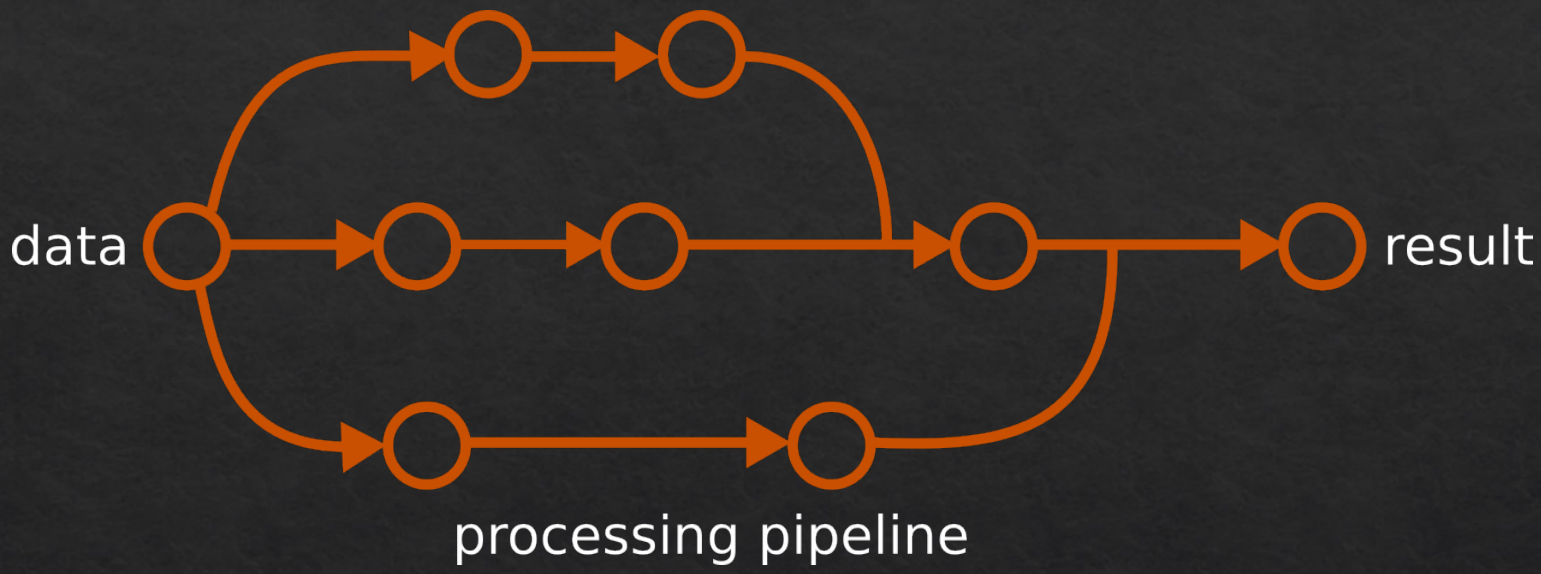
Making neuroimaging processing pipelines reproducible

Pierre Bellec
pierre.bellec@criugm.qc.ca

HBM Educational workshop on reproducible neuroimaging - 2016/06

The problem



$\mathcal{Q}$: How to make neuroimaging data analysis reproducible?

$\mathcal{A}$: (short version) automate everything.

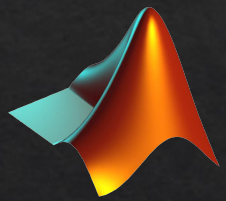
Step 1: Learn how to code (a bit)

Because there is nothing quite like a `For` loop to automate stuff.

Step 1: Learn how to code (a bit)

Which language (non-comprehensive list, pick one or more)?

Python is a general purpose language, widely used in the industry, which features powerful libraries for neuroimaging, such as nilearn and nipype.



Matlab is widely used in the neuroimaging community, and includes packages of reference such as SPM or EEGlab. It is proprietary but has a free, open-source "clone": GNU Octave.

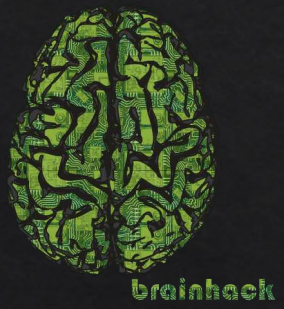
R is the language of statistics. It features an incredibly rich catalogue of statistical tools.

Step 1: Learn how to code (a bit)

Lots of great learning resources exist online and offline:

