Guía de conversión de imágenes fMRI crudas a formato BIDS (Linux)

Introducción a BIDS y preparación de los datos

El **Brain Imaging Data Structure (BIDS)** es un estándar para organizar datos de neuroimagen de forma consistente. En BIDS, cada estudio se estructura en carpetas por sujeto/sesión y cada imagen NIfTI tiene un archivo JSON acompañante con metadatos (parámetros de adquisición, etc.) extraídos del header DICOM [1]. Esto asegura que no se pierda información importante al convertir de DICOM a NIfTI, y facilita la **automatización de preprocesamiento**: existen herramientas como *fMRIPrep* y *MRIQC* capaces de procesar conjuntos de datos BIDS casi sin intervención del usuario [2].

En esta guía nos centraremos en convertir datos **fMRI crudos de ratones** al formato BIDS. El estándar BIDS es aplicable a datos de animales (no solo humanos); por ejemplo, es recomendable añadir una columna `species` con el valor binomial de la especie (p. ej. "mus musculus" para ratón) en el archivo `participants.tsv` [3]. **NOTA:** No cubriremos la conversión de datos Bruker (formato propietario común en escáneres preclínicos) porque ya ha sido excluida en la solicitud. Nos enfocaremos en los formatos crudos más comunes como **DICOM** (y también Philips PAR/REC o NIfTI sin organizar) obtenidos de los escáneres.

Requisitos previos

- **Python 3**: Varias herramientas son paquetes de Python. Asegúrate de tener Python ≥3.7 instalado (en Linux suele venir preinstalado). Puedes verificar con `python --version` [4].
- **dcm2niix**: Es la utilidad base que convierte DICOM a NIfTI. Muchas herramientas la utilizan internamente, así que conviene instalarla por separado. En Ubuntu/Debian puedes hacerlo fácilmente con: `sudo apt update && sudo apt install dcm2niix` [5]. (Esto instala *dcm2niix*, un conversor DICOM→NIfTI compatible con múltiples modalidades [6]).
- **Docker (opcional)**: Algunas herramientas ofrecen imágenes de Docker para simplificar la instalación. Si prefieres usar Docker, consulta la siguiente sección.

Instalación de Docker (opcional)

Si decides usar contenedores Docker para ejecutar las herramientas (opción recomendada para principiantes en algunos casos), sigue estos pasos para instalar Docker en Linux:

1. **Instalar Docker Engine:** Actualiza la lista de paquetes e instala los paquetes necesarios y Docker desde el repositorio oficial. En Ubuntu, por ejemplo:

```
sudo apt update
sudo apt install apt-transport-https ca-certificates curl software-properties-
```

```
common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/
ubuntu focal stable"
sudo apt update
sudo apt install docker-ce
```

Esto instalará Docker Community Edition y arrancará el servicio Docker [7] [8]. Puedes verificar que Docker está activo con `sudo systemctl status docker` [9].

1. **Post-instalación (opcional):** Por defecto, el comando `docker` requiere privilegios de superusuario. Para evitar tener que anteponer `sudo` en cada comando, agrega tu usuario al grupo `docker` con: `sudo usermod -aG docker ${USER}` [10]. Cierra sesión y vuelve a iniciarla para aplicar el cambio, o ejecuta `su - ${USER}` [11]. Tras esto, usar `docker run ...` ya no requerirá `sudo`.

   Nota: Si prefieres no usar Docker, todas las herramientas mencionadas se pueden instalar nativamente (p. ej. vía `pip`/conda). Docker simplemente evita gestionar dependencias manualmente, pero implica familiarizarse con contenedores.

A continuación, describimos **5 métodos comunes** para convertir datos fMRI crudos a BIDS en Linux, dirigidos a *principiantes*. Incluimos tanto herramientas de línea de comandos (CLI) como con interfaz gráfica (GUI), priorizando facilidad de instalación y uso. Cada método incluirá instrucciones de descarga/instalación y pasos básicos de ejecución.

Método 1: Dcm2Bids (CLI en Python)

Descripción: *Dcm2Bids* es una herramienta comunitaria escrita en Python que combina la conversión DICOM→NIfTI de **dcm2niix** con un sistema de configuración para renombrar y organizar los archivos según BIDS [12]. Su objetivo es ser una solución **amigable y fácil de usar** para la conversión a BIDS, enfocada en simplificar los casos típicos sin añadir complejidad innecesaria [13]. En lugar de requerir programación, Dcm2Bids utiliza un archivo de configuración JSON donde el usuario mapea las series DICOM (identificadas por nombre, número de serie u otros metadatos) a nombres y carpetas BIDS apropiados.

