nestly Documentation

Release 0.2

Erick Matsen et al.

March 25, 2014

Nestly is a small package designed to ease running software with combinatorial choices of parameters. It can easily do so for "cartesian products" of parameter choices, but can do much more– arbitrary "backwards-looking" dependencies can be used.

To find out more, look in the `examples/` subdirectory.

Contents:

CHAPTER 1

nestly Package

1.1 nestly Package

nestly is a collection of functions designed to make running software with combinatorial choices of parameters easier.

1.2 core Module

Core functions for building nests.

class nestly.core.**Nest**(*control_name='control.json'*, *indent=2*, *fail_on_clash=False*, *warn_on_clash=True*, *base_dict=None*)

Bases: object

Nests are used to build nested parameter selections, culminating in a directory structure representing choices made, and a JSON dictionary with all selections.

Build parameter combinations with Nest.add(), then create a nested directory structure with Nest.build().

Parameters

- **control_name** – Name JSON file to be created in each leaf
- **indent** – Indentation level in json file
- **fail_on_clash** – Error if a nest level attempts to overwrite a previous value
- **warn_on_clash** – Print a warning if a nest level attempts ot overwrite a previous value
- **base_dict** – Base dictionary to start all control dictionaries from (default: {})

add(*name*, *nestable*, *create_dir=True*, *update=False*, *label_func=<type 'str'>*, *template_subs=False*)

Add a level to the nest

Parameters

- **name** (*string*) – Name of the level. Forms the key in the output dictionary.
- **nestable** – Either an iterable object containing values, _or_ a function which takes a single argument (the control dictionary) and returns an iterable object containing values
- **create_dir** (*boolean*) – Should a directory level be created for this nestable?

- **update** (*boolean*) – Should the control dictionary be updated with the results of each value returned by the nestable? Only valid for dictionary results; useful for updating multiple values. At a minimum, a key-value pair corresponding to `name` must be returned.
- **label_func** – Function to be called to convert each value to a directory label.
- **template_subs** (*boolean*) – Should the strings in / returned by nestable be treated as templates? If true, str.format is called with the current values of the control dictionary.

build(*root='runs'*)
Build a nested directory structure, starting in `root`

Parameters **root** – Root directory for structure

iter(*root=None*)
Create an iterator of (directory, control_dict) tuples for all valid parameter choices in this `Nest`.

Parameters **root** – Root directory

Return type Generator of `(directory, control_dictionary)` tuples.

nestly.core.**control_iter**(*base_dir*, *control_name='control.json'*)
Generate the names of all control files under base_dir

nestly.core.**nest_map**(*control_iter*, *map_fn*)
Apply `map_fn` to the directories defined by `control_iter`

For each control file in control_iter, map_fn is called with the directory and control file contents as arguments.

Example:

```
>>> list(nest_map(['run1/control.json', 'run2/control.json'],
...               lambda d, c: c['run_id']))
[1, 2]
```

Parameters

- **control_iter** – Iterable of paths to JSON control files
- **map_fn** (*function*) – Function to run for each control file. It should accept two arguments: the directory of the control file and the json-decoded contents of the control file.

Returns A generator of the results of applying `map_fn` to elements in `control_iter`

nestly.core.**stripext**(*path*)
Return the basename, minus extension, of a path.

Parameters **path** (*string*) – Path to file

1.3 Subpackages

1.3.1 scripts Package

nestrun Module

nestrun.py - run commands based on control dictionaries.

class nestly.scripts.nestrun.**NestlyProcess**(*command*, *working_dir*, *popen*, *log_name='log.txt'*)
Bases: `object`

Metadata about a process run

complete(*return_code*)
Mark the process as complete with provided return_code

log_tail(*nlines=10*)
Return the last `nlines` lines of the log file

running_time

terminate()

`nestly.scripts.nestrun.`**extant_file**(*x*)
'Type' for argparse - checks that file exists but does not open.

