datalad_neuroimaging Documentation

Release 0.1.0

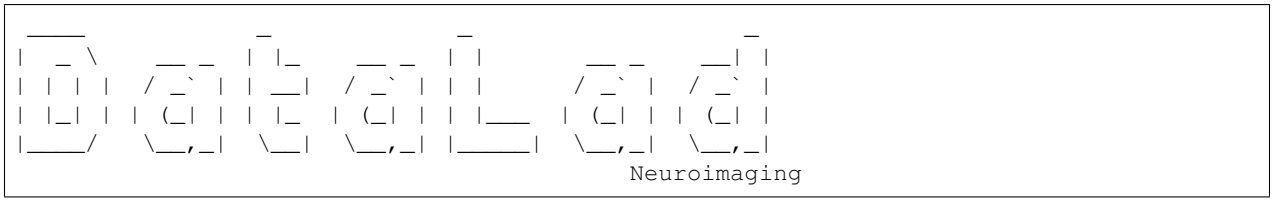
DataLad team

Feb 18, 2021

Contents

CHAPTER 1

Change log



This is a high level and scarce summary of the changes between releases. We would recommend to consult log of the DataLad git repository for more details.

1.1 0.3.1 (Jun 3, 2020) – Uncles aren't always nice either

- be compatible with pydicom 2.0.0

1.2 0.3.0 (Feb 26, 2020) – . . . and parents become grumpier

- DataLad 0.12 series is now minimal supported

1.3 0.2.4 (Feb 05, 2020) – Kids are growing

Minor bugfix release to account for changes in pybids and datalad

- pybids:
  - demand pybids >=0.9.2
  - account for new field "extension" provided by BIDS now
  - use get_dataset_description if available

- pandas:
  - use .iloc instead of deprecated .ix
- datalad
  - use -d^ instead of deprecated (in 0.12) -S in an example script

1.4 0.2.3 (May 24, 2019) – Old is not bad

Minor quick bugfix release to demand pybids < 0.9 since we are not yet fully ready for its full glory

1.5 0.2.2 (May 20, 2019) – It was like that way before!

Minor bugfix release - revert back to old ways of installing package data so test data and procedures get properly installed

1.6 0.2.1 (May 17, 2019) – Why wasn't it that way before?

- include a procedure 'cfg_bids' to configure BIDS datasets
- fix several issues with troublesome dependency declarations

1.7 0.2.0 (Feb 09, 2019) – Am I compatible with you honey?

- Make compatible with (and demand) pybids 0.7.{0,1}. 0.7.0 introduced change of terms: modality -> suffix, and type -> datatype, which would now require to either reaggregate all previous metadata or somehow fixup in-place existing metadata files. And for 0.7.1 workaround was added to not return `suffix` at least when participants.tsv was queried.
- Make compatible with pydicom 1.0 in treatment of MultiValue
- Refactorings:
  - tests:
    - use `export_archive` instead of plain `tarfile.open`
    - make compatible with recent (0.11.x) DataLad which now extract annex keys etc

1.8 0.1.5 (Sep 28, 2018) – BIDS robustness

- Assorted improvements of the BIDS metadata extractor performance on datasets in the wild.

1.9 0.1.4 (Aug 02, 2018) – PyBIDS

- Fixed compatibility with pybids 0.6.4 and now demand it as the minimal PyBIDS version

1.10 0.1 (Apr 28, 2018) – The Release

1.10.1 Major refactoring and deprecations

- This is the first separate release of DataLad's neuroimaging functionality as an extension module.
- Metadata
  - BIDS metadata now uniformly refers to subjects and participants using the metadata key 'subject'

1.10.2 Enhancements and new features

- Extractors now report progress (with DataLad 0.10+)
- BIDS participant metadata is now read via pybids

1.10.3 Fixes

- Fix issue with unicode characters in BIDS metadata
- DICOM metadata now also contains the 'PatientName' field that was previously excluded due to a too restrictive data type filter

