

協助完成家庭作業



協助我完成家庭作業

我可以協助你理解與完成作業的**思考過程**，但不會直接給答案。請先說明：這是哪一科、哪個主題、你目前卡在哪個部分。



openspec 操作指令與重要文檔說明 例如 `openspec list`

先釐清：你指的「openspec」是

1. 一個具體工具（例如 Linux CLI 套件、開源規範檢視器），還是
2. 你作業中的某個教材或自訂模擬指令環境？

兩者差異很大。請告訴我它出現的上下文（例如課程名稱、教材章節、或題目原文一段），我再一步步帶你拆解指令結構與常用文件來源。



openspec 操作指令與重要文檔說明 例如 `openspec list`

你說的 OpenSpec 指令與文件（例如 `openspec list`）是來自這個工具的一部分。下面我直接列出常用操作、重要文件說明，並指出你作業時應了解的重點。

常用指令

以下為 OpenSpec CLI 的指令範例與用途：

- `openspec --version`：檢查版本。 [GitHub](#)
- `openspec init`：在專案中初始化 OpenSpec，包括建立 `openspec/` 目錄與配置。 [GitHub+1](#)
- `openspec list`：列出目前“活躍變更 (active change folders)”或專案中用 OpenSpec 管理的變更。 [GitHub+1](#)
- `openspec show <change>`：顯示某一變更（如 proposal、tasks、spec delta）細節。 [GitHub](#)
- `openspec validate <change>`：驗證某變更的規範格式與結構是否符合。 [GitHub](#)
- `openspec archive <change> [--yes]`：將完成的變更歸檔。 [GitHub](#)

重要文件與資料夾說明

在使用 OpenSpec 時，以下檔案／目錄是你應熟悉的：

- `openspec/` 目錄：在初始化後，專案根目錄會建立此資料夾，用來存放變更檔案。 [GitHub](#)
- `openspec/changes/<change-name>/`：每一個變更(feature 或修正)會有一個子資料夾。裡面常有 `proposal.md`、`tasks.md`、`specs/` 等。 [GitHub](#)
- `openspec/project.md`：定義專案層級的約定、標準、技術棧、架構規範等，是讓團隊／AI工具理解專案背景的文件。 [GitHub](#)
- `AGENTS.md`：當你選用支持的 AI 工具時 (例如內建 slash commands)，該文件作為工具與變更流程的配合約定。 [GitHub](#)

作業你應注意的盲點

- 確切理解「變更 (change)」在此流程中的意義：不是指版本控制的 commit，而是你使用 OpenSpec 管理的 `proposal` → `tasks` → `apply` → `archive` 這一整流程。
- 在 `list` 指令裡，你看到的活躍變更代表那些還沒歸檔、還在進行中的規範／任務。完成後應用 `archive` 命令。
- 驗證 (`validate`) 不只是語法檢查，它確保你的 spec 結構、任務與目標符合流程標準，這對作業寫出來的內容許多老師／工具會要求。
- 專案級文件如 `project.md` 並非“可選”，對於標準化流程與團隊／AI工具協作流程尤為重要。忽略這項可能導致工具沒正確讀取規定。
- 作業中提到的「如 `openspec list`」表示老師期望你能使用這些 CLI 指令並理解它們在流程中的位置，而不只是知道名稱。

如你願意，我可以幫你整理一份「OpenSpec CLI 指令參考表」+「文件模版 (`proposal.md`, `tasks.md`, `spec.md`) 範本」你可以拿來直接套用作業。要嗎？



OK 假設我是一個全新的專案 且已經init 接下來我要怎麼做呢才可以有task

很好，從這裡開始就是 OpenSpec 的核心流程。你已經 `openspec init` 過，代表目錄中出現了 `openspec/`。接下來的重點是產生「變更 (change)」，因為所有 task 都附屬在某個變更底下。流程如下：

步驟 1：建立一個變更 (change)

```
bash

openspec new <change-name>
```

例：

```
bash

openspec new feature-user-login
```

這會建立：

```
pgsql

openspec/changes/feature-user-login/
├── proposal.md
├── tasks.md
└── specs/
```

步驟 2：撰寫 proposal.md

這份文件描述「為什麼要做這項變更」與「期望結果」。
範例結構：

```
markdown

# Proposal: User Login Feature
## Goal
讓使用者能以帳號密碼登入系統
## Context
目前所有頁面均為公開，無登入流程
## Acceptance Criteria
- 使用者可註冊
- 使用者可登入並驗證身分
```

