datalad$_crawler Documentation$

Release 0.1.0

DataLad team

Jun 20, 2019

Contents

CHAPTER 1

Change log

```
 ____            _             _                   _ 
|  _ \    __ _  | |_    __ _  | |       __ _    __| |
| | | |  / _` | | __|  / _` | | |      / _` |  / _` |
| |_| | | (_| | | |_  | (_| | | |___  | (_| | | (_| |
|____/   \__,_|  \__|  \__,_| |_____|  \__,_|  \__,_|
                                           Crawler
```

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 0.1 (May 11, 2018) – The Release

- First release as a DataLad extension. Functionality remains identical to DataLad 0.10.0.rc2

CHAPTER 2

Acknowledgments

DataLad development is being performed as part of a US-German collaboration in computational neuroscience (CR-CNS) 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

DataLad Crawler 101

3.1 Nodes

A node in a pipeline is just a callable (function of a method of a class) which takes a dictionary, and yields a dictionary any number of times. The simplest node could look like

```
>>> def add1_node(data, field='input'):
...     data_ = data.copy()
...     data_['input'] += 1
...     yield data_
```

which creates a simple node, intended to increment an arbitrary (specified by *field* keyword argument) field in the input dictionary and yields a modified dictionary as its output once.

```
>>> next(add1_node({'input': 1}))
{'input': 2}
```

Nodes are generators which yield a dictionary zero, one, or multiple times and yield a dictionary. For more on generators, reference the Python documentation on Generators.

Note: Nodes should not have side-effects, i.e. they should not modify input data, but yield a shallow copy if any of the field values need to be added, removed, or modified. To help with creation of a new shallow copy with some fields adjusted, use `updated()`.

3.2 Pipelines

A pipeline is a series of generators ordered into a list. Each generator takes the output of its predecessor as its own input. The first node in the pipeline would need to be provided with specific input. The simplest pipeline could look like

```
>>> from datalad.crawler.nodes.crawl_url import crawl_url
>>> from datalad.crawler.nodes.matches import a_href_match
>>> from datalad.crawler.nodes.annex import Annexificator
>>> annex = Annexificator(allow_dirty=True)  # so we could demo right within
>>> pipeline = \
...     [
...     crawl_url('http://map.org/datasets'),
...     a_href_match(".*\.mat"),
...     annex
...     ]
```

in which the first node (method of a class) is provided with input and crawls a website. *a_href_match* then works to output all files that end in *.mat*, and those files are lastly inputted to *annex*, another node, which simply annexes them.

Note: Since pipelines depend heavily on nodes, these nodes must yield in order for an output to be produced. If a generator fails to yield, then the pipeline can no longer continue and it is stopped at that node.

3.3 Subpipelines

A subpipline is a pipeline that lives within a greater pipeline and is also denoted by *[]*. Two subpipelines that exist on top of one another will take in the same input, but process it with different generators. This functionality allows for the same input to be handled in two or more (depending on the number of subpipelines) different manners.

TODO: ‘FinishPipeline‘ exception here `FinishPipeline`

CHAPTER 4

Demo

4.1 Track data from a webpage

With a few lines DataLad is set up to track data posted on a website, and obtain changes made in the future…

The website http://www.fmri-data-analysis.org/code provides code and data file for examples in a text book.

We will set up a dataset that DataLad uses to track the content linked from this webpage

Let's create the dataset, and configure it to track any text file directly in Git. This will make it very convenient to see how source code changed over time.

```
~ % datalad create --text-no-annex demo
[INFO   ] Creating a new annex repo at /demo/demo
create(ok): /demo/demo (dataset)
~ % cd demo
```

DataLad's crawler functionality is used to monitor the webpage. It's configuration is stored in the dataset itself.