Instalación: Para instalar Dcm2Bids necesitarás Python 3.7+ instalado. Luego puedes usar pip:

```
pip install dcm2bids
```

Este comando descargará e instalará la última versión estable desde PyPI [14]. Alternativamente, está disponible vía Conda: `conda install -c conda-forge dcm2bids`. *Nota:* Dcm2Bids no incluye *dcm2niix* en el paquete, por lo que **asegúrate de tener** `dcm2niix` **en tu PATH** (por ejemplo, instalándolo con apt como se indicó arriba, o con conda) [15]. Si `dcm2niix` no está disponible, Dcm2Bids no podrá realizar la conversión de imágenes.

Uso básico: La conversión con Dcm2Bids se hace típicamente en tres pasos:

1. **Estructura BIDS vacía:** Primero, crea la estructura de carpetas y archivos mínimos de BIDS. Dcm2Bids proporciona un comando para ello:

```
dcm2bids_scaffold -o /ruta/al/directorio_BIDS
```

Esto generará en la carpeta destino la estructura BIDS inicial con subdirectorios (`anat/`, `func/`, etc.), archivos vacíos como `dataset_description.json`, `participants.tsv`, `README`, y un directorio `code/` para colocar configuraciones [16] [17]. (Internamente, este *scaffold* toma plantillas del BIDS Starter Kit [18]).

1. **Configurar mapeo (archivo JSON):** A continuación, debes crear el archivo de configuración JSON (`dcm2bids_config.json`) que indica cómo mapear cada serie DICOM a un nombre BIDS. Para facilitar esto, Dcm2Bids incluye la utilidad `dcm2bids_helper` que puedes ejecutar sobre una carpeta de DICOM de un sujeto para listar las series encontradas. Por ejemplo:

```
dcm2bids_helper -d /ruta/DICOMs_sujeto -o /ruta/al/directorio_BIDS/tmp_dcm2bids
```

El helper convertirá los DICOM de muestra a NIfTI (en una carpeta temporal) para que puedas inspeccionar los nombres de serie, y generará un archivo de ejemplo con esa información. Con base en esa salida, editarás (o crearás) `dcm2bids_config.json` dentro de `directorio_BIDS/code/`. En este JSON definirás entradas para cada tipo de serie: por ejemplo, mapear una serie llamada "T1w" al destino BIDS `anat/T1w.nii.gz`, una serie "BOLD_rest" a `func/sub-*_task-rest_bold.nii.gz`, etc. La documentación oficial proporciona guías para construir este archivo [19]. *(Ver el tutorial enlazado en [20] para detalles de sintaxis del config.)*

1. **Ejecutar la conversión:** Una vez listo el archivo de configuración, ejecuta Dcm2Bids para procesar uno o todos los sujetos. El comando principal es `dcm2bids -d /ruta/DICOMs -o /ruta/BIDS -p <ID_sujeto> -c /ruta/config.json`, donde:
2. `-d` especifica la carpeta con los DICOM (puede ser la raíz que contenga subcarpetas por sujeto).
3. `-p` indica el identificador del sujeto a convertir (por ejemplo `01` para *sub-01*).
4. `-c` el path al archivo de configuración JSON que creaste.
5. `-o` la carpeta de salida BIDS (la misma que usaste en el scaffold).

Dcm2Bids leerá todos los DICOM del sujeto, aplicará *dcm2niix* para convertir a NIfTI+JSON, y luego reorganizará/copiará esos archivos a la estructura BIDS de salida según las reglas del config file. Si todo está correcto, obtendrás dentro de `sub-01/` los archivos `.nii.gz` y `.json` renombrados según BIDS, separados en `anat/`, `func/`, etc. Dcm2Bids generará también un archivo de registro (`conversion.log`) en `tmp_dcm2bids/log/` con los detalles de la conversión.

Ventajas: Dcm2Bids es ideal para principiantes porque **evita programación** y usa un enfoque declarativo mediante un archivo JSON. La versión 3 en adelante ha simplificado la creación del config y añadido comandos como *scaffold* para ayudarte [21]. Una vez que tienes un config funcionando, puedes reutilizarlo para muchos sujetos y estudios con estructura similar. Además, Dcm2Bids cuenta con imágenes Docker

preconstruidas (por ejemplo, `unfmontreal/dcm2bids` en DockerHub [22]) si prefieres no instalar nada en el sistema – aunque el uso local con pip es bastante directo.

Para datos de ratón: Dcm2Bids no hace distinción si los DICOM provienen de humano o animal; simplemente asegúrate de nombrar los sujetos como `sub-<ID>` (por ejemplo `sub-mouse01`) y luego en `participants.tsv` incluir la especie. Dcm2Bids copiará los metadatos de los DICOM a los JSON (ej. TR, TE) pero probablemente tendrás que agregar manualmente al JSON de anatomía el campo `Species` con `Mus musculus` (ya que dcm2niix por sí solo no establece la especie).

