ChatGPT

Guía para convertir datos fMRI crudos de ratón a formato BIDS (Windows)

La **Estructura de Datos de Imágenes Cerebrales (BIDS)** es un estándar para organizar y nombrar datos de neuroimagen (como fMRI) de forma coherente [1]. Convertir sus archivos crudos (por ejemplo, DICOM de MRI) al formato BIDS facilita el preprocesamiento con *pipelines* estandarizados (p. ej. *fMRIPrep*, *MRIQC*), el análisis reproducible y el intercambio de datos [2] [3]. A continuación, se presenta una guía paso a paso, en español y orientada a principiantes, de cinco métodos populares para convertir imágenes **fMRI de ratones** a BIDS en sistemas Windows. *Nota:* Nos enfocamos en formatos crudos comunes como DICOM/NIfTI (excluyendo datos Bruker, ya tratados en otros tutoriales).

Ejemplo conceptual: conversión de una colección de archivos DICOM crudos (izquierda) a un conjunto de carpetas y archivos organizados según el estándar BIDS (derecha). Las herramientas presentadas a continuación automatizan este proceso [4].

1. Dcm2Bids (herramienta Python de línea de comandos)

¿Qué es? Dcm2Bids es una herramienta escrita en Python que automatiza la conversión de DICOM a BIDS aprovechando el convertidor *dcm2niix* para generar archivos NIfTI+JSON, y luego reorganiza esos archivos en la estructura de directorios BIDS adecuada [5]. En esencia, *dcm2bids* toma las imágenes convertidas por dcm2niix y las renombra/ubica según las convenciones BIDS (por ejemplo, `sub-01/anat/sub-01_T1w.nii.gz`, `sub-01/func/sub-01_task-rest_bold.nii.gz`, etc.). Es una herramienta popular y flexible, pero requiere crear un archivo de **configuración JSON** para mapear sus series DICOM a los nombres BIDS correctos [6].

Preparativos: Antes de usar Dcm2Bids, asegúrese de tener instalado **Python 3** en Windows (se recomienda usar Anaconda o el instalador oficial de Python, recordando marcar "Add Python to PATH") [7]. Asimismo, necesita el programa **dcm2niix** instalado, ya que Dcm2Bids lo utiliza para convertir DICOM a NIfTI [8]. Puede descargar dcm2niix como ejecutable para Windows desde NITRC [9], y añadirlo al PATH o indicar su ubicación en la configuración.

Instalación de Dcm2Bids: Abra una consola de comandos (o Anaconda Prompt) en Windows y ejecute:

```
pip install dcm2bids
```

Esto instalará la última versión estable de Dcm2Bids y sus dependencias. *(Nota: Dcm2Bids requiere Python y el módulo `future`; la versión actual utiliza Python 3)* [8].

Crear el archivo de configuración: Dcm2Bids funciona mediante un archivo JSON de configuración específico de su estudio. Este archivo describe cómo reconocer cada tipo de adquisición y cómo nombrarla

en BIDS [6]. Por ejemplo, debe definir campos como `dataType` (tipo de dato BIDS: anat, func, dwi, fmap, etc.) y `modalityLabel` (modalidad específica: p. ej. `T1w`, `T2w`, `bold`) para cada serie, junto con criterios para identificar esa serie a partir de los metadatos DICOM (como el `SeriesDescription` o TE) [10]. A continuación se muestra una porción de ejemplo de un archivo de configuración JSON para ilustrar la sintaxis:

```
{
    "descriptions": [
        {
            "dataType": "anat",
            "modalityLabel": "T2w",
            "criteria": {
                "SeriesDescription": "*T2*",
                "EchoTime": 0.1
            }
        },
        {
            "dataType": "func",
            "modalityLabel": "bold",
            "customLabels": "task-rest",
            "criteria": {
                "SidecarFilename": "006*"
            }
        }
    ]
}
```

Ejemplo: La primera entrada indica que cualquier serie cuyo **SeriesDescription** contenga "T2" y TE=0.1 se etiquetará como imagen anatómica T2w [11]. La segunda mapea un fichero cuyo nombre comienza con "006" a una imagen funcional BOLD de tarea "rest" (reposo) [12] [13]. En la práctica, usted deberá editar el JSON según las convenciones de nomenclatura de sus DICOM. Dcm2Bids proporciona un ejemplo de config en su repositorio, y puede usar editores con resaltado de sintaxis JSON para facilitar esta tarea [14]. *Consejo:* Si sus series DICOM contienen información consistente en los nombres (p. ej., "T1", "BOLD") puede utilizar comodines `*` para emparejar esas cadenas [15].

