heudiconv Documentation

Release 1.1.1

Heudiconv team

May 02, 2024

CONTENTS

a heuristic-centric DICOM converter

CHAPTER

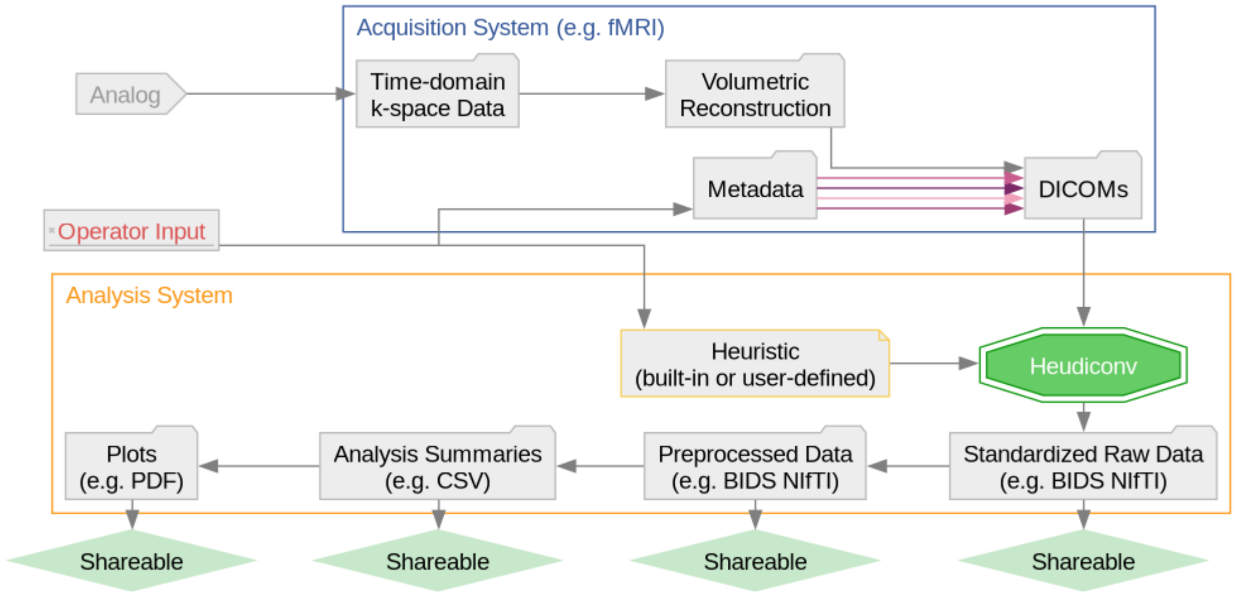
ONE

ABOUT

`heudiconv` is a flexible DICOM converter for organizing brain imaging data into structured directory layouts.

- It allows flexible directory layouts and naming schemes through customizable heuristics implementations.
- It only converts the necessary DICOMs and ignores everything else in a directory.
- You can keep links to DICOM files in the participant layout.
- Using dcm2niix under the hood, it's fast.
- It can track the provenance of the conversion from DICOM to NIfTI in W3C PROV format.
- It provides assistance in converting to BIDS.
- It integrates with DataLad to place converted and original data under git/git-annex version control while automatically annotating files with sensitive information (e.g., non-defaced anatomicals, etc).

Heudiconv can be inserted into your workflow to provide automatic conversion as part of a data acquisition pipeline, as seen in the figure below:



Acquisition System (e.g. fMRI)
Analog
Time-domain k-space Data
Volumetric Reconstruction
Metadata
DICOMs
Operator Input
Analysis System
Heuristic (built-in or user-defined)
Heudiconv
Plots (e.g. PDF)
Analysis Summaries (e.g. CSV)
Preprocessed Data (e.g. BIDS NIfTI)
Standardized Raw Data (e.g. BIDS NIfTI)
Shareable
Shareable
Shareable
Shareable

CHAPTER

TWO

INSTALLATION

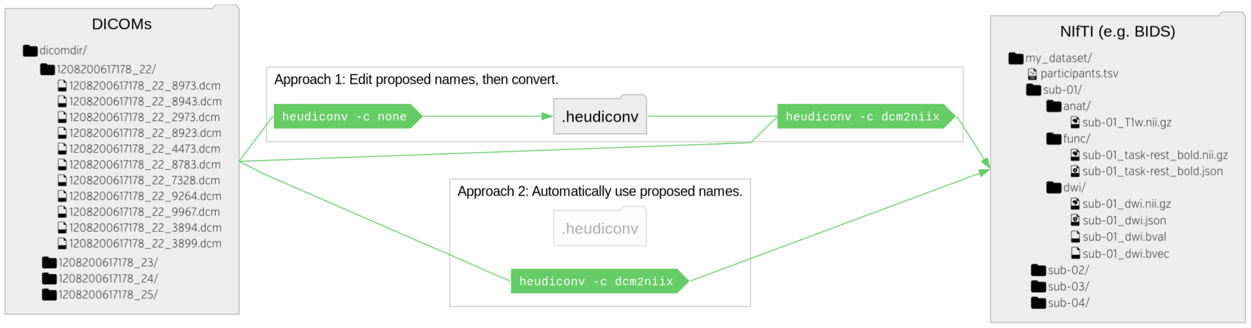
See our installation page on heudiconv.readthedocs.io .

CHAPTER

THREE

HOWTO 101

In a nutshell – `heudiconv` is given a file tree of DICOMs, and it produces a restructured file tree of NifTI files (conversion handled by dcm2niix) with accompanying metadata files. The input and output structure is as flexible as your data, which is accomplished by using a Python file called a `heuristic` that knows how to read your input structure and decides how to name the resultant files. You can run your conversion automatically (which will produce a `.heudiconv` directory storing the used parameters), or generate the default parameters, edit them to customize file naming, and continue conversion via an additional invocation of *heudiconv*:


DICOMs
dicomdir/
1208200617178_22/
1208200617178_22_8973.dcm
1208200617178_22_8943.dcm
1208200617178_22_2973.dcm
1208200617178_22_8923.dcm
1208200617178_22_4473.dcm
1208200617178_22_8783.dcm
1208200617178_22_7328.dcm
1208200617178_22_9264.dcm
1208200617178_22_9967.dcm
1208200617178_22_3894.dcm
1208200617178_22_3899.dcm
1208200617178_23/
1208200617178_24/
1208200617178_25/
Approach 1: Edit proposed names, then convert.
heudiconv -c none
.heudiconv
heudiconv -c dcm2niix
Approach 2: Automatically use proposed names.
.heudiconv
heudiconv -c dcm2niix
NIfTI (e.g. BIDS)
my_dataset/
participants.tsv
sub-01/
anat/
sub-01_T1w.nii.gz
func/
sub-01_task-rest_bold.nii.gz
sub-01_task-rest_bold.json
dwi/
sub-01_dwi.nii.gz
sub-01_dwi.json
sub-01_dwi.bval
sub-01_dwi.bvec
sub-02/
sub-03/
sub-04/

`heudiconv` comes with existing heuristics which can be used as is, or as examples. For instance, the Heuristic convertall extracts standard metadata from all matching DICOMs. `heudiconv` creates mapping files, `<something>.edit.text` which lets researchers simply establish their own conversion mapping.

In most use-cases of retrospective study data conversion, you would need to create your custom heuristic following the examples and the "Heuristic" section in the documentation. **Note** that ReproIn heuristic is generic and powerful enough to be adopted virtually for *any* study: For prospective studies, you would just need to name your sequences following the ReproIn convention, and for retrospective conversions, you often would be able to create a new versatile heuristic by simply providing remappings into ReproIn as shown in this issue (documentation is coming).

Having decided on a heuristic, you could use the command line:

```
heudiconv -f HEURISTIC-FILE-OR-NAME -o OUTPUT-PATH --files INPUT-PATHs
```

with various additional options (see `heudiconv --help` or "Usage" in documentation) to tune its behavior to convert your data.

For detailed examples and guides, please check out ReproIn conversion invocation examples and the user tutorials in the documentation.

CHAPTER

FOUR

HOW TO CITE

Please use Zenodo record for your specific version of HeuDiConv. We also support gathering all relevant citations via DueCredit.

CHAPTER FIVE

HOW TO CONTRIBUTE

For a detailed into, see our contributing guide.

Our releases are packaged using Intuit auto, with the corresponding workflow including Docker image preparation being found in `.github/workflows/release.yml`.

CHAPTER SIX

3-RD PARTY HEURISTICS

- https://github.com/courtois-neuromod/ds_prep/blob/main/mri/convert/heuristics_unf.py

CHAPTER

SEVEN

SUPPORT

All bugs, concerns and enhancement requests for this software can be submitted here: https://github.com/nipy/heudiconv/issues.

If you have a problem or would like to ask a question about how to use `heudiconv`, please submit a question to NeuroStars.org with a `heudiconv` tag. NeuroStars.org is a platform similar to StackOverflow but dedicated to neuroinformatics.

All previous `heudiconv` questions are available here: http://neurostars.org/tags/heudiconv/

CHAPTER

EIGHT

CONTRIBUTING TO HEUDICONV

8.1 Files organization

- heudiconv/ is the main Python module where major development is happening, with major submodules being:
  - `cli/` - wrappers and argument parsers bringing the HeuDiConv functionality to the command line.
  - `external/` - general compatibility layers for external functions HeuDiConv depends on.
  - `heuristics/` - heuristic evaluators for workflows, pull requests here are particularly welcome.
- docs/ - documentation directory.
- utils/ - helper utilities used during development, testing, and distribution of HeuDiConv.

8.2 How to contribute

The preferred way to contribute to the HeuDiConv code base is to fork the main repository on GitHub.

If you are unsure what that means, here is a set-up workflow you may wish to follow:

0. Fork the project repository on GitHub, by clicking on the “Fork” button near the top of the page — this will create a copy of the repository writeable by your GitHub user.
1. Set up a clone of the repository on your local machine and connect it to both the “official” and your copy of the repository on GitHub:

```
git clone git://github.com/nipy/heudiconv
cd heudiconv
git remote rename origin official
git remote add origin git://github.com/YOUR_GITHUB_USERNAME/heudiconv
```

2. When you wish to start a new contribution, create a new branch:

```
git checkout -b topic_of_your_contribution
```

3. When you are done making the changes you wish to contribute, record them in Git:

```
git add the/paths/to/files/you/modified can/be/more/than/one
git commit
```

3. Push the changes to your copy of the code on GitHub, following which Git will provide you with a link which you can click to initiate a pull request:

```
git push -u origin topic_of_your_contribution
```

(If any of the above seems overwhelming, you can look up the Git documentation on the web.)

8.3 Releases and Changelog

HeuDiConv uses the auto tool to generate the changelog and automatically release the project.

auto is used in the HeuDiConv GitHub actions, which monitors the labels on the pull request. HeuDiConv automation can add entries to the changelog, cut releases, and push new images to dockerhub.

The following pull request labels are respected:

- major: Increment the major version when merged
- minot: Increment the minot version when merged
- patch: Increment the patch version when merged
- skip-release: Preserve the current version when merged
- release: Create a release when this pr is merged
- internal: Changes only affect the internal API
- documentation: Changes only affect the documentation
- tests: Add or improve existing tests
- dependencies: Update one or more dependencies version
- performance: Improve performance of an existing feature

8.4 Development environment

We support Python 3 only (>= 3.7).

Dependencies which you will need are listed in the repository. Note that you will likely have these will already be available on your system if you used a package manager (e.g. Debian's `apt-get`, Gentoo's `emerge`, or simply PIP) to install the software.

Development work might require live access to the copy of HeuDiConv which is being developed. If a system-wide release of HeuDiConv is already installed, or likely to be, it is best to keep development work sandboxed inside a dedicated virtual environment. This is best accomplished via:

```
cd /path/to/your/clone/of/heudiconv
mkdir -p venvs/dev
python -m venv venvs/dev
source venvs/dev/bin/activate
pip install -e .[all]
```

8.5 Documentation

To contribute to the documentation, we recommend building the docs locally prior to submitting a patch.

To build the docs locally:

1. From the root of the heudiconv repository, *pip install -r docs/requirements.txt*
2. From the *docs/* directory, run *make html*

8.6 Additional Hints

It is recommended to check that your contribution complies with the following rules before submitting a pull request:

- All public functions (i.e. functions whose name does not start with an underscore) should have informative docstrings with sample usage presented as doctests when appropriate.
- Docstrings are formatted in NumPy style.
- Lines are no longer than 120 characters.
- All tests still pass:

  ```
  cd /path/to/your/clone/of/heudiconv
  pytest -vvs .
  ```

- New code should be accompanied by new tests.

8.7 Contents

8.7.1 Installation

`Heudiconv` is packaged and available from many different sources.

Local

Released versions of HeuDiConv are available on PyPI and conda. If installing through `PyPI`, eg:

```
pip install heudiconv[all]
```

Manual installation of dcm2niix is required. You can also benefit from an installer/downloader helper `dcm2niix` package on `PyPI`, so you can simply `pip install dcm2niix` if you are installing in user space so subsequently it would be able to download and install dcm2niix binary.

On Debian-based systems, we recommend using NeuroDebian, which provides the heudiconv package.

Containers

Our container image releases are availe on our Docker Hub

If Docker is available on your system, you can pull the latest release:

```
$ docker pull nipy/heudiconv:latest
```

Additionally, HeuDiConv is available through the Docker image at repronim/reproin provided by ReproIn heuristic project, which develops the `reproin` heuristic.

To maintain provenance, it is recommended that you use the `latest` tag only when testing out heudiconv. Otherwise, it is recommended that you use an explicit version and record that information alongside the produced data.

Singularity

If Singularity is available on your system, you can use it to pull and convert our Docker images! For example, to pull and build the latest release, you can run:

```
$ singularity pull docker://nipy/heudiconv:latest
```

Singularity YODA style using ///repronim/containers

ReproNim provides a large collection of Singularity container images of popular neuroimaging tools, e.g. all the BIDS-Apps. This collection also includes the forementioned container images for HeuDiConv and ReproIn in the Singularity image format. This collection is available as a DataLad dataset at ///repronim/containers on datasets.datalad.org and as a GitHub repo. The HeuDiConv and ReproIn container images are named `nipy-heudiconv` and `repronim-reproin`, respectively, in this collection. To use them, you can install the DataLad dataset and then use the `datalad containers-run` command to run. For a more detailed example of using images from this collection while fulfilling the YODA Principles, please check out A typical YODA workflow in the documentation of this collection.

Note: With the `datalad containers-run` command, the images in this collection work on macOS (OSX) as well for `repronim/containers` helpers automagically take care of running the Singularity containers via Docker.