CHAPTER 2

Acknowledgments

DataLad development is being performed as part of a US-German collaboration in computational neuroscience (CRCNS) project "DataGit: converging catalogues, warehouses, and deployment logistics into a federated 'data distribution'" (Halchenko/Hanke), co-funded by the US National Science Foundation (NSF 1429999) and the German Federal Ministry of Education and Research (BMBF 01GQ1411). Additional support is provided by the German federal state of Saxony-Anhalt and the European Regional Development Fund (ERDF), Project: Center for Behavioral Brain Sciences, Imaging Platform

DataLad is built atop the git-annex software that is being developed and maintained by Joey Hess.

CHAPTER 3

Demos

3.1 Data management use cases

3.1.1 An automatically reproducible neuroimaging data analysis

Scientific studies should be reproducible, and with the increasing accessibility of data, there is not much excuse for lack of reproducibility anymore.

DataLad can help with the technical aspects of reproducible science...

It always starts with a dataset

```
~ % datalad create demo
[INFO   ] Creating a new annex repo at /demo/demo
create(ok): /demo/demo (dataset)
~ % cd demo
```

For this demo we are using two public brain imaging datasets that were published on OpenFMRI.org, and are available from DataLad's datasets.datalad.org

```
~/demo % datalad install -d . -s ///openfmri/ds000001 inputs/ds000001
[INFO   ] Cloning http://datasets.datalad.org/openfmri/ds000001 to '/demo/demo/inputs/
↪ds000001'
add(ok): inputs/ds000001 (dataset) [added new subdataset]
add(notneeded): inputs/ds000001 (dataset) [nothing to add from /demo/demo/inputs/
↪ds000001]
add(notneeded): .gitmodules (file) [already included in the dataset]
save(ok): /demo/demo (dataset)
[INFO   ] access to dataset sibling "datalad" not auto-enabled, enable with:
|             datalad siblings -d "/demo/demo/inputs/ds000001" enable -s datalad
install(ok): inputs/ds000001 (dataset)
action summary:
  add (notneeded: 2, ok: 1)
```

(continues on next page)

(continued from previous page)

```
  install (ok: 1)
  save (ok: 1)
```

BTW: '///' is just short for http://datasets.datalad.org

```
~/demo % datalad install -d . -s ///openfmri/ds000002 inputs/ds000002
[INFO   ] Cloning http://datasets.datalad.org/openfmri/ds000002 to '/demo/demo/inputs/
↪ds000002'
add(ok): inputs/ds000002 (dataset) [added new subdataset]
add(notneeded): inputs/ds000002 (dataset) [nothing to add from /demo/demo/inputs/
↪ds000002]
add(notneeded): .gitmodules (file) [already included in the dataset]
save(ok): /demo/demo (dataset)
[INFO   ] access to dataset sibling "datalad" not auto-enabled, enable with:
|            datalad siblings -d "/demo/demo/inputs/ds000002" enable -s datalad
install(ok): inputs/ds000002 (dataset)
action summary:
  add (notneeded: 2, ok: 1)
  install (ok: 1)
  save (ok: 1)
```

Both datasets are now registered as subdatasets, and their precise versions are on record

```
~/demo % datalad --output-format '{path}: {revision_descr}' subdatasets
[WARNING] Result rendering failed for: {'status': 'ok', 'gitmodule_name': u'inputs/
↪ds000001', 'parentds': '/demo/demo', 'gitmodule_url': u'http://datasets.datalad.org/
↪openfmri/ds000001/.git', 'action': 'subdataset', 'path': '/demo/demo/inputs/ds000001
↪', 'type': 'dataset', 'refds': '/demo/demo', 'revision':
↪'f47099a5124e8f619f763f44f70e1faf5154d41a'} [u'revision_descr' [base.py:<lambda>
↪:412]]
[WARNING] Result rendering failed for: {'status': 'ok', 'gitmodule_name': u'inputs/
↪ds000002', 'parentds': '/demo/demo', 'gitmodule_url': u'http://datasets.datalad.org/
↪openfmri/ds000002/.git', 'action': 'subdataset', 'path': '/demo/demo/inputs/ds000002
↪', 'type': 'dataset', 'refds': '/demo/demo', 'revision':
↪'e1b7df06da8dd8f1d8802d699d9ad7781fad8bb6'} [u'revision_descr' [base.py:<lambda>
↪:412]]
```

