recipy Documentation

the recipy team

Nov 23, 2018

Contents:

CHAPTER 1

Installation

The easiest way to install is by simply running:

```
pip install recipy
```

Alternatively, you can clone this repository and run:

```
python setup.py install
```

If you want to install the dependencies manually (they should be installed automatically if you're following the instructions above) then run:

```
pip install -r requirements.txt
```

You can upgrade from a previous release by running:

```
pip install -U recipy
```

To find out what has changed since the last release, see the changelog

Note: Previous (unreleased) versions of recipy required MongoDB to be installed and set up manually. This is no longer required, as a pure Python database (TinyDB) is used instead. Also, the GUI is now integrated fully into recipy and does not require installing separately.

CHAPTER 2

User Manual

With the addition of a single line of code to the top of a Python script, recipy logs each run of your code to a database, keeping track of the input files, output files and the version of your code. It then lets you query this database to help you to recall the exact steps you took to create a certain output file.

2.1 Logging Provenance Information

To log provenance information, simply add the following line to the top of your code:

```
import recipy
```

Note that this **must** be the **very top** line of your script, before you import anything else.

Then just run your script as usual, and all of the data will be logged into the TinyDB database (don't worry, the database is automatically created if needed). You can then use the `recipy` command to quickly query the database to find out what run of your code produced what output file. So, for example, if you run some code like this:

```
import recipy
import numpy

arr = numpy.arange(10)
arr = arr + 500

numpy.save('test.npy', arr)
```

(Note the addition of `import recipy` at the beginning of script - but there are no other changes from a standard script.)

Alternatively, run an unmodified script with `python -m recipy SCRIPT [ARGS ...]` to enable recipy logging. This invokes recipy's module entry point, which takes care of import recipy for you, before running your script.

2.2 Retrieving Information about Runs

it will produce an output called `test.npy`. To find out the details of the run which created this file you can search using

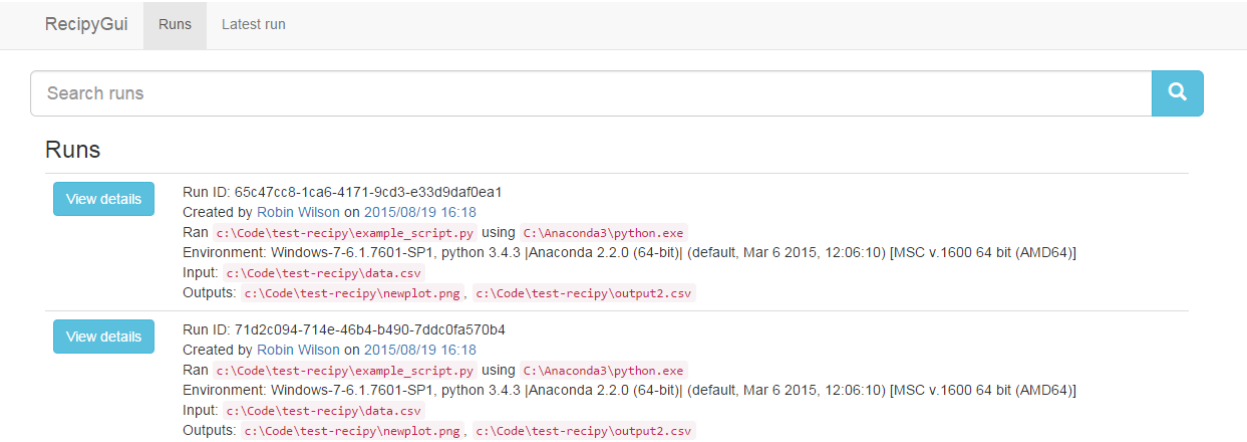
```
recipy search test.npy
```

and it will display information like the following:

```
Created by robin on 2015-05-25 19:00:15.631000
Ran /Users/robin/code/recipy/example_script.py using /usr/local/opt/python/bin/
↪python2.7
Git: commit 91a245e5ea82f33ae58380629b6586883cca3ac4, in repo /Users/robin/code/
↪recipy, with origin git@github.com:recipy/recipy.git
Environment: Darwin-14.3.0-x86_64-i386-64bit, python 2.7.9 (default, Feb 10 2015,␣
↪03:28:08)
Inputs:

Outputs:
  /Users/robin/code/recipy/test.npy
```

An alternative way to view this is to use the GUI. Just run `recipy gui` and a browser window will open with an interface that you can use to search all of your recipy 'runs':



2.3 Logging Files Using Built-In Open

If you want to log inputs and outputs of files read or written with built-in open, you need to do a little more work. Either use `recipy.open` (only requires `import recipy` at the top of your script), or add `from recipy import open` and just use `open`.

This workaround is required, because many libraries use built-in open internally, and you only want to record the files you explicitly opened yourself.

If you use Python 2, you can pass an `encoding` parameter to `recipy.open`. In this case `codecs` is used to open the file with proper encoding.

2.4 Annotating Runs

Once you've got some runs in your database, you can 'annotate' these runs with any notes that you want to keep about them. This can be particularly useful for recording which runs worked well, or particular problems you ran into. This can be done from the 'details' page in the GUI, or by running

```
recipy annotate [run-id]
```

which will open an editor to allow you to write notes that will be attached to the run. These will then be viewable via the command-line and the GUI when searching for runs.

2.5 Saving Custom Values

In your script, you can also add custom key-value pairs to the run:

```
recipy.log_values(key='value')
recipy.log_values({'key': 'value'})
```

Please note that, at the moment, these values are not displayed in the CLI or in the GUI.

2.6 Command Line Interface

There are other features in the command-line interface too: `recipy --help` to see the other options. You can view diffs, see all runs that created a file with a given name, search based on ids, show the latest entry and more:

```
recipy - a frictionless provenance tool for Python

Usage:
  recipy search [options] <outputfile>
  recipy latest [options]
  recipy gui [options]
  recipy annotate [<idvalue>]
  recipy pm [--format <rst|plain>]
  recipy (-h | --help)
  recipy --version

Options:
  -h --help         Show this screen
  --version         Show version
  -p --filepath     Search based on filepath rather than hash
  -f --fuzzy        Use fuzzy searching on filename
  -r --regex        Use regex searching on filename
  -i --id           Search based on (a fragment of) the run ID
  -a --all          Show all results (otherwise just latest result given)
  -v --verbose      Be verbose
  -d --diff         Show diff
  -j --json         Show output as JSON
  --no-browser      Do not open browser window
  --debug           Turn on debugging mode
```

2.7 Configuration

By default, recipy stores all of its configuration and the database itself in `~/.recipy`. Recipy's main configuration file is inside this folder, called `recipyrc`. The configuration file format is very simple, and is based on Windows INI files - and having a configuration file is completely optional: the defaults will work fine with no configuration file.

An example configuration is:

```
[ignored metadata]
diff

[general]
debug
```

This simply instructs recipy not to save `git diff` information when it records metadata about a run, and also to print debug messages (which can be really useful if you're trying to work out why certain functions aren't patched). At the moment, the only possible options are:

- `[general]`
  - `debug` - print debug messages
  - `editor = vi` - Configure the default text editor that will be used when recipy needs you to type in a message. Use notepad if on Windows, for example
  - `quiet` - don't print any messages
  - `port` - specify port to use for the GUI
- `[data]`
  - `file_diff_outputs` - store diff between the old output and new output file, if the output file exists before the script is executed
- `[database]`
  - `path = /path/to/file.json` - set the path to the database file
- `[ignored metadata]`
  - `diff` - don't store the output of `git diff` in the metadata for a recipy run
  - `git` - don't store anything relating to git (origin, commit, repo etc) in the metadata for a recipy run
  - `input_hashes` - don't compute and store SHA-1 hashes of input files
  - `output_hashes` - don't compute and store SHA-1 hashes of output files
- `[ignored inputs]`
  - List any module here (eg. `numpy`) to instruct recipy *not* to record inputs from this module, or `all` to ignore inputs from all modules
- `[ignored outputs]`
  - List any module here (eg. `numpy`) to instruct recipy *not* to record outputs from this module, or `all` to ignore outputs from all modules

By default all metadata is stored (ie. no metadata is ignored) and debug messages are not shown. A `.recipyrc` file in the current directory takes precedence over the `~/.recipy/recipyrc` file, allowing per-project configurations to be easily handled.

