Porekit Documentation

Release 0.1.0alpha

Andreas Klostermann

April 09, 2016

Contents

A pythonic toolkit for working with Oxford Nanopore Data

This is a kit of tools for handling data from Oxford Nanopore Technologies' sequencers, built for integration into the scientific Python ecosystem, including Jupyter Notebook.

This library is meant for use both as an interactive toolkit for use in Jupyter notebooks, as well as custom scripts. In the future a command line tool may be added as well.

Feature requests and bug reports are wellcome. Please use the github issues.

CHAPTER 1

Porekit

A pythonic toolkit for working with Oxford Nanopore Data This is a kit of tools for handling data from Oxford Nanopore Technologies' sequencers, built for integration into the scientific Python ecosystem, including Jupyter Notebook.

This library is meant for use both as an interactive toolkit for use in Jupyter notebooks, as well as custom scripts. In the future a command line tool may be added as well.

Feature requests and bug reports are wellcome. Please use the github issues.

Notes from original author and maintainer:

- I am not affiliated with Oxford Nanopore Technologies
- Neither am I affiliated or in contact with any participant in the MinION MAP
- This work has been done without any "official documentation", and I don't even know if there is such a thing
- The documentation of porekit represents my best guess about how MinION sequencing and the file format works. It probably contains many factual errors or misinterpretations!
- For me, porekit is mainly an educational effort to learn about nanopore sequencing without having access to it.

1.1 Main features

- **Gathering Metadata about reads**
  - Use as Pandas DataFrame
  - Export to many different formats
  - Helper functions for custom scripts
- Plots * Read length distribution * Channel Occupancy over time * reads over time * Yield over time * template length vs complement length

1.2 Planned

- **Jupyter integration**
  - Interactive squiggle viewer

- Free software: ISC license
- Documentation: https://porekit-python.readthedocs.org.

1.3 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Use Python 3.5!

I recommend installing this inside an Anaconda virtual environment. Anaconda is a distribution of common scientific Python packages and the conda package manager simplifies handling binary dependencies for Python and R packages.

As Porekit is at a very early state in its development, you should probably clone this github repository and run *python setup.py develop*.

```
In [1]: import pandas as pd
        import numpy as np
        import seaborn as sb
        import matplotlib.pyplot as plt
        %matplotlib inline
```

CHAPTER 3

Introduction to Porekit-Python

3.1 Disclaimer

Porekit is the result of my personal interest in nanopore sequencing. I'm not affiliated with Oxford Nanopore Technologies, or any MAP participant. This means a lot of the factual information presented in this notebook may be wrong.

3.2 General philosophy and goal of Porekit-Python

This library is meant to provide tools for interactively exploring nanopore data inside the Jupyter notebook, and for writing simple scripts or more complex software dealing with nanopore data. Therefore a lot of attention has been given to make interactive use easy and painless, and to keep the code in the background flexible and exposed to library users.

3.3 What Oxford Nanopore Data looks like

The MinION sequencer is attached to a laptop running MinKnow. This program connects directly to the MinION device and tells it what to do. Optionally, third party software can connect to an API inside the primary software to remote control the sequencing process. That is not covered here, though.

In a nutshell, nanopore sequencing works by dragging a DNA molecule through a tiny pore in a membrane. As the DNA passes, the voltage difference between the two sides of the membrane change, depending on the electrochemical properties of the passing nucleotides. This means that, at the core of the nanopore data, there is a timeseries of voltage measurements, which is called the "squiggle".

The process to convert the squiggle into a sequence of DNA letters is called base calling. The current MinKnow software uploads the squiggle to Metrichor servers, which perform the base calling, and send the result back to the user's computer.

The result of a sequencing run is a collection of FAST5 files, each containing data on one molecule of DNA which passed through one of currently 512 channels in the flowcell. These files are stored on disk, usually in one directory per run. A convention seems to be to name each file with a unique and descriptive string:

```
In [4]: !ls /home/andi/nanopore/GenomeRU2/downloads/pass/ | tail -n 10
```

```
PLSP57501_20151028_Mk1_lambda_RU9_2752_1_ch9_file64_strand.fast5
PLSP57501_20151028_Mk1_lambda_RU9_2752_1_ch9_file67_strand.fast5
PLSP57501_20151028_Mk1_lambda_RU9_2752_1_ch9_file6_strand.fast5
PLSP57501_20151028_Mk1_lambda_RU9_2752_1_ch9_file74_strand.fast5
```

```
PLSP57501_20151028_Mk1_lambda_RU9_2752_1_ch9_file86_strand.fast5
PLSP57501_20151028_Mk1_lambda_RU9_2752_1_ch9_file90_strand.fast5
PLSP57501_20151028_Mk1_lambda_RU9_2752_1_ch9_file92_strand.fast5
PLSP57501_20151028_Mk1_lambda_RU9_2752_1_ch9_file95_strand.fast5
PLSP57501_20151028_Mk1_lambda_RU9_2752_1_ch9_file97_strand.fast5
PLSP57501_20151028_Mk1_lambda_RU9_2752_1_ch9_file9_strand.fast5
```