However, very little data were actually downloaded (the full datasets are several gigabytes in size):

```
~/demo % du -sh inputs/
20M  inputs/
```

DataLad datasets are fairly lightweight in size, they only contain pointers to data and history information in their minimal form.

Both datasets contain brain imaging data, and are compliant with the BIDS standard. This makes it really easy to locate particular images and perform analysis across datasets.

Here we will use a small script that performs 'brain extraction' using FSL as a stand-in for a full analysis pipeline

```
~/demo % mkdir code
~/demo % cat << EOT > code/brain_extraction.sh
> # enable FSL
> . /etc/fsl/5.0/fsl.sh
>
> # obtain all inputs
> datalad get \$@
```

(continues on next page)

(continued from previous page)

```
> # perform brain extraction
> count=1
> for nifti in \$@; do
>   subdir="sub-\$(printf %03d \$count)"
>   mkdir -p \$subdir
>   echo "Processing \$nifti"
>   bet \$nifti \$subdir/anat -m
>   count=\$((count + 1))
> done
> EOT
```

Note that this script uses the 'datalad get' command which automatically obtains the required files from their remote source – we will see this in action shortly

We are saving this script in the dataset. This way we will know exactly which code was used for the analysis. Also, we track this code file with Git, so we can see more easily how it was edited over time.

```
~/demo % datalad add code -m "Brain extraction script" --to-git
add(ok): /demo/demo/code/brain_extraction.sh (file) [non-large file; adding content␣
↪to git repository]
add(ok): /demo/demo/code (directory)
save(ok): /demo/demo (dataset)
action summary:
  add (ok: 2)
  save (ok: 1)
```

In addition, we will "tag" this state of the dataset. This is optional, but it can help to identify important milestones more easily

```
~/demo % datalad save --version-tag setup_done
save(ok): /demo/demo (dataset)
```

Now we can run our analysis code to produce results. However, instead of running it directly, we will run it with DataLad – this will automatically create a record of exactly how this script was executed

For this demo we will just run it on the structural images of the first subject from each dataset. The uniform structure of the datasets makes this very easy. Of course we could run it on all subjects; we are simply saving some time for this demo. While the command runs, you should notice a few things:

1) We run this command with 'bash -e' to stop at any failure that may occur

2) You'll see the required data files being obtained as they are needed – and only those that are actually required will be downloaded

```
~/demo % datalad run bash -e code/brain_extraction.sh inputs/ds*/sub-01/anat/sub-01_
↪T1w.nii.gz
[INFO   ] == Command start (output follows) =====
get(ok): /demo/demo/inputs/ds000001/sub-01/anat/sub-01_T1w.nii.gz (file)
get(ok): /demo/demo/inputs/ds000002/sub-01/anat/sub-01_T1w.nii.gz (file)
action summary:
  get (ok: 2)
Processing inputs/ds000001/sub-01/anat/sub-01_T1w.nii.gz
Processing inputs/ds000002/sub-01/anat/sub-01_T1w.nii.gz
[INFO   ] == Command exit (modification check follows) =====
add(ok): sub-002/anat.nii.gz (file)
add(ok): sub-001/anat.nii.gz (file)
add(ok): sub-002/anat_mask.nii.gz (file)
```

(continues on next page)

(continued from previous page)

```
add(ok): sub-001/anat_mask.nii.gz (file)
save(ok): /demo/demo (dataset)
action summary:
  add (ok: 4)
  save (ok: 1)
```