Método 2: HeuDiConv (CLI avanzado en Python)

Descripción: *HeuDiConv* es una potente herramienta de conversión DICOM→BIDS basada en **heurísticas programables** [23] [24] . A diferencia de Dcm2Bids (configuración estática), HeuDiConv te permite escribir un *script en Python* (heuristic file) que contiene la lógica para clasificar y renombrar las series de manera flexible. Es muy usada en entornos de investigación por su versatilidad: se puede adaptar a prácticamente cualquier convención de nombres o requerimiento especial. Internamente, HeuDiConv utiliza *dcm2niix* para convertir a NIfTI, pero aporta una capa que "decide" cómo nombrar y dónde ubicar cada archivo [25] [26] . También puede **rastrear la procedencia** de las conversiones y se integra con sistemas como DataLad (para versionado de datos), aunque estos son usos avanzados [27] .

Consideraciones: Debido a su enfoque, HeuDiConv **requiere conocimientos básicos de Python** y entender la estructura de tus datos para escribir la heurística. Esto implica una curva de aprendizaje más pronunciada para principiantes [28] . Sin embargo, ya existen heurísticas de ejemplo que se pueden reutilizar en muchos casos (el propio paquete incluye algunas, como `reproin` y `convertall`) [29] . En resumen, HeuDiConv es muy poderoso pero demanda un esfuerzo inicial mayor para configurarlo.

Instalación: HeuDiConv se puede instalar via pip o con Docker. Para principiantes, puede ser más sencillo usar Docker:

- **Opción 1 – Docker:** Hay una imagen oficial (por ejemplo `nipype/heudiconv`). Tras instalar Docker, puedes ejecutar HeuDiConv sin instalar Python en tu sistema. Un ejemplo sería:

```
docker run --rm -it -v /datos/DICOM:/data:ro -v /datos/BIDS:/output nipy/
heudiconv:latest \
    -d "/data/{subject}/*.dcm" -o /output -f convertall -s sub-01 -c dcm2niix -b
```

Donde montamos la carpeta con DICOM en `/data` de solo lectura, y la carpeta de salida en `/output` . Este comando usará la heurística `convertall` (incluida en HeuDiConv) que simplemente convierte *todas* las series para el sujeto `sub-01` sin asignar etiquetas específicas (te tocará luego renombrar/organizar). La opción `-c dcm2niix -b` indica que genere también los archivos JSON BIDS usando dcm2niix y copie los metadatos básicos.

- **Opción 2 – Nativo:** Alternativamente, instala con `pip install heudiconv` . Asegúrate de tener también *dcm2niix* disponible en el sistema. Verifica con `heudiconv --version` que todo esté

correcto. Si instalas nativo, también necesitarás crear un entorno (por ejemplo con conda) e instalar dependencias como numpy, nibabel, etc., que pip manejará automáticamente.

Uso básico: Usar HeuDiConv implica:

1. **Crear la heurística:** Es un archivo `.py` donde defines funciones o diccionarios que mapean las series a nombres BIDS. Por ejemplo, podrías hacer que toda serie cuyo nombre contenga "T1" vaya a `anat/T1w`, las que contengan "BOLD" y "rest" vayan a `func/task-rest_bold`, etc. La heurística tiene acceso a los metadatos DICOM de cada serie (ProtocolName, TE/TR, etc.) para tomar decisiones. *(Como principiante, puedes empezar tomando una heurística existente y adaptándola. La comunidad comparte heurísticas en el repo de HeuDiConv, y hay una genérica llamada* `reproin` *que funciona si tus protocolos siguen ciertos patrones predefinidos.)*

2. **Ejecutar heudiconv:** Una vez tienes `mi_heuristica.py`, ejecutas el comando de conversión. Por ejemplo:

```
heudiconv -d "/ruta/DICOM/{subject}/*/*.dcm" -o /ruta/BIDS -f /ruta/
mi_heuristica.py \
          -s sub-01 -c dcm2niix -b
```

   - `-d` indica el patrón de búsqueda de DICOM (usando `{subject}` como comodín para el ID de sujeto en la ruta).
   - `-o` la carpeta de salida BIDS.
   - `-f` la heurística a usar (archivo `.py`).
   - `-s` el sujeto a procesar (sin el prefijo `sub-`).
   - `-c dcm2niix` para usar dcm2niix en la conversión y `-b` para obtener JSON sidecars con metadatos.

HeuDiConv hará dos pasadas: primero identifica las series y genera un fichero intermedio `.heuristic.yml` con la correspondencia encontrada, luego aplica la conversión según la heurística. Al finalizar, tendrás la estructura BIDS en `/ruta/BIDS` con el sujeto convertido. Si falta algún archivo o hay series no asignadas (HeuDiConv las denomina `unassigned` si no entran en ninguna regla), deberás ajustar la heurística y re-ejecutar.

1. **Conversión por lotes:** Puedes convertir múltiples sujetos pasando varios IDs con `-s`. HeuDiConv procesará cada uno secuencialmente. Es recomendable convertir primero un sujeto de prueba, verificar la estructura BIDS generada, y luego escalar a todos los sujetos.