The crawler comes with a bunch of configuration templates. Here we are using one that extract all URLs that match a particular pattern, and obtains the linked data. In case of this webpage, all URLs of interest on that page seems to have 'd=1' suffix

```
~/demo % datalad crawl-init --save --template=simple_with_archives url=http://www.
↪fmri-data-analysis.org/code 'a_href_match_=.*d=1$'
[INFO   ] Creating a pipeline to crawl data files from http://www.fmri-data-analysis.
↪org/code
[INFO   ] Initiating special remote datalad-archives
[INFO   ] Not adding annex.largefiles=exclude=README* and exclude=LICENSE* to git␣
↪annex calls because already defined to be (not(mimetype=text/*))
~/demo % datalad diff --revision @~1
         added(file): .datalad/crawl/crawl.cfg
~/demo % cat .datalad/crawl/crawl.cfg
[crawl:pipeline]
template = simple_with_archives
```

(continues on next page)

(continued from previous page)

```
_url = http://www.fmri-data-analysis.org/code
_a_href_match_ = .*d=1$
```

With this configuration in place, we can ask DataLad to crawl the webpage.

```
~/demo % datalad crawl
[INFO   ] Loading pipeline specification from ./.datalad/crawl/crawl.cfg
[INFO   ] Creating a pipeline to crawl data files from http://www.fmri-data-analysis.
↪org/code
[INFO   ] Not adding annex.largefiles=exclude=README* and exclude=LICENSE* to git␣
↪annex calls because already defined to be (not(mimetype=text/*))
[INFO   ] Running pipeline [<function switch_branch at 0x7f9147061488>, [[<datalad.
↪crawler.nodes.crawl_url.crawl_url object at 0x7f9135c6ad50>, a_href_match(query='.
↪*d=1$'), <function fix_url at 0x7f914b7a9cf8>, <datalad.crawler.nodes.annex.
↪Annexificator object at 0x7f9135c4b810>]], <function switch_branch at␣
↪0x7f9135c51de8>, [<function merge_branch at 0x7f9135c51050>, [find_files(dirs=False,
↪ fail_if_none=True, regex='\\.(zip|tgz|tar(\\..+)?)$', topdir='.'), <function _add_
↪archive_content at 0x7f9135c51e60>]], <function switch_branch at 0x7f9135c51ed8>,
↪<function merge_branch at 0x7f9135c51f50>, <function _finalize at 0x7f9135c74050>]
[INFO   ] Found branch non-dirty -- nothing was committed
[INFO   ] Checking out master into a new branch incoming
[INFO   ] Fetching 'http://www.fmri-data-analysis.org/code'
[INFO   ] Need to download 950 Bytes from http://www.fmri-data-analysis.org/code/
↪figure_2_12.R?attredirects=0&d=1. No progress indication will be reported
[INFO   ] Need to download 656 Bytes from http://www.fmri-data-analysis.org/code/
↪figure_2_14.m?attredirects=0&d=1. No progress indication will be reported
[INFO   ] Need to download 4.3 kB from http://www.fmri-data-analysis.org/code/figure_
↪2_3.m?attredirects=0&d=1. No progress indication will be reported
[INFO   ] Need to download 453.1 kB from http://www.fmri-data-analysis.org/code/
↪figure_3_14.tgz?attredirects=0&d=1. No progress indication will be reported
[INFO   ] Need to download 486 Bytes from http://www.fmri-data-analysis.org/code/
↪figure_3_8.m?attredirects=0&d=1. No progress indication will be reported
[INFO   ] Need to download 255 Bytes from http://www.fmri-data-analysis.org/code/
↪figure_3_9.m?attredirects=0&d=1. No progress indication will be reported
[INFO   ] Need to download 321.6 kB from http://www.fmri-data-analysis.org/code/
↪figure_4_7.tgz?attredirects=0&d=1. No progress indication will be reported
[INFO   ] Need to download 2.1 kB from http://www.fmri-data-analysis.org/code/figure_
↪5_10.m?attredirects=0&d=1. No progress indication will be reported
[INFO   ] Need to download 1.1 kB from http://www.fmri-data-analysis.org/code/figure_
↪5_11.m?attredirects=0&d=1. No progress indication will be reported
[INFO   ] Need to download 2.5 kB from http://www.fmri-data-analysis.org/code/figure_
↪5_12.zip?attredirects=0&d=1. No progress indication will be reported
[INFO   ] Need to download 1.5 kB from http://www.fmri-data-analysis.org/code/figure_
↪5_3.m?attredirects=0&d=1. No progress indication will be reported
[INFO   ] Need to download 110.4 kB from http://www.fmri-data-analysis.org/code/
↪figure_8_11.tgz?attredirects=0&d=1. No progress indication will be reported
[INFO   ] Need to download 1.7 kB from http://www.fmri-data-analysis.org/code/figure_
↪8_2.m?attredirects=0&d=1. No progress indication will be reported
[INFO   ] Need to download 3.2 kB from http://www.fmri-data-analysis.org/code/figure_
↪9_1.R?attredirects=0&d=1. No progress indication will be reported
[INFO   ] Need to download 9.8 kB from http://www.fmri-data-analysis.org/code/figure_
↪9_2.R?attredirects=0&d=1. No progress indication will be reported
[INFO   ] Need to download 9.8 kB from http://www.fmri-data-analysis.org/code/figure_
↪9_3.R?attredirects=0&d=1. No progress indication will be reported
                                                                              [INFO␣
↪   ] Repository found dirty -- adding and committing
                                                                              [INFO␣
↪   ] Checking out a new detached branch incoming-processed
```