The analysis step is done, all generated results were saved in the dataset. All changes, including the command that caused them are on record

```
~/demo % git show --stat
commit 7607ddef8c03dc5516869f1e35025083772efc5a (HEAD -> master)
Author: DataLad Demo <demo@datalad.org>
Date:   Fri Mar 16 08:26:11 2018 +0100

    [DATALAD RUNCMD] bash -e code/brain_extraction.sh inputs/...

    === Do not change lines below ===
    {
     "pwd": ".",
     "cmd": [
      "bash",
      "-e",
      "code/brain_extraction.sh",
      "inputs/ds000001/sub-01/anat/sub-01_T1w.nii.gz",
      "inputs/ds000002/sub-01/anat/sub-01_T1w.nii.gz"
     ],
     "exit": 0,
     "chain": []
    }
    ^^^ Do not change lines above ^^^

 sub-001/anat.nii.gz      | 1 +
 sub-001/anat_mask.nii.gz | 1 +
 sub-002/anat.nii.gz      | 1 +
 sub-002/anat_mask.nii.gz | 1 +
 4 files changed, 4 insertions(+)
```

DataLad has enough information stored to be able to re-run a command.

On command exit, it will inspect the results and save them again, but only if they are different.

In our case, the re-run yields bit-identical results, hence nothing new is saved.

```
~/demo % datalad rerun
unlock(ok): sub-001/anat.nii.gz (file)
unlock(ok): sub-001/anat_mask.nii.gz (file)
unlock(ok): sub-002/anat.nii.gz (file)
unlock(ok): sub-002/anat_mask.nii.gz (file)
[INFO   ] == Command start (output follows) =====
get(notneeded): /demo/demo/inputs/ds000001/sub-01/anat/sub-01_T1w.nii.gz (file)␣
↪[already present]
get(notneeded): /demo/demo/inputs/ds000002/sub-01/anat/sub-01_T1w.nii.gz (file)␣
↪[already present]
action summary:
  get (notneeded: 2)
Processing inputs/ds000001/sub-01/anat/sub-01_T1w.nii.gz
Processing inputs/ds000002/sub-01/anat/sub-01_T1w.nii.gz
```

(continues on next page)

(continued from previous page)

```
[INFO   ] == Command exit (modification check follows) =====
add(ok): sub-002/anat.nii.gz (file)
add(ok): sub-001/anat.nii.gz (file)
add(ok): sub-002/anat_mask.nii.gz (file)
add(ok): sub-001/anat_mask.nii.gz (file)
save(notneeded): /demo/demo (dataset)
action summary:
  add (ok: 4)
  save (notneeded: 1)
  unlock (ok: 4)
```

Now that we are done, and have checked that we can reproduce the results ourselves, we can clean up

DataLad can easily verify if any part of our input dataset was modified since we configured our analysis

```
~/demo % datalad diff --revision setup_done inputs
```

Nothing was changed.

With DataLad with don't have to keep those inputs around – without losing the ability to reproduce an analysis.

Let's uninstall them – checking the size on disk before and after

```
~/demo % du -sh
32M  .
~/demo % datalad uninstall inputs/*
drop(ok): /demo/demo/inputs/ds000002/sub-01/anat/sub-01_T1w.nii.gz (file) [checking␣
↪http://openneuro.s3.amazonaws.com/ds000002/ds000002_R2.0.0/uncompressed/sub-01/anat/
↪sub-01_T1w.nii.gz?versionId=vXK2.bQ360phhPqbVV_n6RMYqaWAy4Dg...]
drop(ok): /demo/demo/inputs/ds000002 (directory)
uninstall(ok): /demo/demo/inputs/ds000002 (dataset)
drop(ok): /demo/demo/inputs/ds000001/sub-01/anat/sub-01_T1w.nii.gz (file) [checking␣
↪http://openneuro.s3.amazonaws.com/ds000001/ds000001_R1.1.0/uncompressed/sub001/
↪anatomy/highres001.nii.gz?versionId=8TJ17W9WInNkQPdiQ9vS7wo8ZJ9llF80...]
drop(ok): /demo/demo/inputs/ds000001 (directory)
uninstall(ok): /demo/demo/inputs/ds000001 (dataset)
action summary:
  drop (ok: 4)
  uninstall (ok: 2)
~/demo % du -sh .
3.0M .
```