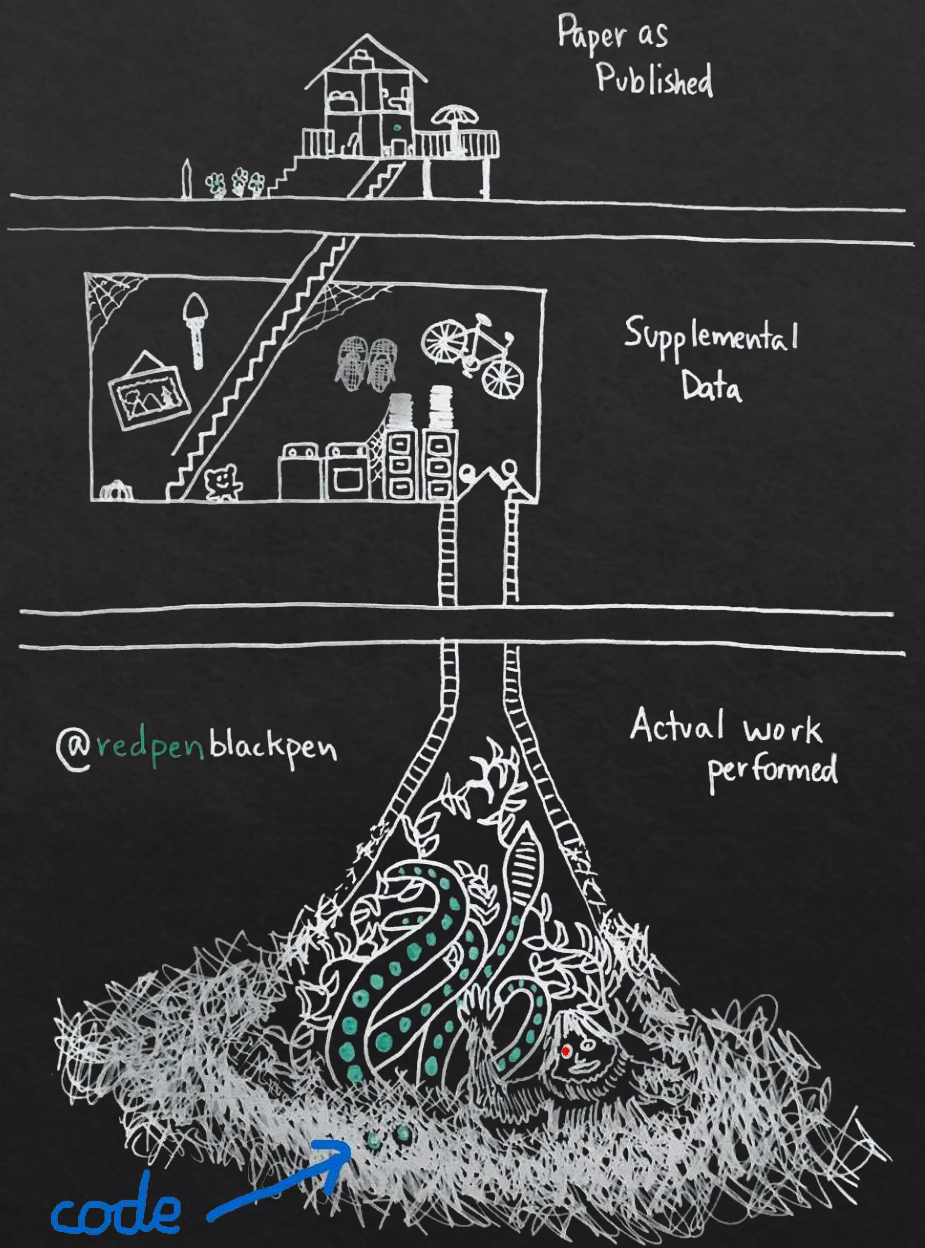
Software carpentry offers many high quality online and offline tutorials.

Neurostars is a good online forum if you have questions.



Brainhack is a series of hackathons where you can learn from peers by collaborating on projects. The brainhack 101 tutorial series at HBM cover many of the basics to get started.

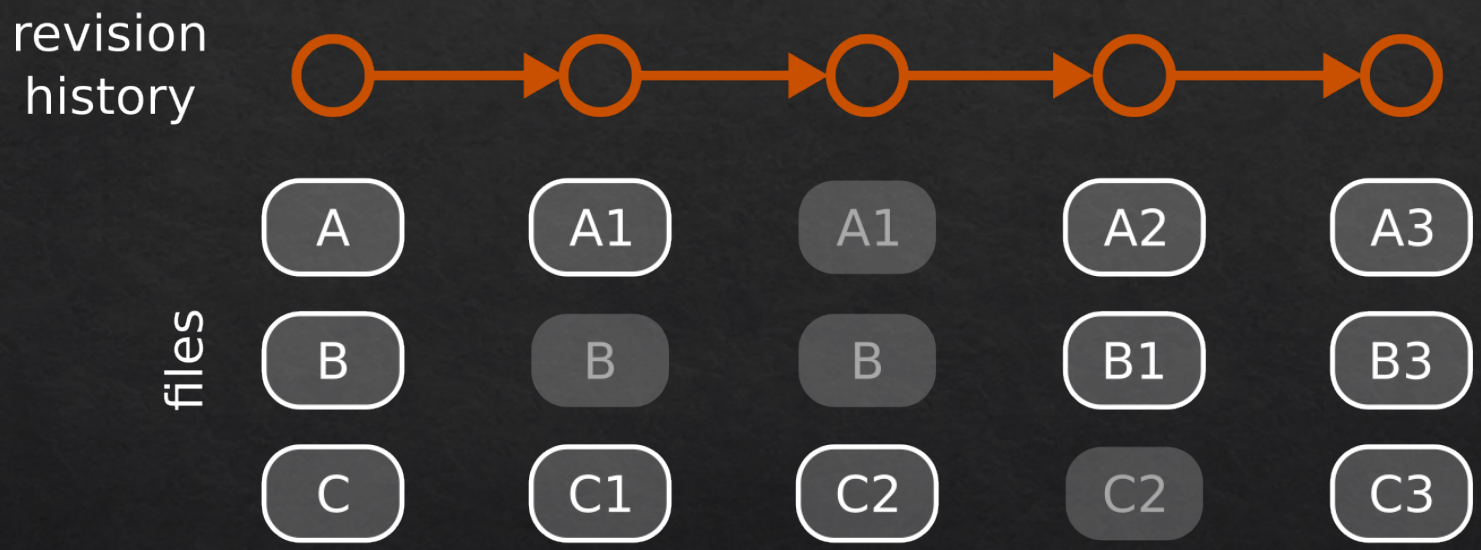
Step 2: Control the versions of your code



Step 2: Control the versions of your code

Version control: How does it work?