8.7.2 Changes

```
# v1.1.1 (Thu May 02 2024)

####  Bug Fix

- Handle cases where dates/times in DICOM are empty strings, not Nones (e.g. after some␣
↪anonymization) [#756](https://github.com/nipy/heudiconv/pull/756) ([@jennan](https://
↪github.com/jennan) [@yarikoptic](https://github.com/yarikoptic))

#### Authors: 2

- Maxime Rio ([@jennan](https://github.com/jennan))
- Yaroslav Halchenko ([@yarikoptic](https://github.com/yarikoptic))

---

# v1.1.0 (Wed Feb 28 2024)
```

(continues on next page)

(continued from previous page)

```
####  Enhancement

- Add support for a  custom seqinfo to extract from DICOMs any additional metadata␣
↪desired for a heuristic [#581](https://github.com/nipy/heudiconv/pull/581)␣
↪([@yarikoptic](https://github.com/yarikoptic) [@bpinsard](https://github.com/bpinsard))
- codespell: ignore "build" folder which might be on the system [#581](https://github.
↪com/nipy/heudiconv/pull/581) ([@yarikoptic](https://github.com/yarikoptic))

#### Authors: 2

- Basile ([@bpinsard](https://github.com/bpinsard))
- Yaroslav Halchenko ([@yarikoptic](https://github.com/yarikoptic))

---

# v1.0.2 (Mon Feb 26 2024)

####  Bug Fix

- properly remove GE multiecho bvals/bvecs [#728](https://github.com/nipy/heudiconv/pull/
↪728) ([@bpinsard](https://github.com/bpinsard))
- datalad sensitive marking fixes [#739](https://github.com/nipy/heudiconv/pull/739)␣
↪([@bpinsard](https://github.com/bpinsard))
- Reject "Missing images" in sensor-dicoms [#735](https://github.com/nipy/heudiconv/pull/
↪735) ([@chaselgrove](https://github.com/chaselgrove))

####  Pushed to `master`

- Adding workflow figure ([@TheChymera](https://github.com/TheChymera))
- Added figures to master branch ([@TheChymera](https://github.com/TheChymera))

####  Internal

- auto 11.0.5 is needed to avoid hitting some "Error: fatal: ... not an integer" bug [
↪#746](https://github.com/nipy/heudiconv/pull/746) ([@yarikoptic](https://github.com/
↪yarikoptic))
- Fix - auto is in ~/, not in the PATH [#745](https://github.com/nipy/heudiconv/pull/
↪745) ([@yarikoptic](https://github.com/yarikoptic))
- Make it possible to review auto version -v output during release + adjust that␣
↪workflow step description [#743](https://github.com/nipy/heudiconv/pull/743)␣
↪([@yarikoptic](https://github.com/yarikoptic))
- [gh-actions](deps): Bump codecov/codecov-action from 3 to 4 [#736](https://github.com/
↪nipy/heudiconv/pull/736) ([@dependabot[bot]](https://github.com/dependabot[bot])␣
↪[@yarikoptic](https://github.com/yarikoptic))
- [gh-actions](deps): Bump actions/setup-python from 4 to 5 [#723](https://github.com/
↪nipy/heudiconv/pull/723) ([@dependabot[bot]](https://github.com/dependabot[bot]))

####  Documentation

- Adjust wording on heuristics page -- do not claim creating some skeleton [#741](https:/
↪/github.com/nipy/heudiconv/pull/741) ([@yarikoptic](https://github.com/yarikoptic))
```

(continues on next page)

(continued from previous page)

```
- Document how to release and add changelog entries [#737](https://github.com/nipy/
↪heudiconv/pull/737) ([@asmacdo](https://github.com/asmacdo) [@yarikoptic](https://
↪github.com/yarikoptic))
- Add dianne tutorials [#734](https://github.com/nipy/heudiconv/pull/734)␣
↪([@asmacdo](https://github.com/asmacdo) [@yarikoptic](https://github.com/yarikoptic))
- Add documentation building instructions [#730](https://github.com/nipy/heudiconv/pull/
↪730) ([@asmacdo](https://github.com/asmacdo))
- Allowing RTD to access images under the same path as README [#734](https://github.com/
↪nipy/heudiconv/pull/734) ([@TheChymera](https://github.com/TheChymera))
- Using environment figure in about section [#730](https://github.com/nipy/heudiconv/
↪pull/730) ([@TheChymera](https://github.com/TheChymera))
- Make README more concrete [#724](https://github.com/nipy/heudiconv/pull/724)␣
↪([@asmacdo](https://github.com/asmacdo))

#### Authors: 6

- [@chaselgrove](https://github.com/chaselgrove)
- [@dependabot[bot]](https://github.com/dependabot[bot])
- Austin Macdonald ([@asmacdo](https://github.com/asmacdo))
- Basile ([@bpinsard](https://github.com/bpinsard))
- Horea Christian ([@TheChymera](https://github.com/TheChymera))
- Yaroslav Halchenko ([@yarikoptic](https://github.com/yarikoptic))

---

# v1.0.1 (Fri Dec 08 2023)

####  Bug Fix

- Drop Python 3.7 support [#722](https://github.com/nipy/heudiconv/pull/722)␣
↪([@yarikoptic](https://github.com/yarikoptic))
- ReproIn: give an informative assertion message when multiple values are found [
↪#718](https://github.com/nipy/heudiconv/pull/718) ([@yarikoptic](https://github.com/
↪yarikoptic))
- Convert assertion into a warning that we would not use dicom dir template option [
↪#709](https://github.com/nipy/heudiconv/pull/709) ([@yarikoptic](https://github.com/
↪yarikoptic))
- Do not demand --files for all commands, even those which do not care about it (like␣
↪populate-intended-for) [#708](https://github.com/nipy/heudiconv/pull/708)␣
↪([@yarikoptic](https://github.com/yarikoptic))

####  Pushed to `master`

- Add script to sensor dicoms -- for the error where dcm2niix might or might not fail␣
↪but issues an Error ([@yarikoptic](https://github.com/yarikoptic))

####  Internal

- Ran pre-commit on everything, black decided to adjust some formatting [#721](https://
↪github.com/nipy/heudiconv/pull/721) ([@yarikoptic](https://github.com/yarikoptic))
- Make sensor-dicoms use gnu-getopt if present (on OSX) [#721](https://github.com/nipy/
↪heudiconv/pull/721) ([@yarikoptic](https://github.com/yarikoptic))
```

(continues on next page)

(continued from previous page)

```
#### Authors: 1

- Yaroslav Halchenko ([@yarikoptic](https://github.com/yarikoptic))

---

# v1.0.0 (Wed Sep 20 2023)

####  Breaking Change

- [gh-actions](deps): Bump actions/checkout from 3 to 4 [#703](https://github.com/nipy/
↪heudiconv/pull/703) ([@dependabot[bot]](https://github.com/dependabot[bot]))

####  Enhancement

- Fix inconsistent behavior of existing session when using -d compared to --files␣
↪option: raise an AssertionError instead of just a warning [#682](https://github.com/
↪nipy/heudiconv/pull/682) ([@neurorepro](https://github.com/neurorepro))

####  Bug Fix

- Various tiny enhancements flake etc demanded [#702](https://github.com/nipy/heudiconv/
↪pull/702) ([@yarikoptic](https://github.com/yarikoptic))
- Boost claimed BIDS version to 1.8.0 from 1.4.1 [#699](https://github.com/nipy/
↪heudiconv/pull/699) ([@yarikoptic](https://github.com/yarikoptic))
- Point to Courtois-neuromod heuristic [#702](https://github.com/nipy/heudiconv/pull/
↪702) ([@yarikoptic](https://github.com/yarikoptic))

####  Internal

- Add codespell to lint tox env [#706](https://github.com/nipy/heudiconv/pull/706)␣
↪([@yarikoptic](https://github.com/yarikoptic))
- test-compare-two-versions.sh: also ignore differences in HeudiconvVersion field in␣
↪jsons since we have it there now [#685](https://github.com/nipy/heudiconv/pull/685)␣
↪([@yarikoptic](https://github.com/yarikoptic))

####  Documentation

- Add description of placeholders which could be used in the produced templates [
↪#681](https://github.com/nipy/heudiconv/pull/681) ([@yarikoptic](https://github.com/
↪yarikoptic))

#### Authors: 3

- [@dependabot[bot]](https://github.com/dependabot[bot])
- Michael ([@neurorepro](https://github.com/neurorepro))
- Yaroslav Halchenko ([@yarikoptic](https://github.com/yarikoptic))

---

# v0.13.1 (Tue May 23 2023)
```

(continues on next page)

(continued from previous page)

```
####  Bug Fix

- Make .subsecond optional in BIDS/DICOM datetime entries [#675](https://github.com/nipy/
↪heudiconv/pull/675) ([@yarikoptic](https://github.com/yarikoptic))

####  Internal

- [gh-actions](deps): Bump codespell-project/actions-codespell from 1 to 2 [#677](https:/
↪/github.com/nipy/heudiconv/pull/677) ([@dependabot[bot]](https://github.com/
↪dependabot[bot]))
- Replace (no longer used) Travis badge with GitHub action one (for test) [#677](https://
↪github.com/nipy/heudiconv/pull/677) ([@yarikoptic](https://github.com/yarikoptic))

#### Authors: 2

- [@dependabot[bot]](https://github.com/dependabot[bot])
- Yaroslav Halchenko ([@yarikoptic](https://github.com/yarikoptic))

---

# v0.13.0 (Mon May 08 2023)

####  Enhancement

- Add type annotations [#656](https://github.com/nipy/heudiconv/pull/656)␣
↪([@jwodder](https://github.com/jwodder) [@yarikoptic](https://github.com/yarikoptic))
- ENH: Support extracting DICOMs from ZIP files (and possibly other archives) by␣
↪switching to use shutil.unpack_archive instead of tarfile module functionality [
↪#471](https://github.com/nipy/heudiconv/pull/471) ([@HippocampusGirl](https://github.
↪com/HippocampusGirl) [@psadil](https://github.com/psadil))
- Allow filling of acq_time when AcquisitionDate AcquisitionTime missing [#614](https://
↪github.com/nipy/heudiconv/pull/614) ([@psadil](https://github.com/psadil))

####  Bug Fix

- BF(?): make _setter images be taken as scouts - only DICOMs are saved [#570](https://
↪github.com/nipy/heudiconv/pull/570) ([@yarikoptic](https://github.com/yarikoptic))
- Adjust .mailmap to account for mapping various folks with multiple emails so that git␣
↪shortlog -sn -e provides entries without duplicates [#570](https://github.com/nipy/
↪heudiconv/pull/570) ([@yarikoptic](https://github.com/yarikoptic))
- Make an `embed_dicom_and_nifti_metadata()` annotation work on Python 3.7 [#673](https:/
↪/github.com/nipy/heudiconv/pull/673) ([@jwodder](https://github.com/jwodder))
- Merge branch 'feature_dicom_compresslevel' [#673](https://github.com/nipy/heudiconv/
↪pull/673) ([@yarikoptic](https://github.com/yarikoptic))
- Update heudiconv/dicoms.py [#669](https://github.com/nipy/heudiconv/pull/669)␣
↪([@octomike](https://github.com/octomike))
- Don't call `logging.basicConfig()` in `__init__.py` [#659](https://github.com/nipy/
↪heudiconv/pull/659) ([@jwodder](https://github.com/jwodder))

####  Pushed to `master`
```

(continues on next page)

(continued from previous page)

```
- Mailmapping more contributors ([@yarikoptic](https://github.com/yarikoptic))
- Adjust comment and remove trailing space flipping linting ([@yarikoptic](https://
↪github.com/yarikoptic))

####  Internal

- Declare `custom_grouping` return type instead of casting [#671](https://github.com/
↪nipy/heudiconv/pull/671) ([@jwodder](https://github.com/jwodder))
- Use `pydicom.dcmread()` instead of `pydicom.read_file()` [#668](https://github.com/
↪nipy/heudiconv/pull/668) ([@jwodder](https://github.com/jwodder))
- Add `sample_nifti.json` to `.gitignore` [#663](https://github.com/nipy/heudiconv/pull/
↪663) ([@jwodder](https://github.com/jwodder))
- Write command arguments as lists of strings instead of splitting strings on whitespace␣
↪[#664](https://github.com/nipy/heudiconv/pull/664) ([@jwodder](https://github.com/
↪jwodder))
- Add & apply pre-commit and lint job [#658](https://github.com/nipy/heudiconv/pull/658)␣
↪([@jwodder](https://github.com/jwodder))
- Fix some strings with \ (make them raw or double-\), improve pytest config: move to␣
↪tox.ini, make unknown warnings into errors [#660](https://github.com/nipy/heudiconv/
↪pull/660) ([@jwodder](https://github.com/jwodder))
- Replace py.path with pathlib [#654](https://github.com/nipy/heudiconv/pull/654)␣
↪([@jwodder](https://github.com/jwodder))

####  Tests

- Make `test_private_csa_header` test write to temp dir [#666](https://github.com/nipy/
↪heudiconv/pull/666) ([@jwodder](https://github.com/jwodder))

####  Dependency Updates

- Replace third-party `mock` library with stdlib's `unittest.mock` [#661](https://github.
↪com/nipy/heudiconv/pull/661) ([@jwodder](https://github.com/jwodder))
- Remove kludgy support for older versions of pydicom and dcmstack [#662](https://github.
↪com/nipy/heudiconv/pull/662) ([@jwodder](https://github.com/jwodder))
- Remove use of `six` [#655](https://github.com/nipy/heudiconv/pull/655)␣
↪([@jwodder](https://github.com/jwodder))

#### Authors: 5

- John T. Wodder II ([@jwodder](https://github.com/jwodder))
- Lea Waller ([@HippocampusGirl](https://github.com/HippocampusGirl))
- Michael ([@octomike](https://github.com/octomike))
- Patrick Sadil ([@psadil](https://github.com/psadil))
- Yaroslav Halchenko ([@yarikoptic](https://github.com/yarikoptic))

---

# v0.12.2 (Tue Mar 14 2023)

####  Internal

- [DATALAD RUNCMD] produce updated dockerfile [#652](https://github.com/nipy/heudiconv/
```

(continues on next page)

(continued from previous page)

```
↪pull/652) ([@yarikoptic](https://github.com/yarikoptic))

#### Authors: 1

- Yaroslav Halchenko ([@yarikoptic](https://github.com/yarikoptic))

---

# v0.12.1 (Tue Mar 14 2023)

####  Bug Fix

- Re-add explicit instructions to install dcm2niix "manually" and remove it from install_
↪requires [#651](https://github.com/nipy/heudiconv/pull/651) ([@yarikoptic](https://
↪github.com/yarikoptic))

####  Documentation

- Contributing guide. [#641](https://github.com/nipy/heudiconv/pull/641)␣
↪([@TheChymera](https://github.com/TheChymera))
- Reword and correct punctuation on installation.rst [#643](https://github.com/nipy/
↪heudiconv/pull/643) ([@yarikoptic](https://github.com/yarikoptic)␣
↪[@candleindark](https://github.com/candleindark))

#### Authors: 3

- Horea Christian ([@TheChymera](https://github.com/TheChymera))
- Isaac To ([@candleindark](https://github.com/candleindark))
- Yaroslav Halchenko ([@yarikoptic](https://github.com/yarikoptic))

---

# v0.12.0 (Tue Feb 21 2023)

####  Enhancement

- strip non-alphanumeric from session ids too [#647](https://github.com/nipy/heudiconv/
↪pull/647) ([@keithcallenberg](https://github.com/keithcallenberg) [@yarikoptic](https:/
↪/github.com/yarikoptic))

####  Bug Fix

- Docker images: tag also as "unstable", strip "v" prefix, and avoid building in non-
↪release workflow for releases. [#642](https://github.com/nipy/heudiconv/pull/642)␣
↪([@yarikoptic](https://github.com/yarikoptic))
- add install link to README [#640](https://github.com/nipy/heudiconv/pull/640)␣
↪([@asmacdo](https://github.com/asmacdo))
- Setting git author and email in test environment [#631](https://github.com/nipy/
↪heudiconv/pull/631) ([@TheChymera](https://github.com/TheChymera))
- Duecredit dcm2niix [#622](https://github.com/nipy/heudiconv/pull/622)␣
↪([@yarikoptic](https://github.com/yarikoptic))
- Do not issue warning if cannot parse _task entity [#621](https://github.com/nipy/
```

(continues on next page)

(continued from previous page)

```
↪heudiconv/pull/621) ([@yarikoptic](https://github.com/yarikoptic))
- Provide codespell config and workflow [#619](https://github.com/nipy/heudiconv/pull/
↪619) ([@yarikoptic](https://github.com/yarikoptic))
- BF: Use .get in group_dicoms_into_seqinfos to not puke if SeriesDescription is missing␣
↪[#622](https://github.com/nipy/heudiconv/pull/622) ([@yarikoptic](https://github.com/
↪yarikoptic))
- DOC: do provide short version for sphinx [#609](https://github.com/nipy/heudiconv/pull/
↪609) ([@yarikoptic](https://github.com/yarikoptic))

#### Pushed to `master`

- DOC: add clarification on where docs/requirements.txt should be "installed" from␣
↪([@yarikoptic](https://github.com/yarikoptic))
- fix minor typo ([@yarikoptic](https://github.com/yarikoptic))
- DOC: fixed the comment. Original was copy/pasted from DataLad ([@yarikoptic](https://
↪github.com/yarikoptic))

#### Internal

- dcm2niix explicitly noted as a (PyPI) dependency and removed from being installed via␣
↪apt-get etc [#628](https://github.com/nipy/heudiconv/pull/628) ([@TheChymera](https://
↪github.com/TheChymera) [@yarikoptic](https://github.com/yarikoptic))

#### Documentation

- Reword number of intended ideas in README.rst [#639](https://github.com/nipy/heudiconv/
↪pull/639) ([@candleindark](https://github.com/candleindark))
- Add a bash anon-cmd to be used to incrementally anonymize sids [#615](https://github.
↪com/nipy/heudiconv/pull/615) ([@yarikoptic](https://github.com/yarikoptic))
- Reword and correct punctuation on usage.rst [#644](https://github.com/nipy/heudiconv/
↪pull/644) ([@candleindark](https://github.com/candleindark) [@yarikoptic](https://
↪github.com/yarikoptic))
- Clarify the infotodict function [#645](https://github.com/nipy/heudiconv/pull/645)␣
↪([@yarikoptic](https://github.com/yarikoptic))
- Added distribution badges [#632](https://github.com/nipy/heudiconv/pull/632)␣
↪([@TheChymera](https://github.com/TheChymera))
- Capitalize sentences and end sentences with period [#629](https://github.com/nipy/
↪heudiconv/pull/629) ([@candleindark](https://github.com/candleindark))
- Tune up .mailmap to harmonize Pablo, Dae and Mathias [#629](https://github.com/nipy/
↪heudiconv/pull/629) ([@yarikoptic](https://github.com/yarikoptic))
- Add HOWTO 101 section, with references to ReproIn to README.rst [#623](https://github.
↪com/nipy/heudiconv/pull/623) ([@yarikoptic](https://github.com/yarikoptic))
- minor fix -- Fix use of code:: directive [#623](https://github.com/nipy/heudiconv/pull/
↪623) ([@yarikoptic](https://github.com/yarikoptic))

#### Tests

- Add 3.11 to be tested etc [#635](https://github.com/nipy/heudiconv/pull/635)␣
↪([@yarikoptic](https://github.com/yarikoptic))

#### Authors: 5
```

(continues on next page)