Note: No default configuration file is provided with recipy, so if you wish to configure anything you will need to create a properly-formatted file yourself.

CHAPTER 3

Patched Modules

This table lists the modules recipy has patches for, and the input and output functions that are patched.

Are you missing a patch for your favourite Python package? Learn how to *create a new patch*. We are looking forward to your pull request!

module_name	input_functions	output_functions
pandas	read_csv, read_table, read_excel, read_hdf, read_pickle, read_stata, read_msgpack	DataFrame.to_csv, DataFrame.to_excel, DataFrame.to_hdf, DataFrame.to_msgpack, DataFrame.to_stata, DataFrame.to_pickle, Panel.to_excel, Panel.to_hdf, Panel.to_msgpack, Panel.to_pickle, Series.to_csv, Series.to_hdf, Series.to_msgpack, Series.to_pickle
matplotlib.pyplot		savefig
numpy	genfromtxt, loadtxt, fromfile	save, savez, savez_compressed, savetxt
lxml.etree	parse, iterparse	
bs4	BeautifulSoup	
gdal	Open	Driver.Create, Driver.CreateCopy
sklearn	datasets.load_svmlight_file	datasets.dump_svmlight_file
nibabel	nifti1.Nifti1Image.from_filename, nifti2.Nifti2Image.from_filename, freesurfer.mghformat.MGHImage.from_filename, spm99analyze.Spm99AnalyzeImage.from_filename, minc1.Minc1Image.from_filename, minc2.Minc2Image.from_filename, analyze.AnalyzeImage.from_filename, parrec.PARRECImage.from_filename, spm2analyze.Spm2AnalyzeImage.from_filename	nifti1.Nifti1Image.to_filename, nifti2.Nifti2Image.to_filename, freesurfer.mghformat.MGHImage.to_filename, spm99analyze.Spm99AnalyzeImage.to_filename, minc1.Minc1Image.to_filename, minc2.Minc2Image.to_filename, analyze.AnalyzeImage.to_filename, parrec.PARRECImage.to_filename, spm2analyze.Spm2AnalyzeImage.to_filename
tifffile	imread	imsave
imageio	core.functions.get_reader, core.functions.read	core.functions.get_writer
netCDF4	Dataset	Dataset
xarray	open_dataset, open_mfdataset, open_rasterio, open_dataarray	Dataset.to_netcdf, DataArray.to_netcdf
iris	iris.load, iris.load_cube, iris.load_cubes, iris.load_raw	iris.save

To generate this table do:

```
recipy pm --format rst
```

CHAPTER 4

For Developers

4.1 How does it work?

For each run of your script, recipy logs information about which script was run, the command line arguments, Python version, git or svn repo, warnings, errors, and inputs and outputs (see *Database Schema* for a complete overview). Gathering most of this information is straightforward using the Python Standard Library and specialized packages (such as, GitPython or svn). Automatically logging inputs and outputs, recipy's most important feature, is more complicated.

When a Python module that reads or writes files is imported, recipy wraps methods for reading and writing files to log file paths to the database. To make this happen, recipy contains a patch for every library that reads or writes files. When you import recipy, the patches are added to `sys.meta_path` so they can be used to wrap a module's functions that read or write files when it is imported. This is why *import recipy* should be called before importing other modules.

Currently, recipy contains patches for many popular (scientific) libraries, including `numpy` and `pandas`. For an overview see *Patched Modules*. Is your favourite module missing? Have a look at *Creating Patches*. We are looking forward to your pull request!

4.2 Creating Patches

Patches are derived from `PatchImporter`, often using the easier interface provided by `PatchSimple`. To create a patch, you need to specify the module name, the input and output functions, and the argument referencing the file path.

4.2.1 Simple patches

Because most of the complexity is hidden away, the actual code to wrap a module is fairly simple. Using `PatchSimple`, the patch for numpy looks like:

```
1  from .PatchSimple import PatchSimple
2  from .log import log_input, log_output, add_module_to_db
3  from recipyCommon.utils import create_wrapper
4
5  # Inherit from PatchSimple
6  class PatchNumpy(PatchSimple):
7    # Specify the full name of the module
8    modulename = 'numpy'
9
10   # List functions that are involved in input/output
11   # these can be anything that can go after "modulename."
12   # so they could be something like "pyplot.savefig" for example
13   input_functions = ['genfromtxt', 'loadtxt', 'load', 'fromfile']
14   output_functions = ['save', 'savez', 'savez_compressed', 'savetxt']
15
16   # Define the functions that will be used to wrap the input/output
17   # functions.
18   # In this case we are calling the log_input function to log it to
19   # the DB and we are giving it the 0th argument from the function
20   # (because all of the functions above take the filename as the
21   # 0th argument), and telling it that it came from numpy.
22   input_wrapper = create_wrapper(log_input, 0, 'numpy')
23   output_wrapper = create_wrapper(log_output, 0, 'numpy')
24
25   # Add the module to the database, so we can list it.
26   add_module_to_db(modulename, input_functions, output_functions)
```

First, we import the required functionality (lines 1-3). In addition to `PatchSimple`, we need functions that do the actual logging (`log_input()` and `log_output()`), write details about the module to the database (`add_module_to_db()`), and a function to create wrappers (`create_wrapper()`). Next, we define a class for the patch that inherits from `PatchSimple` (line 6). In this class, we need to define the module name (line 8), and the input and output functions (line 13 and 14). Because the patches rely on class attributes, it is important to use the variable names specified in the example code. Then, we need to define the input and output wrapper (line 22 and 23). The wrappers are created using a predefined function, that needs as inputs a logging function (i.e., `log_input()` or `log_output()`), the index of the argument that specifies the file path in the input or output function, and the name of the module. Finally, the module is added to the database (line 26).

4.2.2 Enabling a patch

Patch objects for specific modules can be found in `PatchBaseScientific` and `PatchScientific`. To enable a new patch, one more step is required; the constructor of the new object should be called inside `multiple_insert()`. This function is called at the bottom of `PatchBaseScientific` and `PatchScientific`.

4.2.3 Patching more complex modules

Most of the patched modules are based on `PatchSimple`. However, some modules require more complexity. Some modules, e.g., netcdf4-python, use a file open like method for reading and writing files; the method called is the same, and whether the file is read or written depends on arguments:

```
1  from netCDF4 import Dataset
2
3  ds = Dataset('test.nc', 'r')  # read test.nc
4  ds = Dataset('test.nc', 'w')  # write test.nc
```

For this situation a different patch type and wrapper creation function are available: `PatchFileOpenLike` in combination with `create_argument_wrapper()`.

The complete code example (`PatchNetCDF4`):

```
1  from .PatchFileOpenLike import PatchFileOpenLike
2  from .log import log_input, log_output, add_module_to_db
3  from recipyCommon.utils import create_argument_wrapper
4
5  class PatchNetCDF4(PatchFileOpenLike):
6    modulename = 'netCDF4'
7
8    functions = ['Dataset']
9
10   wrapper = create_argument_wrapper(log_input, log_output, 0, 'mode', 'ra',
11                                     'aw', 'r', 'netCDF4')
12
13   add_module_to_db(modulename, functions, functions)
```

Another instance where it isn't possible to use `PatchSimple` is when not all input or output functions have the file path in the same position. For example, xarray has three functions for writing files, i.e., `to_netcdf()`, `to_netcdf()`, and `save_mfdataset()`. For `to_netcdf()` and `to_netcdf()`, the file paths are argument 0, as in the previous examples. However, the argument for the file paths of `save_mfdataset()` (it is a method for writing multiple files at once) is 1. With `PatchSimple` there is no way to represent this.