(continues on next page)

(continued from previous page)

```
[INFO   ] Initiating 1 merge of incoming using strategy theirs
                                                                              [INFO␣
↪   ] Adding content of the archive ./figure_4_7.tgz into annex <AnnexRepo path=/demo/
↪demo (<class 'datalad.support.annexrepo.AnnexRepo'>)>
[INFO   ] Finished adding ./figure_4_7.tgz: Files processed: 4, +git: 1, +annex: 3
[INFO   ] Adding content of the archive ./figure_8_11.tgz into annex <AnnexRepo path=/
↪demo/demo (<class 'datalad.support.annexrepo.AnnexRepo'>)>
[INFO   ] Finished adding ./figure_8_11.tgz: Files processed: 7, renamed: 7, +git: 4,␣
↪+annex: 3
[INFO   ] Adding content of the archive ./figure_5_12.zip into annex <AnnexRepo path=/
↪demo/demo (<class 'datalad.support.annexrepo.AnnexRepo'>)>
[INFO   ] Finished adding ./figure_5_12.zip: Files processed: 3, skipped: 1, renamed:␣
↪2, +git: 2
[INFO   ] Adding content of the archive ./figure_3_14.tgz into annex <AnnexRepo path=/
↪demo/demo (<class 'datalad.support.annexrepo.AnnexRepo'>)>
[INFO   ] Finished adding ./figure_3_14.tgz: Files processed: 6, renamed: 6, +annex: 6
[INFO   ] Repository found dirty -- adding and committing
                                                                              [INFO␣
↪   ] Checking out an existing branch master
[INFO   ] Initiating 1 merge of incoming-processed using strategy None
[INFO   ] Found branch non-dirty -- nothing was committed
[INFO   ] House keeping: gc, repack and clean
[INFO   ] Finished running pipeline: URLs processed: 16, downloaded: 16, size: 923.4␣
↪kB,  Files processed: 40, skipped: 1, renamed: 15, +git: 19, +annex: 16,  Branches␣
↪merged: incoming->incoming-processed
[INFO   ] Total stats: URLs processed: 16, downloaded: 16, size: 923.4 kB,  Files␣
↪processed: 40, skipped: 1, renamed: 15, +git: 19, +annex: 16,  Branches merged:␣
↪incoming->incoming-processed,  Datasets crawled: 1
```