(continued from previous page)

```
- Austin Macdonald ([@asmacdo](https://github.com/asmacdo))
- Horea Christian ([@TheChymera](https://github.com/TheChymera))
- Isaac To ([@candleindark](https://github.com/candleindark))
- Keith Callenberg ([@keithcallenberg](https://github.com/keithcallenberg))
- Yaroslav Halchenko ([@yarikoptic](https://github.com/yarikoptic))

---

# v0.11.6 (Thu Nov 03 2022)

####  Internal

- Delete .dockerignore [#607](https://github.com/nipy/heudiconv/pull/607)␣
↪([@jwodder](https://github.com/jwodder))

####  Documentation

- DOC: Various fixes to make RTD build the docs again [#608](https://github.com/nipy/
↪heudiconv/pull/608) ([@yarikoptic](https://github.com/yarikoptic))

#### Authors: 2

- John T. Wodder II ([@jwodder](https://github.com/jwodder))
- Yaroslav Halchenko ([@yarikoptic](https://github.com/yarikoptic))

---

# v0.11.5 (Thu Nov 03 2022)

####  Bug Fix

- Fix certificate issue as indicated in #595 [#597](https://github.com/nipy/heudiconv/
↪pull/597) ([@neurorepro](https://github.com/neurorepro))
- BF docker build: use python3.9 (not 3.7 which gets upgraded to 3.9) and newer dcm2niix␣
↪[#596](https://github.com/nipy/heudiconv/pull/596) ([@yarikoptic](https://github.com/
↪yarikoptic))
- Fixup miniconda spec for neurodocker so it produces dockerfile now [#596](https://
↪github.com/nipy/heudiconv/pull/596) ([@yarikoptic](https://github.com/yarikoptic))

####  Internal

- Update GitHub Actions action versions [#601](https://github.com/nipy/heudiconv/pull/
↪601) ([@jwodder](https://github.com/jwodder))
- Set action step outputs via $GITHUB_OUTPUT [#600](https://github.com/nipy/heudiconv/
↪pull/600) ([@jwodder](https://github.com/jwodder))

####  Documentation

- DOC: codespell fix a few typos in code comments [#605](https://github.com/nipy/
↪heudiconv/pull/605) ([@yarikoptic](https://github.com/yarikoptic))

#### Authors: 3
```

(continues on next page)

(continued from previous page)

```
- John T. Wodder II ([@jwodder](https://github.com/jwodder))
- Michael ([@neurorepro](https://github.com/neurorepro))
- Yaroslav Halchenko ([@yarikoptic](https://github.com/yarikoptic))

---

# v0.11.4 (Thu Sep 29 2022)

####  Bug Fix

- install dcmstack straight from github until it is released [#593](https://github.com/
↪nipy/heudiconv/pull/593) ([@yarikoptic](https://github.com/yarikoptic))
- DOC: provide rudimentary How to contribute section in README.rst ([@yarikoptic](https:/
↪/github.com/yarikoptic))

####  Pushed to `master`

- Check out a full clone when testing ([@jwodder](https://github.com/jwodder))
- Convert Travis workflow to GitHub Actions ([@jwodder](https://github.com/jwodder))
- BF(docker): replace old -tipsy with -y -all for conda clean as neurodocker does now␣
↪([@yarikoptic](https://github.com/yarikoptic))
- adjusted script for neurodocker although it does not work ([@yarikoptic](https://
↪github.com/yarikoptic))

####  Internal

- 0.9 of dcmstack was released, no need for github version [#594](https://github.com/
↪nipy/heudiconv/pull/594) ([@yarikoptic](https://github.com/yarikoptic))
- Minor face-lifts to ReproIn: align doc and code better to BIDS terms, address␣
↪deprecation warnings etc [#569](https://github.com/nipy/heudiconv/pull/569)␣
↪([@yarikoptic](https://github.com/yarikoptic))

#### Authors: 2

- John T. Wodder II ([@jwodder](https://github.com/jwodder))
- Yaroslav Halchenko ([@yarikoptic](https://github.com/yarikoptic))

---

# v0.11.3 (Thu May 12 2022)

####  Internal

- BF: add recently tests data missing from distribution [#567](https://github.com/nipy/
↪heudiconv/pull/567) ([@yarikoptic](https://github.com/yarikoptic))

#### Authors: 1

- Yaroslav Halchenko ([@yarikoptic](https://github.com/yarikoptic))

---
```

(continues on next page)

(continued from previous page)

```
# v0.11.2 (Thu May 12 2022)

####  Internal

- Make versioningit write version to file; make setup.py read version as fallback [
↪#566](https://github.com/nipy/heudiconv/pull/566) ([@jwodder](https://github.com/
↪jwodder))
- BF: add fetch-depth: 0 to get all tags into docker builds of master [#566](https://
↪github.com/nipy/heudiconv/pull/566) ([@yarikoptic](https://github.com/yarikoptic))

#### Authors: 2

- John T. Wodder II ([@jwodder](https://github.com/jwodder))
- Yaroslav Halchenko ([@yarikoptic](https://github.com/yarikoptic))

---

# v0.11.1 (Tue May 10 2022)

####  Internal

- Remove .git/ from .dockerignore so that versioning works while building docker image [
↪#564](https://github.com/nipy/heudiconv/pull/564) ([@yarikoptic](https://github.com/
↪yarikoptic))

#### Authors: 1

- Yaroslav Halchenko ([@yarikoptic](https://github.com/yarikoptic))

---

# v0.11.0 (Tue May 10 2022)

####  Enhancement

- RF: drop Python 3.6 (EOLed), fix dcm2niix version in neurodocker script [#555](https://
↪github.com/nipy/heudiconv/pull/555) ([@yarikoptic](https://github.com/yarikoptic))
- ENH: Adds populate_intended_for for fmaps [#482](https://github.com/nipy/heudiconv/
↪pull/482) ([@pvelasco](https://github.com/pvelasco) [@yarikoptic](https://github.com/
↪yarikoptic) bids@dbic.dartmouth.edu [@neurorepro](https://github.com/neurorepro))

####  Bug Fix

- bids_ME heuristic: add test for the dataset that raised #541, add support for MEGRE [
↪#547](https://github.com/nipy/heudiconv/pull/547) ([@pvelasco](https://github.com/
↪pvelasco) [@yarikoptic](https://github.com/yarikoptic))
- reproin heuristic: specify POPULATE_INTENDED_FOR_OPTS [#546](https://github.com/nipy/
↪heudiconv/pull/546) ([@yarikoptic](https://github.com/yarikoptic))
- FIX: Convert sets to lists for filename updaters [#461](https://github.com/nipy/
↪heudiconv/pull/461) ([@tsalo](https://github.com/tsalo))
- Added new infofilestyle compatible with BIDS [#12](https://github.com/nipy/heudiconv/
```

(continues on next page)

(continued from previous page)

```
↪pull/12) ([@chrisgorgo](https://github.com/chrisgorgo))
- try a simple fix for wrongly ordered files in tar file [#535](https://github.com/nipy/
↪heudiconv/pull/535) ([@bpinsard](https://github.com/bpinsard))
- BF: Fix the order of the 'echo' entity in the filename [#542](https://github.com/nipy/
↪heudiconv/pull/542) ([@pvelasco](https://github.com/pvelasco))
- ENH: add HeudiconvVersion to sidecar .json files [#529](https://github.com/nipy/
↪heudiconv/pull/529) ([@yarikoptic](https://github.com/yarikoptic))
- BF (TST): make anonymize_script actually output anything and map deterministically [
↪#511](https://github.com/nipy/heudiconv/pull/511) ([@yarikoptic](https://github.com/
↪yarikoptic))
- Rename DICOMCONVERT_README.md to README.md [#4](https://github.com/nipy/heudiconv/pull/
↪4) ([@satra](https://github.com/satra))

####  Pushed to `master`

- Dockerfile - use bullseye for the base and fresh dcm2niix ([@yarikoptic](https://
↪github.com/yarikoptic))

####  Internal

- Run codespell on some obvious typos [#563](https://github.com/nipy/heudiconv/pull/563)␣
↪([@yarikoptic](https://github.com/yarikoptic))
- Set up auto [#558](https://github.com/nipy/heudiconv/pull/558) ([@jwodder](https://
↪github.com/jwodder) [@yarikoptic](https://github.com/yarikoptic))

####  Tests

- BF(TST): use caplog to control logging level, use python3 in shebang [#553](https://
↪github.com/nipy/heudiconv/pull/553) ([@yarikoptic](https://github.com/yarikoptic))
- BF(TST): use caplog instead of capfd for testing if we log a warning [#534](https://
↪github.com/nipy/heudiconv/pull/534) ([@yarikoptic](https://github.com/yarikoptic))
- Travis - Use bionic for the base [#533](https://github.com/nipy/heudiconv/pull/533)␣
↪([@yarikoptic](https://github.com/yarikoptic))

#### Authors: 9

- Basile ([@bpinsard](https://github.com/bpinsard))
- Chris Gorgolewski ([@chrisgorgo](https://github.com/chrisgorgo))
- DBIC BIDS Team (bids@dbic.dartmouth.edu)
- John T. Wodder II ([@jwodder](https://github.com/jwodder))
- Michael ([@neurorepro](https://github.com/neurorepro))
- Pablo Velasco ([@pvelasco](https://github.com/pvelasco))
- Satrajit Ghosh ([@satra](https://github.com/satra))
- Taylor Salo ([@tsalo](https://github.com/tsalo))
- Yaroslav Halchenko ([@yarikoptic](https://github.com/yarikoptic))

---

# [0.10.0] - 2021-09-16

Various improvements and compatibility/support (dcm2niix, datalad) changes.
```

(continues on next page)

(continued from previous page)

```
## Added

- Add "AcquisitionTime" to the seqinfo ([#487][])
- Add support for saving the Phoenix Report in the sourcedata folder ([#489][])

## Changed

- Python 3.5 EOLed, supported (tested) versions now: 3.6 - 3.9
- In reprorin heuristic, allow for having multiple accessions since now there is
  `-g all` grouping ([#508][])
- For BIDS, produce a singular `scans.json` at the top level, and not one per
  sub/ses (generates too many identical files) ([#507][])


## Fixed

- Compatibility with DataLad 0.15.0. Minimal version is 0.13.0 now.
- Try to open top level BIDS .json files a number of times for adjustment,
  so in the case of competition across parallel processes, they just end up
  with the last one "winning over" ([#523][])
- Don't fail if etelemetry.get_project returns None ([#501][])
- Consistently use `n/a` for age/sex, also handle ?M for months ([#500][])
- To avoid crashing on unrelated derivatives files etc, make `find_files` to
  take list of topdirs (excluding `derivatives/` etc),
  and look for _bold only under sub-* directories ([#496][])
- Ensure bvec/bval files are only created for dwi output ([#491][])

## Removed

- In reproin heuristic, old hardcoded sequence renamings and filters ([#508][])


# [0.9.0] - 2020-12-23

Various improvements and compatibility/support (dcm2niix, datalad,
duecredit) changes.  Major change is placement of output files to the
target output directory during conversion.

## Added

- #454 zenodo referencing in README.rst and support for ducredit for
  heudiconv and reproin heuristic
- #445 more tutorial references in README.md

## Changed

- [#485][] placed files during conversion right away into the target
  directory (with a `_heudiconv???` suffix, renamed into ultimate target
  name later on), which avoids hitting file size limits of /tmp ([#481][]) and
  helped to avoid a regression in dcm2nixx 1.0.20201102
- [#477][] replaced `rec-<magnitude|phase>` with `part-<mag|phase>` now
  hat BIDSsupports the part entity
```

(continues on next page)

(continued from previous page)

```
- [#473][] made default for CogAtlasID to be a TODO URL
- [#459][] made AcquisitionTime used for acq_time scans file field
- [#451][] retained sub-second resolution in scans files
- [#442][] refactored code so there is now heudiconv.main.workflow for
  more convenient use as a Python module

## Fixed

- minimal version of nipype set to 1.2.3 to guarantee correct handling
  of DWI files ([#480][])
- `heudiconvDCM*` temporary directories are removed now ([#462][])
- compatibility with DataLad 0.13 ([#464][])

## Removed

- #443 pathlib as a dependency (we are Python3 only now)


# [0.8.0] - 2020-04-15

## Enhancements

- Centralized saving of .json files.  Indentation of some files could
  change now from previous versions where it could have used `3`
  spaces. Now indentation should be consistently `2` for .json files
  we produce/modify ([#436][]) (note: dcm2niix uses tabs for indentation)
- ReproIn heuristic: support SBRef and phase data ([#387][])
- Set the "TaskName" field in .json sidecar files for multi-echo data
  ([#420][])
- Provide an informative exception if command needs heuristic to be
  specified ([#437][])

## Refactored

- `embed_nifti` was refactored into `embed_dicom_and_nifti_metadata`
  which would no longer create `.nii` file if it does not exist
  already ([#432][])

## Fixed

- Skip datalad-based tests if no datalad available ([#430][])
- Search heuristic file path first so we do not pick up a python
  module if name conflicts ([#434][])

# [0.7.0] - 2020-03-20

## Removed

- Python 2 support/testing

## Enhancement

```

(continues on next page)

(continued from previous page)

```
- `-g` option obtained two new modes: `all` and `custom`. In case of `all`,
  all provided DICOMs will be treated as coming from a single scanning session.
  `custom` instructs to use `.grouping` value (could be a DICOM attribute or
  a callable)provided by the heuristic ([#359][]).
- Stop before reading pixels data while gathering metadata from DICOMs ([#404][])
- reproin heuristic:
  - In addition to original "md5sum of the study_description" `protocols2fix`
    could now have (and applied after md5sum matching ones)
    1). a regular expression searched in study_description,
    2). an empty string as "catch all".
    This features could be used to easily provide remapping into reproin
    naming (documentation is to come to http://github.com/ReproNim/reproin)
    ([#425][])

## Fixed

- Use nan, not None for absent echo value in sorting
- reproin heuristic: case seqinfos into a list to be able to modify from
  overloaded heuristic ([#419][])
- No spurious errors from the logger upon a warning about `etelemetry`
  absence ([#407][])

# [0.6.0] - 2019-12-16

This is largely a bug fix.  Metadata and order of `_key-value` fields in BIDS
could change from the result of converting using previous versions, thus minor
version boost.
14 people contributed to this release -- thanks
[everyone](https://github.com/nipy/heudiconv/graphs/contributors)!

## Enhancement

- Use [etelemetry](https://pypi.org/project/etelemetry) to inform about most
  recent available version of heudiconv. Please set `NO_ET` environment variable
  if you want to disable it ([#369][])
- BIDS:
  - `--bids` flag became an option. It can (optionally) accept `notop` value
    to avoid creation of top level files (`CHANGES`, `dataset_description.json`,
    etc) as a workaround during parallel execution to avoid race conditions etc.
    ([#344][])
  - Generate basic `.json` files with descriptions of the fields for
    `participants.tsv` and `_scans.tsv` files ([#376][])
  - Use `filelock` while writing top level files. Use
    `HEUDICONV_FILELOCK_TIMEOUT` environment to change the default timeout value
    ([#348][])
  - `_PDT2` was added as a suffix for multi-echo (really "multi-modal")
    sequences ([#345][])
- Calls to `dcm2niix` would include full output path to make it easier to
  discern in the logs what file it is working on ([#351][])
- With recent [datalad]() (>= 0.10), created DataLad dataset will use
  `--fake-dates` functionality of DataLad to not leak data conversion dates,
  which might be close to actual data acquisition/patient visit ([#352][])
```

(continues on next page)

(continued from previous page)

```
- Support multi-echo EPI `_phase` data ([#373][] fixes [#368][])
- Log location of a bad .json file to ease troubleshooting ([#379][])
- Add basic pypi classifiers for the package ([#380][])

## Fixed
- Sorting `_scans.tsv` files lacking valid dates field should not cause a crash
  ([#337][])
- Multi-echo files detection based number of echos ([#339][])
- BIDS
  - Use `EchoTimes` from the associated multi-echo files if `EchoNumber` tag is
    missing ([#366][] fixes [#347][])
  - Tolerate empty ContentTime and/or ContentDate in DICOMs ([#372][]) and place
    "n/a" if value is missing ([#390][])
  - Do not crash and store original .json file is "JSON pretification" fails
    ([#342][])
- ReproIn heuristic
  - tolerate WIP prefix on Philips scanners ([#343][])
  - allow for use of `(...)` instead of `{...}` since `{}` are not allowed
    ([#343][])
  - Support pipolar fieldmaps by providing them with `_epi` not `_magnitude`.
    "Loose" BIDS `_key-value` pairs might come now after `_dir-` even if they
    came first before ([#358][] fixes [#357][])
- All heuristics saved under `.heudiconv/` under `heuristic.py` name, to avoid
  discrepancy during reconversion ([#354][] fixes [#353][])
- Do not crash (with TypeError) while trying to sort absent file list ([#360][])
- heudiconv requires nipype >= 1.0.0 ([#364][]) and blacklists `1.2.[12]` ([#375][])

# [0.5.4] - 2019-04-29

This release includes fixes to BIDS multi-echo conversions, the
re-implementation of queuing support (currently just SLURM), as well as
some bugfixes.

Starting today, we will (finally) push versioned releases to DockerHub.
Finally, to more accurately reflect on-going development, the `latest`
tag has been renamed to `unstable`.

## Added
- Readthedocs documentation ([#327][])

## Changed
- Update Docker dcm2niix to v.1.0.20190410 ([#334][])
- Allow usage of `--files` with basic heuristics. This requires
  use of `--subject` flag, and is limited to one subject. ([#293][])

## Deprecated

## Fixed
- Improve support for multiple `--queue-args` ([#328][])
- Fixed an issue where generated BIDS sidecar files were missing additional
  information - treating all conversions as if the `--minmeta` flag was
  used ([#306][])
```

