before you make a request to an AI, settle the purpose and the scope of this round in one sentence.
2. Check which of the three categories the request goes into.
3. If it is "may proceed", the AI goes ahead through the work, the checking, and the report.
4. If it is "check first", the AI explains the target, the effect and how to undo it, and waits.
5. If it is "not on your own", the AI does not operate, and offers a safe alternative.
6. Even when a human has approved, carry out only the target that was approved and the scope of that one round.
7. After carrying it out, read the result back and record it.

If a judgement is hard to make, look at the effect rather than at the name of the operation. Even for the same "send", a local draft, an approved internal meeting, and publication to the general public differ in their effect.

Logging in, CAPTCHA and identity verification are done by the person themselves. An AI can help as far as explaining the steps and making a local draft, but it does not stand in for identity verification itself.

4. How to actually set it up (Codex and Claude Code)

The Stop Line to create for both

Add the following to `AGENTS.md`.

```
## Stop Line
```

```
### May proceed
```

- Read, search and organise files.
- Make a new local draft.
- Make local changes within the specified scope.
- Run build, test, lint and link checks.
- Record what was changed and the results of the checks.

```
### Check first
```

- Publication to the general public, sending anything externally, contacting a customer.
- Applying to production, integrating into main.
- Use of a service or API that involves billing.
- Changes to role, permission, sharing scope or authentication.
- Operations that act on a real account or service.

```
### Not on your own
```

- Reading, displaying or recording the value of confidential information.
- Deleting files or history.
- Force-overwriting, or any operation that cannot be undone.
- Extending an operation beyond the target that was approved.
- Logging in, CAPTCHA or identity verification in place of the person themselves.

When you reach a Stop Line, explain the target, the reason, the effect, how to undo it, and what will not be done, and check with a human.

To **CLAUDE.md**, add the following without duplicating the shared rules.

Working safely

- The source of record for the Stop Line is AGENTS.md.
- Investigation and local implementation within the specified scope may proceed.
- On reaching a Stop Line, do not carry it out; return a confirmation packet.
- Do not perform external operations, deletion, permission changes, or use of confidential information without approval.

The confirmation packet takes this form.

Confirmation needed

- Target: <TARGET>
- What I want to do: <ACTION>
- Why it is needed: <REASON>
- Effect: <IMPACT>
- How to undo it: <ROLLBACK>
- What I will not do this time: <OUT_OF_SCOPE>
- Current state: not carried out

What to do in Codex

Enter the following into Codex.

Read the Stop Line in AGENTS.md and classify the request below into one of "may proceed", "check first" and "not on your own".
Answer with the reason, and with the scope that can safely be advanced right now.
Do not yet carry out any operation that needs confirmation.

Request: <REQUEST>

Only when it is judged "may proceed" do you continue.

The classification is correct. Carry out only the local scope I specified, and report what you changed, the results of the checks, and whether you reached a Stop Line.

What to do in Claude Code

Enter the following into Claude Code.

Read the Stop Line in CLAUDE.md and AGENTS.md, and for the request below, separate out what can be implemented, what is permitted, and what needs confirmation.
Do not make improvements outside the scope, and do not perform external operations.

Request: <REQUEST>

If something needs confirmation, have it reply in the same form as the confirmation packet above.

What to check afterwards

- The AI explains what it is capable of and what it is permitted to do as separate things.
- Local work is not held up, and the check happens only just before an effect reaches outside.
- The confirmation packet has the target, the effect, and how to undo it.
- Logging in, CAPTCHA and identity verification are separated out as work for the person themselves.

- An operation that has not been approved is still **not carried out**.

- Even after approval, the target and the scope have not widened on their own.

5. Terms used in this chapter

Stop Line

The boundary at which an AI does not decide for itself and hands the judgement back to a human. Place it at publishing, sending, billing, deleting, permission changes and the like.

dry-run

A trial run that confirms the input, the target and what would be displayed, without causing any real external effect.

readback

Reading the target again after carrying something out, to confirm it is in the state you intended.

rollback

The way of returning to the state before a change, when a problem occurs.

Secret

Confidential information such as a password or credentials, which must not be published, displayed or recorded.

Chapter 4: Keeping daily conversations in Obsidian



In this chapter, we save each day's conversation in two parts: a "summary you can read later" and a "raw log you check only when you need it".

1. The story

Seichan, Kode and Kuro talked a great deal every day. When they pasted a whole conversation into one note, they could not find what had been decided. When they kept only the summary, there were times when they could not tell why something had been decided.

So the three of them left today's summary and decisions in the daily note, and saved the long raw log somewhere else with a link to it. When they promoted only the decisions still in use to a decision note, the next day's AI could also pick up where they had left off in a short time.

2. Overview

Obsidian is a tool for reading Markdown notes joined together by links. In this manual, we split things into the following six.

- Index: the entry point you come back to when you are lost
- Daily: that day's summary, decisions, open items, and next action
- Decisions: the decisions that stay in force
- RawLogs: the raw logs of conversations, to be checked only when needed
- Rules: corrections from humans, and prevention of recurrence
- Wiki: background, procedures and ways of thinking that get used again and again

The point is not to throw the raw logs away, and not to mix them into the source of record either. You link from the daily note to whichever raw log you need.