Uso de Dcm2Bids: Una vez preparado el archivo de configuración (por ejemplo, `config.json`), ejecute Dcm2Bids desde la consola con los parámetros adecuados. Los argumentos básicos son: el directorio de DICOM de entrada, el identificador de sujeto (y sesión si aplica), la ruta al JSON de config, y la carpeta de salida BIDS. Por ejemplo, en Windows:

```
dcm2bids -d "C:\Datos\DICOM\sub001" -p sub001 -s ses1 ^
         -c "C:\mi_config\config.json" -o "C:\Datos\BIDS_output"
```

En este ejemplo, `-d` apunta al folder con los DICOM del sujeto, `-p` define el ID del participante (sin el prefijo `sub-`), `-s` la sesión (opcional), `-c` la ruta del archivo de configuración y `-o` el directorio donde

se creará la estructura BIDS. Al ejecutarlo, Dcm2Bids llamará internamente a *dcm2niix* para convertir los DICOM a NIfTI+JSON (archivos "sidecar") y luego moverá/renombrará esos archivos según el config, creando las carpetas `sub-<ID>/anat/`, `sub-<ID>/func/`, etc., con los nombres BIDS correctos [16] [17]. El resultado será un conjunto de directorios por sujeto (sub-001, sub-002, ...) con subdirectorios `anat`, `func`, etc., listos para análisis. Recuerde que *dcm2bids* procesa un sujeto (y sesión) a la vez [18], por lo que deberá ejecutar el comando para cada sujeto o escribir un pequeño script *batch* si tiene muchos.

Verificación: Tras la conversión, verifique la estructura generada. Debería contener archivos `.nii.gz` y sus `.json` correspondientes, con nombres válidos BIDS, y archivos como `dataset_description.json`, `participants.tsv` (si agregó info de participantes) y quizás archivos de eventos TSV si correspondiera. Es recomendable correr el **BIDS Validator** sobre la carpeta resultante para asegurar que cumple con el estándar (existe una versión web y una de línea de comando del validador).

Resumen Dcm2Bids: Es una solución robusta para automatizar conversiones complejas. Requiere un esfuerzo inicial en crear el archivo de configuración, pero una vez hecho, la conversión es rápida y consistente. Este método soporta datos de **fMRI de ratón** en la medida que sus DICOM tengan metadata reconocible; solo deberá mapear adecuadamente las secuencias (por ejemplo, diferenciar anatómicos vs funcionales según el protocolo de su escáner preclínico).

2. HeuDiConv (convertidor flexible con "heurísticas" en Python)

¿Qué es? *HeuDiConv* es otra herramienta de conversión DICOM→BIDS muy utilizada en la neuroimagen. Su nombre proviene de "Heuristic DICOM Converter", ya que se basa en **heurísticas personalizables** para clasificar y nombrar los datos [19] [20]. En lugar de un archivo JSON de configuración, HeuDiConv utiliza un **script Python de heurística** en el que el usuario define cómo deben nombrarse y organizarse las series. Esta herramienta es muy poderosa y adaptable (permite manejar estructuras de directorios de entrada variadas, múltiples sesiones, etc.), e incluso registra la procedencia de la conversión y se integra con *DataLad* para control de versiones [21] [22]. Internamente, HeuDiConv también emplea *dcm2niix* para la conversión de DICOM a NIfTI de forma rápida y sin pérdidas [23], pero delega al usuario la lógica de organización mediante la heurística.

Instalación: HeuDiConv se distribuye como un paquete de Python. Puede instalarlo en Windows usando pip, asegurándose de tener Python 3 instalado. En la consola de comandos, ejecute:

```
pip install heudiconv
```

Esto instalará *heudiconv* y sus dependencias. Como con Dcm2Bids, necesitará también tener disponible *dcm2niix* (instalado por separado o mediante una versión de *heudiconv* que lo incluya). Si la instalación nativa en Windows presenta dificultades, otra opción es utilizar **Docker**: existe una imagen oficial de heudiconv en Docker Hub que puede ejecutarse en Windows 10+ (requiere habilitar Docker Desktop) [24]. No obstante, aquí nos centraremos en el uso nativo para principiantes.

Concepto de heurística: Al usar HeuDiConv, debe proporcionar una "heurística" que le indique al programa cómo mapear los datos de origen a BIDS. Puede ser un archivo `.py` escrito por usted o una heurística

predefinida. Afortunadamente, HeuDiConv incluye algunas heurísticas genéricas ya incorporadas [25]. Por ejemplo:

- La heurística `"convertall"` simplemente convertirá **todas** las series DICOM a NIfTI y las colocará en BIDS asignando nombres genéricos basados en la información presente (útil para un volcado rápido) [26].
- La heurística `"reproin"` implementa un *convenio de nomenclatura* específico: si sus secuencias DICOM fueron nombradas siguiendo ciertas reglas (*ReproIn*), entonces HeuDiConv puede identificar automáticamente anatómicos, funcionales (y sus tareas), etc., sin configuración adicional [27]. Esta es ideal si puede controlar los nombres de secuencia desde la adquisición.

Para la mayoría de los casos de conversiones retrospectivas (datos ya adquiridos), probablemente tendrá que crear su propio fichero de heurística en Python [27]. Pero si es principiante, puede comenzar usando `convertall` para volcar los datos a BIDS sin preocuparse por detalles, y luego reorganizar/renombrar si es necesario.