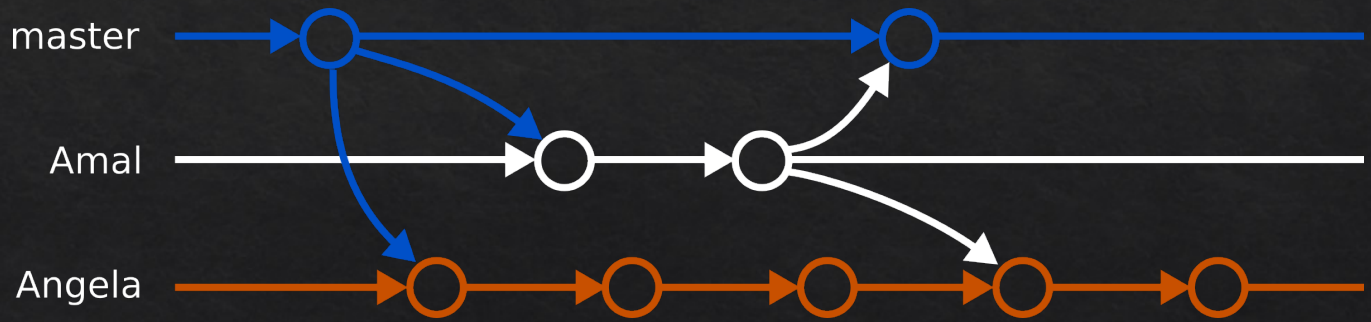
Every ***commit*** is saving the state of the files in your project



Only the changes made to files are actually stored

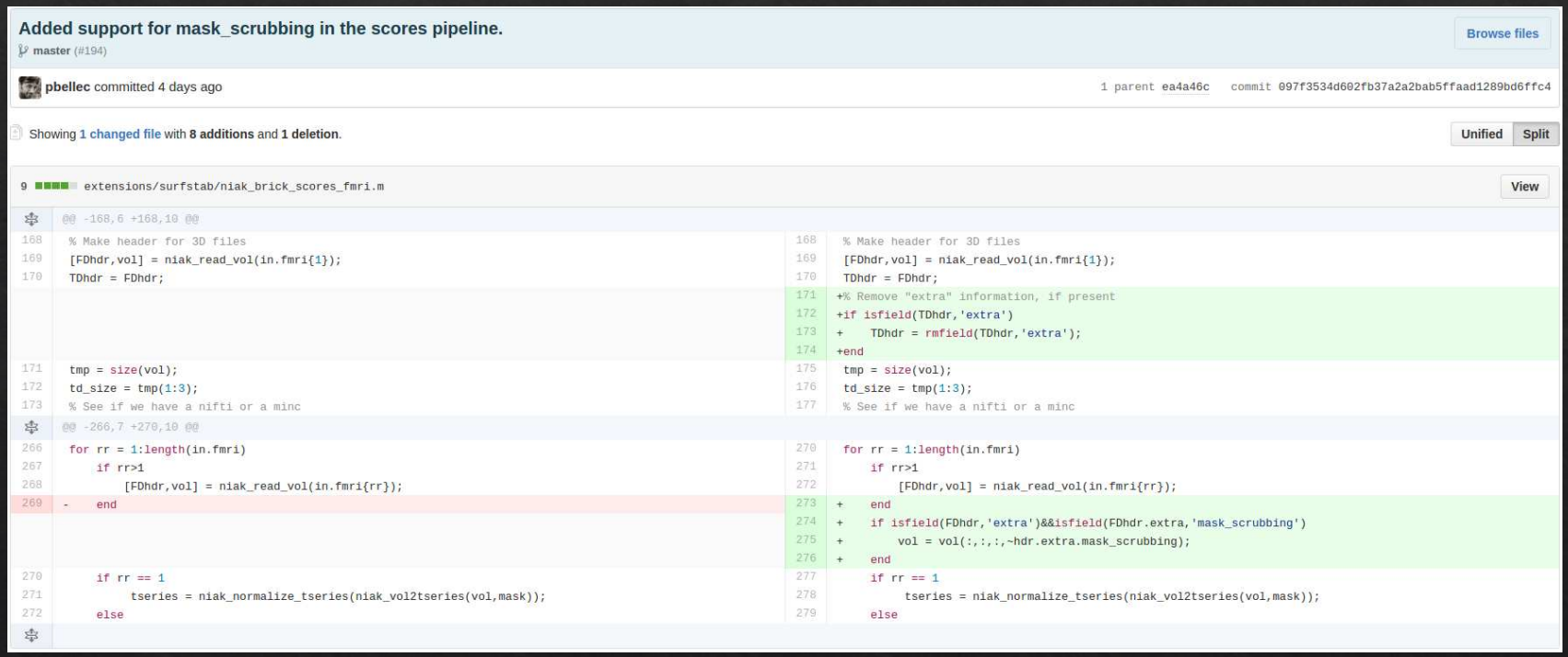
Step 2: Control the versions of your code

The Github platform, based on the git version control system, hosts projects for free, and enable users to easily (sort of) **branch** and **merge** code revisions. Git kraken adds an easy-to-use desktop GUI for git.



Step 2: Control the versions of your code

Github provides tools to easily compare versions of the code.