A patch class that allows specifying separate wrappers for different functions is `PatchMultipleWrappers`. Using this class involves defining a `WrapperList` to which inputs and outputs can be added. As can be seen in the following code example (`PatchXarray`), you can specify a wrapper for a list of functions (line 13 and 14) or for one function (line 15).

```
1  from .PatchMultipleWrappers import PatchMultipleWrappers, WrapperList
2  from .log import log_input, log_output, add_module_to_db
3
4  class PatchXarray(PatchMultipleWrappers):
5    modulename = 'xarray'
6
7    wrappers = WrapperList()
8
9    input_functions = ['open_dataset', 'open_mfdataset', 'open_rasterio',
10                      'open_dataarray']
11   output_functions = ['Dataset.to_netcdf', 'DataArray.to_netcdf']
12
13   wrappers.add_inputs(input_functions, log_input, 0, modulename)
14   wrappers.add_outputs(output_functions, log_output, 0, modulename)
15   wrappers.add_outputs('save_mfdataset', log_output, 1, modulename)
16
17   add_module_to_db(modulename, input_functions, output_functions)
```

4.2.4 Writing tests

If you make a new patch, please include tests (your pull request won't be accepted without them)! Recipy has a testing framework that checks whether inputs and outputs are actually logged when a function is called (see *Test Framework* for more details).

If you create a patch for a module, follow these steps to create tests:

1. Prepare small data files for testing, and create a directory `integration_test/packages/data/<module name>` containing these files.

- Small means kilobytes!

2. Create a test script for the patch. The name of the script should be `run_<module name>.py`.
   - It is probably easiest to copy one of the existing scripts in `integration_test/packages/`, so you can reuse the set up (it is pretty self-explanatory).
3. Write a test method for each input/output method.
   - Be sure to add docstrings!
4. Add the test configuration.
   - Open `integration_test/config/test_packages.yml`
   - Add a new section by typing:

```
---
script: run_<module name>.py
libraries: [ modulename ]
test_cases:
```

   - Add a test case for each method in `run_<module name>.py`, specifying `arguments`, a list containing the name of the test method, `inputs`, a list of the input files this method needs (if any), and `outputs`, a list of the output files this method creates (if any):

```
- arguments: [ <test method name> ]
  inputs: [<input names>]
  outputs: [<output names>]
```

   - Specified input files should exist in `integration_test/packages/data/<module name>`
   - Test cases can skipped by adding the line: `skip: "<reason for skipping>"`
   - If you need to skip a test, please add a description of the problem plus how to reproduce it in the *Recipy and Third-Party Package Issues* section
   - If a test case should be skipped in certain Python versions, add `skip_py_version: [ <python versions> ]`
5. Run the tests by typing: `py.test -v integration_test/`

4.3 Database Schema

Field	Description	Example
`unique_id`		ff129fee-c0d9-47cc-b0a9-997a530230a8
`author`	Username of the one running the script	
`description`	Currently not used?	
`inputs`	List of input files (can be empty)	
`outputs`	List of output files (can be empty)	
`script`	Path to the script that was run	
`command`	Path to the python executable that was used to run the script	`/usr/bin/python2.7`
`environment`	Information about the environment in which the script was run (including Python version)	Linux-4.15.0-29-generic-x86_64-with-debian-stretch-sid python 2.7.15 -Anaconda, Inc.- (default, May 1 2018, 23:32:55)
`date`	Date and time the script was run (in UTC)	
`command_args`	Command line arguments of the script (can be empty)	
`warnings`	Warnings issued during execution of the script (if any)	
`exception`	Exception raised during execution of the script (if any)	
`libraries`	Names and versions of patched libraries used in this script	
`notes`	Notes added by the user by running `recipy annotate` or in the gui	
`custom_values`	Values logged explicitly in the script by calling `recipy.log_values({'name': 'value'})`	
`gitrepo/svnrepo`		
`gitcommit/svncommit`	Current git/svn commit	
`gitorigin`		
`diff`		

4.4 Test Framework

recipy's test framework is in `integration_test`. The test framework has been designed to run under both Python 2.7+ and Python 3+.

4.4.1 Running Tests with py.test

The tests use the py.test test framework, and are run using its `py.test` command. Useful `py.test` flags and command-line options include:

- `-v`: increase verbosity. This shows the name each test function that is run.
- `-s`: show any output to standard output.
- `-rs`: show extra test summary information for tests that were skipped.
- `--junit-xml=report.xml`: create a JUnit-style test report in the file `report.xml`.

4.4.2 Running General Tests

To run tests of recipy's command-line functions, run:

```
py.test -v integration_test/test_recipy.py
```

To run tests of recipy's `.recipyrc` configuration file, run:

```
py.test -v integration_test/test_recipyrc.py
```

To run tests of recipy invocations using `python -m recipy script.py`, run:

```
py.test -v integration_test/test_m_flag.py
```

4.4.3 Running Package-Specific Tests

To run tests that check recipy logs information about packages it has been configured to log, run:

```
py.test -v -rs integration_test/test_packages.py
```

Note: this assumes that all the packages have been installed.

To run a single test, provide the name of the test, for example:

```
$ py.test -v -rs integration_test/test_packages.py::test_scripts\[run_numpy_py_
↪loadtxt\]
```

Note: `[` and `]` need to be prefixed by `\`.

Package-specific tests use a test configuration file located in `integration_test/config/test_packages.yml`.

You can specify a different test configuration file using a `RECIPY_TEST_CONFIG` environment variable. For example:

```
RECIPY_TEST_CONFIG=test_my_package.yml py.test -v -rs \
    integration_test/test_packages.py
```

Note: the above command should be typed on one line, omitting `\`.

For Windows, run:

```
set RECIPY_TEST_CONFIG=test_my_package.yml
py.test -v -rs integration_test\test_packages.py
```

Test Configuration File

The test configuration file is written in YAML (YAML Ain't Markup Language). YAML syntax is:

- `---` indicates the start of a document.
- `:` denotes a dictionary. `:` must be followed by a space.
- `-` denotes a list.

The test configuration file has format:

```
---
script: SCRIPT
standalone: True | False
libraries: [LIBRARY, LIBRARY, ... ]
test_cases:
- libraries: [LIBRARY, LIBRARY, ... ]
  arguments: [..., ..., ...]
  inputs: [INPUT, INPUT, ...]
  outputs: [OUTPUT, OUTPUT, ...]
- libraries: [LIBRARY, LIBRARY, ... ]
  arguments: [..., ..., ...]
  inputs: [INPUT, INPUT, ...]
  outputs: [OUTPUT, OUTPUT, ...]
  skip: "Known issue with recipy"
  skip_py_version: [3.4, ...]
- etc
---
script: SCRIPT
etc
```

Each script to be tested is defined by:

- `SCRIPT`: Python script, with a relative or absolute path. For recipy sample scripts (see below), the script is assumed to be in a sub-directory `integration_test/packages`.
- `standalone`: is the script a standalone script? If `False`, or if omitted, then the script is assumed to be a recipy sample script (see below).
- `libraries`: A list of zero or more Python libraries used by the script, which are expected to be logged by recipy when the script is run regardless of arguments (i.e. any libraries common to all test cases). If none, then this can be omitted.