All inputs are gone…

```
~/demo % ls inputs/*
inputs/ds000001:

inputs/ds000002:
```

Only the remaining data (our code and the results) need to be kept and require a backup for long term archival. Everything else can be re-obtained as needed, when needed.

As DataLad knows everything needed about the inputs, including where to get the right version, we can re-run the analysis with a single command. Watch how DataLad re-obtains all required data, re-runs the code, and checks that none of the results changed and need saving

```
~/demo % datalad rerun
unlock(ok): sub-001/anat.nii.gz (file)
```

(continues on next page)

(continued from previous page)

```
unlock(ok): sub-001/anat_mask.nii.gz (file)
unlock(ok): sub-002/anat.nii.gz (file)
unlock(ok): sub-002/anat_mask.nii.gz (file)
[INFO   ] == Command start (output follows) =====
[INFO   ] Cloning http://datasets.datalad.org/openfmri/ds000001/.git to '/demo/demo/
↪inputs/ds000001'
[INFO   ] access to dataset sibling "datalad" not auto-enabled, enable with:
|             datalad siblings -d "/demo/demo/inputs/ds000001" enable -s datalad
install(ok): /demo/demo/inputs/ds000001 (dataset) [Installed subdataset in order to␣
↪get /demo/demo/inputs/ds000001/sub-01/anat/sub-01_T1w.nii.gz]
[INFO   ] Cloning http://datasets.datalad.org/openfmri/ds000002/.git to '/demo/demo/
↪inputs/ds000002'
[INFO   ] access to dataset sibling "datalad" not auto-enabled, enable with:
|             datalad siblings -d "/demo/demo/inputs/ds000002" enable -s datalad
install(ok): /demo/demo/inputs/ds000002 (dataset) [Installed subdataset in order to␣
↪get /demo/demo/inputs/ds000002/sub-01/anat/sub-01_T1w.nii.gz]
get(ok): /demo/demo/inputs/ds000001/sub-01/anat/sub-01_T1w.nii.gz (file)
get(ok): /demo/demo/inputs/ds000002/sub-01/anat/sub-01_T1w.nii.gz (file)
action summary:
  get (ok: 2)
  install (ok: 2)
Processing inputs/ds000001/sub-01/anat/sub-01_T1w.nii.gz
Processing inputs/ds000002/sub-01/anat/sub-01_T1w.nii.gz
[INFO   ] == Command exit (modification check follows) =====
add(ok): sub-002/anat.nii.gz (file)
add(ok): sub-001/anat.nii.gz (file)
add(ok): sub-002/anat_mask.nii.gz (file)
add(ok): sub-001/anat_mask.nii.gz (file)
save(notneeded): /demo/demo (dataset)
action summary:
  add (ok: 4)
  save (notneeded: 1)
  unlock (ok: 4)
```

Reproduced!

This dataset could now be published and enable anyone to replicate the exact same analysis. Public data for the win!

3.1.2 Creating a "new" derived dataset for Nipype workshop

For more information about the workshop, please visit http://nipy.org/workshops/2017-03-boston/index.html . As we will present git-annex and DataLad, I have decided to prepare a DataLad dataset from the tarball Satrajit Ghosh has shared URL to. That tarball is a trimmed down version of OpenfMRI ds000114 dataset which we also crawl and provide as a DataLad dataset.