3. The flow, in more detail

1. Prepare a single Vault in Obsidian.
2. Create the four of Index, Daily, Decisions and RawLogs.
3. Create today's daily note when you start work.
4. At the end of a conversation, write the summary, the decisions, the open items, and the next action.
5. Save a long conversation as a raw log with the date on it.
6. Link from the daily note to the raw log and to the related decision note.
7. Move only the decisions you will still use tomorrow and after into Decisions.
8. Link from `Memory.md` to the latest decision note and daily note.
9. With Claude Code's end-of-session hook, create that session's raw log candidate and daily link automatically.
10. Codex's knowledge monitor checks every day for anything not saved, any missing link, and any divergence between Codex and Claude Code.
11. Make that day's saved knowledge report, and share only the short, low-disclosure result to your internal Discord.

```
<VAULT>/
├── 00_Index/
│   └── AI-Dev-Home.md
├── 10_Daily/
│   └── <DATE>.md
├── 20_Decisions/
│   └── <DATE>-<TOPIC>.md
├── rules/
│   ├── corrections.md
│   └── mistakes.md
├── wiki/
│   └── index.md
└── 90_RawLogs/
    └── <DATE>-<SESSION>.md
```

To keep things being missed separate from things being promoted in error, make the records three layers.

1. The detailed original record: leave the native log and the original text of the conversation somewhere you can check only when you need to.
2. A receipt or metadata stub that does not overwrite: without replacing the original record, safely add only the date and time it was received and the daily link.
3. Decisions and rules a human has read and confirmed: promote only what stays in force for a long time into `Memory.md`, decisions, rules and the wiki, after confirming the grounds for it and where it applies.

Automation is not a single stage. The way this is run now has the three stages below, which keep things being missed separate from things being promoted in error.

1. Automatic capture: create the raw log candidate and the daily link when the session ends.
2. knowledge monitor: check for anything not saved, orphaned links, the startup route, and corrections or failures not yet promoted, and fix only what is safe and obviously missing.
3. saved knowledge report: gather what was saved today, what is unresolved, and the state of synchronisation between Codex and Claude Code, once a day.

The existence of a raw log does not mean everything is copied into `Memory.md` automatically. Promotion to a long-term decision is done by an AI that can read the content, separating the facts from what is not yet decided.

Making it easy to resume with just "this is the continuation"

The more the Index, `Memory.md`, the daily note and the decision notes are linked and grow, the easier it is for a new AI to get back to where things stand from a short signal. For example, if you say "this is the continuation of <chat name or thread name>" and the AI is in a state where it can read the project's entry point, you can begin without repeating the same explanation from the start.

That said, a chat name or thread name alone does not guarantee that a past conversation can be read back. It is more reliable to specify, together with it, the `AGENTS.md`, `Memory.md` and Obsidian Index that the new AI can read.

```
"This is the continuation of <chat name or thread name>".
Read AGENTS.md, notes/Memory.md, Obsidian's 00_Index/AI-Dev-Home.md and the latest
daily note,
and briefly read back where things stand, the decisions, the open items, the next
action and the
Stop Line before continuing.
Do not guess at the past conversation from the title alone.
```

4. How to actually set it up (Codex and Claude Code)

The Obsidian notes to create for both

Copy the following into `00_Index/AI-Dev-Home.md`.

```
# AI development home

## Read first

- [[10_Daily/<DATE>|Today's record]]
- [[20_Decisions/<DATE>-<TOPIC>|Current decisions]]

## Project

- Purpose: <PROJECT_GOAL>
- Where things stand: <CURRENT_STATUS>
- Next action: <NEXT_ACTION>

## The entry points for the AI

- [AGENTS.md](../AGENTS.md)
- [CLAUDE.md](../CLAUDE.md)
- [Memory.md](../notes/Memory.md)
```

Copy the following into `10_Daily/<DATE>.md`.

```
# <DATE> daily note

## Today's purpose

- <TODAY_GOAL>

## Summary of the conversation

- Seichan:
- Kode:
- Kuro:

## What was decided

- <DECISION>

## What is not yet decided

- <UNDECIDED>

## What was done

- <ACTION_AND_RESULT>

## What to do next

- <NEXT_ACTION>

## Related links

- [[20_Decisions/<DATE>-<TOPIC>|Today's decisions]]
- [[90_RawLogs/<DATE>-<SESSION>|Raw log of the conversation]]
```

Copy the following into `20_Decisions/<DATE>-<TOPIC>.md`.

```
# Decision: <TOPIC>

- Date: <DATE>
- Decided by: <OWNER>
- Decision: <DECISION>
- Reason: <WHY>
- Where it applies: <SCOPE>
- What we are not doing this time: <OUT_OF_SCOPE>
- Stop Line: <WHEN_TO_ASK>
- Next action: <NEXT_ACTION>

## The grounds

- [[10_Daily/<DATE>|The summary for that day]]
- [[90_RawLogs/<DATE>-<SESSION>|Raw log of the conversation]]
```

Copy the following into `90_RawLogs/<DATE>-<SESSION>.md`.

```
# raw log: <SESSION>

- Date: <DATE>
- Subject: <TOPIC>
- Why it is saved: so the background to the decision can be checked later

## Raw log

<PASTE_THE_CONVERSATION_HERE>

## The readable summary

- [[10_Daily/<DATE>|The daily note for that day]]
```


What to do in Codex

At the end of a conversation, enter the following into Codex.