All files have been obtained and are ready to use. Here is what DataLad recorded for this update

```
~/demo % git show @ -s
commit 3a8033d45cf7a96b523d927e02cf9d6a79f8e30e (HEAD -> master, incoming-processed)
Author: DataLad Demo <demo@datalad.org>
Date:   Fri Mar 16 08:41:22 2018 +0100

    [DATALAD] Added files from extracted archives

    Files processed: 24
     skipped: 1
     renamed: 15
     +git: 7
     +annex: 12
    Branches merged: incoming->incoming-processed
```

Any file from the webpage is available locally.

```
~/demo % ls
all_rois.txt    figure_4_7.sh  figure_9_3.R
dat.txt                figure_5_10.m  flirt_thresh_zstat1.nii.gz
fair_abbrevs.txt   figure_5_11.m  fnirt_thresh_zstat1.nii.gz
fair_networks.txt  figure_5_12.m  mean_func.nii.gz
figure_2_12.R          figure_5_3.m   zstat1_0mm.nii.gz
figure_2_14.m          figure_8_11.R  zstat1_16mm.nii.gz
figure_2_3.m    figure_8_2.m   zstat1_32mm.nii.gz
figure_3_8.m    figure_9_1.R   zstat1_4mm.nii.gz
figure_3_9.m    figure_9_2.R   zstat1_8mm.nii.gz
```

(continues on next page)

(continued from previous page)

```
~/demo % #
```

The webpage can be queried for potential updates at any time by re-running the ‘crawl’ command.

```
~/demo % datalad crawl
[INFO   ] Loading pipeline specification from ./.datalad/crawl/crawl.cfg
[INFO   ] Creating a pipeline to crawl data files from http://www.fmri-data-analysis.
↪org/code
[INFO   ] Not adding annex.largefiles=exclude=README* and exclude=LICENSE* to git␣
↪annex calls because already defined to be (not(mimetype=text/*))
[INFO   ] Running pipeline [<function switch_branch at 0x7f47abaf3488>, [[<datalad.
↪crawler.nodes.crawl_url.crawl_url object at 0x7f479a6fcd50>, a_href_match(query='.
↪*d=1$'), <function fix_url at 0x7f47b023acf8>, <datalad.crawler.nodes.annex.
↪Annexificator object at 0x7f479a6dd810>]], <function switch_branch at␣
↪0x7f479a6e3de8>, [<function merge_branch at 0x7f479a6e3050>, [find_files(dirs=False,
↪ fail_if_none=True, regex='\\.(zip|tgz|tar(\\..+)?)$', topdir='.'), <function _add_
↪archive_content at 0x7f479a6e3e60>]], <function switch_branch at 0x7f479a6e3ed8>,
↪<function merge_branch at 0x7f479a6e3f50>, <function _finalize at 0x7f479a706050>]
[INFO   ] Found branch non-dirty -- nothing was committed
[INFO   ] Checking out an existing branch incoming
[INFO   ] Fetching 'http://www.fmri-data-analysis.org/code'
                                                                                [INFO␣
↪   ] Found branch non-dirty -- nothing was committed
[INFO   ] Checking out an existing branch incoming-processed
[INFO   ] Found branch non-dirty -- nothing was committed
[INFO   ] Checking out an existing branch master
[INFO   ] Finished running pipeline: URLs processed: 16,  Files processed: 16,␣
↪skipped: 16
[INFO   ] Total stats: URLs processed: 16,  Files processed: 16, skipped: 16, ␣
↪Datasets crawled: 1
```

Files can be added, or removed from this dataset without impairing the ability to get updates from the webpage. DataLad keeps the necessary information in dedicated Git branches.

```
~/demo % git branch
  git-annex
  incoming
  incoming-processed
* master
```

CHAPTER 5

Command line reference

5.1 datalad-crawl

5.1.1 Synopsis

```
datalad-crawl [-h] [--is-pipeline] [-t] [-r] [-C CHDIR] [file]
```

5.1.2 Description

Crawl online resource to create or update a dataset.