Each script also has one or more test cases, each of which defines:

- `libraries`: A list of zero or more Python libraries used by the script, which are expected to be logged by recipy when the script is run with the given arguments for this test case. If none, then this can be omitted.
- `arguments`: A list of arguments to be passed to the script. If none, then this can be omitted.
- `inputs`: A list of zero or more input files which the script will read, and which are expected to be logged by recipy when running the script with the arguments. If none, then this can be omitted.
- `outputs`: A list of zero or more output files which the script will write, and which are expected to be logged by recipy when running the script with the arguments. If none, then this can be omitted.
- `skip`: An optional value. If present this test case is marked as skipped. The value is the reason for skipping the test case.
- **`skip_py_version\`** [An optional value. If present this test case is marked] as skipped if the current Python version is in the list of values. Should be used when a patched library does not support a Python version that is supported by recipy.

For example:

```
---
script: run_numpy.py
libraries: [numpy]
test_cases:
- arguments: [loadtxt]
  inputs: [input.csv]
```

(continues on next page)

(continued from previous page)

```
- arguments: [savetxt]
  outputs: [output.csv]
- arguments: [load_and_save_txt]
  inputs: [input.csv]
  outputs: [output.csv]
---
script: "/home/users/user/run_my_script.py"
standalone: True
test_cases:
- arguments: [ ]
  libraries: [ numpy ]
  outputs: [ data.csv ]
---
script: run_nibabel.py
libraries: [ nibabel ]
test_cases:
- arguments: [ analyze_from_filename ]
  inputs: [ analyze_image ]
- arguments: [ analyze_to_filename ]
  outputs: [ out_analyze_image ]
- arguments: [ minc1_from_filename ]
  inputs: [ minc1_image ]
- arguments: [ minc1_to_filename ]
  outputs: [ out_minc1_image ]
  skip: "nibabel.minc1.Minc1Image.to_filename raises NotImplementedError"
```

There may be a number of entries for a single script, if desired. For example:

```
---
script: run_numpy.py
libraries: [numpy]
test_cases:
- arguments: [loadtxt]
  inputs: [input.csv]
- arguments: [savetxt]
  outputs: [output.csv]
---
script: run_numpy.py
libraries: [numpy]
test_cases:
- arguments: [load_and_save_txt]
  inputs: [input.csv]
  outputs: [output.csv]
```

It is up to you to ensure the `library`, `input` and `output` file names record the libraries, input and output files used by the associated script, and which you expect to be logged by recipy.

Comments can be added to configuration files, prefixed by `#`, for example:

```
# This is a comment
```

4.4.4 Issues

The sample scripts in `integration_tests/packages` may fail to run with older versions of third-party packages. Known package versions that can cause failures are listed in Package versioning problems.

Certain third-party packages gave rise to issues, when attempting to configure the test framework for these. The packages and issues, and how the test framework has been configured to currently skip these are described in recipy and third-party package issues.

4.4.5 How the Test Framework uses Test Configuration Files

A test configuration file is used to auto-generate test functions for each test case using py.test's support for parameterization.

In the first example above, 8 test functions are created, 3 for `run_numpy.py` and 1 for `run_my_scripy.py` and 4 for `run_nibabel.py` (of which 1 is marked to be skipped. In the second example, 3 test functions are created, 2 for the first group of `run_numpy.py` test cases and 1 for the second group.

Test function names are auto-generated according to the following template:

```
test_scripts[SCRIPT_ARGUMENTS]
```

where `SCRIPT` is the `script` value and arguments the `argument` values, concatenated using underscores (_) and with all forward slashes, backslashes, colons, semi-colons and spaces also replaced by _. For example, `test_scripts[run_nibabel_py_analyze_from_filename]`.

The test framework runs the script with its arguments as follows. For recipy sample scripts:

```
python -m integration_test.package.SCRIPT ARGUMENTS
```

For scripts marked `standalone: True`:

```
python SCRIPT ARGUMENTS
```

Once the script has run, the test framework carries out the following checks on the recipy database:

- There is only one new run in the database i.e. number of logs has increased by 1.
- `script` refers to the same file as the `script`.
- `command_args` matches the test case's `arguments`.
- `libraries` matches all the test case's `libraries` and all the `libraries` common to all test case's for a script, and the recorded versions match the versions used when `script` was run.
- `inputs` match the test case's `inputs` (in terms of local file names).
- `outputs` match test case's `outputs` (in terms of local file names).
- `date` is a valid date.
- `exit_date` is a valid date and is <= `date`.
- `command` holds the current Python interpreter.
- `environment` holds the operating system and version of the current Python interpreter.
- `author` holds the current user.
- `description` is empty.

Recipy Sample Scripts

`integration_test/packages` has a collection of package-specific scripts. Each script corresponds to one package logged by recipy. Each script has a function that invokes each of the input/output functions of a specific package logged by recipy. For example, `run_numpy.py` has functions that invoke:

- `numpy.loadtxt`
- `numpy.savetxt`
- `numpy.fromfile`
- `numpy.save`
- `numpy.savez`
- `numpy.savez_compressed`
- `numpy.genfromtxt`

Each function is expected to invoke input and/or output functions using one or more functions which recipy can log.

Input and output files are the responsibility of each script itself. It can either create its own input and output files, or read these in from somewhere (but it is not expected that the caller (i.e. the test framework) create these.

Each test class has access to its own directory, via a `self.current_dir` field. It can use this to access any files it needs within the current directory or, by convention, within a sub-directory of `data` (for example `run_numpy.py` assumes a `data/numpy` sub-directory).

These scripts consist of classes that inherit from `integration_test.base.Base` which provides sub-classes with a simple command-line interface which takes a function name as argument and, if that function is provided by the script's class (and takes no arguments beyond `self`), invokes that function. For example:

```
python SCRIPT.py FUNCTION
```

A sample script can be run as follows:

```
python -m integration_test.packages.SCRIPT FUNCTION
```

For example:

```
python -m integration_test.packages.run_numpy loadtxt
```

`test_packages.py` assumes that if it is given a relative path to a script, then that script is in `integration_test/packages` and will create this form of invocation.

Running scripts as modules

Note that the script needs to be specified as a module that is run as a script (the `-m` flag). Running it directly as a script e.g.

```
$ python integration_test/packages/run_numpy.py loadtxt
```

will fail:

```
Traceback (most recent call last):
  File "integration_test/packages/run_numpy.py", line 17, in <module>
    from integration_test.packages.base import Base
ImportError: No module named 'integration_test.packages'
```

For the technical detail of why this is so, please see Execution of Python code with -m option or not.

4.4.6 Providing a Test Configuration for Any Script

A recipy test configuration can be written for any script that uses recipy. For example, to test a script that uses `numpy.loadtxt` you could write a configuration file which specifies:

- Full path to your script.
- Command-line arguments to be passed to your script.
- Libraries you expect to be logged by recipy.
- Local names of input files you expect to be logged by recipy.
- Local names of output files you expect to be logged by recipy.

For example, `my_tests.yml`:

```
---
script: "/home/ubuntu/sample/run_numpy.py"
standalone: True
test_cases:
- arguments: [ "/home/ubuntu/data/data.csv",
               "/home/ubuntu/data/out.csv" ]
  libraries: [ numpy ]
  inputs: [ data.csv ]
  outputs: [ out.csv ]
```

You can run this as follows:

```
RECIPY_TEST_CONFIG=my_tests.yml py.test -v -rs \
    integration_test/test_packages.py
```

The output might look like

```
============================= test session starts ==============================
platform linux2 -- Python 2.7.12, pytest-2.9.2, py-1.4.31, pluggy-0.3.1 -- /home/
↪ubuntu/anaconda2/bin/python
cachedir: .cache
rootdir: /home/ubuntu/recipy, inifile:
collected 1 items

integration_test/test_packages.py::test_scripts[run_numpy_py__home_ubuntu_data_data_
↪csv__home_ubuntu_data_out_csv] PASSED

=========================== 1 passed in 4.39 seconds ===========================
```

If using Anaconda and Git Bash on Windows, the file might look like:

```
---
script: "c:/Users/mjj/Local\ Documents/sample/run_numpy.py"
standalone: True
test_cases:
- arguments: [ "c:/Users/mjj/Local\ Documents/data/data.csv",
               "c:/Users/mjj/Local\ Documents/data/out.csv" ]
  libraries: [ numpy ]
  inputs: [ data.csv ]
  outputs: [ out.csv ]
```

Note the escaped spaces in the path.