Deriving (cloning) a dataset

I have decided to first create a superdataset for the workshop (may be more datasets besides ds114 will be later) outside of our master superdataset which we distribute from http://datasets.datalad.org . So I just created a new dataset in a random directory. I wanted our ds114 dataset to be "derived" from original openfmri ds000114 dataset so we could readily reuse all the knowledge git-annex has about where files might be coming from. To achieve that I have just installed existing ds000114 dataset into our superdataset:

```
# Create dataset
datalad create --no-annex nipype-workshop-2017
cd nipype-workshop-2017
datalad install -d . ///openfmri/ds000114
cd ds000114
```

To make both original and this derived dataset accessible from the same repo I generated a detached branch (since I did not know at that point on which version of openfmri it is based on). And then added content from the tarball available from the OSF.

A few tricky points which you do not necessarily would run into in a more typical workflow:

- since branch is detached, it would be empty to start with, but we want to preserve the settings within .gitattributes (such as git annex backend). I could have just *git add .gitattributes* after *checkout –orphan* but I haven't thought about that and created a new file from scratch.
- Original openfmri dataset did not have settings within .gitattributes to add all text files straight into git, so I have added those settings within a new .gitattributes . For more about settings git-annex understands as to what files should it handled (*largefiles*) or otherwise just pass to git to handle see https://git-annex.branchable.com/tips/largefiles/ .

```
cat .gitattributes                    # sneak preview
git checkout --orphan nipype_test1    # generate a new detached branch
git clean -dfx                        # remove
git reset --hard                      # everything, to end up with super clean directory

#  I did vim .gitattributes, but replacing here with echo
echo -e "* annex.backend=MD5E
* annex.largefiles=(not(mimetype=text/*))
" > .gitattributes

datalad add --to-git .gitattributes  # storing this file within git, default commit␣
↪msg
```

Adding new content from a tarball off the web

Next goal was to download and add to annex the tarball Satra prepared for the workshop, and add its content under git-annex control. In datalad we have 'download-url' command, BUT unfortunately it has failed to download via https for this website (see issue 1416 if resolved already) So I have reverted to using git annex directly which uses wget which worked out correctly

```
# download (~800MB) and add that file under git-annex without
git annex addurl --file=ds114_test1_with_freesurfer.tar.gz "$URL"

datalad save -m "Downloaded the tarball with derived data into annex"
datalad add-archive-content --delete --strip-leading-dirs ds114_test1_with_freesurfer.
↪tar.gz
```

Above *add-archive-content* command extracted content from the archive, stripping leading directory, and added all extracted files under git/git-annex using those rules specified in *.gitattributes* file:

- use MD5E (annex keys are based on md5 checksum with extension appended) backend
- add text files directly under git control, so only binary files are added under annex control and the entire repository's *.git/objects* is only around 30MB while pointing to all openfmri releases, and this derived data

Peering inside

Because I have reused original *///openfmri/ds000114* dataset, I have gained knowledge about all the files which originated from that dataset. E.g. compare output of *whereis* command on *sub-01/anat* (which is also available from original openfmri) and derivatives:

```
# in case of a fake tarball, output will not be very interesting
git annex whereis sub-01/anat
git annex whereis derivatives
```

and you can see that derivatives are available only locally or from "magical" datalad-archives remote which refers to the original tarball. So, even if we drop those files locally, they could get extracted from the tarball. And even if you do not have a tarball, git-annex would happily first download it from the OSF website for you.

