Modulos del pipeline

Esta pagina es solo de referencia de los modulos Perl que dan soporte al pipeline de neuroimagen.

NEURO4

This is a set of functions for helping in the pipeline

- print_help

  just print the help

  this funtions reads the path of a TXT file and print it at STDOUT

  usage:

  ```
  print_help(help_file);
  ```

- escape_name

  This function takes a string and remove some especial characters in order to escape directory names with a lot of strange symbols.

  It returns the escaped string

  usage:

  ```
  escape_name(string);
  ```

- trim

  This function takes a string and remove any trailing spaces after and before the text

  usage:

  ```
  trim(string);
  ```

- check_or_make

  This is mostly helpless, just takes a path, checks if exists and create it otherwise

  usage:

  ```
  check_or_make(path);
  ```

- inplace

This function takes a path and a file name or two paths and returns a string with a single path as result of the concatenation of the first one plus the second one

usage:

```
inplace(path, filename);
```

- load_project

  This function take the name of a project, reads the configuration file that is located at ~/.config/neuro/ and return every project configuration stored as a hash that can be used at the scripts

  usage:

  ```
  load_project(project_name);
  ```

- check_subj

  Here the fun begins

  This function takes as input the name of the project and the subject ID Then it seeks along the BIDS structure for this subject and returns a hash, containing the MRI proper images.

  It should return a single value, except for the T1w images, where an array is returned. This was though this way because mostly a single session is done. However, the skill to detect more than one MRI was introduced to allow the movement correction when ADNI images are analyzed

  So, for T1w images the returned hash should be asked as

  ```
  @{$nifti{'T1w'}}
  ```

  but for other kind of image it should asked as

  ```
  $nifti{'T2w'}
  ```

  usage:

  ```
  check_subj(project_path, bids_id);
  ```

- check_pet

  This function takes as input the name of the project and the subject ID Then it seeks along the BIDS structure for this subject and returns a hash, containing the PET proper images.

  If also a tracer is given as input, then the returned hash contains the PET-tau associated to this tracer. This was introduced as part of a project were the subjects were analyzed with different radiotracers.

  If no tracer is given, it will seek for the FBB PETs. Those PETs are stored as

```
      - single: 4x5min
      - combined: 20min
```

usage:

```
      check_pet(project_path, bids_id, $optional_radiotracer);
```

- check_fs_subj

  This function checks if the Freesurfer directory of a given subjects exists

  usage:

  ```
      check_fs_subj(freesurfer_id)
  ```

- get_lut

  I really don't even remenber what this shit does

- run_dckey

  Get the content of a public tag from a DICOM file.

  usage:

  ```
      run_dckey(key, dicom)
  ```

- dclokey

  Get the content of a private tag from a DICOM file.

  usage:

  ```
      dclokey(key, dicom)
  ```

- centiloid_fbb

  Returns the proper centiloid value for a given SUVR. Only valid for FBB.

  usage:

  ```
      centiloid_fbb(suvr);
  ```

- populate

  Takes a pattern and a filename and stores the content of the file into a HASH according to the given pattern

usage:

```
populate(pattern, filename);
```

- get_subjects

  Parse a project database taking only the subjects and storing them into an array. The databse is expected to be build as,

  ```
  0000;name
  ```

  usage:

  ```
  get_subjects(filename);
  ```

- get_list

  Parse a project database taking only the subjects and storing them into an array. The databse is expected to be build with a four digits number at the beginning of line. Is similar to get_subjects() function but less restrictive

  usage:

  ```
  get_list(filename);
  ```

- get_pair

  A single file is loaded as input and parse into a HASH. The file should be written in the format:

  ```
  key;value
  ```

  usage:

  ```
  get_pair(filename);
  ```

- shit_done

  this function is intended to be used after a script ends and then an email is send to the user with the name of the script, the name of the project and th results attached

  usage:

  ```
  shit_done(script_name, project_name, attached_file)
  ```

- cut_shit

  This function takes a project database and a file with a list, then returns the elements that are common to both. It is intended to be used to restrict the scripts action over a few elements. It

returns a single array.