4.4.7 Test Framework Limitations

The test framework does not support filtering tests depending upon which versions of packages are being tested e.g. specific versions of numpy or matplotlib. The test framework is designed to run tests within a single execution envi-

ronment: a Python interpreter and a set of installed libraries.

If wishing to test different versions of packages then this could be done by:

- Writing a Python script that invokes input/output functions of that package.
- Writing a test configuration file that just runs that script.
- Setting up a test environment (e.g. as part of a Travis CI or AppVeyor configuration file) that installs the specific package and runs `py.test integration_test/test_packages.py` using the test configuration file.

`test_recipy.py` does not validate whether multiple test results are returned by `recipy search -i`.

4.4.8 Recipy Dependencies

The test framework has no dependencies on any other part of the recipy repository: it uses recipy as if it were a stand-alone tool and queries the recipy database directly.

CHAPTER 5

Known Problems

5.1 Recipy and Third-Party Package Issues

Issues encountered during test framework development and how the test framework has been configured to get round these issues.

5.1.1 No information logged by recipy (recipy code is commented-out)

All these tests are marked as skipped.

`PIL` operations

To replicate:

```
python -m integration_test.packages.run_pil image_open
recipy latest
```

```
Run ID: cf695a6c-e9f0-4f4b-896f-9fa64127e2e8
Created by ubuntu on 2016-10-26 11:59:04 UTC
Ran /home/ubuntu/recipy/integration_test/packages/run_pil.py using /home/ubuntu/
↪anaconda2/bin/python
Using command-line arguments: image_open
Git: commit 5dc58a3cf3432441c83fd9899768eb9b63583208, in repo /home/ubuntu/recipy,␣
↪with origin https://mikej888@github.com/mikej888/recipy
Environment: Linux-3.19.0-25-generic-x86_64-with-debian-jessie-sid, python 2.7.12␣
↪|Anaconda custom (64-bit)| (default, Jul  2 2016, 17:42:40)
Libraries: recipy v0.3.0
Inputs: none
Outputs: none
```

`skimage` operations

To replicate:

```
python -m integration_test.packages.run_skimage io_imread
recipy latest
```

```
Run ID: 10c803f7-3741-4008-8548-2f8d7ba4462c
Created by ubuntu on 2016-10-26 11:57:07 UTC
Ran /home/ubuntu/recipy/integration_test/packages/run_skimage.py using /home/ubuntu/
↪anaconda2/bin/python
Using command-line arguments: io_imread
Git: commit 5dc58a3cf3432441c83fd9899768eb9b63583208, in repo /home/ubuntu/recipy,␣
↪with origin https://mikej888@github.com/mikej888/recipy
Environment: Linux-3.19.0-25-generic-x86_64-with-debian-jessie-sid, python 2.7.12␣
↪|Anaconda custom (64-bit)| (default, Jul  2 2016, 17:42:40)
Libraries: recipy v0.3.0
Inputs: none
Outputs: none
```

5.1.2 Inaccurate Information Logged by Recipy

None.

5.1.3 Bugs Arising within Recipy Logging

All these tests are marked as skipped.

`recipy.open`

To replicate:

```
python -m integration_test.packages.run_python open
```

Under Python 3 this fails with:

```
recipy run inserted, with ID 9abe4113-5158-4dcb-8fe8-afd1b1f6505e

Traceback (most recent call last):
  File "c:\Users\mjj\AppData\Local\Continuum\Anaconda3\lib\runpy.py", line 170,in _
↪run_module_as_main
    "__main__", mod_spec)
  File "c:\Users\mjj\AppData\Local\Continuum\Anaconda3\lib\runpy.py", line 85, in _
↪run_code
    exec(code, run_globals)
  File "c:\Users\mjj\Local Documents\recipy\recipy\integration_test\packages\run_
↪python.py", line 45, in <module>
    PythonSample().invoke(sys.argv)
  File "c:\Users\mjj\Local Documents\recipy\recipy\integration_test\packages\base.py",
↪ line 57, in invoke
    function()
  File "c:\Users\mjj\Local Documents\recipy\recipy\integration_test\packages\run_
↪python.py", line 39, in open
```

(continues on next page)

(continued from previous page)

```
    with recipy.open('out.txt', 'w') as f:
  File "c:\Users\mjj\Local Documents\recipy\recipy\recipy\utils.py", line 20, in open
    mode = kwargs['mode']
KeyError: 'mode'
```

Under Python 2 this fails with:

```
recipy run inserted, with ID 5d80b88b-0d56-428d-b9e0-d95eca423044

Traceback (most recent call last):
  File "/home/ubuntu/anaconda2/lib/python2.7/runpy.py", line 174, in _run_module_as_
↪main
    "__main__", fname, loader, pkg_name)
  File "/home/ubuntu/anaconda2/lib/python2.7/runpy.py", line 72, in _run_code
    exec code in run_globals
  File "/home/ubuntu/recipy/integration_test/packages/run_python.py", line 42, in
↪<module>
    python_sample.invoke(sys.argv)
  File "integration_test/packages/base.py", line 57, in invoke
    function()
  File "/home/ubuntu/recipy/integration_test/packages/run_python.py", line 35, in open
    with recipy.open('out.txt', 'w') as f:
  File "recipy/utils.py", line 35, in open
    log_output(args[0], 'recipy.open')
  File "recipy/log.py", line 153, in log_output
    db.update(append("libraries", get_version(source), no_duplicates=True), eids=[RUN_
↪ID])
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/tinydb/database.py", line␣
↪377, in update
    cond, eids
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/tinydb/database.py", line␣
↪230, in process_elements
    data = self._read()
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/tinydb/database.py", line␣
↪277, in _read
    return self._storage.read()
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/tinydb/database.py", line␣
↪31, in read
    raw_data = (self._storage.read() or {})[self._table_name]
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/tinydb_serialization/__
↪init__.py", line 139, in read
    data = self.storage.read()
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/tinydb/storages.py", line␣
↪93, in read
    self._handle.seek(0, 2)
ValueError: I/O operation on closed file
```

bs4.beautifulsoup.prettify

To replicate:

```
python -m integration_test.packages.run_bs4 beautifulsoup
```

```
Traceback (most recent call last):
  File "/home/ubuntu/anaconda2/lib/python2.7/runpy.py", line 174, in _run_module_as_
↪main
```

(continues on next page)