These files belong to data publishd by Quick et al. http://europepmc.org/abstract/MED/25386338;jsessionid=ijHIHUVXlcpxeTzVUihz.0

3.4 Gathering Metadata

The following snippet will extract meta data from all of my downloaded nanopore data, searching directories recursively.

```
In [6]: import porekit
        everything = porekit.gather_metadata("/home/andi/nanopore/")
```

The result is a Pandas DataFrame object, which is too big to comfortably view in its entirety, but still comparatively "small data". Here is a subset of it:

```
In [7]: everything[['asic_id', 'channel_number', 'template_length', 'complement_length']].h
```

All of the columns available:

```
In [8]: everything.columns
```

```
Out[8]: Index(['absolute_filename', 'format', 'run_id', 'asic_id', 'version_name',
                'device_id', 'flow_cell_id', 'asic_temp', 'heatsink_temp',
                'channel_number', 'channel_range', 'channel_sampling_rate',
                'channel_digitisation', 'channel_offset', 'has_basecalling',
                'basecall_timestamp', 'basecall_version', 'basecall_name',
                'has_template', 'template_length', 'has_complement',
                'complement_length', 'has_2D', '2D_length', 'read_start_time',
                'read_duration', 'read_end_time'],
               dtype='object')
```

The file names are used as an index, because they are assumed to be unique and descriptive. Even when the absolute/physical location of the dataset changes, data or analytics based on these filenames are still useful.

The philosopy of porekit is to gather the metadata once and then store it in a different format. This makes it easier to analyse the metadata or use it in another context, for example with alignment data.

The following will store the metadata in an HDF5 File:

```
In [10]: everything.to_hdf("everything.h5", "meta")
```

```
/home/andi/anaconda3/lib/python3.5/site-packages/pandas/core/generic.py:1096: PerformanceWa
your performance may suffer as PyTables will pickle object types that it cannot
map directly to c-types [inferred_type->mixed,key->block3_values] [items->['absolute_filena

  return pytables.to_hdf(path_or_buf, key, self, **kwargs)
```

3.5 Grouping by Device, ASIC and Run Ids

```
In [13]: g = everything.groupby(['device_id', 'asic_id', 'run_id'])
```

```
In [14]: df = g.template_length.agg([lambda v: len(v), np.mean, np.max])
         df.columns = ['Count', 'Mean template length', 'Max template_length']
         df
```

As you can see, I have downloaded several nanopore sets from ENA. These are mostly incomplete sets, since I was interested more in the variety of data rather than the completeness. You can easily use `wget` to download a tarball from ENA, then extract the partial download. The last file will be truncated, but the rest is usable.

```
In [1]: import pandas as pd
        import numpy as np
        import seaborn as sb
        import matplotlib.pyplot as plt
        import porekit
        %matplotlib inline
```