Ejemplo: Si tus carpetas DICOM están organizadas como `raw/sub-01/[archivos DICOM]`, `raw/sub-02/[...]`, podrías usar `-d "raw/{subject}/*dcm" -s sub-01 sub-02 ...`. Si además hay sesiones, puedes incorporar `{session}` en el patrón e indicar `-ss` para las sesiones.

Ventajas y cuándo usarlo: HeuDiConv es muy útil cuando tus datos tienen *nombres poco estandarizados o múltiples modalidades* y quieres automatizar la conversión repetible para diversos estudios. La capacidad de programar reglas complejas lo hace adecuado para centros que integran la conversión en el pipeline de adquisición (ej. convertir automáticamente al terminar cada sesión de escáner). Sin embargo, para un

usuario principiante con unos pocos sujetos, puede ser excesivo. Recomendamos HeuDiConv principalmente si estás dispuesto a invertir tiempo en aprender su funcionamiento o si tus datos no se adaptan bien a las configuraciones estáticas de otras herramientas.

Caso de datos de ratón: Al igual que Dcm2Bids, HeuDiConv tratará los DICOM de ratón sin problemas. Podrías necesitar heurísticas especializadas si los protocolos tienen nombres genéricos (p. ej. "RareFSE_T2" podría mapearlo a `anat/T2w`). Una ventaja es que, si tus DICOM de ratón incluyen campos como *PatientName* con el ID del animal, puedes usarlos dinámicamente en la heurística para los nombres BIDS. Recuerda luego añadir la información de especie en los archivos de participantes.

Método 3: BIDScoin (GUI/CLI híbrido, Python)

Descripción: *BIDScoin* es una aplicación completa que ofrece una **interfaz gráfica (GUI)** para convertir datos crudos a BIDS de forma interactiva [30] [31]. Está diseñada para no requerir conocimientos de programación, guiando al usuario a través de la configuración de la conversión. BIDScoin utiliza un enfoque de **mapeo inteligente**: escanea automáticamente tus datos de origen, detecta las distintas series (por sus atributos, nombres, etc.) y genera una propuesta de mapeo a BIDS. Luego te permite revisar y ajustar ese mapeo en la GUI, y finalmente ejecutar la conversión completa [32] [33]. Es compatible con múltiples formatos de entrada: desde DICOM y PAR/REC, hasta archivos NIfTI ya convertidos sin organizar [34].

Instalación: BIDScoin es un paquete Python 3.8+ que puedes instalar con pip. Si solo planeas convertir MRI convencional (no espectroscopía ni fisiológicos), basta con:

```
pip install bidscoin
```

Esto instalará BIDScoin y el *plugin* principal que usa dcm2niix para MRI [35]. (*Nota:* Si quieres soporte a Spectroscopy u otros, hay extras como `pip install bidscoin[spec2nii2bids]`, pero para fMRI estándar no hace falta [35]). Tras la instalación, tendrás disponibles en tu sistema varios comandos: `bidscoin`, `bidsmapper`, `bidseditor`, `bidscoiner`, entre otros. **Importante:** Asegúrate nuevamente de tener *dcm2niix* instalado por fuera, ya que BIDScoin lo invoca para convertir DICOM→NIfTI [36]. Si no está en PATH, debes instalarlo y luego indicar a BIDScoin dónde encontrarlo (esto se puede hacer en las *Options* de la GUI la primera vez).

Uso (flujo de trabajo): El proceso típico con BIDScoin consta de dos fases principales (mapeo y conversión), pero la interfaz las encadena casi automáticamente:

- **Paso 1: Mapear las series (bidsmapper + bidseditor).** Ejecuta el comando `bidsmapper` apuntando a la carpeta de datos fuente y a la carpeta BIDS de salida deseada. Por ejemplo:

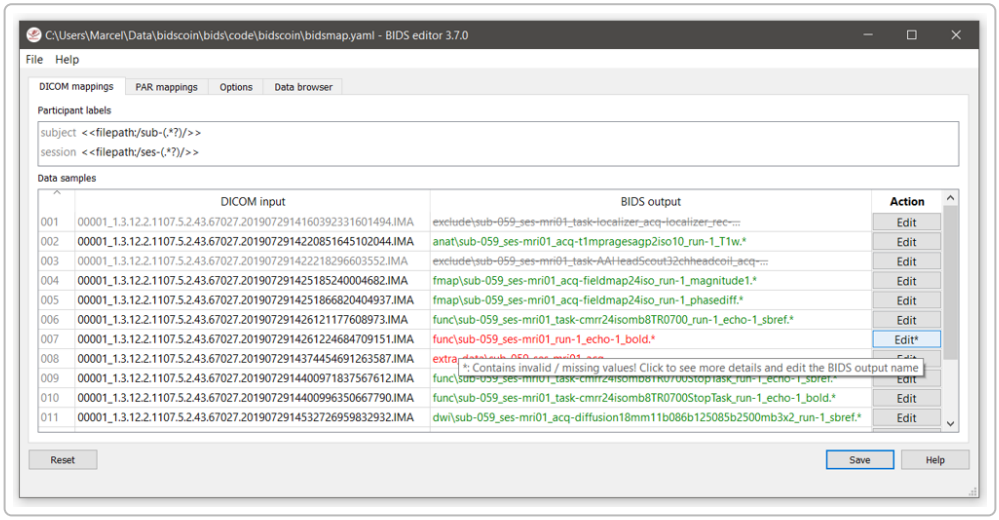
```
bidsmapper /ruta/datos_crudos /ruta/datos_BIDS
```