(continues on next page)

(continued from previous page)

```
- Re-enable SLURM queuing support ([#304][])
- BIDS multi-echo support for EPI + T1 images ([#293][])
- Correctly handle the case when `outtype` of heuristic has "dicom"
  before '.nii.gz'. Previously would have lead to absent additional metadata
  extraction etc ([#310][])

## Removed
- `--sbargs` argument was renamed to `--queue-args` ([#304][])

## Security


# [0.5.3] - 2019-01-12

Minor hot bugfix release

## Fixed
- Do not shorten spaces in the dates while pretty printing .json

# [0.5.2] - 2019-01-04

A variety of bugfixes

## Changed
- Reproin heuristic: `__dup` indices would now be assigned incrementally
  individually per each sequence, so there is a chance to properly treat
  associate for multi-file (e.g. `fmap`) sequences
- Reproin heuristic: also split StudyDescription by space not only by ^
- `tests/` moved under `heudiconv/tests` to ease maintenance and facilitate
  testing of an installed heudiconv
- Protocol name will also be accessed from private Siemens
  csa.tProtocolName header field if not present in public one
- nipype>=0.12.0 is required now

## Fixed
- Multiple files produced by dcm2niix are first sorted to guarantee
  correct order e.g. of magnitude files in fieldmaps, which otherwise
  resulted in incorrect according to BIDS ordering of them
- Aggregated top level .json files now would contain only the fields
  with the same values from all scanned files. In prior versions,
  those files were not regenerated after an initial conversion
- Unicode handling in anonimization scripts

# [0.5.1] - 2018-07-05
Bugfix release

## Added
- Video tutorial / updated slides
- Helper to set metadata restrictions correctly
- Usage is now shown when run without arguments
- New fields to Seqinfo
  - series_uid
```

(continues on next page)

(continued from previous page)

```
- Reproin heuristic support for xnat
## Changed
- Dockerfile updated to use `dcm2niix v1.0.20180622`
- Conversion table will be regenerated if heurisic has changed
- Do not touch existing BIDS files
  - events.tsv
  - task JSON
## Fixed
- Python 2.7.8 and older installation
- Support for updated packages
  - `Datalad` 0.10
  - `pydicom` 1.0.2
- Later versions of `pydicom` are prioritized first
- JSON pretty print should not remove spaces
- Phasediff fieldmaps behavior
  - ensure phasediff exists
  - support for single magnitude acquisitions

# [0.5] - 2018-03-01
The first release after major refactoring:

## Changed
- Refactored into a proper `heudiconv` Python module
  - `heuristics` is now a `heudiconv.heuristics` submodule
  - you can specify shipped heuristics by name (e.g. `-f reproin`)
    without providing full path to their files
  - you need to use `--files` (not just positional argument(s)) if not
    using `--dicom_dir_templates` or `--subjects` to point to data files
    or directories with input DICOMs
- `Dockerfile` is generated by [neurodocker](https://github.com/kaczmarj/neurodocker)
- Logging verbosity reduced
- Increased leniency with missing DICOM fields
- `dbic_bids` heuristic renamed into reproin
## Added
- [LICENSE](https://github.com/nipy/heudiconv/blob/master/LICENSE)
  with Apache 2.0 license for the project
- [CHANGELOG.md](https://github.com/nipy/heudiconv/blob/master/CHANGELOG.md)
- [Regression testing](https://github.com/nipy/heudiconv/blob/master/tests/test_
↪regression.py) on real data (using datalad)
- A dedicated [ReproIn](https://github.com/repronim/reproin) project
  with details about ReproIn setup/specification and operation using
  `reproin` heuristic shipped with heudiconv
- [utils/test-compare-two-versions.sh](utils/test-compare-two-versions.sh)
  helper to compare conversions with two different versions of heudiconv
## Removed
- Support for converters other than `dcm2niix`, which is now the default.
  Explicitly specify `-c none` to only prepare conversion specification
  files without performing actual conversion
## Fixed
- Compatibility with Nipype 1.0, PyDicom 1.0, and upcoming DataLad 0.10
- Consistency with converted files permissions
- Ensured subject id for BIDS conversions will be BIDS compliant
```

(continues on next page)

(continued from previous page)

```
- Re-add `seqinfo` fields as column names in generated `dicominfo`
- More robust sanity check of the regex reformatted .json file to avoid
  numeric precision issues
- Many other various issues

# [0.4] - 2017-10-15
A usable release to support [DBIC][] use-case
## Added
- more testing
## Changes
- Dockerfile updates (added pigz, progressed forward [dcm2niix][])
## Fixed
- correct date/time in BIDS `_scans` files
- sort entries in `_scans` by date and then filename

# [0.3] - 2017-07-10
A somewhat working release on the way to support [DBIC][] use-case
## Added
- more tests
- grouping of dicoms by series if provided
- many more features and fixes

# [0.2] - 2016-10-20
An initial release on the way to support [DBIC][] use-case
## Added
- basic Python project assets (`setup.py`, etc)
- basic tests
- [datalad][] support
- dbic_bids heuristic
- `--dbg` command line flag to enter `pdb` environment upon failure
# Fixed
- Better Python3 support
- Better PEP8 compliance

# [0.1] - 2015-09-23

Initial version

---

## References
[DBIC]: http://dbic.dartmouth.edu
[datalad]: http://datalad.org
[dcm2niix]: https://github.com/rordenlab/dcm2niix
[#301]: https://github.com/nipy/heudiconv/issues/301
[#353]: https://github.com/nipy/heudiconv/issues/353
[#354]: https://github.com/nipy/heudiconv/issues/354
[#357]: https://github.com/nipy/heudiconv/issues/357
[#358]: https://github.com/nipy/heudiconv/issues/358
[#347]: https://github.com/nipy/heudiconv/issues/347
[#366]: https://github.com/nipy/heudiconv/issues/366
[#368]: https://github.com/nipy/heudiconv/issues/368
```

(continues on next page)

(continued from previous page)

```
[#373]: https://github.com/nipy/heudiconv/issues/373
[#485]: https://github.com/nipy/heudiconv/issues/485
[#442]: https://github.com/nipy/heudiconv/issues/442
[#451]: https://github.com/nipy/heudiconv/issues/451
[#459]: https://github.com/nipy/heudiconv/issues/459
[#473]: https://github.com/nipy/heudiconv/issues/473
[#477]: https://github.com/nipy/heudiconv/issues/477
[#293]: https://github.com/nipy/heudiconv/issues/293
[#304]: https://github.com/nipy/heudiconv/issues/304
[#306]: https://github.com/nipy/heudiconv/issues/306
[#310]: https://github.com/nipy/heudiconv/issues/310
[#327]: https://github.com/nipy/heudiconv/issues/327
[#328]: https://github.com/nipy/heudiconv/issues/328
[#334]: https://github.com/nipy/heudiconv/issues/334
[#337]: https://github.com/nipy/heudiconv/issues/337
[#339]: https://github.com/nipy/heudiconv/issues/339
[#342]: https://github.com/nipy/heudiconv/issues/342
[#343]: https://github.com/nipy/heudiconv/issues/343
[#344]: https://github.com/nipy/heudiconv/issues/344
[#345]: https://github.com/nipy/heudiconv/issues/345
[#348]: https://github.com/nipy/heudiconv/issues/348
[#351]: https://github.com/nipy/heudiconv/issues/351
[#352]: https://github.com/nipy/heudiconv/issues/352
[#359]: https://github.com/nipy/heudiconv/issues/359
[#360]: https://github.com/nipy/heudiconv/issues/360
[#364]: https://github.com/nipy/heudiconv/issues/364
[#369]: https://github.com/nipy/heudiconv/issues/369
[#372]: https://github.com/nipy/heudiconv/issues/372
[#375]: https://github.com/nipy/heudiconv/issues/375
[#376]: https://github.com/nipy/heudiconv/issues/376
[#379]: https://github.com/nipy/heudiconv/issues/379
[#380]: https://github.com/nipy/heudiconv/issues/380
[#387]: https://github.com/nipy/heudiconv/issues/387
[#390]: https://github.com/nipy/heudiconv/issues/390
[#404]: https://github.com/nipy/heudiconv/issues/404
[#407]: https://github.com/nipy/heudiconv/issues/407
[#419]: https://github.com/nipy/heudiconv/issues/419
[#420]: https://github.com/nipy/heudiconv/issues/420
[#425]: https://github.com/nipy/heudiconv/issues/425
[#430]: https://github.com/nipy/heudiconv/issues/430
[#432]: https://github.com/nipy/heudiconv/issues/432
[#434]: https://github.com/nipy/heudiconv/issues/434
[#436]: https://github.com/nipy/heudiconv/issues/436
[#437]: https://github.com/nipy/heudiconv/issues/437
[#462]: https://github.com/nipy/heudiconv/issues/462
[#464]: https://github.com/nipy/heudiconv/issues/464
[#480]: https://github.com/nipy/heudiconv/issues/480
[#481]: https://github.com/nipy/heudiconv/issues/481
[#487]: https://github.com/nipy/heudiconv/issues/487
[#489]: https://github.com/nipy/heudiconv/issues/489
[#491]: https://github.com/nipy/heudiconv/issues/491
[#496]: https://github.com/nipy/heudiconv/issues/496
```

(continues on next page)

(continued from previous page)

```
[#500]: https://github.com/nipy/heudiconv/issues/500
[#501]: https://github.com/nipy/heudiconv/issues/501
[#507]: https://github.com/nipy/heudiconv/issues/507
[#508]: https://github.com/nipy/heudiconv/issues/508
[#523]: https://github.com/nipy/heudiconv/issues/523
```

8.7.3 Tutorials

Quickstart

This tutorial is based on Dianne Patterson's University of Arizona tutorials

This guide assumes you have already *installed heudiconv and dcm2niix* and demonstrates how to use the heudiconv tool with a provided *heuristic.py* to convert DICOMS into the BIDS data structure.

Prepare Dataset

Download and unzip sub-219_dicom.zip.

We will be working from a directory called MRIS. Under the MRIS directory is the *dicom* subdirectory: Under the subject number *219* the session *itbs* is nested. Each dicom sequence folder is nested under the session:

```
dicom
└── 219
    └── itbs
        ├── Bzero_verify_PA_17
        ├── DTI_30_DIRs_AP_15
        ├── Localizers_1
        ├── MoCoSeries_19
        ├── MoCoSeries_31
        ├── Post_TMS_restingstate_30
        ├── T1_mprage_1mm_13
        ├── field_mapping_20
        ├── field_mapping_21
        └── restingstate_18
Nifti
└── code
    └── heuristic1.py
```

Basic Conversion

Next we will use heudiconv convert DICOMS into the BIDS data structure. The example dataset includes an example heuristic file, *heuristic1.py*. Typical use of heudiconv will require the creation and editing of your *heuristics file*, which we will cover in a *later tutorial*.

Note: Heudiconv requires you to run the command from the parent directory of both the Dicom and Nifti directories, which is *MRIS* in our case.

Run the following command:

```
heudiconv  --files dicom/219/itbs/*/*.dcm -o Nifti -f Nifti/code/heuristic1.py -s 219 -
↪ss itbs -c dcm2niix -b --minmeta --overwrite
```

- We specify the dicom files to convert with *–files*
- The heuristic file is provided with the *-f* option
- We tell heudiconv to place our output in the Nifti dir with *-o*
- *-b* indicates that we want to output in BIDS format
- *–minmeta* guarantees that meta-information in the dcms does not get inserted into the JSON sidecar. This is good because the information is not needed but can overflow the JSON file causing some BIDS apps to crash.

Output

The *Nifti* directory will contain a bids-compliant subject directory:

```
└── sub-219
    └── ses-itbs
        ├── anat
        ├── dwi
        ├── fmap
        └── func
```

The following required BIDS text files are also created in the Nifti directory. Details for filling in these skeleton text files can be found under tabular files in the BIDS specification:

```
CHANGES
README
dataset_description.json
participants.json
participants.tsv
task-rest_bold.json
```

Validation

Ensure that everything is according to spec by using bids validator

Click *Choose File* and then select the *Nifti* directory. There should be no errors (though there are a couple of warnings).

Note: Your files are not uploaded to the BIDS validator, so there are no privacy concerns!

Next

In the following sections, you will modify *heuristic.py* yourself so you can test different options and understand how to work with your own data.

Custom Heuristics

This tutorial is based on Dianne Patterson's University of Arizona tutorials

In this tutorial we go more in depth, creating our own *heuristic.py* and modifying it for our needs:

1. *Step1* Generate a heuristic (translation) file skeleton and some associated descriptor text files.
2. *Step2* Modify the *heuristic.py* to specify BIDS output names and directories, and the input DICOM characteristics.
3. *Step3* Call HeuDiConv to run on more subjects and sessions.

Prerequisites:

1. Ensure *heudiconv and dcm2niix* is installed.
2. *Prepare the dataset* used in the quickstart.

Step 1: Generate Skeleton

Note: Step 1 only needs to be completed once for each project. If repeating this step, ensure that the .heudiconv directory is removed.

From the *MRIS* directory, run the following command to process the `dcm` files that you downloaded and unzipped for this tutorial.:

```
heudiconv --files dicom/219/*/*/*.dcm -o Nifti/ -f convertall -s 219 -c none
```

- `--files dicom/{subject}/*/*/*.dcm` identifies the path to the DICOM files and specifies that they have the extension `.dcm` in this case.
- `-o Nifti/` is the output in *Nifti*. If the output directory does not exist, it will be created.
- `-f convertall` This creates a *heuristic.py* template from an existing heuristic module. There are other heuristic modules , but *convertall* is a good default.
- `-s 219` specifies the subject number.
- `-c none` indicates you are not actually doing any conversion right now.

You will now have a heudiconv skeleton in the *<output_dir>/.heudiconv* directory, in our case *Nifti/.heudiconv*

The `.heudiconv` hidden directory

Take a look at *MRIS/Nifti/.heudiconv/219/info/*, heudiconv has produced two files of interest: a skeleton *heuristic.py* and a *dicominfo.tsv* file. The generated heuristic file template contains comments explaining usage.

Warning:

- **The Good** Every time you run conversion to create the BIDS NIfTI files and directories, a detailed record of what you did is recorded in the *.heudiconv* directory. This includes a copy of the *heuristic.py* module that you ran for each subject and session. Keep in mind that the hidden *.heudiconv* directory gets updated every time you run heudiconv. Together your *code* and *.heudiconv* directories provide valuable provenance information that should remain with your data.
- **The Bad** If you rerun *heuristic.py* for some subject and session that has already been run, heudiconv quietly uses the conversion routines it stored in *.heudiconv*. This can be really annoying if you are troubleshooting *heuristic.py*.
- **More Good** You can remove subject and session information from *.heudiconv* and run it fresh. In fact, you can entirely remove the *.heudiconv* directory and still run the *heuristic.py* you put in the *code* directory.

Step 2: Modify Heuristic

We will modify the generated *heuristic.py* so heudiconv will arrange the output in a BIDS directory structure.

It is okay to rename this file, or to have several versions with different names, just be sure to pass the intended filename with *-f*. See *Heuristics File* docs for more info.

- I provide three section labels (1, 1b and 2) to facilitate exposition here. Each of these sections should be manually modified by you for your project.

Section 1

- This *heuristic.py* does not import all sequences in the example *Dicom* directory. This is a feature of heudiconv: You do not need to import scouts, motion corrected images or other DICOMs of no interest.
- You may wish to add, modify or remove keys from this section for your own data:

```
# Section 1: These key definitions should be revised by the user
###################################################################
# For each sequence, define a key variables (e.g., t1w, dwi etc) and template using␣
↪the create_key function:
# key = create_key(output_directory_path_and_name).

###### TIPS #######
# If there are sessions, then session must be subfolder name.
# Do not prepend the ses key to the session! It will be prepended automatically for␣
↪the subfolder and the filename.
# The final value in the filename should be the modality.  It does not have a key,␣
↪just a value.
# Otherwise, there is a key for every value.
# Filenames always start with subject, optionally followed by session, and end with␣
↪modality.

