ChunkyPipes Documentation

Release 0.2.3

Dominic Fitzgerald

Jul 19, 2017

Contents

ChunkyPipes is a framework for easily designing and distributing NGS pipelines written in Python.

Running a pipeline with ChunkyPipes can be as simple as:

```
$ chunky install ngs-pipeline.py
$ chunky configure ngs-pipeline
$ chunky run ngs-pipeline [arguments]
```

CHAPTER 1

Table of Contents

Getting Started

To install with pip:

```
$ pip install chunkypipes
```

Before ChunkyPipes can function, it needs to be initialized with a call to `chunky init`:

```
$ chunky init
> ChunkyPipes successfully initialized at /home/user
```

To install a pipeline, point ChunkyPipes to the python source file:

```
$ chunky install /path/to/pipeline.py
> Pipeline pipeline.py successfully installed
```

To configure a pipeline to run on the current platform, execute the configuration subcommand:

```
$ chunky configure pipeline
```

To run the pipeline, execute the run subcommand:

```
$ chunky run pipeline [options]
```

Using ChunkyPipes

ChunkyPipes is designed to make running pipelines as painless as possible.

Initializing ChunkyPipes

Although ChunkyPipes can run without initializing a hidden directory, doing so makes using ChunkyPipes and running pipelines much easier and more organized. By keeping pipeline files and various configurations in an internal hidden directory structure, ChunkyPipes abstracts out filepath details involved in running pipelines.

```
$ chunky init
```

If no argument is provided, ChunkyPipes initializes a hidden directory in the home directory. This is also where ChunkyPipes looks by default to install, configure, and run pipelines. If the user wishes to initialize a hidden directory at a location other than the home directory, a path argument may be specified. To change where ChunkyPipes looks when operating, introduce a `CHUNKY_HOME` environment variable and point it to the directory containing the ChunkyPipes hidden directory.

```
$ chunky init /path/to/other/place
> Please set a CHUNKY_HOME environment variable to /path/to/other/place
$ export CHUNKY_HOME=/path/to/other/place
```

Note: The above `export` statement will only persist for the life of the terminal session. To introduce a `CHUNKY_HOME` environment variable permanently, add the export statement to `~/.bashrc` or the platform equivalent.

Installing Pipelines

To install pipelines into ChunkyPipes:

```
$ chunky install /path/to/ngs-pipeline.py
```

This will install the pipeline into the ChunkyPipes hidden directory, whether that be the default home directory or the directory pointed to by `CHUNKY_HOME`.

If the pipeline requires any Python package dependencies, `chunky install` will prompt the user to install these dependencies via pip. Though this step is optional, it's likely that the installed pipeline won't run without the declared dependencies.

Warning: `chunky install` will feed the developer-provided package name and version directly to pip using the `--upgrade` option. The exact version specified will be installed, even if it's an older version than a package that's currently installed. If this behavior isn't desired, the user can pick and choose dependencies to install by running `chunky show` on the pipeline to get all dependencies and installing those desired.

Note: If the user is not in a virtual environment at the time `chunky install` is run, `sudo` or the platform equivalent may need to be prepended to the command in order to install Python packages into system files.

To show a list of installed pipelines:

```
$ chunky list
```

This list also shows which pipelines have corresponding configuration files.

Configuring Pipelines

To configure a pipeline:

```
$ chunky configure <pipeline-name>
```

ChunkyPipes will present an interactive configuration, asking the user for any platform-specific information required by the pipeline.

```
$ chunky configure ngs-pipeline
> Full path to software1 []: (User enters) /path/to/soft1
> Provide a value for arg1 []: (User enters) 45
> Full path to software2 []: (User enters) /path/to/soft2
```

If no `--location` parameter is given, the configuration file is stored in the ChunkyPipes hidden directory as a JSON formatted file with the same base filename as the pipeline. If the user doesn't provide a `--config` parameter when running a pipeline, ChunkyPipes uses the config file in the hidden directory.

If the user is overwriting an existing configuration, the existing value will be displayed in square brackets (`[]`) as a part of the prompt. Leaving this field blank will cause the existing value to be used.

As of version 0.2.0, ChunkyPipes interactive configuration supports TAB-completion for filesystem paths, but not left-and-right character seeking.

Showing Pipelines

To show information about a pipeline:

```
$ chunky show <pipeline-name>
```

This command will show the pipeline description, arguments, dependencies, configuration dictionary, and the current default configuration, if it exists.

Running Pipelines

To run a pipeline:

```
$ chunky run <pipeline-name or path> [-h] [--config CONFIG] [pipeline_args]
```

The pipeline can be either the name of an installed pipeline or a path to a Python file containing a properly formatted Pipeline class. If an installed pipeline name is given without a `--config` parameter, both components will come from the ChunkyPipes hidden directory. If a path is given, `--config` must also be given a value.