BIDScoin espera que los datos crudos estén organizados al menos por sujeto (y opcionalmente por sesión) en subcarpetas [37] [38]. Si tus DICOM están todos en un solo directorio, BIDScoin proporciona utilidades para reorganizarlos en `sourcedata/sub-*/ses-*` fácilmente (ver comando `dicomsort` en su

documentación). Suponiendo que ya tienes `/ruta/datos_crudos/sub-001/` etc., el *bidsmapper* escaneará cada carpeta de sujeto/sesión e identificará series únicas. Por defecto, tras escanear, **lanza automáticamente la GUI de BIDScoin llamada *bidseditor*** [39] [40].

En la ventana principal de *BIDS Editor* podrás ver en una tabla todas las series detectadas (columna "DICOM input") y una propuesta de nombre BIDS para cada una (columna "BIDS output") [41] [42]. Las entradas aparecen en rojo si no cumplen completamente BIDS o requieren atención. Aquí es donde interviene el usuario: puedes hacer doble click en una serie para abrir la ventana de edición detallada y asignar el tipo de dato correcto (anat, func, fmap, etc.), el sufijo (p. ej., T1w, bold, etc.), tareas o cualquier entidad BIDS, hasta que el nombre de salida quede **verde** (válido) [43] [44]. La GUI te permite también añadir metadata extra si lo deseas (por ejemplo, campo `IntendedFor` en fieldmaps, notas adicionales, etc.) [45] [46].

[47]



Interfaz principal de BIDScoin: a la izquierda se listan los **datos de entrada DICOM** (una fila por tipo de serie identificado) y a la derecha la **salida BIDS propuesta**. Las filas aparecen en verde cuando el mapeo es válido; filas en rojo indican valores faltantes o nombres no conformes (por ejemplo, la marcada "extra" en la imagen señala que falta asignar un label correcto) [42]. El usuario puede editar cada ítem hasta que todos estén resueltos, ayudándose de menús desplegables y vista previa instantánea.

La idea es que *solo la primera vez* tendrás que ajustar muchas cosas; BIDScoin aprende la correspondencia en un archivo YAML llamado `bidsmap.yaml` que guarda en `code/bidscoin/`. Si en el futuro agregas más sujetos con las mismas series, no tendrás que volver a editar nada mientras el protocolo de adquisición sea igual – el mapa se reutiliza y aplica automáticamente a nuevas sesiones [48]. Cuando estés satisfecho con los mappings (todas las salidas en verde), pulsa **Save** en la GUI para guardar el `bidsmap.yaml`.

- **Paso 2: Convertir datos a BIDS (bidscoiner).** Ahora ejecuta el segundo comando para realizar la conversión efectiva:

```
bidscoiner /ruta/datos_crudos /ruta/datos_BIDS
```

El *bidscoiner* leerá el `bidsmap.yaml` (que refleja tus ediciones) y procederá a convertir cada archivo DICOM a NIfTI+JSON usando dcm2niix, colocando cada resultado en la ruta BIDS apropiada [39] [49]. En esencia, aplica el mapa "moneda" que definiste – de ahí el nombre BIDS *coin* ("acuñar" los datos en BIDS). Este paso ya no requiere intervención y suele tardar dependiendo del volumen de datos. Al finalizar, tendrás tu directorio `/ruta/datos_BIDS` completo con subdirectorios `sub-001`, `sub-002`, etc., cada uno con sus archivos convertidos y listos. Si agregas más sujetos posteriormente, puedes solo correr `bidscoiner` nuevamente (usando el mismo mapa) y convertirá los faltantes [48].

Ventajas: BIDScoin destaca por su **facilidad para principiantes**, gracias a la GUI interactiva [30] [31]. No hace falta escribir código ni editar JSON a mano; la interfaz muestra sugerencias y hasta valida en tiempo real si los nombres cumplen las reglas BIDS (coloreando texto en verde/rojo) [42] [50]. Además, soporta múltiples formatos de entrada en un solo flujo (por ejemplo, si en tu estudio tienes además archivos PAR/REC de Philips, los detecta en otra pestaña "PAR mappings"). Su diseño evita errores comunes y guía al usuario con *tooltips* en los campos (al pasar el ratón) con explicaciones de términos BIDS [41] [51].