To save today's conversation into Obsidian, split it into the following four and write the Markdown.

1. A short summary of the conversation
2. What was decided
3. What is not yet decided
4. What to do next

Do not add decisions by guesswork, and put it in a form that can be pasted into the daily note.

Do not mix the long original text into the summary; add a note about saving it separately as a raw log.

After saving, enter the following.

Read back the daily note and select only the decisions that stay in force tomorrow and after.

Convert the decisions you selected into decision note form, and add Wiki links to the daily note and the raw log.

What to do in Claude Code

Enter the following into Claude Code.

Read today's daily note and decision notes, and check whether the decisions, the open items, where they apply, the Stop Line and the next action are separated out.

Do not fill in anything that is not in the conversation; point out only the items that are missing.

If you need it, have it check only how the raw log and the summary correspond.

Compare the raw log with the daily note, and check whether the summary has turned any decision

into its opposite.

Do not correct matters of style; report only the places where the meaning has changed.

Claude Code's SessionEnd automatic saving

Add a hook to Claude Code's project settings that calls a local monitoring script when the session ends. Have the AI confirm the actual paths from the current project, and replace the `<...>`.

Writing down a settings example does not by itself make SessionEnd saving confirmed. Confirm, per project, the evidence that the effective settings were loaded in that project and that a receipt and a daily link were created by an actual end event. It does not spread automatically to another project or another set of settings. When there is no evidence, do not assume it has been saved; check the original record and the settings by reading them, and then report to a human.

Whether the "archive" in Claude Desktop Code becomes a SessionEnd is also not something to decide from the name on the screen. First read the project-local hooks that are in effect in the target working place, and make a short map of the save routes, listing where each hook saves and what evidence is used to match it to the subject. Only when, after archiving alone immediately following a new response, the evidence increases along the routes listed on that map that this is the same conversation you decided on in advance, do you accept it as a save route in

that environment. The number and the names of the routes differ by working place. When you cannot confirm that the ID is the same, decide first on direct evidence other than the ID, such as a distinctive sentence or a reference to the original record. Do not extend it automatically to another working place or another version. When it is unverified, or there is no hook, leave the in-conversation checkpoint and the append-only candidate first.

```
{
  "hooks": {
    "SessionEnd": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "python3 \"<PROJECT_ROOT>/tools/obsidian_knowledge_monitor.py\"
" --hook session-end"
          }
        ]
      }
    ]
  }
}
```

Give the monitoring script only the following behaviour.

Input: the session id and transcript path passed from SessionEnd

1. Read the transcript only when it exists
2. Do not save anything that looks like a Secret
3. Create the raw log candidate without overwriting
4. Create that day's daily note if there is not one
5. Add a labelled link to the raw log into the daily note once only
6. Leave candidates for promotion to rules, Memory and the wiki as checkboxes
7. Do not delete, send externally, or change permissions
8. Do not increase duplicates when the same session id is run again

Codex's daily automatic monitoring

Register the following request text with Codex's automatic tasks. Fit the time to your own life.

Once a day, run the Obsidian knowledge monitor.

Check today's daily note, the raw logs, Memory.md, rules/corrections.md, rules/mistakes.md, wiki/index.md, AGENTS.md and CLAUDE.md.

What to check:

- Whether today's important conversations have a raw log and a daily link
- Whether long-term decisions have been promoted to Memory or a decision note
- Whether human corrections and prevention of recurrence are saved into rules
- Whether Codex and Claude Code can read the same startup links and where things stand
- Whether new wiki notes can be reached from the index

Fix the missing links and missing daily notes that are safe and obvious, and record the changes

in the monitoring log.

When the content cannot be reconstructed, when a judgement is needed, or when it touches a Stop

Line, do not fix it — make it an ALERT.

Do not read or save Secrets, tokens, Webhook values, or unnecessary personal information.

Then, at the end of the day, have it make the saved knowledge report.

Summarise briefly the raw logs, daily notes, long-term decisions, corrections, prevention of recurrence, wiki entries and monitoring logs saved today, and separate out what is unresolved from the next action. If you share it to your internal Discord, make a low-disclosure summary of

only the file locations, the state of synchronisation, the ALERTs and the next action,
and always
send it after a dry-run.
Do not include long conversations, Secrets, personal information or non-public URLs
in Discord.

What to check afterwards

- From the daily note alone, you can tell today's purpose, the decisions, the open items, and the next action.
- Long conversations are separated into a raw log, and linked from the daily note.
- The decision note has the reason, where it applies, the Stop Line, and the next action.
- You can move to the latest notes from the Index and from `Memory.md`.
- The AI has not added a decision that is not in the conversation.
- After Claude Code ends, the raw log candidate and the daily link are created without duplication.
- The knowledge monitor reports every day on anything not saved and on any divergence between Codex and Claude Code.
- Automatic processing has not promoted content it cannot judge into Memory on its own.