```

(continues on next page)

(continued from previous page)

```
###### Definitions #######
# The "data" key creates sequential numbers which can be used for naming sequences.
# This is especially valuable if you run the same sequence multiple times at the␣
↪scanner.
data = create_key('run-{item:03d}')

t1w = create_key('sub-{subject}/{session}/anat/sub-{subject}_{session}_T1w')

dwi = create_key('sub-{subject}/{session}/dwi/sub-{subject}_{session}_dir-AP_dwi')

# Save the RPE (reverse phase-encode) B0 image as a fieldmap (fmap).  It will be␣
↪used to correct
# the distortion in the DWI
fmap_rev_phase =  create_key('sub-{subject}/{session}/fmap/sub-{subject}_{session}_
↪dir-PA_epi')

fmap_mag =  create_key('sub-{subject}/{session}/fmap/sub-{subject}_{session}_
↪magnitude')

fmap_phase = create_key('sub-{subject}/{session}/fmap/sub-{subject}_{session}_
↪phasediff')

# Even if this is resting state, you still need a task key
func_rest = create_key('sub-{subject}/{session}/func/sub-{subject}_{session}_task-
↪rest_run-01_bold')
func_rest_post = create_key('sub-{subject}/{session}/func/sub-{subject}_{session}_
↪task-rest_run-02_bold')
```

- **Key**
  - Define a short informative key variable name for each image sequence you wish to export. Note that you can use any key names you want (e.g. *foo* would work as well as *fmap_phase*), but you need to be consistent.
  - The `key` name is to the left of the = for each row in the above example.
- **Template**
  - Use the variable `{subject}` to make the code general purpose, so you can apply it to different subjects in Step 3.
  - Use the variable `{session}` to make the code general purpose only if you have multiple sessions for each subject.
    - Once you use the variable `{session}`:
    - Ensure that a session gets added to the **output path**, e.g., `sub-{subject}/{session}/anat/` AND
    - Session gets added to the **output filename**: `sub-{subject}_{session}_T1w` for every image in the session.
    - Otherwise you will get bids validator errors
  - Define the output directories and file names according to the BIDS specification
  - Note the output names for the fieldmap images (e.g., *sub-219_ses-itbs_dir-PA_epi.nii.gz*, *sub-219_ses-itbs_magnitude1.nii.gz*, *sub-219_ses-itbs_magnitude2.nii.gz*, *sub-219_ses-itbs_phasediff.nii.gz*).
  - The reverse_phase encode dwi image (e.g., *sub-219_ses-itbs_dir-PA_epi.nii.gz*) is grouped with the fieldmaps because it is used to correct other images.

    – Data that is not yet defined in the BIDS specification will cause the bids-validator to produce an error unless you include it in a .bidsignore file.

- **data**
    – a key definition that creates sequential numbering
    – `03d` means *create three slots for digits* `3d`, *and pad with zeros* `0`.
    – This is useful if you have a scanner sequence with a single name but you run it repeatedly and need to generate separate files for each run. For example, you might define a single functional sequence at the scanner and then run it several times instead of creating separate names for each run.

Note: It is usually better to name your sequences explicitly (e.g., run-01, run-02 etc.) rather than depending on sequential numbering. There will be less confusion later.

    – If you have a sequence with the same name that you run repeatedly WITHOUT the sequential numbering, HeuDiConv will overwrite earlier sequences with later ones.
    – To ensure that a sequence includes sequential numbering, you also need to add `run-{item:03d}` (for example) to the key-value specification for that sequence.
    – Here I illustrate with the t1w key-value pair:
        ∗ If you started with:
            · `t1w = create_key('sub-{subject}/anat/sub-{subject}_T1w')`,
        ∗ You could add sequence numbering like this:
            · `t1w = create_key('sub-{subject}/anat/sub-{subject}_run-{item:03d}_T1w')`.
        ∗ Now if you export several T1w images for the same subject and session, using the exact same protocol, each will get a separate run number like this:
            · *sub-219_ses_run-001_T1w.nii.gz, sub-219_ses_run-002_T1w.nii.gz* etc.

Section 1b

- Based on your chosen keys, create a data dictionary called *info*:

```
# Section 1b: This data dictionary (below) should be revised by the user.
###########################################################################
# info is a Python dictionary containing the following keys from the infotodict␣
↪defined above.
# This list should contain all and only the sequences you want to export from the␣
↪dicom directory.
info = {t1w: [], dwi: [], fmap_rev_phase: [], fmap_mag: [], fmap_phase: [], func_
↪rest: [], func_rest_post: []}

# The following line does no harm, but it is not part of the dictionary.
last_run = len(seqinfo)
```

- Enter each key in the dictionary in this format `key:  []`, for example, `t1w:  []`.
- Separate the entries with commas as illustrated above.

Section 2

- Define the criteria for identifying each DICOM series that corresponds to one of the keys you want to export:

```
# Section 2: These criteria should be revised by the user.
##########################################################
# Define test criteria to check that each DICOM sequence is correct
# seqinfo (s) refers to information in dicominfo.tsv. Consult that file for
# available criteria.
# Each sequence to export must have been defined in Section 1 and included in␣
↪Section 1b.
# The following illustrates the use of multiple criteria:
for idx, s in enumerate(seqinfo):
    # Dimension 3 must equal 176 and the string 'mprage' must appear somewhere in the␣
↪protocol_name
    if (s.dim3 == 176) and ('mprage' in s.protocol_name):
        info[t1w].append(s.series_id)

    # Dimension 3 must equal 74 and dimension 4 must equal 32, and the string 'DTI'␣
↪must appear somewhere in the protocol_name
    if (s.dim3 == 74) and (s.dim4 == 32) and ('DTI' in s.protocol_name):
        info[dwi].append(s.series_id)

    # The string 'verify_P-A' must appear somewhere in the protocol_name
    if ('verify_P-A' in s.protocol_name):
        info[fmap_rev_phase] = [s.series_id]

    # Dimension 3 must equal 64, and the string 'field_mapping' must appear somewhere␣
↪in the protocol_name
    if (s.dim3 == 64) and ('field_mapping' in s.protocol_name):
        info[fmap_mag] = [s.series_id]

    # Dimension 3 must equal 32, and the string 'field_mapping' must appear somewhere␣
↪in the protocol_name
    if (s.dim3 == 32) and ('field_mapping' in s.protocol_name):
        info[fmap_phase] = [s.series_id]

    # The string 'resting_state' must appear somewhere in the protocol_name and the␣
↪Boolean field is_motion_corrected must be False (i.e. not motion corrected)
    # This ensures I do NOT get the motion corrected MOCO series instead of the raw␣
↪series!
    if ('restingstate' == s.protocol_name) and (not s.is_motion_corrected):
        info[func_rest].append(s.series_id)

    # The string 'Post_TMS_resting_state' must appear somewhere in the protocol_name␣
↪and the Boolean field is_motion_corrected must be False (i.e. not motion␣
↪corrected)

    # This ensures I do NOT get the motion corrected MOCO series instead of the raw␣
↪series.
    if ('Post_TMS_restingstate' == s.protocol_name) and (not s.is_motion_corrected):
        info[func_rest_post].append(s.series_id)
```

  – To define the criteria, look at *dicominfo.tsv* in *.heudiconv/info*. This file contains tab-separated values so you can easily view it in Excel or any similar spreadsheet program. *dicominfo.tsv* is not used programmatically to run heudiconv (i.e., you could delete it with no adverse consequences), but it is very useful for defining the test criteria for Section 2 of *heuristic.py*.

- Some values in *dicominfo.tsv* might be wrong. For example, my reverse phase encode sequence with two acquisitions of 74 slices each is reported as one acquisition with 148 slices (2018_12_11). Hopefully they'll fix this. Despite the error in *dicominfo.tsv*, dcm2niix reconstructed the images correctly.
- You will be adding, removing or altering values in conditional statements based on the information you find in *dicominfo.tsv*.
- `seqinfo` (s) refers to the same information you can view in *dicominfo.tsv* (although seqinfo does not rely on *dicominfo.tsv*).
- Here are two types of criteria:
  * `s.dim3 == 176` is an **equivalence** (e.g., good for checking dimensions for a numerical data type). For our sample T1w image to be exported from DICOM, it must have 176 slices in the third dimension.
  * `'mprage' in s.protocol_name` says the protocol name string must **include** the word *mprage* for the *T1w* image to be exported from DICOM. This criterion string is case-sensitive.
- `info[t1w].append(s.series_id)` Given that the criteria are satisfied, the series should be named and organized as described in *Section 1* and referenced by the info dictionary. The information about the processing steps is saved in the *.heudiconv* subdirectory.
- Here I have organized each conditional statement so that the sequence protocol name comes first followed by other criteria if relevant. This is not necessary, though it does make the resulting code easier to read.

Step 3:

- You have now done all the hard work for your project. When you want to add a subject or session, you only need to run this third step for that subject or session (A record of each run is kept in .heudiconv for you):

```
heudiconv --files dicom/{subject}/*/*.dcm -o Nifti/ -f Nifti/code/heuristic.py -s␣
↪219 -ss itbs -c dcm2niix -b --minmeta --overwrite
```

- The first time you run this step, several important text files are generated (e.g., CHANGES, dataset_description.json, participants.tsv, README etc.). On subsequent runs, information may be added (e.g., *participants.tsv* will be updated). Other files, like the *README* and *dataset_description.json* should be updated manually.
- This Docker command is slightly different from the previous Docker command you ran.
  - `-f Nifti/code/heuristic.py` now tells HeuDiConv to use your revised *heuristic.py* in the *code* directory.
  - In this case, we specify the subject we wish to process `-s 219` and the name of the session `-ss itbs`.
  - We could specify multiple subjects like this: `-s 219 220 -ss itbs`
  - `-c dcm2niix -b` indicates that we want to use the dcm2niix converter with the -b flag (which creates BIDS).
  - `--minmeta` ensures that only the minimum necessary amount of data gets added to the JSON file when created. On the off chance that there is a LOT of meta-information in the DICOM header, the JSON file will not get swamped by it. fmriprep and mriqc are very sensitive to this information overload and will crash, so *minmeta* provides a layer of protection against such corruption.

- `--overwrite` This is a peculiar option. Without it, I have found the second run of a sequence does not get generated. But with it, everything gets written again (even if it already exists). I don't know if this is my problem or the tool... but for now, I'm using `--overwrite`.
- Step 3 should produce a tree like this:

```
Nifti
├── CHANGES
├── README
├── code
│   ├── __pycache__
│   │   └── heuristic1.cpython-36.pyc
│   ├── heuristic1.py
│   └── heuristic2.py
├── dataset_description.json
├── participants.json
├── participants.tsv
├── sub-219
│   └── ses-itbs
│       ├── anat
│       │   ├── sub-219_ses-itbs_T1w.json
│       │   └── sub-219_ses-itbs_T1w.nii.gz
│       ├── dwi
│       │   ├── sub-219_ses-itbs_dir-AP_dwi.bval
│       │   ├── sub-219_ses-itbs_dir-AP_dwi.bvec
│       │   ├── sub-219_ses-itbs_dir-AP_dwi.json
│       │   └── sub-219_ses-itbs_dir-AP_dwi.nii.gz
│       ├── fmap
│       │   ├── sub-219_ses-itbs_dir-PA_epi.json
│       │   ├── sub-219_ses-itbs_dir-PA_epi.nii.gz
│       │   ├── sub-219_ses-itbs_magnitude1.json
│       │   ├── sub-219_ses-itbs_magnitude1.nii.gz
│       │   ├── sub-219_ses-itbs_magnitude2.json
│       │   ├── sub-219_ses-itbs_magnitude2.nii.gz
│       │   ├── sub-219_ses-itbs_phasediff.json
│       │   └── sub-219_ses-itbs_phasediff.nii.gz
│       ├── func
│       │   ├── sub-219_ses-itbs_task-rest_run-01_bold.json
│       │   ├── sub-219_ses-itbs_task-rest_run-01_bold.nii.gz
│       │   ├── sub-219_ses-itbs_task-rest_run-01_events.tsv
│       │   ├── sub-219_ses-itbs_task-rest_run-02_bold.json
│       │   ├── sub-219_ses-itbs_task-rest_run-02_bold.nii.gz
│       │   └── sub-219_ses-itbs_task-rest_run-02_events.tsv
│       ├── sub-219_ses-itbs_scans.json
│       └── sub-219_ses-itbs_scans.tsv
└── task-rest_bold.json
```

TIPS

- **Name Directories as you wish**: You can name the project directory (e.g., **MRIS**) and the output directory (e.g., **Nifti**) as you wish (just don't put spaces in the names!).
- **Age and Sex Extraction**: Heudiconv will extract age and sex info from the DICOM header. If there is any reason to believe this information is wrong in the DICOM header (for example, it was made-up because no one knew how old the subject was, or it was considered a privacy concern), then you need to check the output. If you have Horos (or another DICOM editor), you can edit the values in the DICOM headers, otherwise you need to edit the values in the BIDS text file *participants.tsv*.
- **Separating Sessions**: If you have multiple sessions at the scanner, you should create an *Exam* folder for each session. This will help you to keep the data organized and *Exam* will be reported in the *study_description* in your *dicominfo.tsv*, so that you can use it as a criterion.
- **Don't manually combine DICOMS from different sessions**: If you combine multiple sessions in one subject DICOM folder, heudiconv will fail to run and will complain about `conflicting study identifiers`. You can get around the problem by figuring out which DICOMs are from different sessions and separating them so you deal with one set at a time. This may mean you have to manually edit the BIDS output.
  - Why might you manually combine sessions you ask? Because you never intended to have multiple sessions, but the subject had to complete some scans the next day. Or, because the scanner had to be rebooted.
- **Don't assume all your subjects' dicoms have the same names or that the sequences were always run in the same order**: If you develop a *heuristic.py* on one subject, try it and carefully evaluate the results on your other subjects. This is especially true if you already collected the data before you started thinking about automating the output. Every time you run HeuDiConv with *heuristic.py*, a new *dicominfo.tsv* file is generated. Inspect this for differences in protocol names and series descriptions etc.
- **Decompressing DICOMS**: Decompress your data, heudiconv does not yet support compressed DICOM conversion. https://github.com/nipy/heudiconv/issues/287
- **Create unique DICOM protocol names at the scanner** If you have the opportunity to influence the DICOM naming strategies, then try to ensure that there is a unique protocol name for every run. For example, if you repeat the fmri protocol three times, name the first one fmri_1, the next fmri_2, and the last fmri_3 (or any variation on this theme). This will make it much easier to uniquely specify the sequences when you convert and reduce your chance of errors.

Exploring Criteria

dicominfo.tsv contains a human readable version of seqinfo. Each column of data can be used as criteria for identifying the correct DICOM image. We have already provided examples of using string types, numbers, and Booleans (True-False). Tuples (immutable lists) are also available and examples of using these are provided below. To ensure that you are extracting the images you want, you need to be very careful about creating your initial *heuristic.py*.

Why Experiment?

- Criteria can be tricky. Ensure the NIfTI files you create are the correct ones (for example, not the derived or motion corrected if you didn't want that). In addition to looking at the images created (which tells you whether you have a fieldmap or T1w etc.), you should look at the dimensions of the image. Not only the dimensions, but the range of intensity values and the size of the image on disk should match for dcm2niix and heudiconv's *heuristic.py*.
- For really tricky cases, download and install dcm2niix on your local machine and run it for a sequence of concern (in my experience, it is usually fieldmaps that go wrong).

- Although Python does not require you to use parentheses while defining criteria, parentheses are a good idea. Parentheses will help ensure that complex criteria involving multiple logical operators `and`, `or`, `not` make sense and behave as expected.

Tuples

Suppose you want to use the values in the field `image_type`? It is not a number or string or Boolean. To discover the data type of a column, you can add a statement like this `print(type(s.image_type))` to the for loop in Section 2 of *heuristic.py*. Then run *heuristic.py* (preferably without any actual conversions) and you should see an output like this `<class 'tuple'>`. Here is an example of using a value from `image_type` as a criterion:

```
if ('ASL_3D_tra_iso' == s.protocol_name) and ('TTEST' in s.image_type):
   info[asl_der].append(s.series_id)
```

Note that this differs from testing for a string because you cannot test for any substring (e.g., 'TEST' would not work). String tests will not work on a tuple datatype.

Note: *image_type* is described in the DICOM specification

Reproin

This tutorial is based on Dianne Patterson's University of Arizona tutorials

Reproin is a setup for automatic generation of sharable, version-controlled BIDS datasets from MR scanners.

If you can control how your image sequences are named at the scanner, you can use the *reproin* naming convention. If you cannot control such naming, or already have collected data, you can provide your custom heuristic mapping into *reproin* and thus in effect use reproin heuristic. That will be a topic for another tutorial but meanwhile you can checkout reproin/issues/18 for a brief HOWTO.

Get Example Dataset

This example uses a phantom dataset: reproin_dicom.zip generated by the University of Arizona on their Siemens Skyra 3T with Syngo MR VE11c software on 2018_02_08.

The `REPROIN` directory is a simple reproin-compliant DICOM (.dcm) dataset without sessions. (Derived dwi images (ADC, FA etc.) that the scanner produced have been removed.:

```
[user@local ~/reproin_dicom/REPROIN]$ tree -I "*.dcm"

    REPROIN
    ├── data
    └── dicom
        └── 001
            └── Patterson_Coben\ -\ 1
                ├── Localizers_4
                ├── anatT1w_acqMPRAGE_6
                ├── dwi_dirAP_9
                ├── fmap_acq4mm_7
                ├── fmap_acq4mm_8
```

(continues on next page)

(continued from previous page)

```
                    ├── fmap_dirPA_15
                    └── func_taskrest_16
```

Convert and organize

From the `REPROIN` directory:

```
heudiconv -f reproin --bids  -o data --files dicom/001 --minmeta
```

- `-f reproin` specifies the converter file to use
- `-o data/` specifies the output directory *data*. If the output directory does not exist, it will be created.
- `--files dicom/001` identifies the path to the DICOM files.
- `--minmeta` ensures that only the minimum necessary amount of data gets added to the JSON file when created. On the off chance that there is a LOT of meta-information in the DICOM header, the JSON file will not get swamped by it. Rumors are that fMRIPrep and MRIQC might be sensitive to excess of metadata and might crash crash, so minmeta provides a layer of protection against such corruption.