Uso básico de HeuDiConv: Supongamos que quiere convertir los DICOM del sujeto `sub001` ubicados en `C:\Datos\DICOM\sub001\...`. Un comando mínimo usando la heurística convertall sería:

```
heudiconv -d "C:/Datos/DICOM/{subject}/*/*.dcm" -o "C:/Datos/BIDS_output" ^
          -f convertall -s sub001 -c dcm2niix -b
```

Explicación de parámetros: - `-d` especifica el patrón de búsqueda de archivos DICOM, donde `{subject}` se reemplaza por el ID (en este caso `sub001`). El comodín `*/*.dcm` asume que los DICOM pueden estar en subcarpetas. Ajuste esto según su estructura de origen. - `-o` indica la carpeta de salida para los datos convertidos en BIDS. - `-f` es la heurística a usar (`convertall` en este ejemplo; podría ser una ruta a un .py personalizado). - `-s` es el sujeto (sin prefijo `sub-`). - `-c dcm2niix` le dice que use *dcm2niix* para la conversión de imágenes (es el método por defecto; `-c` significa *converter*). - `-b` indica que agregue archivos *sidecar* BIDS adicionales si aplica (por ejemplo, archivos `.bval`, `.bvec` para DWI, etc., y un archivo `dataset_description.json` mínimo).

Al ejecutar, HeuDiConv recorrerá los DICOM, creará un directorio `sub-001` en la salida, y colocará allí los NIfTI y JSON. Con `convertall`, la organización por defecto suele ser sub-001/ses-?/**serie_descripción**. Si usa `reproin` y sus datos siguen ese estándar, la organización ya será BIDS-compliant. De lo contrario, los nombres pueden requerir edición manual posterior.

Heurística personalizada: Para una conversión fina, deberá crear un archivo Python, digamos `myheuristic.py`, donde define una función `infotoids` (para obtener IDs) y un diccionario `heuristic` que mapea cada serie a un nombre BIDS. Esto requiere conocimientos básicos de Python. La documentación de HeuDiConv y ejemplos en su repo GitHub son referencias útiles [27] [28]. Sin embargo, dado que esta guía es para principiantes absolutos, profundizar en escritura de heurísticas podría ser abrumador. Una estrategia más simple es usar primero Dcm2Bids o BIDScoin para generar un dataset BIDS, y posteriormente adoptar HeuDiConv si se necesita mayor flexibilidad en proyectos complejos.

Resumen HeuDiConv: Es altamente **flexible** y potente – preferida en entornos de investigación con múltiples estudios – pero para principiantes la curva de aprendizaje es mayor porque implica scripting. Aun

así, puede usarse en Windows (ya sea instalándolo con pip o mediante Docker) y soporta datos de ratón de forma similar a humanos, siempre que usted defina la heurística apropiada. Recuerde que, como con cualquier método, los archivos NIfTI producidos deben seguir las convenciones BIDS. HeuDiConv **no** adivina completamente la intención de cada serie a menos que se le indique; por ello, a veces es conveniente comenzar con otra herramienta más sencilla y luego migrar a HeuDiConv cuando adquiera experiencia.

3. BIDScoin (suite con GUI para conversión amigable)

¿Qué es? *BIDScoin* es un conjunto de herramientas (con comandos CLI y **interfaz gráfica**) diseñado para convertir datos brutos a BIDS de manera **amigable para el usuario**, minimizando la necesidad de programación [4]. A diferencia de heurísticas o config files escritos a mano, BIDScoin utiliza un enfoque de **mapeo interactivo**: escanea sus datos de origen para identificar las distintas series presentes y le permite mapear cada tipo de serie a un formato BIDS mediante una interfaz visual [29] [30]. Esto resulta muy útil para principiantes, ya que uno puede "apuntar y clickear" para asignar, por ejemplo, qué series son anatómicas, cuáles funcionales (y con qué nombres de tarea), etc., con sugerencias iniciales proporcionadas por el programa. BIDScoin es multiplataforma (Windows, Mac, Linux) al estar basado en Python, y no requiere conocimientos de programación para casos comunes [31] [4].

Instalación: Primero, instale Python 3 en Windows si no lo tiene (como se indicó antes, usar el instalador de Python y habilitar PATH) [7]. Luego, se recomienda crear un entorno virtual para aislar la instalación [32], pero no es obligatorio. Para instalar BIDScoin, abra la consola y ejecute:

```
pip install bidscoin[dcm2niix2bids]
```

El *extra* `[dcm2niix2bids]` instalará también la dependencia *dcm2niix* dentro del entorno, específicamente para manejar DICOM a NIfTI [33]. Alternativamente, `pip install bidscoin` a secas instalará la suite base y usted tendría que proveer el ejecutable dcm2niix por su cuenta [33]. Tras la instalación, debería disponer de comandos como `bidsmapper`, `bidseditor` y `bidscoiner`.