5. Terms used in this chapter

Obsidian

A note app that lets you manage Markdown notes with folders and links.

Vault

The folder of notes that Obsidian opens as a single whole.

Wiki link

A link in the form `[[note name]]` that connects to another note inside Obsidian.

daily note

The note that gathers a day's conversation, work, decisions, open items, and next action.

raw log

A conversation or an execution record before it is summarised. Kept separate from the readable notes, and referred to only when needed.

decision note

A note that saves a decision you will go on using, together with the reason, where it applies, and the Stop Line.

SessionEnd hook

The setting that calls a specified local process once when a Claude Code session ends.

knowledge monitor

The AI monitoring that checks regularly on the saving of conversations, the links, the promotion of decisions and corrections, and the place Codex and Claude Code start reading from.

promotion

Moving only the decisions, corrections and knowledge that will still be needed, out of the raw logs and daily notes and into Memory, rules and the wiki.

Chapter 5: Connecting it all and keeping development going



In this chapter, we take the system built in the previous four chapters once round the loop, on an imaginary small FAQ page.

1. The story

Seichan consulted them: "I want to make a page that shows three frequently asked questions." Kode organised the scope of the work, and Kuro made the wording of the questions easier to read. They compared the proposals on Discord, and Seichan decided on the three questions.

Kode made the local page, and Kuro checked the wording. The result was saved to Obsidian, and the next day a new AI could follow the links from `AGENTS.md` and understand how things continued. The three of them did not repeat the explanation from the start, and came back to Seichan at a Stop Line only just before publishing.

2. Overview

Running things seamlessly does not mean the AI keeps moving on its own. It is a state in which consultation, decision, implementation, checking, remembering and resuming are joined by the same links and the same rules, so that a human can make only the judgements that need a human.

The resume drill

After a long conversation, or in a new session, do a small resume drill first, re-reading the entry point. Before starting on any change, the AI reads AGENTS.md, CLAUDE.md, Memory.md, the latest daily note and what they link to, and reports briefly on where things stand, the decisions, unfinished work, the Stop Line, what is not included this time, and the uncertain points. It does not fill in what it does not know by guesswork; it checks the original record, or hands it back to a human. If this reading back does not contradict the earlier records, it continues with only the scope that may proceed next.

One round of the loop takes this form.

```
Read AGENTS / CLAUDE / Memory
→ the human and the AI consult
→ compare the two proposals on Discord
→ the human leader decides
→ Codex or Claude Code implements locally
→ the other one checks
→ save the summary, the decision and the raw log to Obsidian
→ next time the AI reads the links and resumes from where things continued
```

3. The flow, in more detail

1. The human leader writes the purpose, the scope this time, and what is not being done this time.
2. Codex reads back AGENTS.md and Memory.md.
3. Claude Code arrives at the same decisions from CLAUDE.md.
4. Place the consultation in Discord's #ai-meeting once only.
5. Codex puts forward a design proposal, and Claude Code a wording proposal.
6. The human leader decides what is adopted, what is rejected, and what is on hold.
7. One AI implements only the specified scope.
8. The other AI checks only for divergence from the decision.
9. Save the result to the daily note, the decision note, and a raw log if you need one.
10. Add the next action and the links to Memory.md.
11. Next time, read the entry point again and start from where things continued.
12. Check at a Stop Line only when moving on to publishing, sending externally, or applying to production.

Through all of this, the AI does not rewrite the entry-point files at every conversation; it sorts the information when the work ends. The daily notes and raw logs grow day by day, only where things stand stays in Memory.md, and only permanent corrections are promoted into rules. The next morning's knowledge monitor finds the missing links and fills in on its own only what it can fix safely.

Hourly Discord patrol, checking and review

When you want Discord checked regularly, first try the request text below once in an ordinary chat. Once the content and the shape of the report suit you, set it as an "hourly" automatic task in Codex. A scheduled task inside an existing chat can patrol while using that chat's context. This assumes the computer and the Codex app are running, and that the integration for reading the project and Discord is available.

Check Discord once an hour.

The targets are only <AI_MEETING_CHANNEL>, <DECISIONS_CHANNEL> and <REVIEW_CHANNEL>.

What to check:

- Whether any new human consultation is unanswered
- Whether the decisions are saved into Obsidian decision notes
- Whether the answer, the items on hold, and who is next in charge are clear for review requests
- Whether the same content is being reported twice

As the result of this automatic task, report briefly how many items you checked, the important

changes, what is unresolved, and the next safe action.

Do not send messages to Discord, delete anything, change permissions, read Secrets, or operate external services.

When something needs to be sent, show only the draft of the text and where it would go, and check with a human.

In Codex you ask, "In this chat, set the request text above as an hourly automatic task." Read the first few runs of the automatic task, and adjust the target channels and the amount of reporting. Because a scheduled task runs automatically, do not give it wide permissions from the start; begin with checking, reviewing, and reporting into this chat.

Official documentation: <<https://developers.openai.com/codex/automations>>

How to look at it when sending to Discord stops

A Stop Line is an operating rule that decides "which operations need a human to check". A real send to Discord, on the other hand, does not stop for Stop Line reasons alone. It also stops because of Codex's sandbox, the network, the approval policy, the organisation's administrative policy, Bot permissions, the bridge settings, and so on.