Output Directory Structure

Heudiconv's Reproin converter produces a hierarchy of directories with the BIDS dataset (here - *Cohen*) at the bottom:

```
data
└── Patterson
    └── Coben
        ├── sourcedata
        │   └── sub-001
        │       ├── anat
        │       ├── dwi
        │       ├── fmap
        │       └── func
        └── sub-001
            ├── anat
            ├── dwi
            ├── fmap
            └── func
```

The specific value for the hierarchy can be specified to HeuDiConv via *–locator PATH* option. If not, ReproIn heuristic bases it on the value of the DICOM "Study Description" field which is populated when user selects a specific *Exam* card located within some *Region* (see ReproIn Walkthrough "Organization").

- The dataset is nested under two levels in the output directory: *Region* (Patterson) and *Exam* (Coben). *Tree* is reserved for other purposes at the UA research scanner.
- Although the Program *Patient* is not visible in the output hierarchy, it is important. If you have separate sessions, then each session should have its own Program name.
- **sourcedata** contains tarred gzipped (*.tgz*) sets of DICOM images corresponding to NIfTI images.
- **sub-001/** contains a single subject data within this BIDS dataset.
- The hidden directory is generated: *REPROIN/data/Patterson/Coben/.heudiconv* to contain derived mapping data, which could potentially be inspected or adjusted/used for re-conversion.

Reproin Scanner File Names

- **For both BIDS and *reproin*, names are composed of an ordered series of key-value pairs, called [*entities*](https://github.com/bids-standard/bids-specification/blob/master/src/schema/objects/entities.yaml).**
  Each key and its value are joined with a dash - (e.g., `acq-MPRAGE`, `dir-AP`). These key-value pairs are joined to other key-value pairs with underscores `_`. The exception is the modality label, which is discussed more below.
- *Reproin* scanner sequence names are simplified relative to the final BIDS output and generally conform to this scheme (but consult the reproin heuristics file for additional options): `sequence type-modality label _ session-session name _ task-task name _ acquisition-acquisition detail _ run-run number _ direction-direction label`:

```
| func-bold_ses-pre_task-faces_acq-1mm_run-01_dir-AP
```

- Each sequence name begins with the seqtype key. The seqtype key is the modality and corresponds to the name of the BIDS directory where the sequence belongs, e.g., `anat`, `dwi`, `fmap` or `func`.
- The seqtype key is optionally followed by a dash - and a modality label value (e.g., `anat-scout` or `anat-T2W`). Often, the modality label is not needed because there is a predictable default for most seqtypes:
- For **anat** the default modality is `T1W`. Thus a sequence named `anat` will have the same output BIDS files as a sequence named `anat-T1w`: *sub-001_T1w.nii.gz*.
- For **fmap** the default modality is `epi`. Thus `fmap_dir-PA` will have the same output as `fmap-epi_dir-PA`: *sub-001_dir-PA_epi.nii.gz*.
- For **func** the default modality is `bold`. Thus, `func-bold_task-rest` will have the same output as `func_task-rest`: *sub-001_task-rest_bold.nii.gz*.
- *Reproin* gets the subject number from the DICOM metadata.
- If you have multiple sessions, the session name does not need to be included in every sequence name in the program (i.e., Program= *Patient* level mentioned above). Instead, the session can be added to a single sequence name, usually the scout (localizer) sequence e.g. `anat-scout_ses-pre`, and *reproin* will propagate the session information to the other sequence names in the *Program*. Interestingly, *reproin* does not add the localizer to your BIDS output.
- When our scanner exports the DICOM sequences, all dashes are removed. But don't worry, *reproin* handles this just fine.
- In the UA phantom reproin data, the subject was named `01`. Horos reports the subject number as `01` but exports the DICOMS into a directory `001`. If the data are copied to an external drive at the scanner, then the subject number is reported as `001_001` and the images are `*.IMA` instead of `*.dcm`. *Reproin* does not care, it handles all of this gracefully. Your output tree (excluding *sourcedata* and *.heudiconv*) should look like this:

```
.
|-- CHANGES
|-- README
|-- dataset_description.json
|-- participants.tsv
|-- sub-001
|   |-- anat
|   |   |-- sub-001_acq-MPRAGE_T1w.json
|   |   `-- sub-001_acq-MPRAGE_T1w.nii.gz
|   |-- dwi
|   |   |-- sub-001_dir-AP_dwi.bval
```

(continues on next page)

(continued from previous page)

```
|   |   |-- sub-001_dir-AP_dwi.bvec
|   |   |-- sub-001_dir-AP_dwi.json
|   |   `-- sub-001_dir-AP_dwi.nii.gz
|   |-- fmap
|   |   |-- sub-001_acq-4mm_magnitude1.json
|   |   |-- sub-001_acq-4mm_magnitude1.nii.gz
|   |   |-- sub-001_acq-4mm_magnitude2.json
|   |   |-- sub-001_acq-4mm_magnitude2.nii.gz
|   |   |-- sub-001_acq-4mm_phasediff.json
|   |   |-- sub-001_acq-4mm_phasediff.nii.gz
|   |   |-- sub-001_dir-PA_epi.json
|   |   `-- sub-001_dir-PA_epi.nii.gz
|   |-- func
|   |   |-- sub-001_task-rest_bold.json
|   |   |-- sub-001_task-rest_bold.nii.gz
|   |   `-- sub-001_task-rest_events.tsv
|   `-- sub-001_scans.tsv
`-- task-rest_bold.json
```

- Note that despite all the the different subject names (e.g., `01`, `001` and `001_001`), the subject is labeled `sub-001`.

External Tutorials

Luckily(?), we live in an era of plentiful information. Below are some links to other users' tutorials covering their experience with `heudiconv`.

- YouTube tutorial by James Kent.
- Walkthrough by the Stanford Center for Reproducible Neuroscience.
- U of A Neuroimaging Core by Dianne Patterson.
- Sample Conversion: Coastal Coding 2019.
- A joined DataLad and HeuDiConv tutorial for reproducible fMRI studies.
- The ReproIn conversion workflow overview.
- Slides and recording of a ReproNim Webinar on `heudiconv`.

Caution: Some of these tutorials may not be up to date with the latest releases of `heudiconv`.

8.7.4 Heuristics File

The heuristic file controls how information about the DICOMs is used to convert to a file system layout (e.g., BIDS).

Provided Heuristics

`heudiconv` provides over 10 pre-created heuristics, which can be seen here .

These heuristic files are documented in their code comments. Some of them, like convertall or ReproIn could be immediately reused and represent two ends of the spectrum in heuristics:

- `convertall` is very simple and does not automate anything – it is for a user to modify filenames in the prepared conversion table, and then rerun with `-c dcm2niix`.
- `reproin` can be used fully automated, if original sequences were named according to its ReproIn convention.

Discover more on their user in the *Tutorials* section.

However, there is a large variety of data out there, and not all DICOMs will be covered by the existing heuristics. This section will outline what makes up a heuristic file, and some useful functions available when making one.

Components

`infotodict(seqinfos)`

The only required function for a heuristic, *infotodict* is used to both define the conversion outputs and specify the criteria for scan to output association. Conversion outputs are defined as keys, a *tuple* consisting of three elements:

- a template path used for the basis of outputs
- *tuple* of output types. Valid types include *nii*, *nii.gz*, and *dicom*.
- *None* - a historical artifact (corresponds to some notion of `annotation_class` no living human is aware about)

The following details of the sequences could also be used as a `{detail}` in the conversion keys:

- `item`: an index of seqinfo (e.g., `1`),
- `subject`: a subject label (e.g., `qa`)
- `seqitem`: sequence item, index with a sequence/protocol name (e.g., `3-anat-scout_ses-{date}`)
- `subindex`: an index within the `seqinfo` (e.g., `1`),
- `session`: empty (no session) or a session entity (along with `ses-`, e.g., `ses-20191216`),
- `bids_subject_session_prefix`: shortcut for BIDS file name prefix combining subject and optional session (e.g., `sub-qa_ses-20191216`),
- `bids_subject_session_dir`: shortcut for BIDS file path combining subject and optional session (e.g., `sub-qa/ses-20191216`).

Note: An example conversion key

`('sub-{subject}/func/sub-{subject}_task-test_run-{item}_bold', ('nii.gz', 'dicom'), None)`

or equivalent in *–bids* mode which would work also if there is a specified session

`('{bids_subject_session_dir}/func/{bids_subject_session_prefix}_task-test_run-{item}_bold', ('nii.gz', 'dicom'), None)`

The `seqinfos` parameter is a list of namedtuples which serves as a grouped and stacked record of the DICOMs passed in. Each item in *seqinfo* contains DICOM metadata that can be used to isolate the series, and assign it to a conversion key.

A function `create_key` is commonly defined by heuristics (internally) to assist in creating the key, and to be used inside `infotodict`.

A dictionary of {`conversion key`: `series_id`} is returned, where `series_id` is the 3rd (indexes as `[2]` or accessed as `.series_id` from `seqinfo`).

`create_key(template, outtype)`

A common helper function used to create the conversion key in `infotodict`. But it is not used directly by HeuDiConv.

`filter_files(fl)`

A utility function used to filter any input files.

If this function is included, every file found will go through this filter. Any files where this function returns `True` will be filtered out.

`filter_dicom(dcm_data)`

A utility function used to filter any DICOMs.

If this function is included, every DICOM found will go through this filter. Any DICOMs where this function returns `True` will be filtered out.

`infotoids(seqinfos, outdir)`

Further processing on `seqinfos` to deduce/customize subject, session, and locator.

A dictionary of {"locator": locator, "session": session, "subject": subject} is returned.

`grouping` string or `grouping(files, dcmfilter, seqinfo)`

Whenever `--grouping custom` (`-g custom`) is used, this attribute or callable will be used to inform how to group the DICOMs into separate groups. From original PR#359:

```
grouping = 'AcquisitionDate'
```

or:

```
def grouping(files, dcmfilter, seqinfo):
    seqinfos = collections.OrderedDict()
    ...
    return seqinfos  # ordered dict containing seqinfo objects: list of DICOMs
```

custom_seqinfo(wrapper, series_files)

If present this function will be called on each group of dicoms with a sample nibabel dicom wrapper to extract additional information to be used in `infotodict`.

Importantly the return value of that function needs to be hashable. For instance the following non-hashable types can be converted to an alternative hashable type: - list > tuple - dict > frozendict - arrays > bytes (tobytes(), or pickle.dumps), str or tuple of tuples.

POPULATE_INTENDED_FOR_OPTS

Dictionary to specify options to populate the `'IntendedFor'` field of the `fmap` jsons.

When a BIDS session has `fmaps`, they can automatically be assigned to be used for susceptibility distortion correction of other non-`fmap` images in the session (populating the `'IntendedFor'` field in the `fmap` json file).

For this automated assignment, `fmaps` are taken as groups (`_phase` and `_phasediff` images and the corresponding `_magnitude` images; consecutive Spin-Echo images collected with opposite phase encoding polarity (`pepolar` case); etc.).

This is achieved by checking, for every non-`fmap` image in the session, which `fmap` groups are suitable candidates to correct for distortions in that image. Then, if there is more than one candidate (e.g., if there was a `fmap` collected at the beginning of the session and another one at the end), the user can specify which one to use.

The parameters that can be specified and the allowed options are defined in `bids.py`:

- `'matching_parameter'`: The imaging parameter that needs to match between the `fmap` and an image for the `fmap` to be considered as a suitable to correct that image. Allowed options are:
  - `'Shims'`: `heudiconv` will check the `ShimSetting` in the `.json` files and will only assign `fmaps` to images if the `ShimSettings` are identical for both.
  - `'ImagingVolume'`: both `fmaps` and images will need to have the same the imaging volume (the header affine transformation: position, orientation and voxel size, as well as number of voxels along each dimensions).
  - `'ModalityAcquisitionLabel'`: it checks for what modality (`anat`, `func`, `dwi`) each `fmap` is intended by checking the `_acq-` label in the `fmap` filename and finding corresponding modalities (e.g. `_acq-fmri`, `_acq-bold` and `_acq-func` will be matched with the `func` modality)
  - `'CustomAcquisitionLabel'`: it checks for what modality images each `fmap` is intended by checking the `_acq-` custom label (e.g. `_acq-XYZ42`) in the `fmap` filename, and matching it with: - the corresponding modality image `_acq-` label for modalities other than `func` (e.g. `_acq-XYZ42` for `dwi` images) - the corresponding image `_task-` label for the `func` modality (e.g. `_task-XYZ42`)
  - `'Force'`: forces `heudiconv` to consider any `fmaps` in the session to be suitable for any image, no matter what the imaging parameters are.
- `'criterion'`: Criterion to decide which of the candidate `fmaps` will be assigned to a given file, if there are more than one. Allowed values are:
  - `'First'`: The first matching `fmap`.
  - `'Closest'`: The closest in time to the beginning of the image acquisition.

Note: Example:

```
POPULATE_INTENDED_FOR_OPTS = {
        'matching_parameters': ['ImagingVolume', 'Shims'],
        'criterion': 'Closest'
}
```

If POPULATE_INTENDED_FOR_OPTS is not present in the heuristic file, IntendedFor will not be populated automatically.

8.7.5 CLI Reference

heudiconv processes DICOM files and converts the output into user defined paths.

Example:
heudiconv -d 'rawdata/{subject}' -o . -f heuristic.py -s s1 s2 s3

```
usage: heudiconv [-h] [--version]
                 [-d DICOM_DIR_TEMPLATE | --files [FILES ...]]
                 [-s [SUBJS ...]] [-c {dcm2niix,none}] [-o OUTDIR]
                 [-l LOCATOR] [-a CONV_OUTDIR] [--anon-cmd ANON_CMD]
                 [-f HEURISTIC] [-p] [-ss SESSION]
                 [-b [BIDSOPTION1 [BIDSOPTION2 ...]]] [--overwrite]
                 [--datalad] [--dbg]
                 [--command {heuristics,heuristic-info,ls,populate-templates,sanitize-
↪jsons,treat-jsons,populate-intended-for}]
                 [-g {studyUID,accession_number,all,custom}] [--minmeta]
                 [--random-seed RANDOM_SEED] [--dcmconfig DCMCONFIG]
                 [-q {SLURM,None}] [--queue-args QUEUE_ARGS]
```

Named Arguments

--version	show program's version number and exit
-d, --dicom_dir_template	Location of dicomdir that can be indexed with subject id {subject} and session {session}. Tarballs (can be compressed) are supported in addition to directory. All matching tarballs for a subject are extracted and their content processed in a single pass. If multiple tarballs are found, each is assumed to be a separate session and the –ses argument is ignored. Note that you might need to surround the value with quotes to avoid {…} being considered by shell
--files	Files (tarballs, dicoms) or directories containing files to process. Cannot be provided if using –dicom_dir_template.
-s, --subjects	List of subjects - required for dicom template. If not provided, DICOMS would first be "sorted" and subject IDs deduced by the heuristic.
-c, --converter	Possible choices: dcm2niix, none Tool to use for DICOM conversion. Setting to "none" disables the actual conversion step – useful for testing heuristics.
-o, --outdir	Output directory for conversion setup (for further customization and future reference. This directory will refer to non-anonymized subject IDs.

-l, --locator	Study path under outdir. If provided, it overloads the value provided by the heuristic. If –datalad is enabled, every directory within locator becomes a super-dataset thus establishing a hierarchy. Setting to "unknown" will skip that dataset.
-a, --conv-outdir	Output directory for converted files. By default this is identical to –outdir. This option is most useful in combination with –anon-cmd.
--anon-cmd	Command to run to convert subject IDs used for DICOMs to anonymized IDs. Such command must take a single argument and return a single anonymized ID. Also see –conv-outdir.
-f, --heuristic	Name of a known heuristic or path to the Python script containing heuristic.
-p, --with-prov	Store additional provenance information. Requires python-rdflib.
-ss, --ses	Session for longitudinal study_sessions. Default is None.
-b, --bids	Possible choices: notop Flag for output into BIDS structure. Can also take BIDS-specific options, e.g., –bids notop. The only currently supported options is "notop", which skips creation of top-level BIDS files. This is useful when running in batch mode to prevent possible race conditions.
--overwrite	Overwrite existing converted files.
--datalad	Store the entire collection as DataLad dataset(s). Small files will be committed directly to git, while large to annex. New version (6) of annex repositories will be used in a "thin" mode so it would look to mortals as just any other regular directory (i.e. no symlinks to under .git/annex). For now just for BIDS mode.
--dbg	Do not catch exceptions and show exception traceback.
--command	Possible choices: heuristics, heuristic-info, ls, populate-templates, sanitize-jsons, treat-jsons, populate-intended-for Custom action to be performed on provided files instead of regular operation.
-g, --grouping	Possible choices: studyUID, accession_number, all, custom How to group dicoms (default: by studyUID).
--minmeta	Exclude dcmstack meta information in sidecar jsons.
--random-seed	Random seed to initialize RNG.
--dcmconfig	JSON file for additional dcm2niix configuration.