(continued from previous page)

```
    "__main__", fname, loader, pkg_name)
  File "/home/ubuntu/anaconda2/lib/python2.7/runpy.py", line 72, in _run_code
    exec code in run_globals
  File "/home/ubuntu/recipy/integration_test/packages/run_bs4.py", line 53, in
↪<module>
    Bs4Sample().invoke(sys.argv)
  File "integration_test/packages/base.py", line 57, in invoke
    function()
  File "/home/ubuntu/recipy/integration_test/packages/run_bs4.py", line 49, in␣
↪beautifulsoup
    print((soup.prettify()))
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/bs4/element.py", line 1160,
↪ in prettify
    return self.decode(True, formatter=formatter)
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/bs4/__init__.py", line 439,
↪ in decode
    return prefix + super(BeautifulSoup, self).decode(
TypeError: super() argument 1 must be type, not FunctionWrapper
```

pandas.Panel.to_excel

To replicate:

```
python -m integration_test.packages.run_pandas panel_to_excel
```

```
Traceback (most recent call last):
  File "/home/ubuntu/anaconda2/lib/python2.7/runpy.py", line 174, in _run_module_as_
↪main
    "__main__", fname, loader, pkg_name)
  File "/home/ubuntu/anaconda2/lib/python2.7/runpy.py", line 72, in _run_code
    exec code in run_globals
  File "/home/ubuntu/recipy/integration_test/packages/run_pandas.py", line 355, in
↪<module>
    PandasSample().invoke(sys.argv)
  File "integration_test/packages/base.py", line 57, in invoke
    function()
  File "/home/ubuntu/recipy/integration_test/packages/run_pandas.py", line 195, in␣
↪panel_to_excel
    panel.to_excel(file_name)
  File "recipyCommon/utils.py", line 91, in f
    return wrapped(*args, **kwargs)
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/pandas/core/panel.py",␣
↪line 460, in to_excel
    df.to_excel(writer, name, **kwargs)
  File "recipyCommon/utils.py", line 90, in f
    function(args[arg_loc], source)
  File "recipy/log.py", line 139, in log_output
    filename = os.path.abspath(filename)
  File "/home/ubuntu/anaconda2/lib/python2.7/posixpath.py", line 360, in abspath
    if not isabs(path):
  File "/home/ubuntu/anaconda2/lib/python2.7/posixpath.py", line 54, in isabs
    return s.startswith('/')
AttributeError: '_XlwtWriter' object has no attribute 'startswith'
```

nibabel.minc2.Minc2Image.from_filename

To replicate:

```
python -m integration_test.packages.run_nibabel minc2_from_filename
```

```
Traceback (most recent call last):
  File "/home/ubuntu/anaconda2/lib/python2.7/runpy.py", line 174, in _run_module_as_
↪main
    "__main__", fname, loader, pkg_name)
  File "/home/ubuntu/anaconda2/lib/python2.7/runpy.py", line 72, in _run_code
    exec code in run_globals
  File "/home/ubuntu/recipy/integration_test/packages/run_nibabel.py", line 302, in
↪<module>
    NibabelSample().invoke(sys.argv)
  File "integration_test/packages/base.py", line 57, in invoke
    function()
  File "/home/ubuntu/recipy/integration_test/packages/run_nibabel.py", line 143, in␣
↪minc2_from_filename
    data = nib.minc2.Minc2Image.from_filename(file_name)
  File "recipyCommon/utils.py", line 91, in f
    return wrapped(*args, **kwargs)
  File "recipyCommon/utils.py", line 91, in f
    return wrapped(*args, **kwargs)
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/nibabel/spatialimages.py",␣
↪line 699, in from_filename
    return klass.from_file_map(file_map)
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/nibabel/minc1.py", line␣
↪299, in from_file_map
    minc_file = Minc1File(netcdf_file(fobj))
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/nibabel/externals/netcdf.py
↪", line 230, in __init__
    self._read()
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/nibabel/externals/netcdf.py
↪", line 513, in _read
    self.filename)
TypeError: Error: None is not a valid NetCDF 3 file
```

nibabel.Nifti2Image.from_filename

To replicate:

```
python -m integration_test.packages.run_nibabel nifti2_from_filename
```

```
sizeof_hdr should be 348; set sizeof_hdr to 348
data code 0 not supported; not attempting fix
Traceback (most recent call last):
  File "/home/ubuntu/anaconda2/lib/python2.7/runpy.py", line 174, in _run_module_as_
↪main
    "__main__", fname, loader, pkg_name)
  File "/home/ubuntu/anaconda2/lib/python2.7/runpy.py", line 72, in _run_code
    exec code in run_globals
  File "/home/ubuntu/recipy/integration_test/packages/run_nibabel.py", line 302, in
↪<module>
    NibabelSample().invoke(sys.argv)
```

(continues on next page)

(continued from previous page)

```
  File "integration_test/packages/base.py", line 57, in invoke
    function()
  File "/home/ubuntu/recipy/integration_test/packages/run_nibabel.py", line 182, in␣
↪nifti2_from_filename
    data = nib.Nifti2Image.from_filename(file_name)
  File "recipyCommon/utils.py", line 91, in f
    return wrapped(*args, **kwargs)
  File "recipyCommon/utils.py", line 91, in f
    return wrapped(*args, **kwargs)
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/nibabel/keywordonly.py",␣
↪line 16, in wrapper
    return func(*args, **kwargs)
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/nibabel/analyze.py", line␣
↪986, in from_filename
    return klass.from_file_map(file_map, mmap=mmap)
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/nibabel/keywordonly.py",␣
↪line 16, in wrapper
    return func(*args, **kwargs)
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/nibabel/analyze.py", line␣
↪947, in from_file_map
    header = klass.header_class.from_fileobj(hdrf)
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/nibabel/nifti1.py", line␣
↪594, in from_fileobj
    hdr = klass(raw_str, endianness, check)
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/nibabel/nifti1.py", line␣
↪577, in __init__
    check)
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/nibabel/analyze.py", line␣
↪252, in __init__
    super(AnalyzeHeader, self).__init__(binaryblock, endianness, check)
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/nibabel/wrapstruct.py",␣
↪line 176, in __init__
    self.check_fix()
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/nibabel/wrapstruct.py",␣
↪line 361, in check_fix
    report.log_raise(logger, error_level)
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/nibabel/batteryrunners.py",
↪ line 275, in log_raise
    raise self.error(self.problem_msg)
nibabel.spatialimages.HeaderDataError: data code 0 not supported
```

`sklearn.load_svmlight_file` and `sklearn.dump_svmlight_file`

To replicate:

```
python -m integration_test.packages.run_sklearn load_svmlight_file
```

Under Python 3 this fails with:

```
Traceback (most recent call last):
  File "/home/ubuntu/anaconda3/lib/python3.5/runpy.py", line 184, in _run_module_as_
↪main
    "__main__", mod_spec)
  File "/home/ubuntu/anaconda3/lib/python3.5/runpy.py", line 85, in _run_code
    exec(code, run_globals)
```

(continues on next page)

(continued from previous page)

```
  File "/home/ubuntu/recipy/integration_test/packages/run_sklearn.py", line 16, in
↪<module>
    from sklearn import datasets
  File "<frozen importlib._bootstrap>", line 969, in _find_and_load
  File "<frozen importlib._bootstrap>", line 958, in _find_and_load_unlocked
  File "<frozen importlib._bootstrap>", line 664, in _load_unlocked
  File "<frozen importlib._bootstrap>", line 634, in _load_backward_compatible
  File "/home/ubuntu/recipy/recipy/PatchImporter.py", line 52, in load_module
    mod = self.patch(mod)
  File "/home/ubuntu/recipy/recipy/PatchSimple.py", line 25, in patch
    patch_function(mod, f, self.input_wrapper)
  File "/home/ubuntu/recipy/recipyCommon/utils.py", line 82, in patch_function
    setattr(mod, old_f_name, recursive_getattr(mod, function))
  File "/home/ubuntu/recipy/recipyCommon/utils.py", line 54, in recursive_getattr
    prev_part = getattr(prev_part, part)
AttributeError: module 'sklearn' has no attribute 'datasets'
```