Added support for mask_scrubbing in the scores pipeline.

master (#194)

Browse files

pbellec committed 4 days ago — 1 parent ea4a46c — commit 097f3534d602fb37a2a2bab5ffaad1289bd6ffc4

Showing 1 changed file with 8 additions and 1 deletion.

Unified | Split

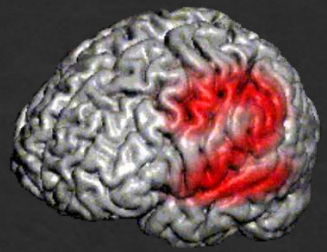
9 extensions/surfstab/niak_brick_scores_fmri.m — View

```
@@ -168,6 +168,10 @@
 % Make header for 3D files
 [FDhdr,vol] = niak_read_vol(in.fmri{1});
 TDhdr = FDhdr;
+% Remove "extra" information, if present
+if isfield(TDhdr,'extra')
+    TDhdr = rmfield(TDhdr,'extra');
+end
 tmp = size(vol);
 td_size = tmp(1:3);
 % See if we have a nifti or a minc
@@ -266,7 +270,10 @@
 for rr = 1:length(in.fmri)
     if rr>1
         [FDhdr,vol] = niak_read_vol(in.fmri{rr});
-    end
+    end
+    if isfield(FDhdr,'extra')&&isfield(FDhdr.extra,'mask_scrubbing')
+        vol = vol(:,:,:,~hdr.extra.mask_scrubbing);
+    end
     if rr == 1
         tseries = niak_normalize_tseries(niak_vol2tseries(vol,mask));
     else
```

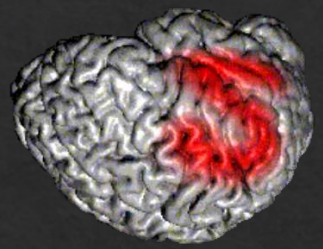
Step 3: Control your environment

SPM

SPM 95, 96, 99



SPM2



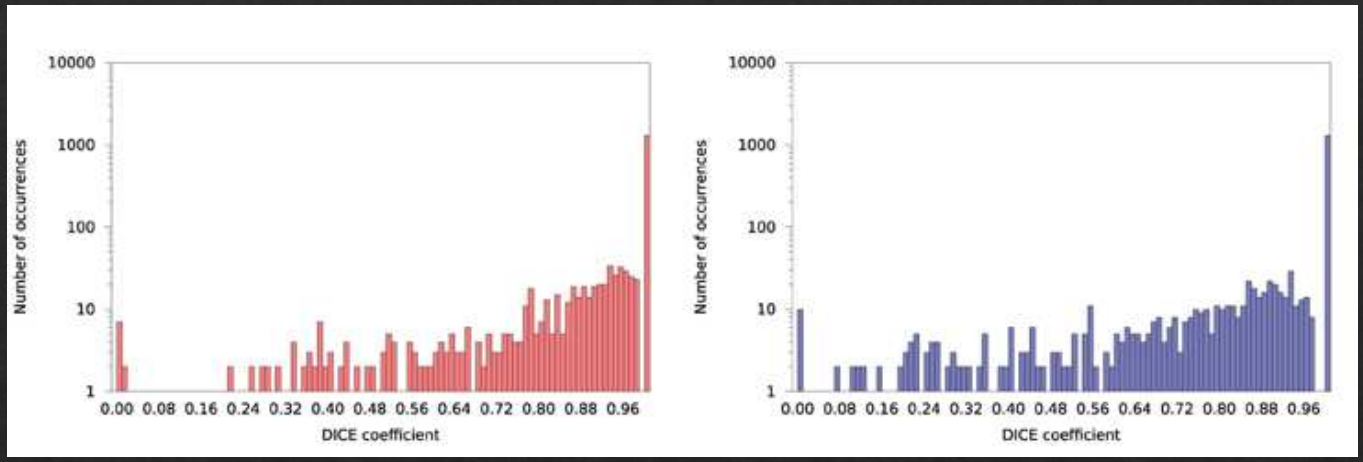
SPM5

SPM8

SPM12

Logos alone already suggest substantial differences across versions of SPM. There are also minor updates, e.g. 9 for SPM8. See, e.g., Malone et al., Neuroimage 2015 for a comparison of brain volumes in SPM8 and SPM12.

Step 3: Control your environment

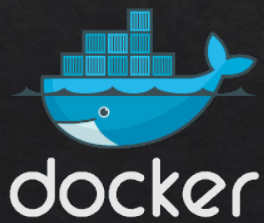


DICE coefficient between matched components from an ICA decomposition across two runs executed on the same system. (1) automatic detection of the number of components; (2) same random seeds; (3) different system libraries (libmath). From Glatard et al., Fontiers in Neuroinformatics 2015.

Step 3: Control your environment

Powerful tools exist to control your production environment:

Neurodebian offers a large catalogue of neuroimaging packages based on the Debian Operating System. It is possible to freeze an entire production environment in a virtual machine, which can then be re-used and shared.



Docker offers a way to build containers where each software has its own dedicated, controlled environment. Containers are easier to build and update than virtual machines, and are also more lightweight.

Step 4: Share your code

For your research to be reproducible, your code needs to be accessible to others, long-term:

The Zenodo data repository offers to publicly archive code, long term and for free. They have partnered with Github for seamless integration.

Archiving through Zenodo, or Figshare, automatically associate your code with a digital object identifier (DOI), which makes it fully citable for proper attribution.

Make clear for others under what terms they can use or modify your code. The MIT and BSD licenses are go-to options, as they impose minimal constraints.