Organización inicial de datos: Antes de ejecutar BIDScoin, organice sus archivos DICOM de manera estructurada. Lo ideal es tener una carpeta raíz (por ejemplo `C:\Datos\Estudio1\DICOM\`) con subcarpetas por sujeto (ej. `sub001`, `sub002`, …), y dentro de cada una las imágenes DICOM (posiblemente en subdirectorios por serie o sesión). BIDScoin puede trabajar con diferentes organizaciones, pero tenerlos separados por sujeto facilita el mapeo.

Flujo de trabajo de BIDScoin: El proceso típico consta de *dos pasos principales* (con un paso intermedio opcional de edición):

1. **Mapeo inicial –** `bidsmapper`**:** Este comando escanea el directorio fuente de DICOM y genera un archivo de mapeo llamado **bidsmap** (formato YAML) que lista cada *run* o serie encontrada y propone un nombre BIDS tentativo [34]. Por ejemplo, puede listar una serie con nombre "RARE_T2" y sugerir `anat/T2w` como destino. El bidsmap es básicamente una tabla de conversión que BIDScoin irá construyendo a medida que descubra tipos de datos [35].

2. **Revisión/edición (opcional) –** `bidseditor`**:** BIDScoin ofrece una GUI para editar el *bidsmap* generado. Al ejecutar `bidseditor`, se abrirá una ventana mostrando en una columna las series DICOM identificadas (por nombre, número de serie, etc.) y en otra columna el nombre BIDS de salida propuesto [36]. Mediante esta interfaz, usted puede ajustar cualquier asignación: por ejemplo, marcar que "RARE_T2" realmente es `anat/T2w` (anatómica T2) o quizá cambiar alguna serie funcional para agregarle el label de tarea correcto (`task-xxx`) [37]. Las entradas aparecen en verde si cumplen con BIDS o en rojo si requieren información adicional [38], lo que guía al usuario sobre qué campos son obligatorios (por ejemplo, a una serie *bold* se le exigirá especificar el `TaskName`). Puede hacer doble clic en los campos para editarlos, usar menús desplegables para valores predefinidos, e incluso inspeccionar metadatos DICOM completos desde la interfaz [39] [40]. Una vez satisfecho con la correspondencia, guarde los cambios en el bidsmap.

3. **Conversión final –** `bidscoiner`**:** Este comando toma el *bidsmap* verificado y procede a convertir y organizar los datos. En esta etapa, BIDScoin utilizará el *plugin* apropiado (por defecto, **dcm2niix**) para convertir los DICOM a NIfTI+JSON [41], creando la estructura de carpetas BIDS de salida. Puede ejecutar `bidscoiner` directamente, o desde la GUI del editor hay un botón para iniciar la conversión una vez listo. Durante el proceso, verá en la terminal cómo cada serie se convierte y copia a su lugar (anat, func, etc.) [42]. Al finalizar, tendrá en la carpeta de salida todos los sujetos como `sub-XX` con sus archivos convertidos y archivos de documentación (`dataset_description.json`, etc.).

Ejemplo rápido: Supongamos que sus DICOM están en `C:\Datos\DICOM\`. Podría hacer:

```
bidsmapper -s C:\Datos\DICOM -o C:\Datos\BIDS_output
bidseditor C:\Datos\BIDS_output\bidsmap.yaml   (edite según necesario y guarde)
bidscoiner -s C:\Datos\DICOM -o C:\Datos\BIDS_output
```

Aquí `-s` indica la carpeta fuente y `-o` la de destino BIDS. El archivo `bidsmap.yaml` se habrá creado en la carpeta de salida tras el primer paso. En la GUI de `bidseditor` confirmará que todo esté correcto. Luego `bidscoiner` realizará la conversión usando ese mapa.

Características adicionales: BIDScoin soporta múltiples formatos de entrada (DICOM, PAR/REC de Philips, NIfTI huérfanos, datos de PET, etc.) mediante *plugins*, pero para fMRI estándar con DICOM el plugin `dcm2niix2bids` es el usado [31]. También permite usar expresiones regulares avanzadas en el bidsmap para coincidir nombres, pero esto es opcional. Una vez configurado un bidsmap para un estudio (p. ej. un protocolo específico de ratón), puede reutilizarlo en lotes de datos similares, ahorrando trabajo. El programa almacena *plantillas* de mapping que pueden ser específicas por centro o estudio, ofreciendo sugerencias inteligentes la próxima vez [43].

Resumen BIDScoin: Es una solución excelente para **usuarios novatos**, ya que combina automatización con intervención guiada: uno deja que detecte todo automáticamente, pero mantiene el control para corregir o completar la información necesaria mediante una interfaz intuitiva. Funciona en Windows sin problema (solo requiere la instalación de Python y Qt para la GUI, que pip maneja automáticamente). Tras la conversión, obtendrá un dataset BIDS completo. Asegúrese de validar el resultado; BIDScoin genera datos compatibles, pero siempre es bueno ejecutar el validador BIDS. En contextos de **imágenes de ratón**,

BIDScoin no diferencia por especie de manera intrínseca, así que usted deberá añadir manualmente la información de especie en los metadatos (ver la nota al final sobre datos de animales). Por lo demás, el proceso es idéntico al de humanos.