Building Pipelines

All ChunkyPipes compatible pipelines exist as a `Pipeline` class that subclasses `chunkypipes.components.BasePipeline` and overrides the following methods:

```
# fun-pipeline.py
from chunkypipes.components import BasePipeline

class Pipeline(BasePipeline):
    def dependencies(self):
```

```
        return []

    def description(self):
        return ''

    def configure(self):
        return {}

    def add_pipeline_args(self, parser):
        pass

    def run_pipeline(self, pipeline_args, pipeline_config):
        pass
```

Pipeline Dependencies

The pipeline dependencies are all Python packages available via pip that that pipeline uses. The list of dependencies is returned as a list of strings, one for each Python package, in the pip format:

```
def dependencies(self):
    return ['numpy', 'scipy==0.17.1']
```

Note: If a package is specified with a version, of the form `package==0.0.0`, ChunkyPipes will attempt to install exactly this version, regardless of what may already be installed on the user's system. Unless a specific version is required to run, don't specify the version.

Pipeline Description

The pipeline description is used as a part of the help message for a pipeline:

```
def description(self):
    return 'This pipeline is crazy fun!'
```

```
$ chunky run fun-pipeline -h
> usage: chunky run fun-pipeline [-h] [-c CONFIG]
>
> This pipeline is crazy fun!
```

Pipeline Configuration

The pipeline configuration includes items in the pipeline logic which may change from platform to platform, but generally won't change from run to run. Paths to software is a common configuration item.

The configuration is returned as a dictionary from the `configure()` method:

```
def configure(self):
    return {
        'software1': {
            'path': 'Full path to software1',
            'arg1': 'Provide a value for arg1'
        },
```

```
        'software2': {
            'path': 'Full path to software2',
        }
    }
```

The configuration dictionary can go arbitrarily deep. All values must be either a dictionary or a string. String values are used as a prompt to the user during configuration and will be replaced with the user-specified values when the pipeline is run.

For the above configuration, the user will see and interactively fill in the prompts:

```
$ chunky configure fun-pipeline
> Full path to software1: (User enters) /path/to/soft1
> Provide a value for arg1: (User enters) 45
> Full path to software2: (User enters) /path/to/soft2
```

When writing pipeline logic in `run_pipeline()`, the following dictionary will be made available in the `pipeline_config` parameter:

```
# Contents of pipeline_config
{
    'software1': {
        'path': '/path/to/soft1',
        'arg1': '45'
    },
    'software2': {
        'path': '/path/to/soft2'
    }
}
```

Pipeline Arguments

The pipeline arguments are items that will change from run to run and are specified by the user on the command line on a per-run basis. Arguments into other programs in the pipeline are common arguments.

The pipeline arguments are added to the `parser` parameter of the `add_pipeline_args()` method. `parser` is an `argparse.ArgumentParser` object, and arguments are added to it using argparse.ArgumentParser.add_argument(). The `argparse` module does not need to be imported by the pipeline.

```
def add_pipeline_args(self, parser):
    parser.add_argument('--read', required=True, help='Path to the read fastq')
    parser.add_argument('--output', required=True, help='Path to output directory')
    parser.add_argument('--lib', default='default_lib', help='Name of the library')
```

These arguments will be exposed to the user according to the rules of the `argparse` module:

```
$ chunky run fun-pipeline -h
> chunky run fun-pipeline [-h] [-c CONFIG] --reads READS --output OUTPUT [--lib LIB]
>
> This pipeline is crazy fun!
>
> optional arguments:
> -h, --help            show this help message and exit
> -c CONFIG, --config CONFIG
>                       Path to a config file to use for this run.
> --read READS          Path to the read fastq
```

```
> --output OUTPUT       Path to output directory
> --lib LIB             Name of the library
>
$ chunky run fun-pipeline --reads /path/to/read.fastq --output /path/to/output/dir --
↪lib LIB33
> ...
```

When writing pipeline logic, the arguments will be made available as a dictionary in the `pipeline_args` parameter:

```
# Contents of pipeline_args
{
    'read': '/path/to/read.fastq',
    'output': '/path/to/output/dir',
    'lib': 'LIB33'
}
```

Note: Parameters in `argparse` can have dashes in them (and should, as command line parameters), but when converted to a Python dictionary object dashes are replaced with underscores.

Ex. `--output-dir` will become `pipeline_args['output_dir']`

Pipeline Logic

All of the pipeline logic goes in the `run_pipeline()` method. Two variables are populated at runtime and passed into the function as parameters: `pipeline_config` and `pipeline_args`. For details on those two parameters, refer to the above sections *Pipeline Configuration* and *Pipeline Arguments*.

From here the logic can be anything, since this is a regular Python function definition. ChunkyPipes provides a couple classes that abstract out details of calling command line programs.