Step 4: Share your code

It is very useful to document the content of the code repository. A simple markdown (text) file `README.md` is enough. Github will make it pretty, e.g. https://github.com/SIMEXP/mcinet:

GLM-connectome analysis

- The preprocessing pipelines for ADNI2, MNI, CRIUGMa, and CRIUGMb
- The region-growing pipeline
- The Boostrap Analysis of Stable Clusters pipeline, with regular grid of resolutions.
- The Boostrap Analysis of Stable Clusters pipeline, with resolutions selected by MSTEPS.
- The Multiscale Statistical Parametric Connectome pipeline, with resolutions selected by MSTEPS

Scripts for a previous version of the above analysis can be found below:

- The Multiscale Statistical Parametric Connectome pipeline, with regular grid of resolutions.
- The Multiscale Statistical Parametric Connectome pipeline, with resolutions selected by MSTEPS

This analysis did not model different scanner models in ADNI2 and instead treated ADNI2 as a single site. The results of this analysis can be found in the following preprint:

A. Tam, C. Dansereau, A. Badhwar, P. Orban, S. Belleville, H. Chertkow, A. Dagher, A. Hanganu, O. Monchi, P. Rosa-Neto, A. Shmuel, S. Wang, J. Breitner, P. Bellec, for the Alzheimer's Disease Neuroimaging Initiative *Consistent inter-protocol differences in resting-state functional connectomes between normal aging and mild cognitive impairment* http://dx.doi.org/10.1101/019646

Step 4: Share your code

Instead of sharing bare scripts, Jupyter notebooks can be used to mix text, code and the output figures in a readable format. Jupyter notebooks support Python, R and Octave, amongst many others. A notebook is ideal to implement and share interactive analysis. Github supports netbooks and will render them online.

jupyter XP_real_data_reproducibility Last Checkpoint: Last Wednesday at 3:57 PM (unsaved changes)

File Edit View Insert Cell Kernel Help Octave

Code CellToolbar

Read data

Let's start by grabbing a small dataset.

In [2]:
```
path_work = ['/home/pbellec/tmp/XP_real_data_reproducibility'];
psom_mkdir(path_work)
cd(path_work)
niak_wget('single_subject_cambridge_preprocessed_nii')
```

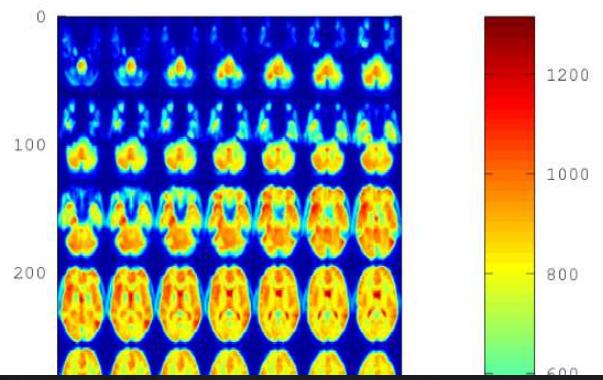
```
ans =  1
ans = 0
```

Check size and visualize one volume:

In [3]:
```
[hdr,vol] = niak_read_vol([path_work '/single_subject_cambridge_preprocessed_nii/fmri_sub00156_session1_rest.nii.gz'
size(vol)
niak_montage(vol(:,:,:,1))
```

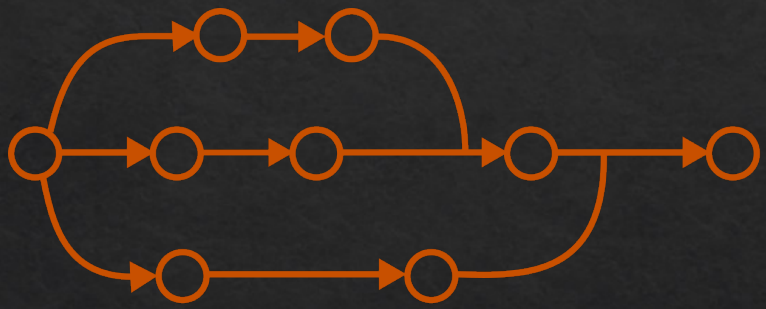
```
ans =

   53   64   52   98
```

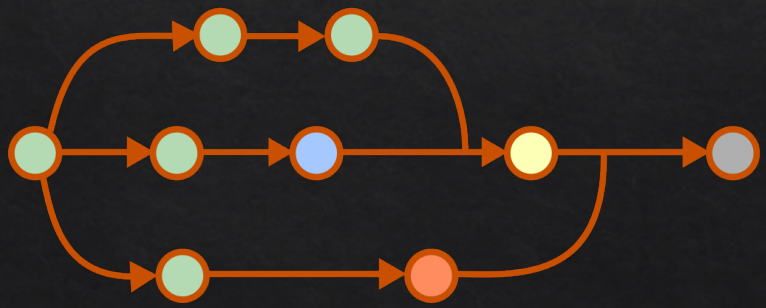


Step 5: Use a pipeline system

For long analyses composed of many steps, called pipelines or workflows, it is possible to automate and accelerate the work using a **pipeline system**.



Composition: specify nodes (typically running a job) and arrows (typically data flowing from one job to another)



Execution: jobs will run in an order that depends on the structure of the pipelines, as well as the resources available to the pipeline engine.