步驟 3：撰寫 tasks.md

這是你問的重點。它是「該變更要完成的實際任務清單」。
範例：

```
markdown

# Tasks
- [ ] 建立 login API endpoint
- [ ] 寫前端登入頁面
- [ ] 整合 session 驗證
```

你可以直接人工建立，也可用 CLI 自動生成空白模板：

```
bash

openspec add task "建立 login API endpoint"
```

步驟 4：檢查變更狀態

```
bash

openspec list
```

會看到：

```
pgsql

feature-user-login (active)
```

步驟 5：驗證格式

```
bash

openspec validate feature-user-login
```

確保文件格式與欄位正確。

步驟 6：完成後歸檔

```
bash

openspec archive feature-user-login
```

想確認你理解：在 OpenSpec 架構中，**task 不能獨立存在**，必須掛在某個 change 下。
你是否想讓我帶你看一次 `proposal.md` 與 `tasks.md` 的最小可執行範本（可直接套進專案用）？



```
(.venv) (base) hongjinglundeMacBook-Pro:openspec alan$ openspec --version
0.13.0
(.venv) (base) hongjinglundeMacBook-Pro:openspec alan$ openspec new hi
error: unknown command 'new'
(Did you mean view?)
```

很好，你已經實際執行到 CLI 階段，錯誤顯示出現關鍵點：

openspec new 不是有效指令（新版 0.13.0 中已移除或改名）。

在 v0.13.0 後，OpenSpec 不再使用 **new** 產生變更，而是改以**「提案導向 (proposal-driven flow)」**。
正確流程如下：

步驟 1：查看可用命令

```
bash

openspec --help
```

你應該會看到像：

```
pgsql

list
view
propose
apply
validate
archive
```

步驟 2：建立變更（新版指令）

```
bash

openspec propose "hi"
```

這會產生：

```
bash

openspec/changes/hi/
├── proposal.md
├── tasks.md
└── specs/
```

步驟 3：編輯 proposal 與 tasks

同前面教過的，你直接在產生的 `proposal.md`、`tasks.md` 中撰寫內容即可。

步驟 4：檢視狀態

```
bash

openspec list
```

步驟 5：檢查內容正確性

```
bash

openspec validate hi
```

你可以試著先執行：

```
bash

openspec propose "hi"
```

回傳內容貼上來，我再幫你對照它生成的結構與用途，確認版本對應無誤。



```
介紹 (base) hongjinglundeMacBook-Pro:HW3-email spam classification alan$ openspec help
Usage: openspec [options] [command]
```

AI-native system for spec-driven development

Options:

```
-V, --version      output the version number
--no-color          Disable color output
-h, --help          display help for command
```

Commands:

```
init [options] [path]      Initialize OpenSpec in your project
update [path]              Update OpenSpec instruction files
list [options]             List items (changes by default). Use --specs to list
                           specs.
view                      Display an interactive dashboard of specs and changes
change                    Manage OpenSpec change proposals
archive [options] [change-name] Archive a completed change and update main specs
spec                      Manage and view OpenSpec specifications
validate [options] [item-name] Validate changes and specs
show [options] [item-name] Show a change or spec
help [command]            display help for command
```

很好，你查到完整的新版 CLI 介面。從這份 help 可以看出核心改變：

新版把「建立與管理變更」整合進 **change** 子命令，而不是舊版的 **new** 或 **propose**。

流程如下：

一、建立新變更（相當於以前的 new）

```
bash

openspec change create hi
```

或縮寫：

```
bash

openspec change add hi
```

這會在 **openspec/changes/hi/** 下產生：

```
proposal.md
tasks.md
specs/
```

二、編輯任務

```
bash

vim openspec/changes/hi/tasks.md
```

或用 CLI 新增一條任務：

```
bash

openspec change add-task hi "建立資料前處理模組"
```

三、查看目前所有變更

```
bash

openspec list
```

四、檢查格式正確性

```
bash

openspec validate hi
```

五、完成後歸檔

```
bash

openspec archive hi
```

這樣你就能進入「有 task」的狀態。

重點：所有任務（tasks）都隸屬於某個 change 目錄。

你可以立即在當前專案中執行這行測試：

```
bash

openspec change create hi
```

執行結果貼上來，我幫你確認生成結構是否完整。



接下來我要怎麼 create 沒辦法 (base) hongjinglundeMacBook-Pro:HW3-email spam classification
alan\$ openspec view