Imagen: Captura de la interfaz gráfica **BIDScoin (bidseditor)** mostrando la asignación de tipos de datos. A la izquierda se enumeran las series detectadas en los DICOM (con sus nombres originales), y a la derecha se muestran los nombres BIDS de salida propuestos (pudiendo editarse). Las entradas en verde cumplen con el estándar, mientras que en rojo requieren atención (por ejemplo, completar campos obligatorios como el nombre de tarea) [38] [44] . Esta GUI permite a usuarios principiantes mapear interactivamente sus datos a BIDS sin escribir código.

4. BIDSKIT (utilidad de conversión en dos pasos con archivo traductor)

¿Qué es? *BIDSKIT* (a veces estilizado como BidsKit) es una herramienta ligera escrita en Python que facilita la conversión de DICOM a BIDS mediante un proceso de **dos pases** con un archivo de traducción intermedio [45] . Fue desarrollada para simplificar la tarea común de convertir estudios de MRI a BIDS, ofreciendo comandos de consola sencillos. Al igual que Dcm2Bids, BIDSKIT utiliza *dcm2niix* para convertir imágenes a NIfTI, pero difiere en que genera automáticamente un archivo (en formato JSON/YAML) que contiene la *información de las series* y espera que el usuario lo revise/edite para completar la conversión [46] . Esto le da al usuario control sobre cómo se nombrará cada serie en BIDS, de una forma quizá más sencilla que escribir una heurística desde cero.

Instalación: BIDSKIT está disponible via pip. En Windows, instale Python 3 y luego en la terminal ejecute:

```
pip install bidskit
```

Esto instalará un comando llamado `bidskit` que podrá usar directamente. Asegúrese igualmente de tener *dcm2niix* accesible, ya que BIDSKIT lo invocará para leer/convertir DICOM (si no está en PATH, puede especificar su ruta en un archivo de configuración avanzado o simplemente colocar el ejecutable junto a los datos).

Organizar datos de origen: BIDSKIT requiere que los DICOM estén organizados por sujeto (y opcionalmente por sesión) en carpetas separadas [47] . Por ejemplo:

```
DICOM_dir/
└── sub01/
    ├── ses1/ (opcional)
    │    └── [archivos DICOM...]
    └── ses2/ ...
└── sub02/
    └── ...
```

Si sus datos no incluyen sesiones múltiples, basta con una carpeta por sujeto con todos sus DICOM. Esta organización es necesaria porque BIDSKIT procesará sujeto por sujeto.

Paso 1 – Conversión inicial y creación del "translator": Ejecute BIDSKIT indicando la carpeta de DICOM y una carpeta de salida BIDS temporal. Por ejemplo:

```
bidskit --init  -i C:\Datos\DICOM\sub01 -o C:\Datos\BIDS_temp\sub-01
```

(Tenga en cuenta que BIDSKIT podría ejecutar ambos pases con un solo comando, pero aquí lo explicamos separadamente para mayor claridad.) Durante este paso "Pass 1", BIDSKIT convertirá todos los DICOM de ese sujeto a NIfTI (colocándolos quizás en una subcarpeta `nifti/` temporal) y **generará un archivo traductor** (translator) que contiene la descripción de cada serie encontrada [45] [48]. Este archivo, posiblemente llamado `protocol_translator.json` o similar, listará entradas como: serie 1: "RARE_T2" -> (vacío), serie 2: "EPI_BOLD_rest" -> (vacío), etc., esperando que se rellenen con el destino BIDS.

- *Nota:* En versiones recientes, el comando podría ser simplemente `bidskit` sin `--init`, y él solo hará ambos pasos, dejando un JSON en el directorio de salida. Si sólo hace la mitad, revise la documentación actual de BIDSKIT (comando `bidskit --help`) ya que la sintaxis puede variar ligeramente. La idea principal es que se realiza una conversión a NIfTI y se saca un "diccionario" de series en este paso 1.

Paso 2 – Editar el archivo traductor: Abra el archivo de traducción generado (es un JSON o YAML). Allí verá cada serie DICOM con campos para asignar el `dataType` (anat, func, etc.), el `modality` (T1w, bold, etc.), e información como el nombre de tarea si corresponde. Edite cada entrada colocando el valor BIDS correcto. Por ejemplo, para la serie "RARE_T2" puede asignar dataType: "anat", modality: "T2w". Para "EPI_BOLD_rest" asignar dataType: "func", modality: "bold", customLabels o TaskName: "rest", etc. El formato es similar al de la config de Dcm2Bids pero más simple (por serie en lugar de patrones). Guarde los cambios. **Este paso es crucial:** hasta que no complete el translator, el programa no sabe cómo renombrar los NIfTI.

