


THE CREW

Ecosistema completo para limpiar, humanizar y organizar exportaciones de ChatGPT en un VAULT de Obsidian (incluyendo imágenes, herramientas internas y andamiajes)


Comando extracción

✖ 1. `batch_sequencer.py`

El director.	Orquesta la extracción de notas y creación de vaults.
	<p>Secuencia todo el flujo: importa exports de ChatGPT (ZIP, JSON o HTML), genera un vault bruto (<code>RAW_VAULT</code>), luego uno limpio (<code>MERGED_VAULT</code>) y otro con bloques invertidos (<code>REVERSE_VAULT</code>).</p> <ul style="list-style-type: none">- Usa <code>split_chatgpt_export.py</code> para dividir exportaciones.- Llama a <code>vault_cleaner.py</code> para fusionar y limpiar.- Admite un <code>gizmo_map.json</code> para asociar IDs internos de ChatGPT a nombres de proyectos.- Crea plantillas de estructura base para los vaults.

👉 Es el **punto de entrada maestro** para convertir dumps de ChatGPT en vaults de Obsidian legibles.


2. `split_chatgpt_export.py`

El detective del sistema.	Busca hilos y los convierte en notas.
	<p>Convierte cada export (ZIP, JSON, HTML) en notas Markdown estructuradas con front-matter YAML.</p> <ul style="list-style-type: none">- Extrae mensajes, autores, fechas, IDs de proyectos, y genera:<ul style="list-style-type: none">- <code>Project_name</code> , <code>source_project</code> , <code>source_project_id</code>- <code>tags</code>: automáticos (si usas un <code>tag_map</code>)- Crea carpetas por año/mes y genera índices (<code>_index.md</code> , <code>_tags/</code>)- Soporta versionado, deduplicación, naming limpio, y control por CLI.

El detective del sistema.	Busca hilos y los convierte en notas.
	- Interpreta bien los formatos de export de ChatGPT con su estructura JSON compleja.


👉 Convierte los datos crudos en notas Obsidian completas.

🔪 3. vault_cleaner.py

Depura y fusiona.	Cocina los ingredientes para crear notas legibles y ordenadas.
	<p>Recoge las notas creadas por el anterior y:</p> <ul style="list-style-type: none"> - Agrupa versiones de la misma conversación. - Elige la más larga o fusiona los mensajes sin duplicar. - Puede invertir el orden de los bloques (### User / ### Assistant). - Mantiene front-matter y metadatos. - Crea un vault limpio, sin ruido ni repeticiones.

👉 Deja un vault depurado y homogéneo listo para análisis o lectura.

📁 4. extract_images_from_zips_dedup.py

Extractor de imágenes.	Extrae las imágenes del ZIP y las deja en un banco común.
	<p>Busca y saca todas las imágenes de los ZIPs de exportación (las de ChatGPT o herramientas), deduplicando por hash SHA-256.</p> <p>Crea un banco común de imágenes únicas.</p>

👉 Alimenta al ImageLinkInjector.py .

🖼️ 5. ImageLinkInjector.py

Repinta las imágenes.	Pinta las imágenes en las notas.
	Reemplaza referencias internas tipo <code>sediment://file_<id></code> dentro de las notas por wikilinks Obsidian (<code>![[IMAGE_BANK/file_xxx.png]]</code>), buscando en el banco de imágenes.

👉 Vuelve a conectar las imágenes originales a sus notas.

Escuadrón de limpieza:

Nuestro escuadrón de limpieza te ayudará a dejar las tus notas de Obsidian legibles para humanos, limpias y ordenadas.

	<ul style="list-style-type: none"> - Humanizadores <ul style="list-style-type: none"> - de bloques de herramientas - de bloques de carga de archivos - Limpieza de metadatos del bloque de imágenes - Limpieza de líneas vacías dobles, etc.
---	--

🔧 1. RoleBlockExtractor.py

Humaniza bloques de cualquier tipo (`### User`, `### Assistant` o `### Tool`)...

Convierte bloques técnicos con JSON (tool calls, audio, canvas, transcripciones, tether_quote, etc.) en texto legible tipo:

```
### Assistant Tool-Block: nombre_del_archivo "Texto procesado..."
```

Opcionalmente conserva el JSON original plegado dentro de `<details>` .

👉 Traduce trazas de herramientas internas en narrativa comprensible.

🌿 2. RenderTetherQuotes.py

Convierte los “tether_quote”.

Detecta los bloques tipo:

```
### Tool {'content_type': 'tether_quote', 'domain': 'Archivo.md', 'text': '...'}
```

y los transforma en:

```
### Assistant 📄 Archivo cargado: **Archivo.md** > contenido...
```

👉 Vuelve legibles las citas y anclajes de documentos cargados.



3. CleanImageToolBlocks.py

Elimina los bloques técnicos de imágenes después de insertar los enlaces visibles.

Detecta fragmentos como:

```
### Tool {'content_type': 'image_asset_pointer', 'asset_pointer':  
'sediment://file_00000...', ...}
```

o

```
### User {'content_type': 'image_asset_pointer', 'asset_pointer': '!  
[[IMAGE_BANK/file_...png]]', ...}
```

y los deja así:

```
![[IMAGE_BANK/file_...png]]
```

👉 **Limpia los metadatos sobrantes** de imágenes ya insertadas, dejando solo el enlace Obsidian visible y eliminando los diccionarios {...} asociados.



4. TidyBlankLines.py

Limpieza estética.

Reduce saltos de línea múltiples a uno solo en todos los .md .

Pequeño pero esencial para dejar todo legible.

Cartógrafos de élite




Crearán dos índices navegables para que tengas una visión por contexto y por proyecto de tu VAULT.

1. `scaffolding_index.py`

Genera el índice de andamiajes.

Escanea todas las notas y crea `scaffolding_index.md` listando los archivos mencionados en

“ Archivo cargado: ...”.

👉 Índice navegable de los documentos que sirvieron como contexto en las conversaciones.

2. `tree_index.py`

Crea un índice tipo árbol.

Organiza las notas por `Project_name` → `Año` → `Mes`, con wikilinks ordenados cronológicamente.

👉 Da una **visión estructurada por proyecto** dentro del vault limpio.

Flujo completo resumido

1. `batch_sequencer_robusto.py` → coordina todo.
2. `split_chatgpt_export.py` → exporta cada conversación a Markdown.
3. `vault_cleaner.py` → fusiona y depura.
4. `extract_images_from_zips_dedup.py` → extrae imágenes únicas.
5. `ImageLinkInjector.py` → reinserta las imágenes en los textos.
6. `ToolBlockExtractor.py` → humaniza bloques de herramientas.
7. `RenderTetherQuotes.py` → convierte los andamiajes (`tether_quote`).
8. `CleanImageToolBlocks.py` → limpia bloques de metadatos de imágenes.
9. `TidyBlankLines.py` → limpieza final de formato.
10. `scaffolding_index.py` → crea índice de archivos de contexto.
11. `tree_index.py` → genera índice jerárquico por proyecto.

En conjunto, este sistema convierte un export caótico de ChatGPT en un **vault Obsidian limpio, con imágenes restauradas, textos legibles, índices, tags y estructura por proyectos.**

Dicho sin rodeos: un pipeline de archivística digital digno de un museo del pensamiento conversacional.