OpenSpec Dashboard

Summary:

- Specifications: 1 specs, 1 requirements
- Active Changes: 0 in progress
- Completed Changes: 0

Specifications

- spam-classification 1 requirement
-
-

Use `openspec list --changes` or `openspec list --specs` for detailed views

這表示你的專案已經成功初始化並偵測到一份主規格 **spam-classification**，但目前沒有任何「變更 (change)」存在。

openspec change create 失敗的原因通常是子指令名稱變動。

新版 OpenSpec 把子命令整合進 **change** 模組，你可以先查它的子指令：

```
bash

openspec help change
```

看裡面列出什麼（可能會有 **add**, **new**, **start**, **init**, **propose** 等）。

如果出現類似：

```
sql

Usage: openspec change [options] [command]
Commands:
  start <name>      Start a new change proposal
  list              List all change proposals
  apply <name>      Apply a change
  ...
```

那正確指令會是：

```
bash

openspec change start hi
```

這會建立 **openspec/changes/hi/** 資料夾。

執行完後可用：

```
bash

openspec list --changes
```

檢查是否出現一個 active change。

你現在先貼出 `openspec help change` 的輸出，我幫你判定正確的「create」指令名稱（每個版本可能不同）。



```
(base) hongjinglundeMacBook-Pro:HW3-email spam classification alan$ openspec help change
Usage: openspec change [options] [command]

Manage OpenSpec change proposals

Options:
  -h, --help            display help for command

Commands:
  show [options] [change-name]  Show a change proposal in JSON or markdown format
  list [options]                List all active changes (DEPRECATED: use "openspec list"
                                instead)
  validate [options] [change-name] Validate a change proposal
  help [command]              display help for command
```

很好，這輸出揭示關鍵：你目前的版本（0.13.0）中，`change` 模組只有管理現有變更的功能（`show`, `list`, `validate`），沒有建立新變更的指令。

這說明 OpenSpec 的初始化方式在新版已經改變。

建立變更（新增一組 proposal + tasks）現在要走 `spec` 模組。

照這個版本邏輯，請你執行：

```
bash

openspec help spec
```

觀察有沒有：

```
sql

propose
add-change
start
new
```

等子命令。

若輸出中有：

```
php-template

openspec spec propose <name>
```

那麼正確建立方式就是：

```
bash

openspec spec propose hi
```

這會在 `openspec/changes/hi/` 下產生一組 `proposal + tasks` 模板。

結論：

- **change**：管理現有變更。
- **spec**：新增或修改規格（含提案階段）。

你現在先執行 `openspec help spec`，貼出內容，我幫你對應出「生成 change 的正確指令」。



```
(base) hongjinglundeMacBook-Pro:HW3-email spam classification alan$ openspec help spec
Usage: openspec spec [options] [command]
```

Manage and view OpenSpec specifications

Options:

```
-h, --help          display help for command
```

Commands:

```
show [options] [spec-id]  Display a specific specification
list [options]             List all available specifications
validate [options] [spec-id] Validate a specification structure
help [command]            display help for command
```

結果確認：你的版本（0.13.0）中，**spec** 模組同樣沒有任何「建立 proposal / change」的子命令。

這代表該版本的 OpenSpec 已經轉為**只讀與驗證導向**（不再支援 CLI 建立變更），而「新增變更」改由在檔案層級手動建立完成。

現在正確做法是手動建立一個變更資料夾，格式如下：

```
bash

mkdir -p openspec/changes/hi/specs
touch openspec/changes/hi/proposal.md
touch openspec/changes/hi/tasks.md
```

結構：

markdown

```
openspec/
├── changes/
│   └── hi/
│       ├── proposal.md
│       ├── tasks.md
│       └── specs/
```

然後在這兩份文件中填入基礎內容：

proposal.md

```
markdown

# Proposal: hi
## Goal
建立基本變更測試
## Context
此變更用於測試 OpenSpec 流程
## Acceptance Criteria
- 能通過 validate
```

tasks.md

```
markdown

# Tasks
- [ ] 測試檔案結構正確
- [ ] 驗證 openspec validate 成功
```

接著執行：

```
bash

openspec list
```

理論上會顯示這個變更。

最後檢查：

```
bash

openspec validate hi
```

結論：

你使用的 OpenSpec 版本仍維持基本功能（list/view/show/validate），但不再提供 CLI 自動建立變更的命令。目前唯一可行路徑是**手動建立變更目錄與文件結構**，再透過 **validate** 驗證它。



```
(base) hongjinglundeMacBook-Pro:HW3-email spam classification alan$ openspec --version
0.13.0 請協助推論作法
```