Paso 3 – Organización final en BIDS: Una vez editado el translator, ejecute BIDSKIT nuevamente para realizar el "Pass 2" y crear la estructura BIDS definitiva. El comando podría detectar automáticamente el translator; si no, sería algo como:

```
bidskit --convert -i C:\Datos\BIDS_temp\sub-01 -o C:\Datos\BIDS_output
```

En este paso, BIDSKIT leerá su archivo de traducción con las asignaciones y **moverá/renombrará los archivos NIfTI** a las carpetas BIDS correspondientes [46]. Por ejemplo, creará `sub-01/anat/sub-01_T2w.nii.gz` para esa serie T2, `sub-01/func/sub-01_task-rest_bold.nii.gz` para la serie BOLD, etc., según lo que indicó. También generará los archivos `.json` sidecar BIDS (tomados de la conversión dcm2niix) en esas ubicaciones, y archivos de registro. Tras este paso, la conversión para sub-01 está completa. Repita el proceso para cada sujeto.

Comprobación: Navegue por `C:\Datos\BIDS_output`. Debería ver subcarpetas `sub-01`, `sub-02`, etc., cada una con `anat/`, `func/`, etc. Revise que los nombres de archivos incluyan los campos

obligatorios (por ejemplo, las bold tengan `_task-..._bold.nii.gz`). Si olvidó algún campo en el translator (p. ej. una tarea), puede editarlo y volver a correr el paso 2.

Ventajas y consideraciones: BIDSKIT es útil porque le guía en dos etapas: primero extrae toda la info de las series (asegurándose de no olvidar ninguna) y luego le deja a usted decidir las etiquetas BIDS sin tener que escribir código. Es más sencillo que HeuDiConv en ese sentido, aunque parecido a Dcm2Bids en que se basa en un archivo de mapeo. Una diferencia es que Dcm2Bids requiere pre-definir el mapeo, mientras que BIDSKIT le ayuda a generarlo a partir de los datos mismos. Para un principiante, esto puede ser más intuitivo. La desventaja es que es un proceso un poco manual (no tan automático como BIDScoin) y puede no ser tan mantenido/actualizado como otras herramientas más populares. Aun así, cumple su función.

Resumen BIDSKIT: En Windows, BIDSKIT corre sin problemas (puede incluso usarse vía Docker si se prefiere). Es una buena opción si quiere un control preciso sin aprender Python, aunque requiere editar archivos de texto estructurados. Tras usarlo, siempre valide el dataset BIDS resultante. BIDSKIT soporta los formatos DICOM estándar; para datos de ratón, el procedimiento es igual: lo importante es que usted asigne correctamente las modalidades (e.j. muchas secuencias anatómicas de animal son T2 en lugar de T1, etc., ajústelo en el translator).

5. BIDS'em ALL (aplicación GUI autónoma en Java)

¿Qué es? *BIDS'em ALL* es una aplicación con interfaz gráfica de usuario, desarrollada en Java, que convierte automáticamente una carpeta de DICOM en un conjunto de archivos NIfTI organizados en BIDS [49]. Su filosofía es "todo en uno": usted selecciona la carpeta con los DICOM, define algunos parámetros básicos en la GUI, y la herramienta se encarga de todo el flujo de conversión, asistido por algunas indicaciones interactivas. Es **multiplataforma** (Windows, Mac, Linux) al correr sobre la Máquina Virtual de Java, y está pensada para ser fácil de usar para quienes no están cómodos con la línea de comandos. Sólo requiere tener instalado Java (versión 8) en el sistema [50].

Instalación/Descarga: BIDS'em ALL no se instala como tal, sino que se distribuye como un archivo ejecutable Java. Debe tener Java Runtime Environment (JRE) 1.8 u 8 instalado en Windows [51]. Si no lo tiene, descárguelo del sitio oficial de Java e instálelo [52]. Luego descargue el archivo `bidsemall.jar` desde la página oficial de BIDS'em ALL [53] (por ejemplo, desde 2vr1.com, que es el sitio del desarrollador). Guarde `bidsemall.jar` en una carpeta conveniente.

Uso de la aplicación: Para lanzar BIDS'em ALL, simplemente haga doble clic sobre el archivo JAR (o ejecútelo con `java -jar bidsemall.jar` en consola). Se abrirá la ventana principal del programa [54]. Los pasos típicos dentro de la GUI son:

1. **Seleccionar directorios:** En la ventana principal, haga clic en "Input DICOM directory" y navegue hasta la carpeta que contiene sus archivos DICOM (puede ser una carpeta raíz con subcarpetas; la aplicación buscará recursivamente, no necesita un `DICOMDIR`) [55]. Luego haga clic en "Output directory" y elija o cree la carpeta donde quiere que se guarde el dataset BIDS convertido.

2. **Analizar DICOM:** Al hacer clic en "Convert", BIDS'em ALL primero escaneará todos los archivos DICOM de entrada. **Paso 1:** Analiza los DICOM para extraer los metadatos de paciente y series (ProtocolName, SeriesDescription, etc.) [56].