Done Running To do

Failed Unavailable

Example of pipeline composition: PSOM

I develop the pipeline system for Octave and Matlab (PSOM), a lightweight open-source (MIT) package, with no dependency. See Bellec et al. Frontiers in Neuroinformatics 2012. Here is how you would describe a job:

Job

A structure with the following fields:

- `command`: (mandatory) the command executed by the job.
- `files_in`: (optional) input files.
- `files_out`: (optional) output files.
- `files_clean`: (optional) files deleted by the job.
- `opt`: (optional) some arbitrary parameters.

Example of pipeline composition: PSOM

You simply compose a pipeline by adding jobs as fields in a structure:

Job "sample": No input, generate a random vector *a*

```
pipe.sample.command      = 'a = randn([opt.nb_samps 1]); save(files_out,''a'')';
pipe.sample.files_out    = '/home/pbellec/tmp/sample.mat';
pipe.sample.opt.nb_samps = 10;
```

Job "quadratic": Compute a^2 and save the results

```
pipe.quadratic.command   = 'load(files_in); b = a.^2; save(files_out,''b'')';
pipe.quadratic.files_in  = pipe.sample.files_out;
pipe.quadratic.files_out = '/home/pbellec/tmp/quadratic.mat';
```

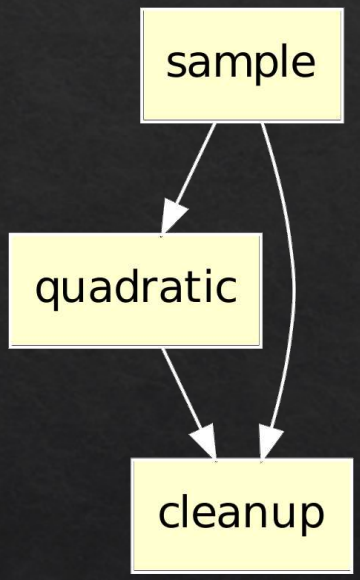
Job "cleanup": delete the output of "sample"

```
pipe.cleanup.command     = 'delete(files_clean)';
pipe.cleanup.files_clean = '/home/pbellec/tmp/sample.mat';
```

Example of pipeline composition: PSOM

Visualize the dependency graph of the pipeline

```
psom_visu_dependencies(pipe)
```



- `sample` → `quadratic`: because `quadratic` uses a file generated by `sample`.
- `sample` → `cleanup`: because `sample` generates a file deleted by `clean-up`.
- `quadratic` → `cleanup`: because `quadratic` uses a file deleted by `clean-up`.

Pipeline engines: features

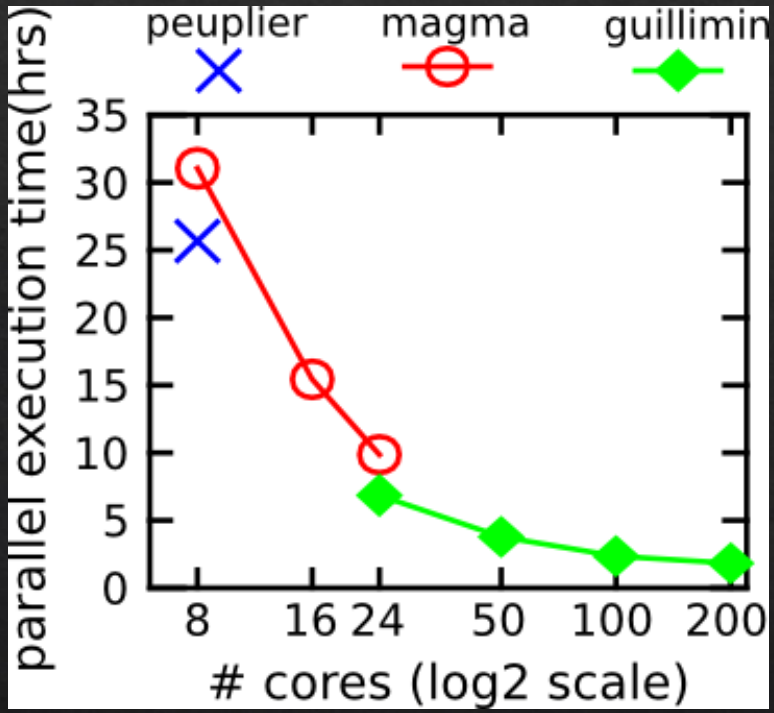
Relatively common features of pipeline engines, including PSOM:

- **Parallel computing**: Detection and execution of parallel components in the pipeline. The same code can run in a variety of execution environments (local, multi-thread, cluster).
- **Provenance tracking**: Generation of a comprehensive record of the pipeline stages and the history of execution.
- **Fault tolerance**: Multiple attempts will be made to run each job before it is considered as failed. Failed jobs can be automatically re-started.
- **Smart updates**: When an analysis is started multiple times, only the parts of the pipeline that need to be reprocessed are executed.

The availabity and exact behaviour of these features will depend on the actual pipeline system.

Pipeline system: parallelization efficiency

A strength of PSOM is that it scales well.