根據版本 **0.13.0** 的指令集（你提供的完整 help output），OpenSpec 在這一版的行為是：

- 不再自動產生變更（change）或提案（proposal）
- 僅保留檢視、驗證、歸檔、列出既有規格的功能
- 所謂「change」流程要靠**手動建立檔案結構**

因此實際能執行的唯一「建立 task」途徑是靠目錄與檔案層級操作。推論出的正確實務流程如下：

一、初始化專案（已完成）

```
bash

openspec init
```

產生：

```
openspec/
specs/
changes/
```

二、手動建立一個變更

```
bash

mkdir -p openspec/changes/hi/specs
```

接著建立兩份文件：

```
bash

touch openspec/changes/hi/proposal.md
touch openspec/changes/hi/tasks.md
```

三、撰寫內容（模板）

openspec/changes/hi/proposal.md

```
markdown

# Proposal: hi
## Goal
測試 OpenSpec 手動建立流程
## Context
建立一個基本變更樣本供 validate 測試
## Acceptance Criteria
- 結構通過 openspec validate
```

openspec/changes/hi/tasks.md

```
markdown

# Tasks
- [ ] 建立 proposal.md 與 tasks.md
- [ ] 驗證 openspec validate hi 能執行
```

四、檢查變更存在

```
bash

openspec list --changes
```

若沒有顯示，也可用：

```
bash

openspec list
```

應會看到一筆 active change。

五、驗證結構是否合規

```
bash

openspec validate hi
```

預期輸出：

Validation passed for change hi

若有錯誤，會指出缺少欄位或格式錯誤。

六、完成後可歸檔

```
bash

openspec archive hi
```

會把內容整合入主 specs。

七、結論性推論

功能	舊版命令	v0.13.0 替代方式
建立變更	openspec new / openspec propose	手動建立資料夾與 proposal.md
建立任務	openspec add task	手動編輯 tasks.md
檢查	openspec validate <change>	相同
檢視	openspec view	相同
歸檔	openspec archive	相同

所以結論：

在 **v0.13.0** 版本中，「有 task」的狀態要靠人工新增檔案。OpenSpec CLI 本身僅負責解析與驗證，不再自動 scaffold。



```
sms_spam_no_header.csv
(base) hongjinglundeMacBook-Pro:HW3-email spam classification alan$ openspec view
```

OpenSpec Dashboard

Summary:

- Specifications: 1 specs, 1 requirements
- Active Changes: 0 in progress
- Completed Changes: 0

Specifications

- spam-classification 1 requirement

Use openspec list --changes or openspec list --specs for detailed views
怎麼 completed changes

在你這個版本的 OpenSpec (0.13.0) 中，「Completed Changes」指的是 **已歸檔 (archived)** 的變更。
流程如下：

1. 先建立一個變更 (Change)

由於這版本不支援 CLI 自動建立，必須手動：

```
bash

mkdir -p openspec/changes/my-spam-change/specs
touch openspec/changes/my-spam-change/proposal.md
touch openspec/changes/my-spam-change/tasks.md
```

填入範例內容：

proposal.md

```
markdown

# Proposal: my-spam-change
## Goal
新增 spam classification 任務
## Context
使用 sms_spam_no_header.csv 進行分類
## Acceptance Criteria
- 任務完成並驗證格式正確
```

tasks.md

```
markdown
```

Tasks

- [] 載入 sms_spam_no_header.csv
- [] 資料前處理
- [] 建立特徵與標籤
- [] 訓練分類模型
- [] 驗證分類準確率

2. 驗證變更結構

```
bash
```

```
openspec validate my-spam-change
```

確認沒有格式錯誤。

3. 歸檔變更

```
bash
```

```
openspec archive my-spam-change
```

歸檔後，OpenSpec 會把這個變更標示為「Completed」。

4. 查看已完成變更

```
bash
```

```
openspec list --changes
```

或者

```
bash
```

```
openspec view
```

這時候 Dashboard 的 **Completed Changes** 就會顯示你的變更數量。

重點：

- Active Changes = 目前還在進行中、尚未 archive 的變更
- Completed Changes = 已經 archive 的變更
- 0 Completed 是因為你目前沒有執行 archive。