Adding dataset into bigger dataset

Having succeeded with construction of the dataset, I have decided to share it as a part of our bigger super dataset at http://datasets.datalad.org . This dataset was the first workshop dataset which was not part of some bigger collection, so I have decided to establish a new subdataset *workshops* within it, and move our nipy workshop superdataset into it.

```
cd $SUPERDATASET
# since in demo we do not have anything there, let's clone our superdataset
datalad install ///
cd datasets*.datalad.org

# -redone because now datasets.datalad.org already has workshops dataset
# and datalad should refuse to create a new one (without removing old one first)
datalad create -d . workshops-redone  # create subdataset to hold various workshops␣
↪datasets
cd workshops-redone
mv "$TOPDIR/nipype-workshop-2017" nipype-2017  # chose shorter name
# add it as a subdataset (git submodule) within
datalad add -d . nipype-2017
```

Adding meta-data descriptors for the dataset(s)

If you ever ran datalad search you know that one of the goals of DataLad is to use metadata associated with the datasets.

```
# created some dataset_description.json (following BIDS format lazy me)
echo -e '{"Name": "Datasets for various workshops",\n "BIDSVersion": "1.0.0"}' >␣
↪dataset_description.json
# and tell datalad that this is using the BIDS metadata standard
git config --file .datalad/config --add datalad.metadata.nativetype bids
# add that file to git
datalad add --to-git --nosave dataset_description.json
# discover and aggregate all meta-data within workshops
datalad aggregate-metadata --nosave -r
# and finally save all accumulated changes from above commands
# while also updating the topmost superdataset about this changes under 'workshops'
datalad save -d'^' -m "Added dataset description and aggregated meta-data" -r
# go upstairs and aggregate meta-information across its direct datasets without␣
↪recursing
# (since might take awhile)
cd ..
datalad aggregate-metadata
```

Publishing

NB instructions here might diverge a little from what was actually performed

Now it was time to publish this dataset as a part of our larger super-dataset. Because our demo superdataset is just a clone (or sibling) of original one, it does not have information about where it must be published to. So we first can create a sibling on remote server where we want also to deploy our web-frontend and then create similar siblings for every

```
datalad create-sibling --shared all \
   -s public --ui=true \
   --publish-by-default 'refs/heads/*' \
   --publish-by-default 'refs/tags/*' \
   "$PUBLISHLOC"

datalad create-sibling -s public --inherit -r --existing skip
```

By default DataLad does not publish any data, and in above create-sibling we also did not provide any *–annex-wanted* settings to instruct annex about what data should be published to our public sibling. So I decided to provide additional instructions for annex directly about what data files I want to be published online from our website. Since original files under *sub-** subdirectories are available from original OpenfMRI S3 bucket, we really needed to publish only *derivatives/** files, which we can describe via

```
git -C workshops-redone/nipype-2017/ds000114 annex wanted public 'include=derivatives/
↪*'
```

And now I was ready to publish changes to the entire collection of datasets with a set of files we decided to share

```
datalad publish -r --to=public
```

Above commands created empty repositories for all the datasets we have locally and now I was ready to “publish” our datasets… Just a few final touches

There is a <”shortcoming” *https://github.com/datalad/datalad/issues/1428*>__ which was discovered just now, because it was the first time we published datasets from non-master branch (*nipype_test1*). Default branch on the remote where we published is master, so we need to checkout nipype_test1 branch and re-run out hooks/post-update hook to re-generate meta-data for dataset listing on web-frontend. Hopefully this portion of explanation will disappear with DataLad 0.5.1 or later ;-)

```
(
  cd $PUBLISHDIR/workshops-redone/nipype-2017/ds000114
  git checkout nipype_test1
  # rerun the hook to regenerate meta-data for web-frontend
  cd .git;  hooks/post-update
)
```

If I do any future changes, and save them, it should be sufficient to just rerun this *publish* command (possibly even without explicit –to=public) and have all datasets updated online, with data files under *derivatives/* in that repository posted as well.

Browsing

If the location where we published our datasets is served by any http server, they now could be used from that location by others, while having complete history of changes stored in annex, and data files available either from that location or from original openfmri S3 bucket.