`nestly.scripts.nestrun.`**invoke**(*max_procs*, *data*, *json_files*)

`nestly.scripts.nestrun.`**main**()

`nestly.scripts.nestrun.`**parse_arguments**()
Grab options and json files.

`nestly.scripts.nestrun.`**template_subs_file**(*in_file*, *out_fobj*, *d*)
Substitute template arguments in in_file from variables in d, write the result to out_fobj.

`nestly.scripts.nestrun.`**worker**(*data*, *json_file*)
Handle parameter substitution and execute command as child process.

`nestly.scripts.nestrun.`**write_summary**(*all_procs*, *summary_file*)
Write a summary of all run processes to summary_file in tab-delimited format.

`nestagg` Module

Aggregate results of `nestly` runs.

`nestly.scripts.nestagg.`**comma_separated_values**(*s*)

`nestly.scripts.nestagg.`**delim**(*arguments*)
Execute delim action.

Parameters **arguments** – Parsed command line arguments from `main()`

`nestly.scripts.nestagg.`**main**(*args=['-b', 'latex', '-D', 'language=en', '-d', '_build/doctrees', '.', '_build/latex']*)
Command-line interface for nestagg

`nestly.scripts.nestagg.`**warn**(*message*)

CHAPTER 2

Command line tools

2.1 nestrun

`nestrun` takes a command template and a list of control.json files with variables to substitute. Substitution is performed using the Python built-in `str.format` method. See the Python Formatter documentation for details on syntax, and `examples/jsonrun/do_nestrun.sh` for an example.

2.1.1 Help

```
usage: nestrun.py [-h] [-j N] [--template 'template text'] [--stop-on-error]
                  [--template-file FILE] [--save-cmd-file SAVECMD_FILE]
                  [--log-file LOG_FILE | --no-log] [--dry-run]
                  [--summary-file SUMMARY_FILE]
                  json_files [json_files ...]

nestrun - substitute values into a template and run commands in parallel.

positional arguments:
  json_files            Nestly control dictionaries

optional arguments:
  -h, --help            show this help message and exit
  -j N, --processes N, --local N
                        Run a maximum of N processes in parallel locally
                        (default: 2)
  --template 'template text'
                        Command-execution template, e.g. bash {infile}. By
                        default, nestrun executes the templatefile.
  --stop-on-error       Terminate remaining processes if any process returns
                        non-zero exit status (default: False)
  --template-file FILE  Command-execution template file path.
  --save-cmd-file SAVECMD_FILE
                        Name of the file that will contain the command that
                        was executed.
  --log-file LOG_FILE   Name of the file that will contain output of the
                        executed command.
  --no-log              Don't create a log file
  --dry-run             Dry run mode, does not execute commands.
  --summary-file SUMMARY_FILE
                        Write a summary of the run to the specified file
```

2.2 nestagg

The `nestagg` command provides a mechanism for combining results of multiple runs. Currently, the only supported action is merging delimited files from a set of leaves, adding values from the control dictionary on each.

2.2.1 Help

```
usage: nestagg.py delim [-h] [-k KEYS | -x EXCLUDE_KEYS] [-m {fail,warn}]
                        [-s SEPARATOR] [-t] [-o OUTPUT]
                        file_template control.json [control.json ...]

positional arguments:
  file_template         Template for the delimited file to read in each
                        directory [e.g. '{run_id}.csv']
  control.json          Control files

optional arguments:
  -h, --help            show this help message and exit
  -k KEYS, --keys KEYS  Comma separated list of keys from the JSON file to
                        include [default: all keys]
  -x EXCLUDE_KEYS, --exclude-keys EXCLUDE_KEYS
                        Comma separated list of keys from the JSON file not to
                        include [default: None]
  -m {fail,warn}, --missing-action {fail,warn}
                        Action to take when a file is missing [default: fail]
  -s SEPARATOR, --separator SEPARATOR
                        Separator [default: ,]
  -t, --tab             Files are tab-separated
  -o OUTPUT, --output OUTPUT
                        Output file [default: stdout]
```

CHAPTER 3

Project Modules

3.1 nestly Package

3.1.1 `nestly` Package

nestly is a collection of functions designed to make running software with combinatorial choices of parameters easier.