3. **Asignar identificadores:** Luego, la aplicación le pedirá que **especifique un ID de sujeto** para usar en BIDS [56]. Por defecto usará el Patient ID de los DICOM, pero puede cambiarlo. Ese ID será el que aparezca como `sub-XXX`. Si hay múltiples sujetos en la carpeta, los listará para que elija o confirme sus IDs.

4. **Clasificar cada serie:** Este es el paso interactivo clave. Para cada serie única detectada (identificada por una combinación de *ProtocolName* y *SeriesDescription*), BIDS'em ALL le solicitará que **identifique el tipo de dato** [57]. Por ejemplo, si encuentra una serie llamada "T2w_TurboRARE", usted debe indicar si es una secuencia anatómica T2 (`anat/T2w`), una funcional, etc. La interfaz probablemente presenta una lista desplegable de modalidades comunes (T1w, T2w, FLAIR, bold, fmap, etc.) y resaltará en rojo los campos obligatorios que falten [58]. Para una serie *funcional (bold)*, típicamente pedirá el `TaskName` (nombre de la tarea) que debe ingresarse. Usted llena los campos necesarios para cumplir con BIDS. **Nota:** Este entrenamiento solo se hace la primera vez por cada tipo de serie; BIDS'em ALL guardará sus elecciones en un archivo de configuración (un `.yml`) [59]. Así, si más adelante convierte otro conjunto de DICOM con las mismas series (por ejemplo, otro sujeto del mismo experimento), **ya no le preguntará de nuevo**: aplicará automáticamente la misma clasificación. Esto acelera el procesamiento en estudios grandes.

5. **Conversión automática:** Tras clasificar las series, la herramienta procede a convertir todo. **Paso 2:** Organiza los DICOM en una estructura tipo BIDS preliminar bajo una carpeta `sourcedata` dentro del directorio de salida, separando por sujeto/sesión según corresponda [60] [61]. **Paso 3:** Llama a *dcm2niix* en segundo plano para convertir cada DICOM en archivos NIfTI (.nii.gz) junto con sus sidecars JSON [61]. Los NIfTI resultantes son guardados en la carpeta final BIDS (denominada posiblemente `rawdata` en esta aplicación) con los nombres correctos y todos los archivos requeridos (incluyendo `.json` de imágenes y archivos `.tsv`/`.json` para participantes, etc. si aplica). En resumen, después de este proceso, usted tendrá en la carpeta de salida una subcarpeta por sujeto (`sub-<ID>`) con sus datos en formato BIDS, listos para usar.

6. **Opciones adicionales:** BIDS'em ALL tiene algunas opciones en el menú "Options", como la posibilidad de **sobrescribir o mantener** datos previamente convertidos al repetir la conversión [62]. También permite saltar directamente al paso de conversión si uno ya tiene los DICOM organizados en `sourcedata` con la estructura BIDS (sirve para reconvertir rápidamente) [62]. Asimismo, guarda el archivo de asociación serie-BIDS (`DIC.yml`) en el directorio de la app, que puede editar manualmente si es necesario para ajustar algún mapeo a mano [63].

Ejemplo simplificado: Usted carga los DICOM de un ratón que incluyen una serie anatómica T2, una serie funcional de resting-state, y quizá un fieldmap. BIDS'em ALL le pedirá: *"¿Qué tipo de serie es 'T2w_TurboRARE'?"* – selecciona "T2w (anatómica)" y llena campos obligatorios (ninguno extra en este caso). Luego: *"¿Qué es 'BOLD_EPI_rest'?"* – selecciona "bold (funcional)", ingresa TaskName = "rest". *"¿Y 'FieldMap'"?* – podría seleccionar "fmap" y elegir el tipo (magnitude/phase o si es un *PEpolar*). Una vez hecho esto, el programa convierte todo. Cuando procese el siguiente sujeto con las mismas secuencias, recordará que *BOLD_EPI_rest = bold task-rest*, etc., y no preguntará de nuevo, haciendo la conversión con un solo clic.

Resumen BIDS'em ALL: Es quizás **la herramienta más sencilla para un principiante absoluto**, ya que prácticamente no requiere usar la terminal ni editar archivos de texto. La interfaz guía al usuario por los mínimos necesarios para cumplir BIDS. A cambio, es menos flexible si se necesitan ajustes muy específicos, pero cumple con la gran mayoría de casos de uso estándar. Funciona perfectamente en Windows (solo

garantice tener Java 8). Al final, valide el dataset BIDS con una herramienta de validación, aunque BIDS'em ALL debería generar todos los metadatos obligatorios. Esta herramienta es relativamente nueva, pero al estar construida sobre dcm2niix, su conversión de imágenes es confiable [61] .

🔍 Notas finales y mejores prácticas

- **Validación:** Una vez obtenga sus datos en BIDS con cualquiera de los métodos, utilice el validador de BIDS para verificar que la estructura y nombres cumplen el estándar. Muchos de los programas mencionados (Dcm2Bids, BIDScoin, etc.) crean datos compatibles, pero el validador puede detectar archivos faltantes (p. ej. un `.json` faltante, o un campo obligatorio no llenado).