If you do not have published to location served by a web server, as the case in our demo script, we could easily start one using the one which comes with Python:

```
cd "$PUBLISHDIR"
# Starting webserver
python -m SimpleHTTPServer 8080 1>/dev/null 2>&1 &
# we started webserver and can browse
PUBLISHURL=http://localhost:8080
browser=$(which x-www-browser 2>/dev/null)
if $browser; then
    echo "Opening browser to visit $PUBLISHURL which would allow to browse
↪$PUBLISHDIR content"
    $browser $PUBLISHURL &
else
    echo "Visit http://localhost:8080 in your browser."
fi
echo "

On that page
Press Enter when you want to finish
"

in=$(read)
kill %2 && echo "stopped browser(?)" || :  # killing our browser job if any
kill %1 && echo "stopped server"
```

Since DataLad datasets are just git/git-annex repositories, we could as well publish them to multiple locations, including github.com, only without data. See datalad-create-sibling-github and *–publish-depends* option to instruct to publish first to our public http server which will host the data and then to github.com for more visibility and collaboration.

CHAPTER 4

API

4.1 Python module reference

This module reference extends the manual with a comprehensive overview of the available functionality. Each module in the package is documented by a general summary of its purpose and the list of classes and functions it provides.

bids2scidata	generate metadata for submission to Scientific Data from a BIDS dataset

4.1.1 datalad_neuroimaging.bids2scidata

generate metadata for submission to Scientific Data from a BIDS dataset

class datalad_neuroimaging.bids2scidata.**BIDS2Scidata**
Bases: datalad.interface.base.Interface

BIDS to ISA-Tab converter

datalad_neuroimaging.bids2scidata.**convert**(*dsmeta*, *filemeta*, *output_directory*, *repository_info=None*)

datalad_neuroimaging.bids2scidata.**getprop**(*obj*, *path*, *default*)
Helper to get a property from a metadata structure that might not be there

datalad_neuroimaging.bids2scidata.**split_term_source_accession**(*val*)

4.2 datalad-bids2scidata

4.2.1 Synopsis

```
datalad-bids2scidata [-h] [--repo-name REPO_NAME] [--repo-accession REPO_ACCESSION] [-
↪-repo-url REPO_URL] [--output OUTPUT] [-d DATASET] path
```

4.2.2 Description

BIDS to ISA-Tab converter

4.2.3 Options

path

path to a BIDS-compatible dataset to export metadata on.

-h, –help, –help-np

show this help message. –help-np forcefully disables the use of a pager for displaying the help message

–repo-name *REPO_NAME*

data repository name to be used in assay tables. Example: OpenfMRI.

–repo-accession *REPO_ACCESSION*

data repository accession number to be used in assay tables. Example: ds000113d.

–repo-url *REPO_URL*

data repository URL to be used in assay tables. Example: https://openfmri.org/dataset/ds000113d.

–output *OUTPUT*

directory where ISA-TAB files will be stored.

-d *DATASET*, –dataset *DATASET*

Dataset to query for metadata of a BIDS-compatible dataset. The queried dataset itself does not have to be a BIDS dataset. If not dataset is given, an attempt is made to identify the dataset based on the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path)

4.2.4 Authors

datalad is developed by The DataLad Team and Contributors <team@datalad.org>.

CHAPTER 5

Metadata

This extension adds metadata extraction support for a range of standards common to neuroimaging data.

5.1 Brain Imaging Data Structure (`bids`)

This extractor has basic support for the BIDS standard. This includes participant information, as well as acquisition properties for individual files. At present, there is no standardized vocabulary for BIDS, instead field names are based on the conventions in the standard description.

5.2 Digital Imaging and Communications in Medicine (`dicom`)

Metadata can be extracted from any standard DICOM file. The extractor yields file-based metadata, and a dataset-level description that identifies individual image series. For each image series, all metadata are reported that are invariant across individual images in a series. The extractor uses an incomplete DICOM vocabulary from http://semantic-dicom.org

5.3 Neuroimaging data exchange format (`nifti1`)

NIfTI-1 metadata is extracted from the header of individual files. Virtually all header information is reported, except for header extensions. An adhoc-vocabulary is used, as no standard vocabulary is available.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index