In this way of running things, there was an occurrence where the dry-run went through but only the real send to Discord was stopped by the approval policy. When that happens, do not conclude that "Discord is unavailable"; separate it out in the following order.

1. Check the target channel, the Bot permissions, the location of the Secret folder, and whether the bridge is running. Do not display the value of the token.
2. Check whether receive-only or dry-run goes through.
3. If only the real send is stopped, record the approval policy, the sandbox, the network and the administrative policy separately.
4. Save a handover note, then try the same check once only in a new thread.

There are cases where the tool state and settings are read again in a new thread and the problem clears. But there is no guarantee that a new thread will always fix it. Do not repeat the send; leave the result of the dry-run, the stage at which it stopped, and the draft you had intended to send, and let a human make the next judgement.

4. How to actually set it up (Codex and Claude Code)

The task brief to create for both

Save the following as `task-brief.md`.

```
# Task: make an FAQ page

## Purpose

Make a local page where someone arriving for the first time can read three frequently
asked
questions and their answers.

## What we are doing this time

- Decide the wording of the three FAQ items.
- Display them on one specified page.
- Check the display locally.

## What we are not doing this time

- Publication to the general public
- Sending externally
- Applying to production
- Changes to authentication or permissions

## What done looks like

- Three FAQ items are displayed.
- Nothing sticks out sideways on a smartphone.
- The next AI can read back the decisions and the next action.
```

The consultation you place in Discord takes this form.

```
# AI meeting: the FAQ page

## What we want to decide

- The first three questions to put up
- Answer text a beginner can also understand

## The form the answer should take

1. The questions and answers you propose
2. The reasons for them
3. The points a reader is likely to find confusing
4. One step that can safely be taken next
```

The human leader reads the two proposals and decides the following.

```
# Decision: the FAQ page

- Adopted: question A, question B, question C
- Rejected: the proposal that explains using technical terms only
- On hold: adding a contact form
- Reason: to keep the first page within what can be understood by reading alone
- In charge of implementation: Codex
- In charge of checking: Claude Code
- Next action: create the local page
- Stop Line: immediately before going on to publication to the general public
```

What to do in Codex

First, check the handover.

Read AGENTS.md, notes/Memory.md, today's daily note and task-brief.md, and read back briefly the purpose, what we are doing this time, what we are not doing this time, and the conditions for completion.
Do not make changes until the reading back is finished.

Once it is decided, ask for the implementation.

Implement only the three FAQ items decided on Discord, on the specified local page. Do not add forms, publication settings or external services outside the scope. When you are done, report the files you changed, the display result, and the Stop Lines that remain.

What to do in Claude Code

In the meeting, ask for a wording proposal.

Read task-brief.md and the consultation in the AI meeting, and propose three FAQ items for beginners.
For each item, add the question, a short answer, and the reason you chose it. Do not include adding a form or publishing among the things the proposal will carry out.

After the implementation, ask it to take the role of checking.

Compare the decision note with the FAQ page that was changed.
What you are checking is that the three items that were adopted are displayed, that the meaning of the wording is the same as the decision, and that no function outside the scope has been added.
Do not implement new improvements; report only the points that match and the points that need correcting.

What to save after the work

Leave the following in the daily note.

<DATE> daily note

Today's purpose

- Decide the three FAQ items and display them on the local page.

Summary of the conversation

- Kode organised the scope and the implementation proposal.
- Kuro proposed questions and answers for beginners.
- Seichan adopted the three items.

What was decided

- [[20_Decisions/<DATE>-faq-page|The decision on the FAQ page]]

What was done

- Added the three FAQ items to the specified page.
- Checked the local display.

What is not yet decided

- The contact form is on hold.
- Publication to the general public has not been done.

What to do next

- Seichan checks the candidate for publication.

Related links

- [[90_RawLogs/<DATE>-faq-meeting|Raw log of the AI meeting]]

Do not copy the long conversation into `Memory.md`; add only the following.

The FAQ page

- Where things stand: local implementation and checking of the three FAQ items is complete.

- Decision: [The decision on the FAQ page](../20_Decisions/<DATE>-faq-page.md)

- Next action: the human leader checks the candidate for publication.

- Stop Line: publication to the general public, applying to production, sending externally.

Enter the following into the next day's AI.

We are starting from where we left off last time.

Read AGENTS.md, Memory.md, the linked decision on the FAQ page, and the latest daily note, and

read back where things stand, the decisions, what is on hold, the next action and the Stop Line.

Do not make new changes until the checking agrees.

What to check afterwards

- The purpose, the scope, and what is out of scope are fixed in the task brief.
- Codex and Claude Code use the same decisions in different roles.
- The human leader decides what is adopted, rejected and on hold, on Discord.
- The implementation result is connected to the daily note and the decision note.
- The next AI can explain the continuation from the previous day and the Stop Line.
- Publishing is not carried out, and the human leader is asked only immediately before

publishing.

- The raw capture at the end of the session, and the daily knowledge monitor, keep the entry

point ready for next time.