3.1.2 `core` Module

Core functions for building nests.

class `nestly.core.`**`Nest`**(*control_name='control.json'*, *indent=2*, *fail_on_clash=False*, *warn_on_clash=True*, *base_dict=None*)

Bases: `object`

Nests are used to build nested parameter selections, culminating in a directory structure representing choices made, and a JSON dictionary with all selections.

Build parameter combinations with `Nest.add()`, then create a nested directory structure with `Nest.build()`.

Parameters

- **control_name** – Name JSON file to be created in each leaf
- **indent** – Indentation level in json file
- **fail_on_clash** – Error if a nest level attempts to overwrite a previous value
- **warn_on_clash** – Print a warning if a nest level attempts ot overwrite a previous value
- **base_dict** – Base dictionary to start all control dictionaries from (default: `{}`)

`add`(*name*, *nestable*, *create_dir=True*, *update=False*, *label_func=<type 'str'>*, *template_subs=False*)

Add a level to the nest

Parameters

- **name** (*string*) – Name of the level. Forms the key in the output dictionary.
- **nestable** – Either an iterable object containing values, _or_ a function which takes a single argument (the control dictionary) and returns an iterable object containing values
- **create_dir** (*boolean*) – Should a directory level be created for this nestable?

- **update** (*boolean*) – Should the control dictionary be updated with the results of each value returned by the nestable? Only valid for dictionary results; useful for updating multiple values. At a minimum, a key-value pair corresponding to `name` must be returned.
- **label_func** – Function to be called to convert each value to a directory label.
- **template_subs** (*boolean*) – Should the strings in / returned by nestable be treated as templates? If true, str.format is called with the current values of the control dictionary.

build(*root='runs'*)
Build a nested directory structure, starting in `root`

Parameters **root** – Root directory for structure

iter(*root=None*)
Create an iterator of (directory, control_dict) tuples for all valid parameter choices in this `Nest`.

Parameters **root** – Root directory

Return type Generator of `(directory, control_dictionary)` tuples.

nestly.core.**control_iter**(*base_dir*, *control_name='control.json'*)
Generate the names of all control files under base_dir

nestly.core.**nest_map**(*control_iter*, *map_fn*)
Apply `map_fn` to the directories defined by `control_iter`

For each control file in control_iter, map_fn is called with the directory and control file contents as arguments.

Example:

```
>>> list(nest_map(['run1/control.json', 'run2/control.json'],
...               lambda d, c: c['run_id']))
[1, 2]
```

Parameters

- **control_iter** – Iterable of paths to JSON control files
- **map_fn** (*function*) – Function to run for each control file. It should accept two arguments: the directory of the control file and the json-decoded contents of the control file.

Returns A generator of the results of applying `map_fn` to elements in `control_iter`

nestly.core.**stripext**(*path*)
Return the basename, minus extension, of a path.

Parameters **path** (*string*) – Path to file

3.1.3 Subpackages

scripts Package

nestrun Module

nestrun.py - run commands based on control dictionaries.

class nestly.scripts.nestrun.**NestlyProcess**(*command*, *working_dir*, *popen*, *log_name='log.txt'*)

Bases: `object`

Metadata about a process run

complete(*return_code*)
Mark the process as complete with provided return_code

log_tail(*nlines=10*)
Return the last `nlines` lines of the log file

running_time

terminate()

`nestly.scripts.nestrun.`**extant_file**(*x*)
'Type' for argparse - checks that file exists but does not open.