If it is correctly used, first the db is identified with *load_project()* function and then passed through this function to get the array of subjects to be analyzed. If the file with the cutting list do not exist, an array with all the subjects is returned.

usage:

```
cut_shit(db, list);
```

- getLoggingTime

  This function returns a timestamp based string intended to be used to make unique filenames

  Stolen from Stackoverflow

  usage:

```
getLoggingTime();
```

FSMetrics

Bunch of helpers for storing ROI structure and relative data

- fs_file_metrics

  This function does not read any input. It sole purpose is to returns a HASH containing the templates of order for converting Freesurfer (FS) results into tables.

  Any hash element is composed by the template ('order'), a boolean ('active') to decide if the FS stats will be processed and the name of the FS stat file ('file'). The order template has two wildcards (<list> and <fs_output>) that should be parsed and changed by the FS subject id and the output directory where the data tables will be stored for each subject

  The function could be invoked as,

```
my %stats = fs_file_metrics();
```

  in any script where this information would be needed.

  The boolean element could be used to choose the stats that should be processed and can be added or modified even at run time if needed. The stored booleans only provided a decent default

- fs_fbb_rois

  deprecated

This function exports a HASH that contains the Freesurfer composition of the usual segmentations used for building the SUVR ROI

- tau_rois

  This function takes a string as input and returns an ARRAY containing the list of ROIs that should be build and where the SUVR should be calculated

  It is intended to be used for PET-Tau but could be used anywhere

  By default a list of Braak areas are returned. If the input string is **alt** a grouping of those Braak areas is returned. If the purpose is to build a meta_temporal ROI the string **meta** should be passed as input

  The main idea here is read the corresponding file for each ROI, stored at `PIPEDIR/lib/tau/` and build each ROI with the FS LUTs store there

- pet_rois

  This function takes a string as input and returns an ARRAY containing the list of ROIs that should be build and where the SUVR should be calculated

  Input values are **parietal**, **frontal**, **pieces** or **global** (default)

  The main idea here is read the corresponding file for each ROI, stored at `PIPEDIR/lib/pet/` and build each ROI with the FS LUTs stored there

SLURM

This module contains just a function to send the jobs to SLURM from the Perl scripts

- send2slurm

  The function takes a HASH as input where all the information relative to the job should be stored. No data is mandatory inside the input HASH, since the minimal values are automagicaly asigned by default as a constructor (no really, but anyway).

  Take into account that this subroutine only pass the parameters to SLURM. So, the logic behind your actions should correspond to what you want to do in any case, exactly as if you were writing sbatch scripts.

  The managed options for SLURM jobs are:

```
      - filename: File where the sbatch script will be stored
      - job_name: Job name for SLURM (-J)
      - cpus: Number of CPUs to be used by each job (-c)
      - mem_per_cpu: Amount of memory to be used for each CPU (--mem-
per-cpu)
      - time: Maximum time that the job will be allowed to run (--time)
```

```
      - output: File where the sbatch script output will be stored (-o)
      - partition: SLURM partition to be used (-p)
      - gres: GPUs to be used (--gres)
      - command: Command to be executed at sbatch script
      - mailtype: Type of warning to be emailed (--mail-type)
      - dependency: Full dependency string to be used at sbatch
execution (--dependency), see more below
```

The function returns the jobid of the queued job, so it can be used to build complex workflows.

usage: my $job_id = send2slurm(\%job_properties);

Warning email: By default, if an empty HASH is passed to the function, a no command sbatch script is launched with *--mail-type=END* option. The intention is that this could be used to warn at the end of any launched swarm. Also, by changing **mailtype** but ommiting the **command** value you can force the function to execute an empty sbatch job with whatever warning behavior that you choose.

Dependencies: If dependencies are going to be used, you need to pass to the function the full string that SLURM expects. That is, you can pass something like *singleton* or *after:000000* or even *afterok:000000,000001,000002*. This last can be build, by example, storing every previous jobid into an ARRAY and passing then as,

```
      ...
              my $jobid = send2slurm(\%previous);
              push @jobids, $jobid;
      ...
      $task{'dependency'} = 'afterok:'.join(',',@jobids);
      ...
      send2slurm(\%task);
```

Of course, if dependencies are not going to be used, the **dependency** option could be safely ignored. But notice that, if you are reusing a HASH then this key should be deleted from it.