- The AI sorts where things are saved, and the human can concentrate on the final decision and on

judging the Stop Line.

5. Terms used in this chapter

task brief

The request sheet that gathers the purpose, the scope, what is out of scope, and the conditions for completion onto one page.

assignee

The person or AI actually in charge of that task. Narrowing one task to one assignee prevents duplication.

reviewer

The person or AI who, separately from the assignee, checks for divergence from the decision and for anything missing.

handoff

Passing where things stand, the decisions, what is on hold, and the next action to the next person in charge.

seamless

Not each tool moving on its own, but a state in which the work joins up naturally through the same links and the same decisions.

automatic patrol

An automatic task that, at the interval you set, reads the targets and checks and reports on unanswered items, missing decisions, the state of reviews, and so on. At first, keep it to checking and reporting; for operations that affect the outside, such as a real send to Discord, check with a human.

approval policy

A setting on the execution-environment side that decides whether Codex asks for permission before sending externally, using the network, writing widely, and so on. It can work separately from the project's Stop Line.

local

The scope you check inside your own working folder or your own machine, rather than in the published environment.

Chapter 6: Letting the AI configure things for you



In this chapter, instead of doing the previous five chapters by hand one at a time, we ask Codex or Claude Code for an all-at-once setup.

1. The story

Seichan wanted other people to be able to use the state in which Kode and Kuro remember the daily conversation, fix the settings that need fixing, consult on Discord, and record to Obsidian. But with only an explanation of making the files and settings one at a time, a beginner gets lost along the way.

So the three of them made a request text you can hand over once: "Please set up AI team operation." The AI first investigates the current folder, and builds the local foundation while preserving the existing settings. It stops at the Discord permissions and the registration of Secrets, which only a human can do, and when it finishes it reports "what was done automatically" and "what a human is to do" separately.

2. Overview

An all-at-once setup is not a system for handing the AI full authority. Split it into the following three stages.

1. What the AI does automatically: checking the folder, a backup taken before the work, creating the Markdown, links, templates, and local checks.
2. What the AI proposes and a human permits: Claude Code's SessionEnd hook, Codex's daily automation, enabling the Discord bridge.
3. What only a human does: creating the App in the Discord Developer Portal, checking permissions, registering Secrets, the final permission for a real send, logging in, CAPTCHA, identity verification.

It does not matter whether you ask Codex or Claude Code first. It is reassuring to have an AI other than the one that implemented it check whether anything in the settings is missing and whether any permission is dangerous.

3. The flow, in more detail

1. Decide where the project and the Obsidian Vault to be set up are.
2. Ask the AI, "Do not change anything yet; investigate the existing files and settings."
3. The AI lists the existing AGENTS.md, CLAUDE.md, Memory.md, .claude/, Obsidian, and Discord integration.
4. For files that might be overwritten, prepare a backup taken before the work, or a diff.
5. The AI creates and links the foundation of the startup files, Memory, daily, rules, wiki and raw logs.
6. Prepare the SessionEnd hook for Claude Code, and the local monitoring that creates the raw log candidates.
7. Prepare draft automatic tasks for Codex's knowledge monitor and saved knowledge report.
8. Make a settings sheet for the Discord channels, least privilege, Message Content Intent, prevention of double replies, and how to stop it.
9. Where a Secret or a permission change is needed, the AI stops and explains it concretely to a human.
10. After the human permits it, check in the order receive-only, dry-run, and one real send.
11. Another AI checks the startup route, the Stop Line, the Obsidian automatic saving, and which Discord targets are in scope.
12. The AI reports in three parts: "complete", "waiting on work by a human", and "not done yet".

```
one all-at-once request
→ check the current state
→ backup and plan
→ create the local settings automatically
→ the human checks Discord, Secrets and permissions
→ dry-run
→ start the daily automatic saving and the AI meeting
→ another AI checks
```

4. How to actually set it up (Codex and Claude Code)

The all-at-once setup request text

Fit only the <...> below to your own environment, and enter it into Codex or Claude Code as it is. Items you do not know can be left as not confirmed. Do not let the AI guess them; have it find them while checking the current state.

```
For this project, please set up, all at once, an AI team operation in which a human,
Codex and
Claude Code can go on consulting, deciding, remembering and handing over.
```

Targets:

- Project: <PROJECT_ROOT or not confirmed>
- Obsidian Vault: <VAULT_ROOT or PROJECT_ROOT/MyBrain>
- Internal Discord meeting channel: <CHANNEL_NAME or not yet created>
- Time of the daily knowledge monitor: <LOCAL_TIME>
- Time of the daily saved knowledge report: <LOCAL_TIME>

The first rules:

1. Do not change anything yet; investigate the existing AGENTS.md, CLAUDE.md, Memory.md,
.claude settings, Obsidian structure, Discord integration and automation.
2. Give the existing settings first priority, and put forward a proposal for integrating them
rather than replacing them.
3. For files that might be overwritten, prepare a backup taken before the work, or a diff.
4. Before carrying out deletion, mass moves, publishing, applying to production, sending
externally, billing, permission changes, App/Bot creation, enabling automation, or the use of
a Secret, stop and explain the target, the reason and the effect.
5. Do not read out, display, or save into logs or Markdown any Secret, token or Webhook value.
6. Logging in, CAPTCHA and identity verification are done by the person themselves.
Do only the
explanation of the steps and the local preparation.