Otra ventaja es que BIDScoin guarda un registro completo de qué se hizo (mapa, logs) y es fácil de **reproducir**: el `bidsmap.yaml` puede ser compartido para que otros conviertan datos similares sin pasar por GUI. También cuenta con un modo no interactivo (p. ej. correr `bidsmapper` con opción `-a` para saltar la GUI) útil para automatizar en el futuro [52].

Consideraciones: La GUI de BIDScoin requiere tener un entorno gráfico (X11 o similar) si se usa en Linux; si estás en servidor sin interfaz gráfica, podrías usar un *workaround* como X11 forwarding o limitarte al modo texto (pero entonces pierdes la facilidad de la GUI). BIDScoin también tiene imagen Singularity para HPC. En cuanto a formatos, para fMRI de ratón en DICOM funciona perfectamente. Solo recuerda indicar en la pestaña **Options** del editor la ruta al ejecutable dcm2niix si no es la por defecto, y configura en `participants.tsv` la especie como mencionamos.

Método 4: BIDSkit (CLI secuencial en Python)

Descripción: *BIDSkit* es una utilidad de Python que convierte DICOM a BIDS en **dos pasos**: primero convierte todos los DICOM de un sujeto a NIfTI+JSON, y luego reorganiza esos archivos a la estructura BIDS [53]. Usa también *dcm2niix* internamente. BIDSkit fue una de las primeras herramientas creadas tras la aparición de BIDS y sigue en desarrollo activo (a fecha 2025) [54]. Se basa en la creación de un **diccionario de traducción** (mapping dictionary) entre nombres de serie y nombres BIDS.

Instalación: Puedes instalarlo vía pip:

```
pip install bidskit
```

Asegúrate de tener Python ≥3.10 (BIDSkit 2025.x lo requiere según PyPI) [55]. Al instalar, obtendrás un comando ejecutable `bidskit`. De nuevo, necesitarás *dcm2niix* disponible en tu sistema, ya que BIDSkit lo invocará.

También existe una imagen Docker (no oficial) mantenida por la comunidad y un contenedor Singularity en algunas instalaciones (como en entornos HPC) [56].

Uso: Supongamos que tienes todos los DICOM de un sujeto en una carpeta. El uso básico sería:

```
bidskit -i /ruta/dicomdir -o /ruta/proyecto_BIDS
```

- `-i` indica la carpeta con los DICOM de entrada (puede contener subcarpetas).
- `-o` la carpeta raíz donde quieres el dataset BIDS de salida.

Al correr este comando, BIDSkit ejecutará el **primer paso** (primera pasada): convertirá cada serie DICOM a NIfTI (usando dcm2niix) y creará una estructura temporal dentro de la carpeta BIDS de salida. En `/ruta/proyecto_BIDS/` verás que creó `sub-XX_tmp/` con todas las imágenes convertidas con nombres genéricos, así como archivos `.json` sidecar con metadatos [57]. **Crucialmente**, BIDSkit genera en este momento un archivo JSON llamado típicamente `Protocol_Translator.json` en la raíz de la carpeta de DICOM de origen [58]. Este archivo contiene un listado de todas las series encontradas y campos para rellenar el destino BIDS deseado. Por ejemplo, puede listar algo así como:

```
{
    "Localizer": [
        "EXCLUDE_BIDS_Directory",
        "EXCLUDE_BIDS_Name",
        "UNASSIGNED"
    ],
    "RARE_T2star": [
        "anat",
        "T2star",
        "UNASSIGNED"
    ],
    "BOLD_rest": [
        "func",
        "task-rest_run-1_bold",
        "UNASSIGNED"
    ]
}
```

En este diccionario, las claves (izquierda) son los *Protocol Name* de tus series (p. ej. "Localizer", "RARE_T2star", "BOLD_rest"), detectados del DICOM [58]. Los valores son listas de tres elementos: `[<Carpeta_BIDS>, <Nombre_Archivo_BIDS>, <Campo_Adicional>]`. BIDSkit habrá auto-rellenado algunos si pudo (por ejemplo, podría reconocer "BOLD" y proponer func/task-*_bold). Muchos aparecerán con* `"UNASSIGNED"`*, lo que significa que debes editar** este JSON manualmente para completar la información según las reglas BIDS [59].

- El primer elemento de cada lista es el directorio BIDS: `"anat"`, `"func"`, `"dwi"`, `"fmap"`, etc., o `"EXCLUDE..."` si esa serie no quieres incluirla en BIDS.
- El segundo elemento es el nombre de archivo (sin `sub-` ni extensiones, BIDSkit se encargará de anteponer sub-ID y extensión `.nii.gz`). Debe seguir la nomenclatura BIDS: por ejemplo `T1w`, `task-rest_run-01_bold`, `dir-AP_epi` (para fieldmaps), etc.

- El tercer elemento suele quedar `"UNASSIGNED"` a menos que la serie corresponda a imágenes que requieren emparejamiento (por ejemplo, en fieldmaps BIDSkit permite aquí listar a qué archivos se aplica el fieldmap).