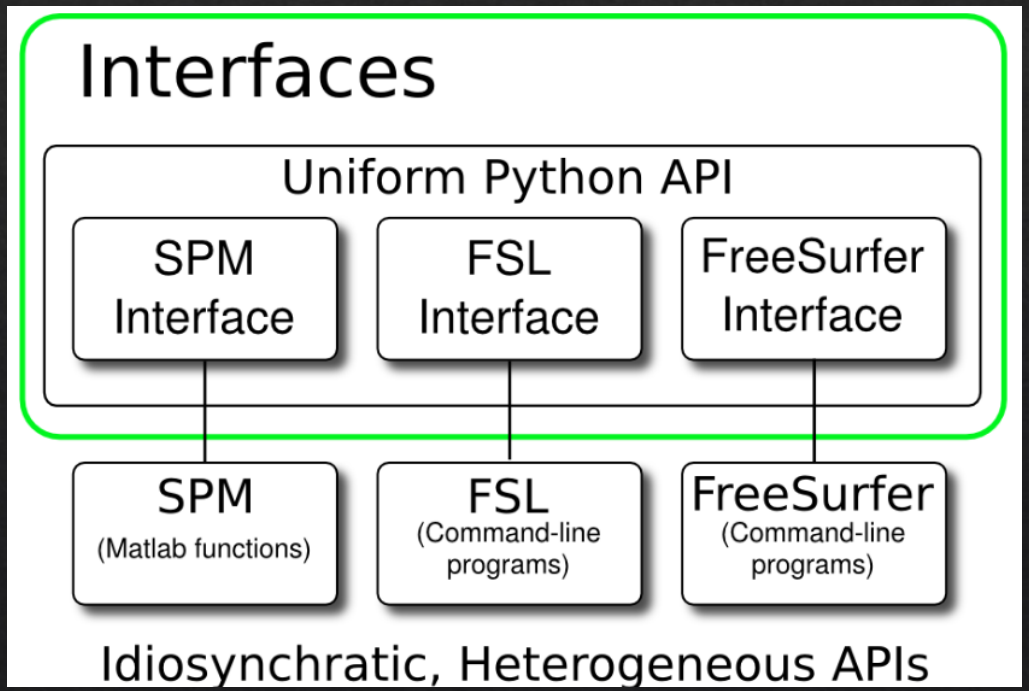
Adapted from Bellec et al. Front. in Neuroinf. 2012.

- Dataset Cambridge, 198 subjects with T1/fMRI.
- 5153 jobs / 7.7 Gb raw input / 21 Gb output / 8348 unique input/output files.
- `peuplier`: single machine (i7, 4 cores / 8 threads), local file system.
- `magma`: single machine (AMD, 24 cores), NFS file system.
- `guillimin`: supercomputer (Xeon, 14000 cores on 2011), infiniband parallel file system.

The PSOM 2.0 release, currently in beta, scales up to 1000s of cores / 10000s of jobs.

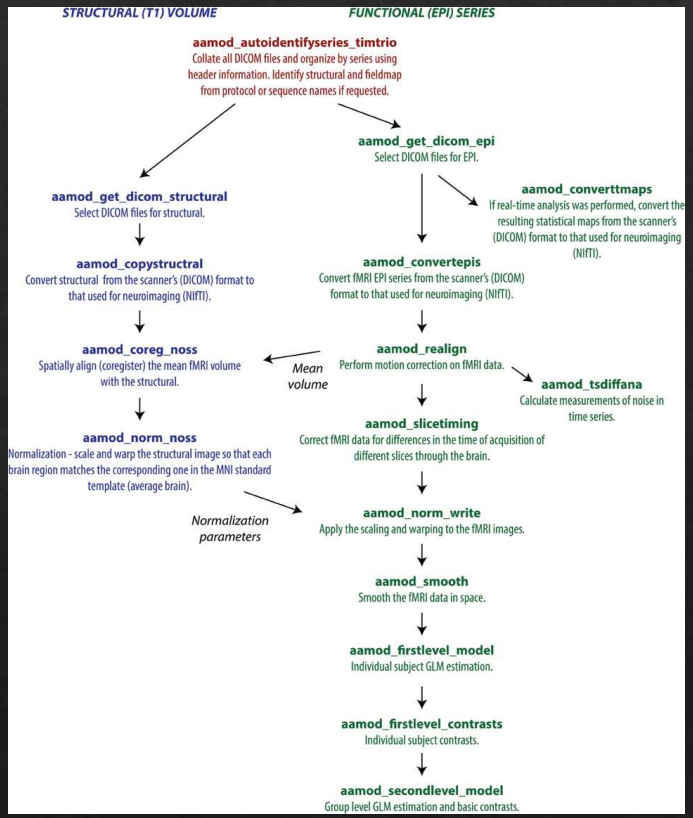
Pipeline system: catalogue of interfaces

Nipype is a Python library with mechanics fairly similar to PSOM. It does feature a vast catalogue of interfaces for most standard neuroimaging tools. See Gorgolewski et al., Frontiers in Neuroinformatics 2011.