CHAPTER 4

Plotting Data

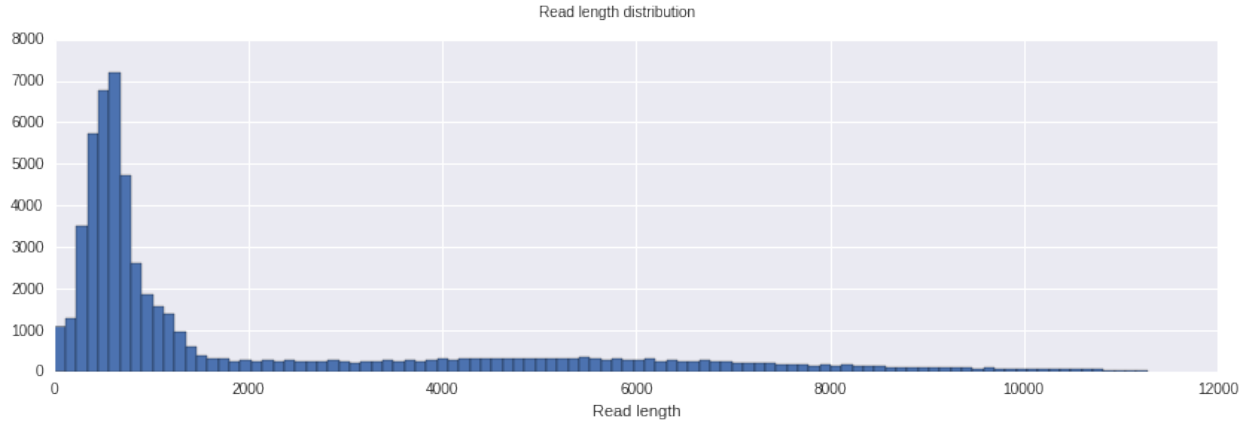
Visualizing the metadata is very useful to get a first look at the nature and quality of the run.

First we need a `DataFrame` with the meta data. You can make one with `porekit.gather_metadata` once, and then load it later from a hdf file or something similar.

```
In [2]: df = pd.read_hdf("../examples/data/ru9_meta.h5", "meta")
```

4.1 Read length distribution

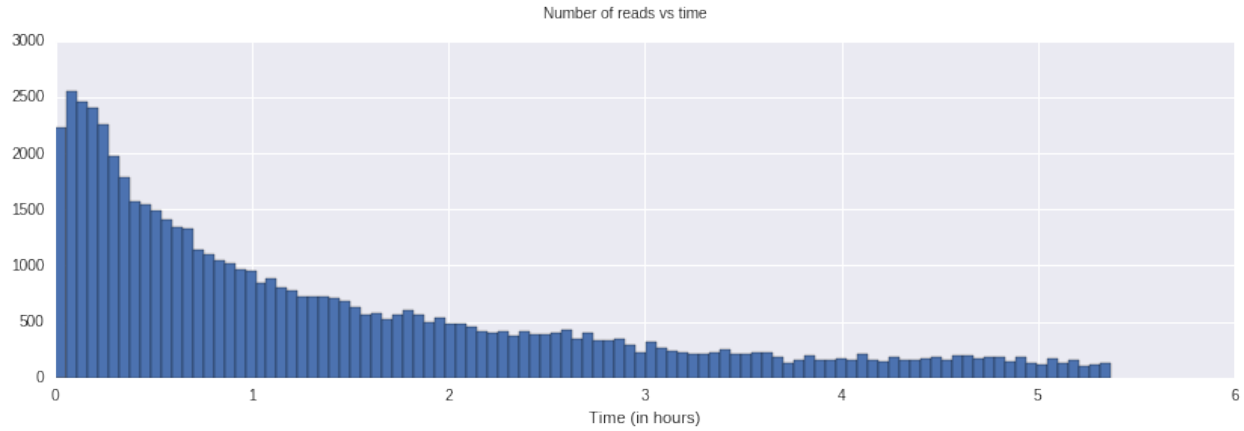
```
In [3]: porekit.plots.read_length_distribution(df);
```



This is a histogram showing the distribution of read length. In this case it's the max of template and complement length. This plots ignores a small part of the longest reads in order to be more readable.

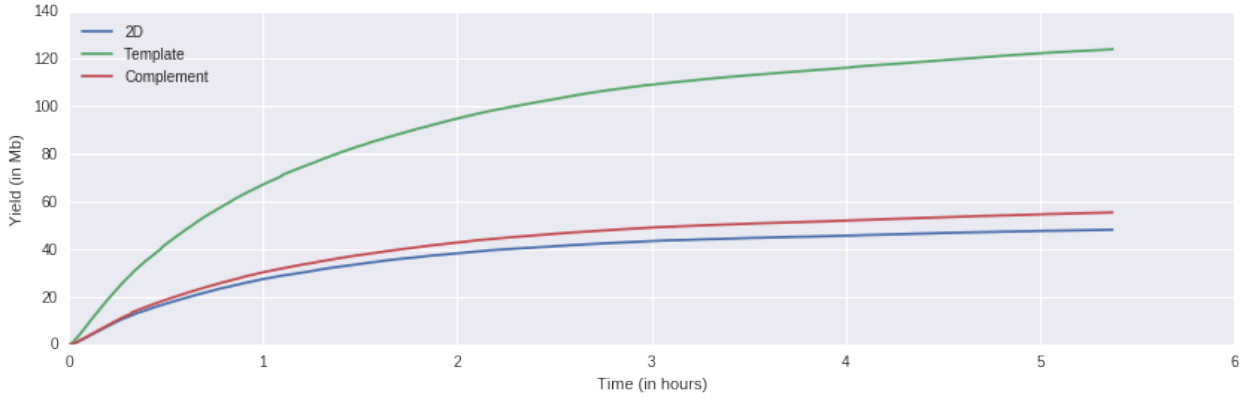
4.2 Reads over time

```
In [4]: porekit.plots.reads_vs_time(df);
```