Tu tarea como usuario es editar **una vez** este `Protocol_Translator.json` colocando los valores correctos para cada serie [60] [61]. En el ejemplo arriba, podrías decidir excluir "Localizer" (ya aparece marcado para excluir), asignar `"RARE_T2star"` a anat T2*, y verificar* `"BOLD_rest"` *ya está en func rest bold. Tras editar y guardar el JSON, procedes al segundo paso**.

Ejecuta nuevamente el mismo comando `bidskit -i /ruta/dicomdir -o /ruta/proyecto_BIDS`. En la segunda pasada, BIDSkit detectará que existe un `Protocol_Translator.json` y lo usará para mover/renombrar los archivos NIfTI convertidos a su ubicación final en la estructura BIDS [62] [63]. Creará los subdirectorios `sub-01/anat`, `sub-01/func`, etc. según lo que indicaste, y colocará ahí los `.nii.gz` y `.json`. Las series marcadas como EXCLUDE no se copiarán a BIDS. Al finalizar, puedes borrar la carpeta temporal `_tmp` y revisar tu dataset BIDS completo.

Ventajas: BIDSkit ofrece un enfoque semiautomático: no tienes que crear el mapa de cero, sino que el programa te lo escribe y tú solo lo editas. Para principiantes, puede ser más intuitivo editar un JSON con ejemplos concretos (los nombres de tus series reales) que escribir una heurística Python desde cero. Además, este método garantiza que **todas las series serán convertidas** inicialmente, así que no hay riesgo de olvidar datos (solo decides qué incluir o excluir en BIDS tras ver todo lo que había). Es útil cuando tienes protocolos heterogéneos o no estás seguro de todas las secuencias: conviertes todo y luego clasificas.

Consideraciones: Requiere el paso manual de editar el JSON, lo cual es propenso a errores si no sigues estrictamente la nomenclatura BIDS. Por suerte, puedes validar el resultado con el *BIDS Validator* posteriormente (ver sección de validación). BIDSkit, al igual que las otras herramientas, no rellena por ti metadatos como nombres de tareas o contra-balanceos: esos detalles experimentales debes incorporarlos al nombrar (e.g., `task-<nombre>` en el archivo). También, si cambias tu protocolo de adquisición a mitad del estudio (nombres de serie diferentes), tendrás que actualizar el translator JSON y reconvertir las nuevas sesiones.

En contextos de ratones, BIDSkit funcionará bien con DICOM (o incluso si convertiste Bruker a DICOM antes). Sólo debes asegurarte de nombrar los sujetos con el prefijo `sub-` en la carpeta de salida y luego ajustar participants/species manualmente. Por ejemplo, para imágenes estructurales de ratón podrías asignar `"anat", "T2w"` a una secuencia de alta resolución. BIDSkit no tiene lógica específica de especie, se basa en los nombres de serie.

Método 5: Conversión manual con dcm2niix + BIDS Starter Kit (enfoque básico)

Como quinta alternativa, mencionamos el método **manual**, apropiado para conjuntos de datos pequeños o para entender la estructura BIDS "desde dentro". Consiste en usar directamente *dcm2niix* para convertir tus DICOM a NIfTI, y luego organizar y renombrar esos archivos a mano según la especificación BIDS, usando guías de ayuda como el *BIDS Starter Kit*.

Paso 1 – Convertir a NIfTI: Usa dcm2niix para convertir tus archivos DICOM (o PAR/REC) a NIfTI. Por ejemplo:

```
dcm2niix -z y -b y -f %p_%s -o /ruta/NIfTI_salidas /ruta/DICOM_entrada
```

- La opción `-z y` comprime a `.nii.gz`.
- La opción `-b y` genera un **sidecar JSON** para cada serie con metadatos BIDS básicos [1] (muy importante: contendrá TE, TR, matriz, etc., extraídos del DICOM).
- `-f %p_%s` define el nombre de archivo de salida como `<ProtocolName>_<SeriesNumber>`. Ajusta esto a tu preferencia; también podrías usar `%p` solo (nombre de protocolo) o `%t` (tipo), etc. Lo crucial es obtener nombres distinguibles para cada serie.

Repite esto para cada sujeto/sesión. Acabarás con directorios llenos de archivos `.nii.gz` y `.json` con nombres provisionales.

Paso 2 – Organizar según BIDS: Crea la carpeta raíz de tu dataset BIDS. Dentro, crea subdirectorios `sub-<ID>` para cada sujeto, y opcionalmente `ses-<label>` si hay sesiones múltiples. Luego, dentro de cada `sub-`/`ses-` crea las carpetas de modalidades necesarias: `anat/` para estructurales (T1, T2, etc.), `func/` para fMRI, `dwi/` para difusores, `fmap/` para fieldmaps, etc. (Consulta la especificación BIDS para la lista completa).