Under Python 2 this fails with:

```
Traceback (most recent call last):
  File "recipy/log.py", line 165, in log_exception
    db.update({"exception": exception}, eids=[RUN_ID])
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/tinydb/database.py", line␣
↪382, in update
    cond, eids
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/tinydb/database.py", line␣
↪235, in process_elements
    func(data, eid)
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/tinydb/database.py", line␣
↪381, in <lambda>
    lambda data, eid: data[eid].update(fields),
KeyError: 316

Original exception was:
Traceback (most recent call last):
  File "/home/ubuntu/anaconda2/lib/python2.7/runpy.py", line 174, in _run_module_as_
↪main
    "__main__", fname, loader, pkg_name)
  File "/home/ubuntu/anaconda2/lib/python2.7/runpy.py", line 72, in _run_code
    exec code in run_globals
  File "/home/ubuntu/recipy/integration_test/packages/run_sklearn.py", line 16, in
↪<module>
    from sklearn import datasets
  File "recipy/PatchImporter.py", line 52, in load_module
    mod = self.patch(mod)
  File "recipy/PatchSimple.py", line 25, in patch
    patch_function(mod, f, self.input_wrapper)
  File "recipyCommon/utils.py", line 82, in patch_function
    setattr(mod, old_f_name, recursive_getattr(mod, function))
  File "recipyCommon/utils.py", line 54, in recursive_getattr
    prev_part = getattr(prev_part, part)
AttributeError: 'module' object has no attribute 'datasets'
Error in atexit._run_exitfuncs:
Traceback (most recent call last):
  File "/home/ubuntu/anaconda2/lib/python2.7/atexit.py", line 24, in _run_exitfuncs
    func(*targs, **kargs)
  File "recipy/log.py", line 244, in hash_outputs
```

(continues on next page)

(continued from previous page)

```
    for filename in run.get('outputs')]
AttributeError: 'NoneType' object has no attribute 'get'
Error in atexit._run_exitfuncs:
Traceback (most recent call last):
  File "/home/ubuntu/anaconda2/lib/python2.7/atexit.py", line 24, in _run_exitfuncs
    func(*targs, **kargs)
  File "recipy/log.py", line 231, in log_exit
    db.update({'exit_date': exit_date}, eids=[RUN_ID])
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/tinydb/database.py", line␣
↪382, in update
    cond, eids
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/tinydb/database.py", line␣
↪235, in process_elements
    func(data, eid)
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/tinydb/database.py", line␣
↪381, in <lambda>
    lambda data, eid: data[eid].update(fields),
KeyError: 316
Error in sys.exitfunc:
Error in sys.excepthook:
Traceback (most recent call last):
  File "recipy/log.py", line 165, in log_exception
    db.update({"exception": exception}, eids=[RUN_ID])
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/tinydb/database.py", line␣
↪382, in update
    cond, eids
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/tinydb/database.py", line␣
↪235, in process_elements
    func(data, eid)
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/tinydb/database.py", line␣
↪381, in <lambda>
    lambda data, eid: data[eid].update(fields),
KeyError: 316

Original exception was:
Traceback (most recent call last):
  File "/home/ubuntu/anaconda2/lib/python2.7/atexit.py", line 24, in _run_exitfuncs
    func(*targs, **kargs)
  File "recipy/log.py", line 231, in log_exit
    db.update({'exit_date': exit_date}, eids=[RUN_ID])
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/tinydb/database.py", line␣
↪382, in update
    cond, eids
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/tinydb/database.py", line␣
↪235, in process_elements
    func(data, eid)
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/tinydb/database.py", line␣
↪381, in <lambda>
    lambda data, eid: data[eid].update(fields),
KeyError: 316
```

5.1.4 Operations Not Implemented by Packages

All these tests are marked as skipped.

nibabel.minc1.Minc1Image.to_filename

To replicate:

```
python -m integration_test.packages.run_nibabel minc1_to_filename
```

```
Traceback (most recent call last):
  File "/home/ubuntu/anaconda2/lib/python2.7/runpy.py", line 174, in _run_module_as_
↪main
    "__main__", fname, loader, pkg_name)
  File "/home/ubuntu/anaconda2/lib/python2.7/runpy.py", line 72, in _run_code
    exec code in run_globals
  File "/home/ubuntu/recipy/integration_test/packages/run_nibabel.py", line 302, in
↪<module>
    NibabelSample().invoke(sys.argv)
  File "integration_test/packages/base.py", line 57, in invoke
    function()
  File "/home/ubuntu/recipy/integration_test/packages/run_nibabel.py", line 134, in␣
↪minc1_to_filename
    img.to_filename(file_name)
  File "recipyCommon/utils.py", line 91, in f
    return wrapped(*args, **kwargs)
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/nibabel/spatialimages.py",␣
↪line 781, in to_filename
    self.to_file_map()
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/nibabel/spatialimages.py",␣
↪line 790, in to_file_map
    raise NotImplementedError
NotImplementedError
```

nibabel.minc2.Minc2Image.to_filename

To replicate:

```
python -m integration_test.packages.run_nibabel minc2_to_filename
```

```
Traceback (most recent call last):
  File "/home/ubuntu/anaconda2/lib/python2.7/runpy.py", line 174, in _run_module_as_
↪main
    "__main__", fname, loader, pkg_name)
  File "/home/ubuntu/anaconda2/lib/python2.7/runpy.py", line 72, in _run_code
    exec code in run_globals
  File "/home/ubuntu/recipy/integration_test/packages/run_nibabel.py", line 302, in
↪<module>
    NibabelSample().invoke(sys.argv)
  File "integration_test/packages/base.py", line 57, in invoke
    function()
  File "/home/ubuntu/recipy/integration_test/packages/run_nibabel.py", line 154, in␣
↪minc2_to_filename
    img.to_filename(file_name)
  File "recipyCommon/utils.py", line 91, in f
    return wrapped(*args, **kwargs)
  File "recipyCommon/utils.py", line 91, in f
    return wrapped(*args, **kwargs)
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/nibabel/spatialimages.py",␣
↪line 781, in to_filename
```

(continues on next page)

(continued from previous page)

```
    self.to_file_map()
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/nibabel/spatialimages.py",␣
↪line 790, in to_file_map
    raise NotImplementedError
NotImplementedError
```

nibabel.parrec.PARRECImage.to_filename

To replicate:

```
python -m integration_test.packages.run_nibabel parrec_to_filename
```

```
Traceback (most recent call last):
  File "/home/ubuntu/anaconda2/lib/python2.7/runpy.py", line 174, in _run_module_as_
↪main
    "__main__", fname, loader, pkg_name)
  File "/home/ubuntu/anaconda2/lib/python2.7/runpy.py", line 72, in _run_code
    exec code in run_globals
  File "/home/ubuntu/recipy/integration_test/packages/run_nibabel.py", line 302, in
↪<module>
    NibabelSample().invoke(sys.argv)
  File "integration_test/packages/base.py", line 57, in invoke
    function()
  File "/home/ubuntu/recipy/integration_test/packages/run_nibabel.py", line 217, in␣
↪parrec_to_filename
    img.to_filename(par_file_name)
  File "recipyCommon/utils.py", line 91, in f
    return wrapped(*args, **kwargs)
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/nibabel/spatialimages.py",␣
↪line 781, in to_filename
    self.to_file_map()
  File "/home/ubuntu/anaconda2/lib/python2.7/site-packages/nibabel/spatialimages.py",␣
↪line 790, in to_file_map
    raise NotImplementedError
NotImplementedError
```

Using py.test and recipy

An issue that does not affect the test framework, but may affect future test development is that recipy and py.test do not integrate. For example, given test_sample.py:

```
class TestSample:

    def test_sample(self):
        pass
```

Running:

```
py.test test_sample.py
```

gives:

```
============================= test session starts ============================
platform win32 -- Python 3.5.1, pytest-3.0.2, py-1.4.31, pluggy-0.3.1
rootdir: c:\Users\mjj\Local Documents, inifile:
collected 1 items

test_sample.py .

========================== 1 passed in 0.02 seconds ==========================
```