`nestly.scripts.nestrun.`**invoke**(*max_procs*, *data*, *json_files*)

`nestly.scripts.nestrun.`**main**()

`nestly.scripts.nestrun.`**parse_arguments**()
Grab options and json files.

`nestly.scripts.nestrun.`**template_subs_file**(*in_file*, *out_fobj*, *d*)
Substitute template arguments in in_file from variables in d, write the result to out_fobj.

`nestly.scripts.nestrun.`**worker**(*data*, *json_file*)
Handle parameter substitution and execute command as child process.

`nestly.scripts.nestrun.`**write_summary**(*all_procs*, *summary_file*)
Write a summary of all run processes to summary_file in tab-delimited format.

`nestagg` Module

Aggregate results of `nestly` runs.

`nestly.scripts.nestagg.`**comma_separated_values**(*s*)

`nestly.scripts.nestagg.`**delim**(*arguments*)
Execute delim action.

Parameters **arguments** – Parsed command line arguments from `main()`

`nestly.scripts.nestagg.`**main**(*args=['-b', 'latex', '-D', 'language=en', '-d', '_build/doctrees', '.', '_build/latex']*)
Command-line interface for nestagg

`nestly.scripts.nestagg.`**warn**(*message*)

CHAPTER 4

Examples

4.1 Building Nests

4.1.1 Basic Nest

From `examples/basic_nest/make_nest.py`, this is a simple, combinatorial example.

```
#!/usr/bin/env python

import glob
import math
import os
import os.path
from nestly import Nest

wd = os.getcwd()
input_dir = os.path.join(wd, 'inputs')

nest = Nest()

# Simplest case: Levels are added with a name and an iterable
nest.add('strategy', ('exhaustive', 'approximate'))

# Items can update the control dictionary
nest.add('run_count', [{'run_count': 10**i, 'function': 'pow'}
                       for i in xrange(3)], update=True)

# label_func is applied to each item create a directory name
nest.add('input_file', glob.glob(os.path.join(input_dir, 'file*')),
        label_func=os.path.basename)

# Items can be added that don't generate directories
nest.add('base_dir', [os.getcwd()], create_dir=False)

# Any function taking one argument (control dictionary) and returning an
# iterable may also be used:
def log_run_count(c):
    run_count = c['run_count']
    return [math.log(run_count, 10)]
nest.add('run_count_log', log_run_count, create_dir=False)

nest.build('runs')
```

4.1.2 Meal

This is quite a bit more complicated, with lookups on previous values of the control dictionary:

```
#!/usr/bin/env python

import glob
import os
import os.path

from nestly import Nest, stripext

wd = os.getcwd()
startersdir = os.path.join(wd, "starters")
winedir = os.path.join(wd, "wine")
mainsdir = os.path.join(wd, "mains")

nest = Nest()

bn = os.path.basename

# start by mirroring the two directory levels in startersdir, and name those
# directories "ethnicity" and "dietary"
nest.add('ethnicity', glob.glob(os.path.join(startersdir, '*')),
    label_func=bn)
nest.add('dietary', lambda c: glob.glob(os.path.join(c['ethnicity'], '*')),
    label_func=bn)

## now get all of the starters
nest.add('starter', lambda c: glob.glob(os.path.join(c['dietary'], '*')),
    label_func=stripext)
## now get the corresponding mains
nest.add('main', lambda c: [os.path.join(mainsdir, bn(c['ethnicity']) + "_stirfry.txt")],
    label_func=stripext)

## get only the tasty wines
nest.add('wine', glob.glob(os.path.join(winedir, '*.tasty')),
    label_func=stripext)
## the wineglasses should be chosen by the wine choice, but we don't want to
## make a directory for those.
nest.add('wineglass', lambda c: [stripext(c['wine']) + ' wine glasses'],
        create_dir=False)

nest.build('runs')
```

CHAPTER 5

Indices and tables

- *genindex*
- *modindex*
- *search*

n