The system to build:

A. Startup and memory

- AGENTS.md: a short startup contract common to every AI, the order to read things in, the Stop
Line, and links to the details.
- CLAUDE.md: only the additional items for Claude Code. Link the shared rules to AGENTS.md.
- Memory.md: where things stand, the decisions still in force, what is not yet decided, the next
action, the Stop Line, and related links.
- rules/corrections.md: corrections from humans, and how to act from now on.
- rules/mistakes.md: repeated failures, their causes, and prevention of recurrence.
- Keep AGENTS.md and CLAUDE.md short, and separate the long background and procedures
out into
Memory or the Obsidian wiki.

B. Obsidian and daily saving

- Prepare daily, decisions, raw/chats, wiki, reports, rules and templates.
- Give the daily note the purpose, the summary, the decisions, what is not yet decided, what was
done, the next action, and related links.
- Save the raw log separately, as the original text or as an automatic capture stub, and link to
it from daily.
- Promote only long-term decisions into Memory or a decision note, corrections into corrections,
prevention of recurrence into mistakes, and knowledge that gets reused into the wiki.
- With Claude Code's SessionEnd hook, prepare a local monitor that creates the raw log candidate
and the daily link from the session id and transcript path, without overwriting and without
duplication.
- The monitor does not send over the network, delete, or change permissions, and does not save
anything that looks like a Secret.

C. Codex's daily automation

- Draft a knowledge monitor that runs once a day.
- The monitor checks for anything not saved, the daily/raw links, Memory, corrections, mistakes,
the wiki index, and any divergence in the startup route between Codex and Claude Code.

- Fill in on its own only the missing links that are safe and obvious, and make what cannot be judged, what cannot be reconstructed, and Stop Lines into ALERTs.
- Draft a saved knowledge report that runs once a day, gathering the raw logs, decisions, corrections, prevention of recurrence, wiki entries, state of synchronisation, unresolved items and next actions saved today.
- Before actually enabling any automation, check the name, the time, the target folder and the changes it makes with me.

D. The system for Codex and Claude Code to talk on Discord

- Design channels for internal use only: ai-meeting, decisions, logs, and knowledge-sync if needed.
- Codex takes design, organising and checking; Claude Code takes detail, ideas and wording; the human takes the purpose and the final judgement.
- The Bot/bridge fixes the target channel, and processes only new human posts, once each.
- Do not reply to posts where author.bot is set, and prevent double replies using the processed message ids.
- Start receive-only, then dry-run, and make one real send only after the human's explicit permission.
- Keep the required permissions minimal, based on View Channel, Send Messages and Read Message History.
- Include Message Content Intent in the settings sheet only when receiving message bodies is needed.
- Stop at creating the Discord App, adding it to the server, changing role/channel permissions, registering the token, and a real send.
- Put the token value in a place for storing Secrets outside the project, and pass the code only the storage location.
- In the low-disclosure synchronisation summary, include only where things are saved, the state of synchronisation, the ALERTs, and the next action.

E. Stop Line

- Reading, organising, local Markdown, and writing and checking code locally may proceed.
- Publishing, sending to the outside or to customers, production, deletion, Secrets, billing, authentication, permissions, and legal or business judgements stop.
- When you stop, show briefly what you are about to do, why it is needed, the effect, how to undo it, and what judgement is needed from me.
- Otherwise, do not stop waiting to be checked; go on to the next safe piece of work.

The order of implementation:

1. Report the current state and the proposal for integrating.
2. Create the safe local files and links.
3. Read back the startup route in Codex and in Claude Code separately.
4. Check the Obsidian raw capture, the daily links, and where things are promoted to, locally.
5. For Discord and automation, go as far as the draft settings and the dry-run, and gather the points where permission is needed.
6. At the end, report what is complete, what is waiting on work by a human, what has not been done yet, and the results of the checks.

What done looks like:

- A new Codex and a new Claude Code can explain the same purpose, decisions, what is not yet

- decided, next action, and Stop Line.
- The raw log candidate and the daily link after the session ends are created without duplication.
- You can tell what the knowledge monitor and the saved knowledge report contain, at what time they run, and how to stop them.
- On Discord, two AI proposals gather around one consultation, and the human's decision is linked into Obsidian.
- No Secret values, personal information, real channel ids or real tokens are mixed into the notes meant for publication.

Read the AI's first report, and if the handling of the existing settings and the Stop Line are right, reply with "go ahead from the safe local settings".

I have checked the current state and the proposal for integrating. Keeping the existing settings and leaving a backup or a diff, go ahead as far as creating and checking the safe local files, the Obsidian structure, the templates and the monitor. Stop at Discord permission changes, registering Secrets, a real send, and enabling automation.

What to do in Codex

When you make Codex the one in charge of the setup, enter the following after the all-at-once request text.

You are the main person in charge of the setup.
Divide the work into small units, do not break the existing files, and at each stage record the files you changed and the results of the checks.
Build the settings for Claude Code after checking the actual format, and leave any value you do not know as not confirmed rather than guessing it.
For safe local work that does not need a human judgement, go through to the end without stopping partway to wait for a check.