- **Documentación adicional:** Si bien aquí proporcionamos una guía en español, las documentaciones oficiales (en inglés) de cada herramienta profundizan en opciones avanzadas. Por ejemplo, Dcm2Bids tiene guías para su versión 3.x con nuevas funciones [64] , HeuDiConv tiene tutoriales en NeuroStars para casos específicos, BIDScoin cuenta con un artículo académico detallando su arquitectura [65] , etc. Consulte esas fuentes conforme adquiera confianza.

- **Datos de animales en BIDS:** El estándar BIDS fue concebido originalmente para humanos, pero se ha ido extendiendo para soportar datos de animales. En el caso de ratones, es importante añadir la información de la especie en los archivos correspondientes. **BIDS recomienda** incluir una columna `species` con el nombre binomial de la especie (ej. *Mus musculus* para ratón) en el archivo `participants.tsv` [66] , así como aclararlo en el `dataset_description.json` si aplica. Ninguno de los convertidores anteriores agrega automáticamente `species`, por lo que deberá hacerlo manualmente después (usando un editor de texto o Excel). Esto asegurará que su dataset cumpla con las extensiones de BIDS para animales y permita a herramientas de análisis inferir correctamente la especie.

- **Elección de método:** ¿Cuál método debería usar? Para un **principiante absoluto con Windows**, si desea evitar la terminal lo más posible, empiece con **BIDS'em ALL** o **BIDScoin**, ya que ofrecen GUI amigables y procesos guiados. Si se siente cómodo editando un JSON simple, **Dcm2Bids** es muy efectivo y ampliamente usado. Para proyectos más complejos o si ya tiene experiencia en Python, **HeuDiConv** ofrece máxima flexibilidad. **BIDSKIT** puede ser un punto intermedio si le gusta entender y controlar el proceso paso a paso. No hay una "única" manera correcta – todas logran el objetivo de obtener un conjunto de datos BIDS válido; la diferencia está en la facilidad de uso vs. flexibilidad.

Esperamos que este manual le haya brindado una orientación clara. Con sus datos fMRI de ratón correctamente organizados en BIDS, estará listo para aplicar pipelines de preprocesamiento estandarizados y realizar análisis reproducibles de manera mucho más sencilla [2] . ¡Éxito en sus conversiones y futuros análisis!

Referencias (fuentes): BIDS estándar y documentación de herramientas relevantes [3] [6] [23] [4] [8] [56] [50] [66] , entre otras. Cada herramienta mencionada cuenta con manuales detallados en línea para profundizar en configuraciones avanzadas o resolver problemas específicos. Buena conversión y que sus datos queden "*BIDSificados*" correctamente.

---

1 2 3 BIDS Overview and Tutorial — Andy's Brain Book 1.0 documentation
https://andysbrainbook.readthedocs.io/en/stable/OpenScience/OS/BIDS_Overview.html

4 29 30 31 41 43 65 BIDScoin: Coin your imaging data to BIDS — BIDScoin 4.5.0 documentation
https://bidscoin.readthedocs.io/en/stable/

5 8 9 11 12 13 14 15 16 17 18 dcm2bids · PyPI
https://pypi.org/project/dcm2bids/1.1.6/

6 10 64 DICOM to BIDS conversion - CDNI's Brain
https://cdnis-brain.readthedocs.io/dcm2bids/

7 32 33 Installation — BIDScoin 4.5.0 documentation
https://bidscoin.readthedocs.io/en/stable/installation.html

19 20 21 22 23 25 26 27 28 HeuDiConv — heudiconv 1.3.3 documentation
https://heudiconv.readthedocs.io/

24 50 DICOM to BIDS conversion on Windows - bids - Neurostars
https://neurostars.org/t/dicom-to-bids-conversion-on-windows/5009

34 The BIDScoin workflow — BIDScoin 4.3.1 documentation
https://bidscoin.readthedocs.io/en/4.3.1/workflow.html

35 The BIDScoin workflow — BIDScoin 3.0.8 documentation
https://bidscoin.readthedocs.io/en/3.0.8/workflow.html

36 37 38 39 40 42 44 Screenshots — BIDScoin 4.6.2 documentation
https://bidscoin.readthedocs.io/en/stable/screenshots.html

45 bidskit/bidskit/__main__.py at master · jmtyszka/bidskit - GitHub
https://github.com/jmtyszka/bidskit/blob/master/bidskit/__main__.py

46 48 Empty output directory and .json file #3 - jmtyszka/bidskit - GitHub
https://github.com/jmtyszka/bidskit/issues/3

47 amiklos/bidskit - Docker Image
https://hub.docker.com/r/amiklos/bidskit

49 51 52 53 54 55 56 57 58 59 60 61 62 63 BIDS'em ALL - Convert DICOM images to BIDS compatbile nifti files
https://2vr1.com/bidsemall/

66 [PDF] Ecosysteme logiciel de segmentation de neurones sur des images ...
https://publications.polymtl.ca/10448/1/2022_MarieHeleneBourget.pdf