Mueve cada par NIfTI+JSON a su carpeta correspondiente y **renómbralos** con el patrón BIDS: `sub-<ID>[_ses-<label>]_<modulidad>[_task-<tarea>]_..._<sufijo>.nii.gz` (y misma base de nombre para .json). Por ejemplo:

- Un archivo anat T1 de sujeto 7: `sub-07/anat/sub-07_T1w.nii.gz` (+ `.json`).
- Un fMRI de resting state de sujeto 7: `sub-07/func/sub-07_task-rest_run-01_bold.nii.gz` (+ `.json`).
- Un fieldmap EPI AP para sujeto 7: `sub-07/fmap/sub-07_dir-AP_epi.nii.gz` (+ `.json`).

Deberás editar ligeramente los archivos JSON para agregar campos requeridos/recomendados que dcm2niix no llena. Por ejemplo, en fMRI BIDS es obligatorio `TaskName` en el JSON de bold (dcm2niix no sabe tu nombre de tarea), así que ábrelo con un editor de texto y añade `"TaskName": "rest"` si es resting-state. Igualmente, para datos de ratón, añade `"Species": "Mus musculus"` en `participants.json` o en cada JSON de imagen (según BEP, aunque lo más formal es en participants). También asegúrate de que las unidades de medidas en JSON (p.ej. *RepetitionTime*) estén según BIDS (segundos para TR, etc., normalmente dcm2niix ya lo hace bien).

Paso 3 – Archivos adicionales: Crea el archivo obligatorio `dataset_description.json` en la raíz. Puedes basarte en una plantilla del BIDS Starter Kit. En este JSON especificas al menos `"Name"` de tu dataset, `"BIDSVersion": "1.X.X"`, y opcionalmente `"License"`, `"Authors"`, etc. Crea también un `participants.tsv` listando los sujetos (`participant_id`) y sus características (columna `species` con "Mus musculus", quizá sexo, edad si aplica). Incluye un `participants.json` describiendo esas columnas (ejemplo disponible en Starter Kit). Añade un archivo `README` con cualquier detalle útil de tu estudio, y opcional un `CHANGES` si hay versión.

Paso 4 – Validación: Es fácil cometer errores manualmente, así que utiliza el **BIDS Validator** para revisar tu estructura. Existe una versión web muy cómoda: abre https://bids-standard.github.io/bids-validator/ en Chrome/Firefox, y selecciona la carpeta raíz de tu dataset (no sube tus datos, corre localmente en el navegador) [64] [65]. Te informará de errores o advertencias de cumplimiento BIDS. Corrige según indique (por ejemplo, nombres mal formados, campos faltantes, etc.) hasta obtener "Success" [66] [67]. También hay versión de línea de comando (instalable con `npm install -g bids-validator`) o incluso un Docker (`docker run --rm -v /path/dataset:/data bids/validator /data`). La validación es un paso fundamental para asegurar que tus datos están **listos para usarse con pipelines automatizados** [68].

Cuándo usar este método: Para *principiantes absolutos*, hacerlo manualmente con uno o dos sujetos es educativo: comprenderás la organización BIDS a detalle. Sin embargo, es **propenso a errores humanos** y muy laborioso si tienes muchos sujetos o series. Recomendamos este enfoque solo para datasets pequeños o cuando las herramientas automatizadas fallen en algún caso especial. En general, los métodos anteriores (Dcm2Bids, BIDScoin, etc.) automatizan gran parte de este proceso reduciendo errores. Aun así, siempre conviene conocer qué hay "bajo el capó" en BIDS por si debes hacer ajustes manuales.

Validación y pasos finales

Independientemente del método utilizado (automático o manual), **siempre valida tu dataset BIDS**. Usa el BIDS Validator como se indicó para detectar problemas. Revisa también que en tus archivos JSON críticos estén los parámetros necesarios (por ejemplo, asegúrate de que los JSON de bold tengan `RepetitionTime` correcto, `TaskName`, etc., los JSON de imágenes anatómicas tengan `MagneticFieldStrength`, `Manufacturer` y otros campos que vienen del DICOM). Si tus datos son de animal, comprueba que has documentado la cepa, especie y cualquier particularidad (existe una recomendación de campos `strain` y `strain_rrid` en `participants.tsv` para animales, si aplica [69] [70]).

Una vez validado, tu conjunto de datos estará **"BIDS compliant"**, listo para alimentar herramientas de análisis. Ahora podrás ejecutar sin problemas pipelines como **fMRIPrep**, que tomará directamente la carpeta BIDS y realizará el preprocesamiento completo [2], o calcular la calidad de tus datos con MRIQC, entre otros.

Por último, te recomendamos conservar los archivos de conversión (ej. heurística de HeuDiConv, config de Dcm2Bids, bidsmap de BIDScoin, translator de BIDSkit) junto con tus datos (quizá en la carpeta `code/`). Esto ayuda a reproducibilidad y a que otros entiendan cómo se obtuvieron esos archivos BIDS a partir de los originales.