Conversion submission options

-q, --queue	Possible choices: SLURM, None Batch system to submit jobs in parallel.
--queue-args	Additional queue arguments passed as a single string of space-separated Argument=Value pairs.

8.7.6 Using heudiconv in a Container

If heudiconv is *installed via a Docker container*, you can run the commands in the following format:

```
docker run nipy/heudiconv:latest [heudiconv options]
```

So a user running via container would check the version with this command:

```
docker run nipy/heudiconv:latest --version
```

Which is equivalent to the locally installed command:

```
heudiconv --version
```

Bind mount

Typically, users of heudiconv will be operating on data that is on their local machine. We can give heudiconv access to that data via a `bind mount`, which is the `-v` syntax.

Once common pattern is to share the working directory with `-v $PWD:$PWD`, so heudiconv will behave as though it is installed on your system. However, you should be aware of how permissions work depending on your container toolset.

Docker Permissions

When you run a container with docker without specifying a user, it will be run as root. This isn't ideal if you are operating on data owned by your local user, so for `Docker` it is recommended to specify that the container will run as your user.:

```
docker run --user=$(id -u):$(id -g) -e "UID=$(id -u)" -e "GID=$(id -g)" --rm -t -v $PWD:
↪$PWD nipy/heudiconv:latest --version
```

Podman Permissions

When running Podman without specifying a user, the container is run as root inside the container, but your user outside of the container. This default behavior usually works for heudiconv users:

```
docker run -v $PWD:PWD nipy/heudiconv:latest --version
```

Other Common Options

We typically recommend users make use of the following flags to Docker and Podman

- `-it` Interactive terminal
- `--rm` Remove the changes to the container when it completes

8.7.7 API Reference

BIDS

Handle BIDS specific operations

exception `heudiconv.bids.`**`BIDSError`**

class `heudiconv.bids.`**`BIDSFile`**(*entities: dict[str, str]*, *suffix: str*, *extension: str | None*)

as defined in https://bids-specification.readthedocs.io/en/stable/99-appendices/04-entity-table.html which might soon become machine readable order matters

classmethod **`parse`**(*filename: str*) → *BIDSFile*

Parse the filename for BIDS entities, suffix and extension

`heudiconv.bids.`**`HEUDICONV_VERSION_JSON_KEY = 'HeudiconvVersion'`**

JSON Key where we will embed our version in the newly produced .json files

`heudiconv.bids.`**`add_rows_to_scans_keys_file`**(*fn: str*, *newrows: dict[str, list[str]]*) → None

Add new rows to the _scans file.

Parameters

- **`fn`** (*`str`*) – filename
- **`newrows`** (*`dict`*) – extra rows to add (acquisition time, referring physician, random string)

`heudiconv.bids.`**`convert_sid_bids`**(*subject_id: str*) → str

Shim for stripping any non-BIDS compliant characters within subject_id

Parameters
`subject_id` (*`string`*)

Returns
sid – New subject ID

Return type
string

`heudiconv.bids.`**`find_compatible_fmaps_for_run`**(*json_file: str*, *fmap_groups: dict[str, list[str]]*, *matching_parameters: list[str]*) → dict[str, list[str]]

Finds compatible fmaps for a given run, for populate_intended_for. (Note: It is the responsibility of the calling function to make sure the arguments are OK)

Parameters

- **`json_file`** (*`str`*) – path to the json file
- **`fmap_groups`** (*`dict`*) – key: prefix common to the group value: list of all fmap paths in the group
- **`matching_parameters`** (*`list of str from AllowedFmapParameterMatching`*) – matching_parameters that will be used to match runs

Returns
compatible_fmap_groups – Subset of the fmap_groups which match json_file, according to the matching_parameters. key: prefix common to the group value: list of all fmap paths in the group

Return type
dict

heudiconv.bids.**find_compatible_fmaps_for_session**(*path_to_bids_session: str*, *matching_parameters: list[str]*) → dict[str, dict[str, list[str]]] | None

Finds compatible fmaps for all non-fmap runs in a session. (Note: It is the responsibility of the calling function to make sure the arguments are OK)

Parameters

- **path_to_bids_session** (*str*) – path to the session folder (or to the subject folder, if there are no sessions).
- **matching_parameters** (*list of str from AllowedFmapParameterMatching*) – matching_parameters that will be used to match runs

Returns
compatible_fmap – Dict of compatible_fmaps_groups (values) for each non-fmap run (keys)

Return type
dict

heudiconv.bids.**find_fmap_groups**(*fmap_dir: str*) → dict[str, list[str]]

Finds the different fmap groups in a fmap directory. By groups here we mean fmaps that are intended to go together (with reversed PE polarity, magnitude/phase, etc.)

Parameters
fmap_dir (*str*) – path to the session folder (or to the subject folder, if there are no sessions).

Returns
fmap_groups – key: prefix common to the group (e.g. no "dir" entity, "_phase"/"_magnitude", . . .) value: list of all fmap paths in the group

Return type
dict

heudiconv.bids.**find_subj_ses**(*f_name: str*) → tuple[str | None, str | None]

Given a path to the bids formatted filename parse out subject/session

heudiconv.bids.**get_formatted_scans_key_row**(*dcm_fn: str | Path*) → list[str]

Parameters
dcm_fn (*str*)

Returns
row – [ISO acquisition time, performing physician name, random string]

Return type
list

heudiconv.bids.**get_key_info_for_fmap_assignment**(*json_file: str*, *matching_parameter: str*) → list[Any]

Gets key information needed to assign fmaps to other modalities. (Note: It is the responsibility of the calling function to make sure the arguments are OK)

Parameters

- **json_file** (*str*) – path to the json file
- **matching_parameter** (*str in AllowedFmapParameterMatching*) – matching_parameter that will be used to match runs

Returns
key_info – part of the json file that will need to match between the fmap and the other image

Return type
list

heudiconv.bids.**get_shim_setting**(*json_file: str*) → Any

Gets the "ShimSetting" field from a json_file. If no "ShimSetting" present, return error

Parameters
`json_file` (*str*)

Return type
str with "ShimSetting" value

heudiconv.bids.**maybe_na**(*val: Any*) → str

Return 'n/a' if non-None value represented as str is not empty

Primarily for the consistent use of lower case 'n/a' so 'N/A' and 'NA' are also treated as 'n/a'

heudiconv.bids.**populate_aggregated_jsons**(*path: str*) → None

Aggregate across the entire BIDS dataset `.json`s into top level `.json`s

Top level .json files would contain only the fields which are common to all `subject[/session]/type/*_modality.json`s.

ATM aggregating only for `*_task*_bold.json` files. Only the task- and OPTIONAL _acq- field is retained within the aggregated filename. The other BIDS _key-value pairs are "aggregated over".

Parameters
path (*str*) – Path to the top of the BIDS dataset

heudiconv.bids.**populate_bids_templates**(*path: str*, *defaults: dict[str, Any] | None = None*) → None

Premake BIDS text files with templates

heudiconv.bids.**populate_intended_for**(*path_to_bids_session: str*, *matching_parameters: str | list[str]*, *criterion: str*) → None

Adds the 'IntendedFor' field to the fmap .json files in a session folder. It goes through the session folders and for every json file, it finds compatible_fmaps: fmaps that have the same matching_parameters as the json file (e.g., same 'Shims').

If there are more than one compatible_fmaps, it will use the criterion specified by the user (default: 'Closest' in time).

Because fmaps come in groups (with reversed PE polarity, or magnitude/ phase), we work with fmap_groups.

Parameters

- **path_to_bids_session** (*str*) – path to the session folder (or to the subject folder, if there are no sessions).
- **matching_parameters** (*list of str from AllowedFmapParameterMatching*) – matching_parameters that will be used to match runs
- **criterion** (*str in ['First', 'Closest']*) – matching_parameters that will be used to decide which of the matching fmaps to use

heudiconv.bids.**sanitize_label**(*label: str*) → str

Strips any non-BIDS compliant characters within label

Parameters
label (*string*)

Returns
clean_label – New, sanitized label

Return type
string

`heudiconv.bids.`**`save_scans_key`**(*item: tuple[str, tuple[str, ...], list[str]]*, *bids_files: list[str]*) → None

Parameters

- **`item`**
- **`bids_files`** (*`list of str`*)

`heudiconv.bids.`**`select_fmap_from_compatible_groups`**(*json_file: str*, *compatible_fmap_groups: dict[str, list[str]]*, *criterion: str*) → str | None

Selects the fmap that will be used to correct for distortions in json_file from the compatible fmap_groups list, based on the given criterion (Note: It is the responsibility of the calling function to make sure the arguments are OK)

Parameters

- **`json_file`** (*`str`*) – path to the json file
- **`compatible_fmap_groups`** (*`dict`*) – fmap_groups that are compatible with the specific json_file
- **`criterion`** (*`str in ['First', 'Closest']`*) – matching_parameters that will be used to decide which fmap to use

Returns
selected_fmap_key – key from the compatible_fmap_groups for the selected fmap group

Return type
str

`heudiconv.bids.`**`treat_age`**(*age: str | float | None*) → str | None

Age might encounter 'Y' suffix or be a float

`heudiconv.bids.`**`tuneup_bids_json_files`**(*json_files: list[str]*) → None

Given a list of BIDS .json files, e.g.

Conversion

`heudiconv.convert.`**`add_taskname_to_infofile`**(*infofiles: str | list[str]*) → None

Add the "TaskName" field to json files with _task- entity in the name.

Note: _task- entity could be present not only in functional data but in many other modalities now.

Parameters
`infofiles` (*`list or str`*) – json filenames or a single filename.

`heudiconv.convert.`**`bvals_are_zero`**(*bval_file: str | list*) → bool

Checks if all entries in a bvals file are zero (or 5, for Siemens files).

Parameters
`bval_file` (*`str`*) – file with the bvals

Return type
True if all are all 0 or 5; False otherwise.

`heudiconv.convert.`**`convert`**(*items: list[tuple[str, tuple[str, ...], list[str]]]*, *converter: str*, *scaninfo_suffix: str*, *custom_callable: Callable[[str, tuple[str, ...], list[str]], Any] | None*, *with_prov: bool*, *bids_options: str | None*, *outdir: str*, *min_meta: bool*, *overwrite: bool*, *symlink: bool = True*, *prov_file: str | None = None*, *dcmconfig: str | None = None*, *populate_intended_for_opts: PopulateIntendedForOpts | None = None*) → None

Perform actual conversion (calls to converter etc) given info from heuristic's *infotodict* `heudiconv.convert.`**`convert_dicom`**(*item_dicoms: list[str]*, *bids_options: str | None*, *prefix: str*, *outdir: str*, *tempdirs: TempDirs*, *_symlink: bool*, *overwrite: bool*) → None

Save DICOMs as output (default is by symbolic link)

Parameters

- **item_dicoms** (*list of filenames*) – DICOMs to save
- **bids_options** (*str or None*) – If not None then save to BIDS format. String may be empty or contain bids specific options
- **prefix** (*string*) – Conversion outname
- **outdir** (*string*) – Output directory
- **tempdirs** (*TempDirs instance*) – Object to handle temporary directories created TODO: remove
- **symlink** (*bool*) – Create softlink to DICOMs - if False, create hardlink instead.
- **overwrite** (*bool*) – If True, allows overwriting of previous conversion

`heudiconv.convert.`**`nipype_convert`**(*item_dicoms: list[str]*, *prefix: str*, *with_prov: bool*, *bids_options: str | None*, *tmpdir: str*, *dcmconfig: str | None = None*) → tuple[Node, str | None]

Converts DICOMs grouped from heuristic using Nipype's Dcm2niix interface.

Parameters

- **item_dicoms** (*list*) – DICOM files to convert
- **prefix** (*str*) – Heuristic output path
- **with_prov** (*bool*) – Store provenance information
- **bids_options** (*str or None*) – If not None then output BIDS sidecar JSONs String may contain bids specific options
- **tmpdir** (*str*) – Conversion working directory
- **dcmconfig** (*str, optional*) – JSON file used for additional Dcm2niix configuration

`heudiconv.convert.`**`save_converted_files`**(*res: Node*, *item_dicoms: list[str]*, *bids_options: str | None*, *outtype: str*, *prefix: str*, *outname_bids: str*, *overwrite: bool*) → list[str]

Copy converted files from tempdir to output directory.

Will rename files if necessary.

Parameters

- **res** (*Node*) – Nipype conversion Node with results
- **item_dicoms** (*list*) – Filenames of converted DICOMs
- **bids** (*list or None*) – If not list save to BIDS List may contain bids specific options
- **prefix** (*str*)

Returns
Converted BIDS files

Return type
bids_outfiles

`heudiconv.convert.`**`update_complex_name`**(*metadata: dict[str, Any]*, *filename: str*) → str

Insert *_part-<mag|phase>* entity into filename if data are from a sequence with magnitude/phase part.

Parameters

- **`metadata`** (*`dict`*) – Scan metadata dictionary from BIDS sidecar file.
- **`filename`** (*`str`*) – Incoming filename

Returns
filename – Updated filename with part entity added in appropriate position.

Return type
str

`heudiconv.convert.`**`update_multiecho_name`**(*metadata: dict[str, Any]*, *filename: str*, *echo_times: list[float]*) → str

Insert *_echo-<num>* entity into filename if data are from a multi-echo sequence.

Parameters

- **`metadata`** (*`dict`*) – Scan metadata dictionary from BIDS sidecar file.
- **`filename`** (*`str`*) – Incoming filename
- **`echo_times`** (*`list`*) – List of all echo times from scan. Used to determine the echo *number* (i.e., index) if field is missing from metadata.

Returns
filename – Updated filename with echo entity added, if appropriate.

Return type
str

`heudiconv.convert.`**`update_uncombined_name`**(*metadata: dict[str, Any]*, *filename: str*, *channel_names: list[str]*) → str

Insert *_ch-<num>* entity into filename if data are from a sequence with "save uncombined".

Parameters

- **`metadata`** (*`dict`*) – Scan metadata dictionary from BIDS sidecar file.
- **`filename`** (*`str`*) – Incoming filename
- **`channel_names`** (*`list`*) – List of all channel names from scan. Used to determine the channel *number* (i.e., index) if field is missing from metadata.

Returns
filename – Updated filename with ch entity added, if appropriate.

Return type
str

DICOMS

class heudiconv.dicoms.**CustomSeqinfoT**(**args*, ***kwargs*)

class heudiconv.dicoms.**SeriesID**(*series_number*, *protocol_name*, *file_studyUID*)

file_studyUID: str | None

Alias for field number 2

protocol_name: str

Alias for field number 1

series_number: int

Alias for field number 0

heudiconv.dicoms.**compress_dicoms**(*dicom_list: list[str]*, *out_prefix: str*, *tempdirs:* TempDirs, *overwrite: bool*) → str | None

Archives DICOMs into a tarball

Also tries to do it reproducibly, so takes the date for files and target tarball based on the series time (within the first file)

Parameters

- **dicom_list** (*list of str*) – list of dicom files
- **out_prefix** (*str*) – output path prefix, including the portion of the output file name before .dicom.tgz suffix
- **tempdirs** (TempDirs) – TempDirs object to handle multiple tmpdirs
- **overwrite** (*bool*) – Overwrite existing tarfiles

Returns

filename – Result tarball

Return type

str

heudiconv.dicoms.**create_seqinfo**(*mw: dw.Wrapper*, *series_files: list[str]*, *series_id: str*, *custom_seqinfo:* CustomSeqinfoT | *None = None*) → *SeqInfo*

Generate sequence info

Parameters

- **mw** (*Wrapper*)
- **series_files** (*list*)
- **series_id** (*str*)

heudiconv.dicoms.**embed_dicom_and_nifti_metadata**(*dcmfiles: List[str]*, *niftifile: str*, *infofile: str* | *Path*, *bids_info: Dict[str, Any]* | *None*) → None

Embed metadata from nifti (affine etc) and dicoms into infofile (json)

niftifile should exist. Its affine's orientation information is used while establishing new *NiftiImage* out of dicom stack and together with *bids_info* (if provided) is dumped into json *infofile*

Parameters

- **dcmfiles**
- **niftifile**

- **infofile**
- **bids_info** (*dict*) – Additional metadata to be embedded. *infofile* is overwritten if exists, so here you could pass some metadata which would overload (at the first level of the dict structure, no recursive fancy updates) what is obtained from nifti and dicoms

`heudiconv.dicoms.`**`embed_metadata_from_dicoms`**(*bids_options: str | None*, *item_dicoms: list[str]*, *outname: str*, *outname_bids: str*, *prov_file: str | None*, *scaninfo: str*, *tempdirs: TempDirs*, *with_prov: bool*) → None

Enhance sidecar information file with more information from DICOMs

Parameters

- **bids_options**
- **item_dicoms**
- **outname**
- **outname_bids**
- **prov_file**
- **scaninfo**
- **tempdirs**
- **with_prov**

`heudiconv.dicoms.`**`get_datetime_from_dcm`**(*dcm_data: FileDataset*) → datetime | None

Extract datetime from filedataset, or return None is no datetime information found.