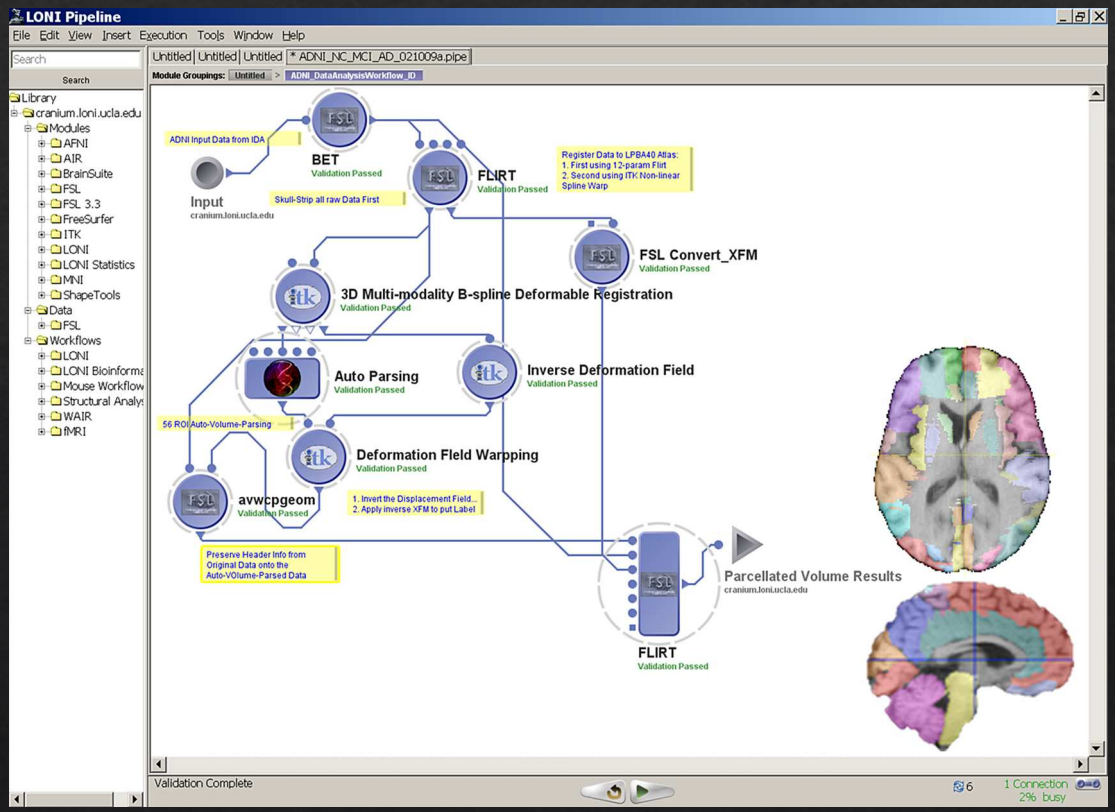
Pipeline system: catalogue of interfaces

The automatic analysis pipeline system also offers a catalogue of interfaces, in particular for SPM and EEGlab, in Matlab (but not Octave). See Cusak et al., Frontiers in Neuroinformatics 2014.



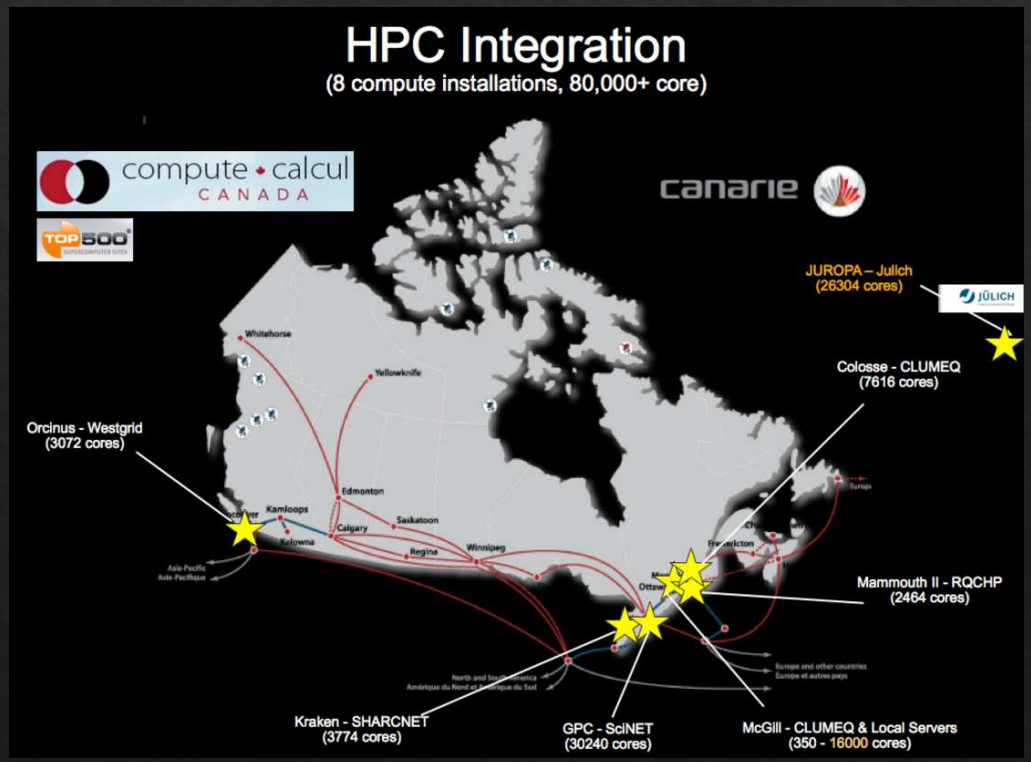
Pipeline system: graphical composition

The LONI pipeline enables composition through a GUI, using box and arrows and standard tools, e.g. FSL, Minc, etc. See Dinov et al., Frontiers in Neuroinformatics 2009.



Plaftorm for pipeline analysis

The CBRAIN webplatform offers a catalogue of mature, standard workflows. Data and processing are seamlessly distributed over a grid of high-performance computing facilities, and tools are encapsulated into containers for reproducibility. See Sherif et al., Frontiers in Neuroinformatics 2014.



Conclusions

Five concrete steps to improve reproducibility of neuroimaging pipeline analyses:

- Learn how to code (a bit).
- Control the versions of your code.
- Control your environment.
- Share your code.
- Use a pipeline system.

See Gorgolewski and Poldrack, BioRXiv 2016, for a short review on this topic.

The hackroom will feature short tutorials on a number of tools for neuroinformatics during HBM. Check the program and join the brainhack slack community for more info.