Examples:

> $ datalad crawl # within a dataset having .datalad/crawl/crawl.cfg

5.1.3 Options

file

configuration (or pipeline if –is-pipeline) file defining crawling, or a directory of a dataset on which to perform crawling using its standard crawling specification. Constraints: value must be a string [Default: None]

-h, –help, –help-np

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

–is-pipeline

flag if provided file is a Python script which defines pipeline(). [Default: False]

-t, –is-template

flag if provided value is the name of the template to use. [Default: False]

-r, –recursive

flag to crawl subdatasets as well (for now serially). [Default: False]

-C *CHDIR*, –chdir *CHDIR*

directory to chdir to for crawling. Constraints: value must be a string [Default: None]

5.1.4 Authors

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

5.2 datalad-crawl-init

5.2.1 Synopsis

```
datalad-crawl-init [-h] [-t TEMPLATE] [-f TEMPLATE_FUNC] [--save] [args [args ...]]
```

5.2.2 Description

Initialize crawling configuration

Allows to specify template and function to generate a crawling pipeline

Examples:

$ datalad crawl-init –template openfmri –template-func superdataset_pipeline

$ datalad crawl-init –template fcptable dataset=Baltimore tarballs=True

5.2.3 Options

args

keyword arguments to pass into the template function generating actual pipeline, organized in key=value pairs. Constraints: value must be a string [Default: None]

-h, –help, –help-np

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

-t *TEMPLATE*, –template *TEMPLATE*

the name of the template. Constraints: value must be a string [Default: None]

-f *TEMPLATE_FUNC*, –template-func *TEMPLATE_FUNC*

the name of the function. [Default: None]

–save

flag to save file into git repo. [Default: False]

5.2.4 Authors

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

CHAPTER 6

Python API

6.1 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.

6.1.1 Commands

crawl	Interface for crawling a webpage and push extracted data into a dataset
crawl_init	Interface for a generic template in which arguments are specified by the user

datalad_crawler.crawl

Interface for crawling a webpage and push extracted data into a dataset

class `datalad_crawler.crawl.`**`Crawl`**

Bases: `datalad.interface.base.Interface`

Crawl online resource to create or update a dataset.

Examples

$ datalad crawl # within a dataset having .datalad/crawl/crawl.cfg

datalad_crawler.crawl_init

Interface for a generic template in which arguments are specified by the user

class `datalad_crawler.crawl_init.`**`CrawlInit`**

Bases: `datalad.interface.base.Interface`

Initialize crawling configuration

Allows to specify template and function to generate a crawling pipeline

Examples:

$ datalad crawl-init –template openfmri –template-func superdataset_pipeline

$ datalad crawl-init –template fcptable dataset=Baltimore tarballs=True

6.1.2 Pipelines

`pipeline`	Pipeline functionality.

datalad_crawler.pipeline

Pipeline functionality.

A pipeline is represented by a simple list or tuple of nodes or other nested pipelines. Each pipeline node is a callable which receives a dictionary (commonly named *data*), does some processing, and yields (once or multiple times) a derived dictionary (commonly a shallow copy of original dict). For a node to be parametrized it should be implemented as a callable (i.e. define __call__) class, which could obtain parameters in its constructor.

TODO: describe PIPELINE_OPTS and how to specify them for a given (sub-)pipeline.

The *data* dictionary is used primarily to carry the scraped/produced data, but besides that it will carry few items which some nodes might use. All those item names will start with the *datalad_* prefix, and will be intended for 'inplace' modifications or querying. The following items are planned to be provided by the pipeline runner:

datalad_settings PipelineSettings object which could be used to provide configuration for the current run of the pipeline. E.g.:

- dry: either nodes are intended not to perform any changes which would reflect on disk
- skip_existing:

datalad_stats ActivityStats/dict object to accumulate statistics on what has been done by the nodes so far