4.3 Yield Curves

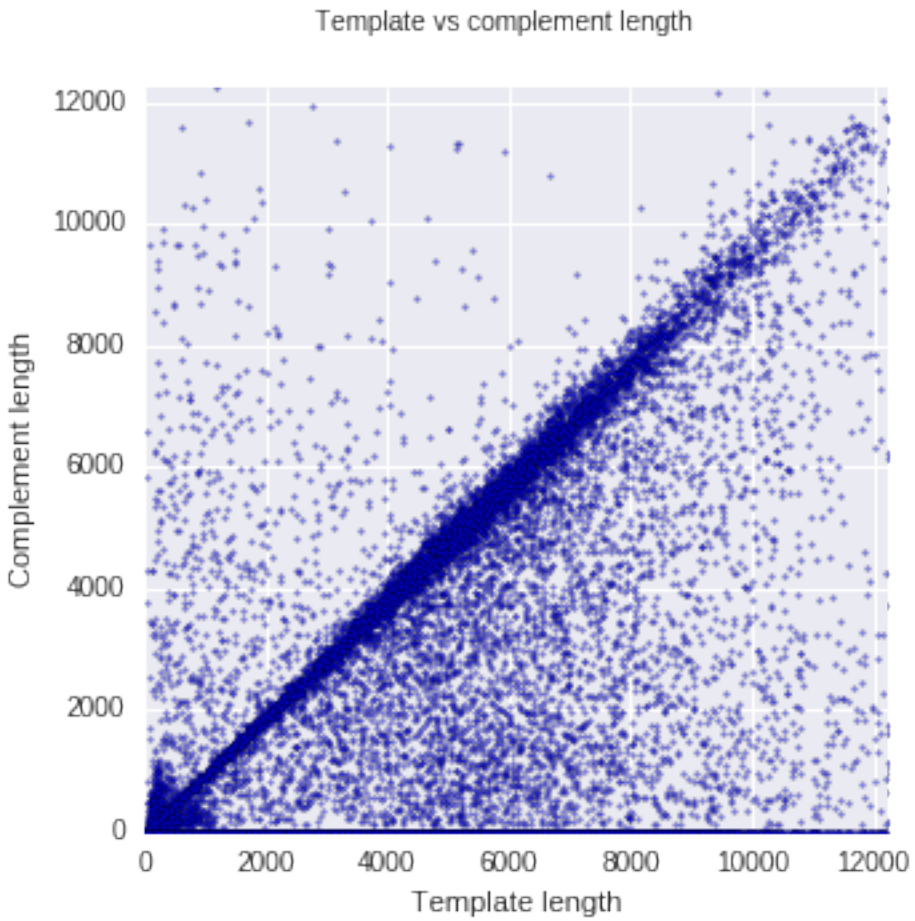
```
In [5]: porekit.plots.yield_curves(df);
```



This plot shows the sequence yields in Megabases over time.