Adding:

```
import recipy
```

Running py.test gives:

```
============================= test session starts ============================
platform win32 -- Python 3.5.1, pytest-3.0.2, py-1.4.31, pluggy-0.3.1
rootdir: c:\Users\mjj\Local Documents, inifile:
collected 0 items / 1 errors

================================== ERRORS ====================================
_____________________ ERROR collecting test_sample.py _____________________
..\appdata\local\continuum\anaconda3\lib\site-packages\_pytest\python.py:209: in fget
    return self._obj
E   AttributeError: 'Module' object has no attribute '_obj'

During handling of the above exception, another exception occurred:
test_sample.py:1: in <module>
    import recipy
..\appdata\local\continuum\anaconda3\lib\site-packages\recipy-0.3.0-py3.5.egg\recipy\_
↪_init__.py:12: in <module>
    log_init()
..\appdata\local\continuum\anaconda3\lib\site-packages\recipy-0.3.0-py3.5.
↪egg\recipy\log.py:74: in log_init
    add_git_info(run, scriptpath)
..\appdata\local\continuum\anaconda3\lib\site-packages\recipy-0.3.0-py3.5.
↪egg\recipyCommon\version_control.py:30: in add_git_info
    repo = Repo(scriptpath, search_parent_directories=True)
..\appdata\local\continuum\anaconda3\lib\site-packages\gitpython-2.0.8-py3.5.
↪egg\git\repo\base.py:139: in __init__
    raise NoSuchPathError(epath)
E   git.exc.NoSuchPathError:␣
↪c:\Users\mjj\AppData\Local\Continuum\Anaconda3\Scripts\py.test
!!!!!!!!!!!!!!!!!!!! Interrupted: 1 errors during collection !!!!!!!!!!!!!!!!!!!!
=========================== 1 error in 4.55 seconds ==========================
Error in atexit._run_exitfuncs:
Traceback (most recent call last):
  File "c:\users\mjj\appdata\local\continuum\anaconda3\lib\site-packages\recipy-0.3.0-
↪py3.5.egg\recipy\log.py", line 242, in hash_outputs
    run = db.get(eid=RUN_ID)
  File "c:\users\mjj\appdata\local\continuum\anaconda3\lib\site-packages\tinydb-3.2.1-
↪py3.5.egg\tinydb\database.py", line 432, in get
TypeError: unhashable type: 'dict'
Error in atexit._run_exitfuncs:
Traceback (most recent call last):
  File "c:\users\mjj\appdata\local\continuum\anaconda3\lib\site-packages\recipy-0.3.0-
↪py3.5.egg\recipy\log.py", line 231, in log_exit
```

(continues on next page)

(continued from previous page)

```
    db.update({'exit_date': exit_date}, eids=[RUN_ID])
  File "c:\users\mjj\appdata\local\continuum\anaconda3\lib\site-packages\tinydb-3.2.1-
↪py3.5.egg\tinydb\database.py", line 382, in update
  File "c:\users\mjj\appdata\local\continuum\anaconda3\lib\site-packages\tinydb-3.2.1-
↪py3.5.egg\tinydb\database.py", line 235, in process_elements
  File "c:\users\mjj\appdata\local\continuum\anaconda3\lib\site-packages\tinydb-3.2.1-
↪py3.5.egg\tinydb\database.py", line 381, in <lambda>
TypeError: unhashable type: 'dict'
```

5.2 Package Versioning Problems

The sample scripts in `integration_tests/packages` may fail to run with older versions of third-party packages. Known package versions that can cause failures are listed below.

These can arise due to differences in versions of packages (especially their APIs) installed by `conda`, `pip`, `easy_install` or within Python-related packages installed via `apt-get` or `yum` under Linux.

5.2.1 NiBabel `AttributeError`

```
$ python -m integration_test.packages.run_nibabel nifti2_from_filename
    data = nib.Nifti2Image.from_filename(file_name)
AttributeError: 'module' object has no attribute 'Nifti2Image'

$ python -m integration_test.packages.run_nibabel nifti2_to_filename
    img = nib.Nifti2Image(self.get_data(), self,get_affine())
AttributeError: 'module' object has no attribute 'Nifti2Image'

$ python -m integration_test.packages.run_nibabel minc1_from_filename
    data = nib.minc1.Minc1Image.from_filename(file_name)
AttributeError: 'module' object has no attribute 'minc1'

$ python -m integration_test.packages.run_nibabel minc1_to_filename
    img = nib.minc1.Minc1Image(self.get_data(), np.eye(4))
AttributeError: 'module' object has no attribute 'minc1'

$ python -m integration_test.packages.run_nibabel minc2_from_filename
    data = nib.minc2.Minc2Image.from_filename(file_name)
AttributeError: 'module' object has no attribute 'minc2'

$ python -m integration_test.packages.run_nibabel minc2_to_filename
    img = nib.minc2.Minc2Image(self.get_data(), np.eye(4))
AttributeError: 'module' object has no attribute 'minc2'

$ python -m integration_test.packages.run_nibabel parrec_from_filename
    data = nib.parrec.PARRECImage.from_filename(file_name)
AttributeError: 'module' object has no attribute 'parrec'

$ python -m integration_test.packages.run_nibabel parrec_to_filename
    img = nib.parrec.PARRECImage.from_filename(file_name)
AttributeError: 'module' object has no attribute 'parrec'
`
```

Fails on nibabel 1.2.2 (bundled in Ubuntu package `python-nibabel`) due to change in package API.

Succeeds on 2.0.2.

5.2.2 PIL `AttributeError`

```
$ python -m integration_test.packages.run_pil image_open
  File "/usr/lib/python2.7/dist-packages/PIL/Image.py", line 528, in __getattr__
    raise AttributeError(name)
AttributeError: __exit__

$ python -m integration_test.packages.run_pil image_save
...as above...
```

Fails on PIL/pillow 2.3.0 (bundled in Ubuntu package `python-pillow`) due to change in package API.

Succeeds on 3.2.0+.

5.2.3 pandas `TypeError`

```
$ python -m integration_test.packages.run_pandas read_excel
TypeError: read_excel() takes exactly 2 arguments (1 given)

$ python3 -m integration_test.packages.run_pandas read_excel
TypeError: read_excel() missing 1 required positional argument: 'sheetname'

$ python -m integration_test.packages.run_pandas read_hdf
TypeError: read_hdf() takes exactly 2 arguments (1 given)

$ python3 -m integration_test.packages.run_pandas read_hdf
TypeError: read_hdf() missing 1 required positional argument: 'key'
```

Fails on pandas 0.13.1 (bundled in Ubuntu package `python-pandas`) due to change in package API.

Succeeds on 0.18.1.

5.2.4 pandas `ImportError`

```
$ python -m integration_test.packages.run_pandas read_pickle
ImportError: No module named indexes.base

$ python3 -m integration_test.packages.run_pandas read_pickle
ImportError: No module named 'pandas.indexes'

During handling of the above exception, another exception occurred:
...
```

Fails on pandas 0.13.1 (bundled in Ubuntu package `python-pandas`) due to change in package API.

Succeeds on 0.18.1.

5.2.5 pandas `ValueError`

```
$ python -m integration_test.packages.run_pandas read_msgpack
ValueError: Unpack failed: error = -1

$ python3 -m integration_test.packages.run_pandas read_msgpack
...as above...
```

Fails on pandas 0.13.1 (bundled in Ubuntu package `python-pandas`) due to change in file format. The scripts work if run using data files created by pandas 0.13.1.

Succeeds on 0.18.1.

5.2.6 skimage `NameError`

```
$ python -m integration_test.packages.run_skimage io_load_sift
NameError: name 'file' is not defined

$ python -m integration_test.packages.run_skimage io_load_surf
NameError: name 'file' is not defined
```

Fails on Python 3 as `file()` built-in removed in Python -M Integration_Test.Packages.3 (see Builtins).

5.2.7 skimage `ImportError`

`run_skimage.py` examples fail with:

```
Traceback (most recent call last):
  File "run_skimage.py", line 13, in <module>
    from skimage import external
ImportError: cannot import name 'external'
```

Commenting out:

```
from skimage import external
```

allows non-external.tifffile examples to run.

Fails on skimage 0.9.3 (bundled in Ubuntu package `python-skimage`) due to change in package API.

Succeeds on 0.12.3.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search