To some degree, we could make an analogy when *blood* is to *data* and *venous system* is to *pipeline*. Blood delivers various elements which are picked up by various parts of our body when they know what to do with the corresponding elements. To the same degree nodes can consume, augment, or produce new items to the *data* and send it down the stream. Since there is no strict typing or specification on what nodes could consume or produce (yet), no verification is done and things can go utterly wrong. So nodes must be robust and provide informative logging.

exception `datalad_crawler.pipeline.`**`FinishPipeline`**

Bases: `exceptions.Exception`

Exception to use to signal that any given pipeline should be stopped

`datalad_crawler.pipeline.`**`get_repo_pipeline_config_path`**(*repo_path='.'*)

Given a path within a repo, return path to the crawl.cfg

`datalad_crawler.pipeline.`**`get_repo_pipeline_script_path`**(*repo_path='.'*)

If there is a single pipeline present among 'pipelines/', return path to it `datalad_crawler.pipeline.`**`initiate_pipeline_config`**(*template*, *template_func=None*, *template_kwargs=None*, *path='.'*, *commit=False*)

TODO Gergana ;)

`datalad_crawler.pipeline.`**`load_pipeline_from_config`**(*path*)

Given a path to the pipeline configuration file, instantiate a pipeline

Typical example description

[crawl:pipeline] pipeline = standard func = pipeline1 _kwarg1 = 1

which would instantiate a pipeline from standard.py module by calling *standard.pipeline1* with *_kwarg1='1'*. This definition is identical to

[crawl:pipeline] pipeline = standard?func=pipeline1&_kwarg1=1

so that theoretically we could specify basic pipelines completely within a URL

`datalad_crawler.pipeline.`**`load_pipeline_from_module`**(*module*, *func=None*, *args=None*, *kwargs=None*, *return_only=False*)

Load pipeline from a Python module

Parameters

- **`module`** (`str`) – Module name or filename of the module from which to load the pipeline
- **`func`** (`str, optional`) – Function within the module to use. Default: *pipeline*
- **`args`** (`list or tuple, optional`) – Positional arguments to provide to the function.
- **`kwargs`** (`dict, optional`) – Keyword arguments to provide to the function.
- **`return_only`** (`bool, optional`) – flag true if only to return pipeline

`datalad_crawler.pipeline.`**`load_pipeline_from_template`**(*name*, *func=None*, *args=None*, *kwargs=None*, *return_only=False*)

Given a name, loads that pipeline from datalad_crawler.pipelines

and later from other locations

Parameters

- **`name`** (`str`) – Name of the pipeline (the template) defining the filename, or the full path to it (TODO), example: openfmri
- **`func`** (`str`) – Name of function from which pipeline to run example: superdataset_pipeline
- **`args`** (`dict, optional`) – Positional args for the pipeline, passed as **args* into the pipeline call
- **`kwargs`** (`dict, optional`) – Keyword args for the pipeline, passed as ***kwargs* into the pipeline call, example: {'dataset': 'ds000001'}
- **`return_only`** (`bool, optional`) – flag true if only to return pipeline

`datalad_crawler.pipeline.`**`reset_pipeline`**(*pipeline*)

Given a pipeline, traverse its nodes and call .reset on them

Note: it doesn't try to call reset if a node doesn't have it

`datalad_crawler.pipeline.`**`run_pipeline`**(**args*, ***kwargs*)

Run pipeline and assemble results into a list

By default, the pipeline returns only its input (see PIPELINE_OPTS), so if no options for the pipeline were given to return additional items, a *[{}]* will be provided as output

`datalad_crawler.pipeline.`**`xrun_pipeline`**(*pipeline*, *data=None*, *stats=None*, *reset=True*)
Yield results from the pipeline.

`datalad_crawler.pipeline.`**`xrun_pipeline_steps`**(*pipeline*, *data*, *output='input'*)
Actually run pipeline steps, feeding yielded results to the next node and yielding results back.

Recursive beast which runs a single node and then recurses to run the rest, possibly multiple times if the current node is a generator. It yields output from the node/nested pipelines, as directed by the output argument.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index