Parameters

dcm_data (*dcm.FileDataset*) – DICOM with header, e.g., as ready by pydicom.dcmread. Objects with __getitem__ and have those keys with values properly formatted may also work

Returns

One of several datetimes that are related to when the scan occurred, or None if no datetime can be found

Return type

Optional[datetime.datetime]

Notes

The following fields are checked in order

1. AcquisitionDate & AcquisitionTime (0008,0022); (0008,0032)
2. AcquisitionDateTime (0008,002A);
3. SeriesDate & SeriesTime (0008,0021); (0008,0031)

`heudiconv.dicoms.`**`get_reproducible_int`**(*dicom_list: list[str]*) → int

Get integer that can be used to reproducibly sort input DICOMs, which is based on when they were acquired.

Parameters

dicom_list (*list[str]*) – Paths to existing DICOM files

Returns

An integer relating to when the DICOM was acquired

Return type

int

Raises
`AssertionError` –

Notes

1. **When date and time for can be read (see `get_datetime_from_dcm()`), return**
   that value as time in seconds since epoch (i.e., Jan 1 1970).
2. **In cases where a date/time/datetime is not available (e.g., anonymization stripped this info), return**
   epoch + AcquisitionNumber (in seconds), which is AcquisitionNumber as an integer
3. If 1 and 2 are not possible, then raise AssertionError and provide message about missing information

Cases are based on only the first element of the dicom_list.

`heudiconv.dicoms.`**`group_dicoms_into_seqinfos`**(*files: list[str]*, *grouping: str*, *file_filter: Callable[[str], Any] | None = None*, *dcmfilter: Callable[[Dataset], Any] | None = None*, *flatten: Literal[False] = False*, *custom_grouping: str | Callable[[list[str], Callable[[dcm.dataset.Dataset], Any] | None, type[SeqInfo]], dict[SeqInfo, list[str]]] | None = None*, *custom_seqinfo: CustomSeqinfoT | None = None*) → dict[str | None, dict[*SeqInfo*, list[str]]]

`heudiconv.dicoms.`**`group_dicoms_into_seqinfos`**(*files: list[str]*, *grouping: str*, *file_filter: Callable[[str], Any] | None = None*, *dcmfilter: Callable[[Dataset], Any] | None = None*, ***, *flatten: Literal[True]*, *custom_grouping: str | Callable[[list[str], Callable[[dcm.dataset.Dataset], Any] | None, type[SeqInfo]], dict[SeqInfo, list[str]]] | None = None*, *custom_seqinfo: CustomSeqinfoT | None = None*) → dict[*SeqInfo*, list[str]]

Process list of dicoms and return seqinfo and file group *seqinfo* contains per-sequence extract of fields from DICOMs which will be later provided into heuristics to decide on filenames

Parameters

- **files** (*list of str*) – List of files to consider
- **grouping** (*{'studyUID', 'accession_number', 'all', 'custom'}*) – How to group DICOMs for conversion. If 'custom', see *custom_grouping* parameter.
- **file_filter** (*callable, optional*) – Applied to each item of filenames. Should return True if file needs to be kept, False otherwise.
- **dcmfilter** (*callable, optional*) – If called on dcm_data and returns True, it is used to set series_id
- **flatten** (*bool, optional*) – Creates a flattened *seqinfo* with corresponding DICOM files. True when invoked with *dicom_dir_template*.
- **custom_grouping** (*str or callable, optional*) – grouping key defined within heuristic. Can be a string of a DICOM attribute, or a method that handles more complex groupings.
- **custom_seqinfo** (*callable, optional*) – A callable which will be provided MosaicWrapper giving possibility to extract any custom DICOM metadata of interest.

Returns

- **seqinfo** (*list of list*) – *seqinfo* is a list of info entries per each sequence (some entry there defines a key for *filegrp*)
- **filegrp** (*dict*) – *filegrp* is a dictionary with files grouped per each sequence

`heudiconv.dicoms.`**`parse_private_csa_header`**(*dcm_data: Dataset*, *_public_attr: str*, *private_attr: str*, *default: str | None = None*) → str

Parses CSA header in cases where value is not defined publicly

Parameters

- **`dcm_data`** (`pydicom Dataset object`) – DICOM metadata
- **`public_attr`** (`string`) – non-private DICOM attribute
- **`private_attr`** (`string`) – private DICOM attribute
- **`(optional)`** (`default`) – default value if private_attr not found

Returns
val (default – private attribute value or default

Return type
empty string)

`heudiconv.dicoms.`**`validate_dicom`**(*fl: str*, *dcmfilter: Callable[[Dataset], Any] | None*) → tuple[Wrapper, tuple[int, str], str | None] | None

Parse DICOM attributes. Returns None if not valid.

Parsing

`heudiconv.parser.`**`find_files`**(*regex: str*, *topdir: list[str] | tuple[str, ...] | str = '.'*, *exclude: str | None = None*, *exclude_vcs: bool = True*, *dirs: bool = False*) → Iterator[str]

Generator to find files matching regex

Parameters

- **`regex`** (`string`)
- **`exclude`** (`string, optional`) – Matches to exclude
- **`exclude_vcs`** – If True, excludes commonly known VCS subdirectories. If string, used as regex to exclude those files (regex: */.(?:git|gitattributes|svn|bzr|hg)(?:/|$)*)
- **`topdir`** (`string or list, optional`) – Directory where to search
- **`dirs`** (`bool, optional`) – Either to match directories as well as files

`heudiconv.parser.`**`get_extracted_dicoms`**(*fl: Iterable[str]*) → ItemsView[str | None, list[str]]

Given a collection of files and/or directories, list out and possibly extract the contents from archives.

Parameters
`fl` – Files (possibly archived) to process.

Returns
The absolute paths of (possibly newly extracted) files.

Return type
ItemsView[str | None, list[str]]

Notes

For 'classical' heudiconv, if multiple archives are provided, they correspond to different sessions, so here we would group into sessions and return pairs *sessionid*, *files* with *sessionid* being None if no "sessions" detected for that file or there was just a single tarball in the list.

When contents of fl appear to be an unpackable archive, the contents are extracted into utils.TempDirs(f'heudiconvDCM') and the mode of all extracted files is set to 700.

When contents of fl are a list of unarchived files, they are treated as a single session.

When contents of fl are a list of unarchived and archived files, the unarchived files are grouped into a single session (key: None). If there is only one archived file, the contents of that file are grouped with the unarchived file. If there are multiple archived files, they are grouped into separate sessions.

`heudiconv.parser.`**`get_study_sessions`**(*dicom_dir_template: str | None*, *files_opt: list[str] | None*, *heuristic: ModuleType*, *outdir: str*, *session: str | None*, *sids: list[str] | None*, *grouping: str = 'studyUID'*) → dict[*StudySessionInfo*, list[str] | dict[*SeqInfo*, list[str]]]

Sort files or dicom seqinfos into study_sessions.

study_sessions put together files for a single session of a subject in a study. Two major possible workflows:

- if dicom_dir_template provided – doesn't pre-load DICOMs and just loads files pointed by each subject and possibly sessions as corresponding to different tarballs.
- if files_opt is provided, sorts all DICOMs it can find under those paths

Batch Queuing

`heudiconv.queue.`**`clean_args`**(*hargs: list[str]*, *iterarg: str*, *iteridx: int*) → list[str]

Filters arguments for batch submission.

Parameters

- **hargs** (*list*) – Command-line arguments
- **iterarg** (*str*) – Multi-argument to index (*subjects* OR *files*)
- **iteridx** (*int*) – *iterarg* index to submit

Returns

cmdargs – Filtered arguments for batch submission

Return type

list

Example

```
>>> from heudiconv.queue import clean_args
>>> cmd = ['heudiconv', '-d', '/some/{subject}/path',
...                     '-q', 'SLURM',
...                     '-s', 'sub-1', 'sub-2', 'sub-3', 'sub-4']
>>> clean_args(cmd, 'subjects', 0)
['heudiconv', '-d', '/some/{subject}/path', '-s', 'sub-1']
```

`heudiconv.queue.queue_conversion`(*queue: str*, *iterarg: str*, *iterables: int*, *queue_args: str | None = None*) → None

Write out conversion arguments to file and submit to a job scheduler. Parses *sys.argv* for heudiconv arguments.

Parameters

- **queue** (*string*) – Batch scheduler to use
- **iterarg** (*str*) – Multi-argument to index (*subjects* OR *files*)
- **iterables** (*int*) – Number of *iterarg* arguments
- **queue_args** (*string (optional)*) – Additional queue arguments for job submission

Utility

Utility objects and functions

class `heudiconv.utils.File`(*name: str*, *executable: bool = False*)

Helper for a file entry in the create_tree/@with_tree

It allows to define additional settings for entries

class `heudiconv.utils.SeqInfo`(*total_files_till_now*, *example_dcm_file*, *series_id*, *dcm_dir_name*, *series_files*, *unspecified*, *dim1*, *dim2*, *dim3*, *dim4*, *TR*, *TE*, *protocol_name*, *is_motion_corrected*, *is_derived*, *patient_id*, *study_description*, *referring_physician_name*, *series_description*, *sequence_name*, *image_type*, *accession_number*, *patient_age*, *patient_sex*, *date*, *series_uid*, *time*, *custom*)

TE: float

Alias for field number 11

TR: float

Alias for field number 10

accession_number: str

Alias for field number 21

custom: Hashable | None

Alias for field number 27

date: str | None

Alias for field number 24

dcm_dir_name: str

Alias for field number 3

dim1: int

Alias for field number 6

dim2: int

Alias for field number 7

dim3: int

Alias for field number 8

dim4: int

Alias for field number 9

example_dcm_file: str
: Alias for field number 1

image_type: tuple[str, ...]
: Alias for field number 20

is_derived: bool
: Alias for field number 14

is_motion_corrected: bool
: Alias for field number 13

patient_age: str | None
: Alias for field number 22

patient_id: str | None
: Alias for field number 15

patient_sex: str | None
: Alias for field number 23

protocol_name: str
: Alias for field number 12

referring_physician_name: str
: Alias for field number 17

sequence_name: str
: Alias for field number 19

series_description: str
: Alias for field number 18

series_files: int
: Alias for field number 4

series_id: str
: Alias for field number 2

series_uid: str | None
: Alias for field number 25

study_description: str
: Alias for field number 16

time: str | None
: Alias for field number 26

total_files_till_now: int
: Alias for field number 0

unspecified: str
: Alias for field number 5

class heudiconv.utils.**StudySessionInfo**(*locator*, *session*, *subject*)

locator: str | None
: Alias for field number 0

session: str | None

Alias for field number 1

subject: str | None

Alias for field number 2

class heudiconv.utils.**TempDirs**

A helper to centralize handling and cleanup of dirs

heudiconv.utils.**anonymize_sid**(*sid: AnyStr*, *anon_sid_cmd: str*) → AnyStr

Raises

`ValueError` – if script returned an empty string (after whitespace stripping), or output with multiple words/lines.

heudiconv.utils.**assure_no_file_exists**(*path: str | Path*) → None

Check if file or symlink (git-annex?) exists, and if so – remove

heudiconv.utils.**clear_temp_dicoms**(*item_dicoms: list[str]*) → None

Ensures DICOM temporary directories are safely cleared

heudiconv.utils.**create_file_if_missing**(*filename: str*, *content: str*) → bool

Create file if missing, so we do not override any possibly introduced changes

heudiconv.utils.**create_tree**(*path: str*, *tree: Sequence[Tuple[str | File, str | TreeSpec | dict]] | Mapping[str, str | TreeSpec | dict]*, *archives_leading_dir: bool = True*) → None

Given a list of tuples (name, load) or a dict create such a tree

if load is a tuple or a dict itself – that would create either a subtree or an archive with that content and place it into the tree if name ends with .tar.gz

heudiconv.utils.**docstring_parameter**(**sub: str*) → Callable[[T], T]

Borrowed from https://stackoverflow.com/a/10308363/6145776

heudiconv.utils.**get_datetime**(*date: str*, *time: str*, ***, *microseconds: bool = True*) → str

Combine date and time from dicom to isoformat.

Parameters

- **date** (`str`) – Date in YYYYMMDD format.
- **time** (`str`) – Time in either HHMMSS.ffffff format or HHMMSS format.
- **microseconds** (`bool, optional`) – Either to include microseconds in the output

Returns

datetime_str – Combined date and time in ISO format, with microseconds as if fraction was provided in 'time', and 'microseconds' was True.

Return type

str

heudiconv.utils.**get_known_heuristic_names**() → list[str]

Return a list of heuristic names present under heudiconv/heuristics

heudiconv.utils.**get_typed_attr**(*obj: Any*, *attr: str*, *_type: type[T]*, *default: V*) → T | V

heudiconv.utils.**get_typed_attr**(*obj: Any*, *attr: str*, *_type: type[T]*, *default: None = None*) → T | None

Typecasts an object's named attribute. If the attribute cannot be converted, the default value is returned instead.

Parameters

- **obj** (*Object*)
- **attr** (*Attribute*)
- **_type** (*Type*)
- **default** (*value, optional*)

`heudiconv.utils.`**`is_readonly`**(*path: str*) → bool

Return True if it is a fully read-only file (dereferences the symlink)

`heudiconv.utils.`**`json_dumps`**(*json_obj: Any*, *indent: int = 2*, *sort_keys: bool = True*) → str

Unified (default indent and sort_keys) invocation of json.dumps

`heudiconv.utils.`**`json_dumps_pretty`**(*j: Any*, *indent: int = 2*, *sort_keys: bool = True*) → str

Given a json structure, pretty print it by colliding numeric arrays into a line.

If resultant structure differs from original – throws exception

`heudiconv.utils.`**`load_heuristic`**(*heuristic: str*) → ModuleType

Load heuristic from the file, return the module

`heudiconv.utils.`**`load_json`**(*filename: str | Path*, *retry: int = 0*) → Any

Load data from a json file

Parameters

- **filename** (*str or Path*) – Filename to load data from.
- **retry** (*int, optional*) – Number of times to retry opening/loading the file in case of failure. Code will sleep for 0.1 seconds between retries. Could be used in code which is not sensitive to order effects (e.g. like populating bids templates where the last one to do it, would make sure it would be the correct/final state).

Returns
data

Return type
dict

`heudiconv.utils.`**`remove_prefix`**(*s: str*, *pre: str*) → str

Remove prefix from the beginning of the string

Parameters

- **s** (*str*)
- **pre** (*str*)

Returns
s – string with "pre" removed from the beginning (if present)

Return type
str

`heudiconv.utils.`**`remove_suffix`**(*s: str*, *suf: str*) → str

Remove suffix from the end of the string

Parameters

- **s** (*str*)
- **suf** (*str*)

Returns
s – string with "suf" removed from the end (if present)

Return type
str

`heudiconv.utils.`**`safe_copyfile`**(*src: str*, *dest: str*, *overwrite: bool = False*) → None

Copy file but blow if destination name already exists

`heudiconv.utils.`**`safe_movefile`**(*src: str*, *dest: str*, *overwrite: bool = False*) → None

Move file but blow if destination name already exists

`heudiconv.utils.`**`save_json`**(*filename: str | Path*, *data: Any*, *indent: int = 2*, *sort_keys: bool = True*, *pretty: bool = False*) → None

Save data to a json file

Parameters

- **filename** (*str or Path*) – Filename to save data in.
- **data** (*dict*) – Dictionary to save in json file.
- **indent** (*int, optional*)
- **sort_keys** (*bool, optional*)
- **pretty** (*bool, optional*)

`heudiconv.utils.`**`set_readonly`**(*path: str*, *read_only: bool = True*) → int

Make file read only or writeable while preserving "access levels"

So if file was not readable by others, it should remain not readable by others.

Parameters

- **path** (*str*)
- **read_only** (*bool, optional*) – If True (default) - would make it read-only. If False, would make it writeable for levels where it is readable

`heudiconv.utils.`**`slim_down_info`**(*j: dict[str, Any]*) → dict[str, Any]

Given an aggregated info structure, removes excessive details

Such as CSA fields, and SourceImageSequence which on Siemens files could be huge and not providing any additional immediately usable information. If needed, could be recovered from stored DICOMs

`heudiconv.utils.`**`strptime_micr`**(*date_string: str*, *fmt: str*) → datetime

Decorate strptime while supporting optional [.%f] in the format at the end

Parameters

- **date_string** (*str*) – Date string to parse
- **fmt** (*str*) – Format string. If it ends with [.%f], we keep it if date_string ends with '.d+' regex and not if it does not.

`heudiconv.utils.`**`treat_infofile`**(*filename: str*) → None

Tune up generated .json file (slim down, pretty-print for humans).

`heudiconv.utils.`**`update_json`**(*json_file: str | Path*, *new_data: dict[str, Any]*, *pretty: bool = False*) → None

Adds a given field (and its value) to a json file

Parameters

- **json_file** (*str or Path*) – path for the corresponding json file
- **new_data** (*dict*) – pair of "key": "value" to add to the json file
- **pretty** (*bool*) – argument to be passed to save_json

PYTHON MODULE INDEX

INDEX