XNATACE

- xconf

  Publish path of xnatapic configuration file

  usage:

  ```
      $path = xconf();
  ```

- xget_conf

  Get the XNAT connection data into a HASH

usage:

```
%xnat_data = xget_conf()
```

- xget_session

  Create a new JSESSIONID on XNAT. Return the connection data for the server AND the ID of the created session

  usage:

```
xget_session();
```

- xget_subjects

  Get the list of subjects of a project into a HASH. El HASH de input, *%sbjs*, se construye como *{ XNAT_ID => Label }*

  usage:

```
%sbjs = xget_subjects(host, jsession, project);
```

- xget_sbj_data

  Get the subjects metadata. Not too much interesting but to extract the subject label.

  usage:

```
$xdata = xget_sbj_data(host, jsession, subject, field);
```

- xget_exp_data

  Get a data field of an experiment. The desired field shoud be indicated as input. By example, if you want the date of the experiment this is seeked as

```
my $xdate = xget_exp_data($host, $session_id, $experiment,
'date')
```

  There are some common fields as *date*, *label* or *dcmPatientId* but in general you should look at,

```
curl -X GET -b JSESSIONID=00000blahblah
"http://myhost/data/experiments/myexperiment?format=json" 2>/dev/null |
jq '.items[0].data_fields'
```

  in order to know the available fields

  usage:

```
$xdata = xget_exp_data(host, jsession, experiment, field);
```

- xget_mri

  Get the XNAT MRI experiment ID

  usage:

```
      xget_mri(host, jsession, project, subject)
```

- xget_fs_data

  Get the full Freesurfer directory in a tar.gz file

  usage:

```
      xget_fs_data(host, jsession, project, experiment, output_path)
```

- xget_fs_stats

  Get a single stats file from Freesurfer segmentation

  usage:

```
      xget_fs_stats(host, jsession, experiment, stats_file,
output_file)
```

- xget_fs_allstats

  Get all stats files from Freesurfer segmentation and write it down at selected directory

  usage:

```
      xget_fs_allstats(host, jsession, experiment, output_dir)
```

- xget_fs_qc

  Get Freeesurfer QC info

  usage:

```
      xget_fs_qc(host, jsession, experiment);
```

  Output is a hash with *rating* and *notes*

- xget_pet

  Get the XNAT PET experiment ID

  usage:

```
xget_pet(host, jsession, project, subject)
```

- xget_pet_reg

  Download de pet registered into native space in nifti format

  usage:

```
xget_pet_reg(host, jsession, experiment, nifti_output);
```

- xget_pet_data

  Get the PET FBB analysis results into a HASH

  usage:

```
%xresult = xget_pet_data(host, jsession, experiment);
```

- xput_report

  Upload a pdf report to XNAT

  usage:

```
xput_report(host, jsession, subject, experiment, pdf_file);
```

- xput_rvr

  Upload a JSON file with VR data

  usage:

```
xput_rvr(host, jsession, experiment, json_file);
```

- xcreate_res

  Create an empty experiment resource

  usage:

```
xcreate_res(host, jsession, experiment, res_name)
```

- xput_res

  Upload data to an experiment resource

  usage:

```
      xput_res(host, jsession, experiment, type, file, filename)
```

- xget_rvr

  Get VR results into a HASH. Output is a hash with filenames and URI of each element stored at RVR

  usage:

```
      xget_rvr(host, jsession, project, experiment);
```

- xget_rvr_data

  Get RVR JSON data into a hash

  usage:

```
      xget_rvr_data(host, jsession, URI);
```

Dependencias

```
Data::Dump
File::Slurp
File::Basename
File::Temp
File::Copy::Recursive
File::Copy
File::Find::Rule
File::Remove
Cwd
Spreadsheet::Write
Text::CSV
File::Path
MIME::Lite
JSON
```