Software

The `chunkypipes.components.Software` object represents a software component of the pipeline. It is instantiated with two arguments, the name of the software and a path to the software executable. The name is only used for logging purposes. Often the software path will come from a configuration value.

```
from chunkypipes.components import Software

software1 = Software('software1', pipeline_config['software1']['path'])
```

To run this software at any point in the pipeline, call the `run()` method and supply any number of Parameters, up to two Redirects, and up to one Pipe.

```
from chunkypipes.components import Parameter, Redirect

software1.run(
    Parameter('-a', '1'),
    Parameter('-b', '2'),
    Parameter('--float', '3.5'),
    Redirect(stream=Redirect.STDOUT, dest='software1.out')
)

software1.run(
```

```
    Parameter('-c', '3'),
    shell=True
)
```

If `shell=True` is given as a parameter, the command will be executed as a string directly in a shell. Otherwise, the command will execute using Python `subprocess.Popen` objects.

Warning: Do not use `shell=True` unless it's certain a program won't run without it. Running commands directly in a shell opens the platform up to shell injection attacks.

Parameter

The `chunkypipes.components.Parameter` object represents a parameter key and value passed into a Software object.

```
from chunkypipes.components import Parameter

Parameter('-a', '1')  # Evaluates to '-a 1'
Parameter('-type', 'gene', 'transcript')  # Evaluates to '-type gene transcript'
Parameter('--output=/path/to/output')  # Evaluates to '--output=/path/to/output'
```

When multiple Parameters are passed into a Software, order is preserved.

Redirect

The `chunkypipes.components.Redirect` object represents a stream redirection. Redirect instantiation accepts two parameters: `stream` and `dest`.

`stream` can be one of the provided constants:

```
Redirect.STDOUT          # >
Redirect.STDOUT_APPEND   # >>
Redirect.STDERR          # 2>
Redirect.STDERR_APPEND   # 2>>
Redirect.BOTH            # &>
Redirect.BOTH_APPEND     # &>>
```

`dest` is the filepath destination of the redirected stream.

Pipe

The `chunkypipes.components.Pipe` object represents piping the output of one program into the input of another. The Software receiving the pipe should call the `pipe()` method instead of `run()`:

```
from chunkypipes.components import Parameter, Redirect, Pipe

software1.run(
    Parameter('-a', '1'),
    Pipe(
        software2.pipe(
            Parameter('-b', '2'),
            Parameter('-c', '3'),
            Redirect(stream=Redirect.STDOUT, dest='software2.out')
```

```
        )
    )
)
# soft1 -a 1 | soft2 -b 2 -c 3 > software2.out
```

If a Pipe is passed into a Software `run()` any Redirects of STDOUT are ignored. Multiple Pipes will be ignored except for the first one.

Pipeline Settings

ChunkyPipes can be configured on a pipeline specific manner to handle certain under-the-hood features. All settings are exposed in the `run_pipeline()` function through the `self.settings` instance variable.

self.settings.logger

The ChunkyPipes logger will by default allow any non-redirected software output to flow to the screen. If a necessary minimum of logger settings are given values, the logger will capture all non-redirected software output to a timestamped log file.

Logger settings are set with the function `self.settings.logger.set()` and given any number of the following keyword arguments:

Keyword Argument	Default	Description
`destination`	`''`	If given a value, all non-redirected stdout streams will go to this file. If `destination_stderr` is not given a value and `log_stderr` is `True` (which it is by default), then all non-redirected stderr streams will go to this file as well.
`destination_mode`	`'w'`	Write mode of the log file. Use standard Python file modes.
`destination_stderr`	`''`	If give a value, all non-redirected stderr streams will go to this file, independent of the value of `destination`.
`destination_stderr_mode`	`'w'`	Write mode of the stderr specific log file, if `destination_stderr` is given a value. Use standard Python file modes.
`log_stdout`	`True`	If `True`, will capture all non-redirected stdout streams.
`log_stderr`	`True`	If `True`, will capture all non-redirected stderr streams.

An example:

```
from chunkypipes.components import Software, Parameter, Redirect

def run_pipeline(self, pipeline_args, pipeline_config):
    ls = Software('ls', '/bin/ls')

    # This run output will go to the screen, since logging settings have not been set
    # nor was any of the output redirected
    ls.run()

    self.settings.logger.set(
        destination='logs/run.log'
    )

    # This run output will go to the log file at logs/run.log, as specified in the␣
↪settings.
    # The log entries will be timestamped
    ls.run()
```

```
    # This run output will go to where it's been redirected, ignoring any logging␣
↪settings
    ls.run(
        Redirect(stream=Redirect.STDOUT, dest='logs/ls.log')
    )
```

Frequently Asked Questions

FAQ is under construction.

Change Log

Version 0.2.4

- Released on 18 April 2016
- Added better logging

Version 0.2.3

- Released on 25 May 2016
- Added `chunky show` subcommand
- Restructured the subcommand system so it uses argparse from the beginning
- Added feature to `confgure` to output a blank configuration
- Removed `__init__` from `list` output
- Added `dependencies()` to the Pipeline class
- `install` attempts to pip install pipeline depencencies, as returned by `depencencies()`

License

OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Contact

Please contact dfitzgerald at uchicago dot edu.