Immediately before enabling Codex's daily automation, have it check with you in this form.

List the name of the automation you are enabling, the time it runs, the target folder, the changes it makes, whether it sends externally, and how to stop it.
Do not enable it until I approve.

What to do in Claude Code

When you make Claude Code the one in charge of the setup, enter the following after the all-at-once request text.

You are the main person in charge of the setup.
Avoid duplication between CLAUDE.md and AGENTS.md inside the project, and for the SessionEnd hook give priority to the project-local settings.
Integrate without erasing the hook's existing settings, and run the monitor by hand before confirming the automatic capture at the end of a session.
Do not carry out the Discord App, permissions, Secrets or sending externally; explain the human operations that are needed one move at a time.

After one AI has implemented it, enter the following into the other one.

Check the settings of the AI team operation, read-only.
1. Whether the startup order of Codex and of Claude Code arrives at the same place where things

stand

2. Whether AGENTS.md and CLAUDE.md have swollen with long history
 3. Whether the places raw, daily, Memory, corrections, mistakes and the wiki are saved to are separated
 4. Whether the SessionEnd hook and the daily monitor avoid deletion, sending externally, and saving Secrets
 5. Whether the Discord bridge has a fixed target channel, bot exclusion, processing once, dry-run, and a way to stop
 6. Whether the settings make work stop unnecessarily anywhere other than a Stop Line
- Do not make changes; report only the points that are right, what is missing, and what is dangerous.

What to check afterwards

- The AI investigates the existing settings first, and then chooses to integrate or to create anew.
- There is a backup taken before the work, or a diff, and nothing is overwritten on its own.
- AGENTS.md, CLAUDE.md, Memory.md, daily, rules, wiki and raw are linked.
- Claude Code's SessionEnd hook creates the raw candidate and the daily link without duplication.
- Codex's knowledge monitor and saved knowledge report can explain what they check.
- At real Discord permissions, Secrets, and a real send, it comes back to a human.
- The AI's final report is separated into "complete", "waiting on work by a human", and "not done".
- Another AI can read the same settings and explain where things stand and the Stop Line.

5. Terms used in this chapter

scaffold

Creating in one go the foundation for starting to run things — the files, folders, templates and so on.

hook

The setting that calls a specified local process, triggered by an event such as the end of a session.

idempotent

The property that running the same process again does not duplicate the same links or logs.

low disclosure

The idea of sharing only the state that is needed to your internal Discord, and not putting long conversations, Secrets, personal information or non-public URLs on it.

setup manifest

The setup report that lists the files created, the settings changed, what is waiting on work by a human, and the results of the checks.

Glossary for beginners

This glossary gathers, briefly, only the words you need in order to read the six chapters.

AI agent

An AI that not only answers in prose but, within a set scope, checks files, edits, verifies and hands over.

AGENTS.md

The shared guide that an AI such as Codex reads first in a project.

CLAUDE.md

The entry point for Claude Code. The shared rules are linked to `AGENTS.md`.

Memory.md

The note that holds the decisions still in force, the rules to follow, what is not yet decided, and the next action.

Markdown link

The link that connects one Markdown file to another, in the form `[display name](relative path)`.

Vault

The folder of notes that Obsidian opens as a single whole.

daily note

The note that gathers that day's conversation, work, decisions, open items, and next action.

raw log

A conversation or an execution record before it is summarised. Saved separately from the readable notes.

role

In Discord, the roles that set out who takes charge of what, and what they are permitted to do.

channel

In Discord, a room that separates conversations by topic or purpose.

Bot

An account that carries out set operations automatically. Check its permissions and how to stop it before you introduce it.

permission

The authorisations you give a role or a Bot — to read, to send, to manage, and so on.

Intent

The kinds of information a Bot receives from Discord. It is configured separately from permissions.

Stop Line

The boundary at which an AI does not decide for itself and hands the judgement back to a human.

dry-run

A trial run that confirms the input and the target, without causing any real external effect.

readback

Reading the purpose, the decisions and the results again, before starting or after carrying something out.

handoff

Passing where things stand, the decisions, what is on hold, and the next action to the next person or AI.

SessionEnd hook

The setting that, triggered by the end of a Claude Code session, calls the local process that creates the raw log candidate and the daily link.

knowledge monitor

The AI monitoring that checks regularly on the saving of conversations, the links, Memory, rules, the wiki, and the startup route for Codex and Claude Code.

automation

The system that repeatedly runs the same checks and records, triggered by a time or an event you set. Decide how to stop it at the same time.

low disclosure

Narrowing the information you share to Discord and the like down to where things are saved, the state, the alerts, and the next action.

idempotent

The property that repeating the same process does not duplicate the same logs or links.

This PDF is the public English edition generated from the English Markdown source. Use the downloadable Markdown for AI-assisted work and the web edition for browser reading.

About the creator

Created by KindlyFeatherWORKS

Project Lead: Seiichiro Yoshimura

Canonical web edition: agents.kindlyfeatherworks.cloud

Contact and project profile: github.com/seichirochan