4.4 Template length vs complement length

```
In [6]: porekit.plots.template_vs_complement(df);
```

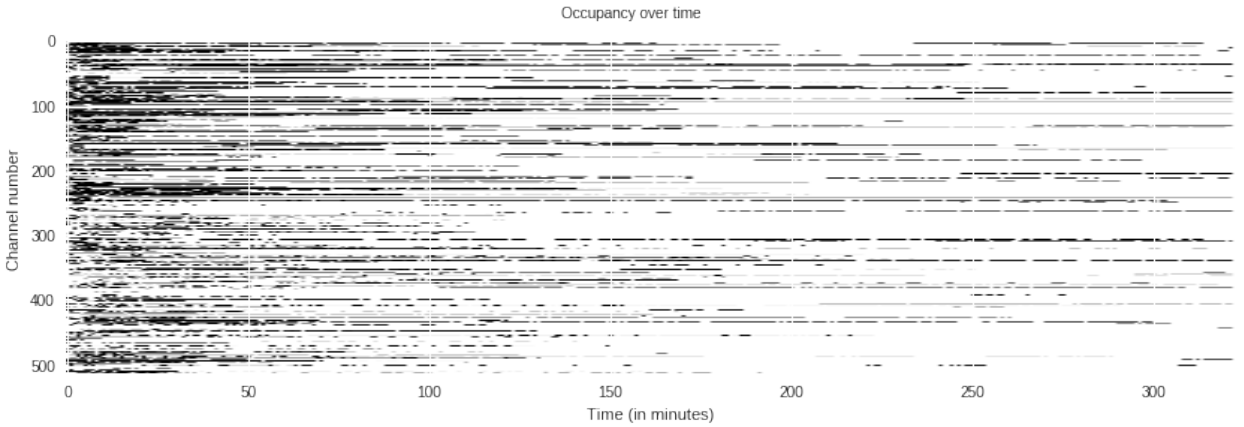


In the standard 2D library preparation, a "hairpin" is attached to one end of double stranded DNA. Then, when the strand goes through the nanopore, first one strand translocates, then the hairpin and finally the complement. Because template and complement both carry the same information, they can be used to improve accuracy of the basecalling.

However, not all molecules have a hairpin attached, not all have a complement strand, and in most cases, the template and complement length does not match completely. This can be seen in the plot above, where most data points are on a diagonal with template and complement length being almost the same. There are more points under the diagonal than above it, and there is a solid line at the bottom, showing reads with no complement.

4.5 Occupancy

```
In [7]: porekit.plots.occupancy(df);
```



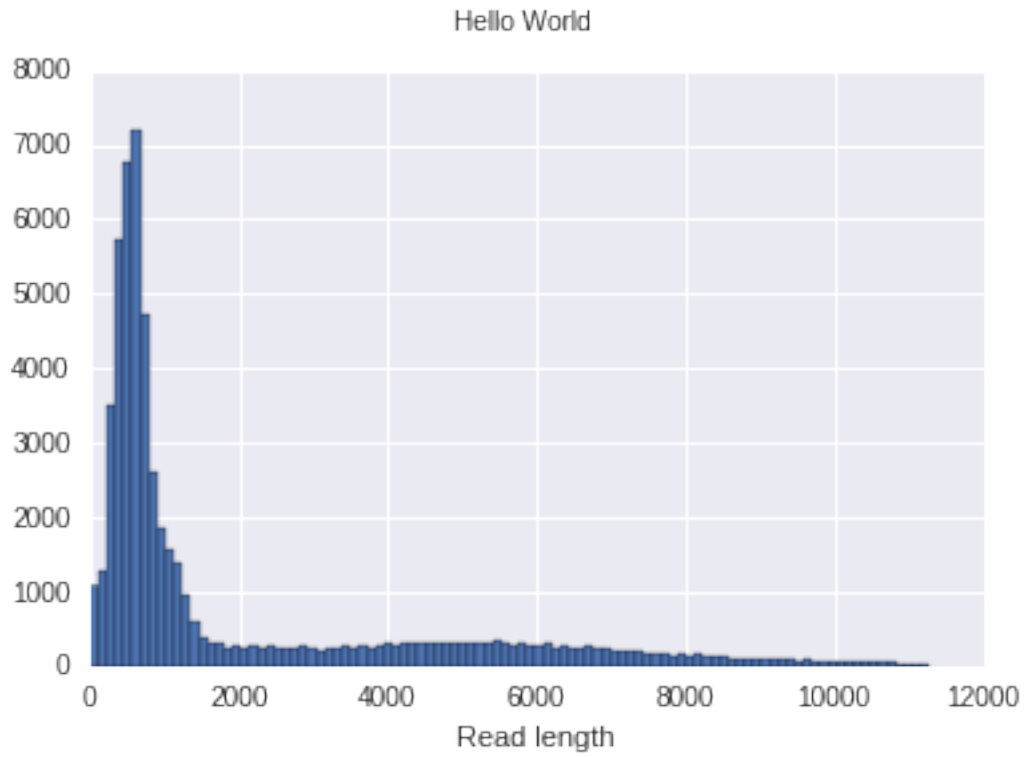
This shows the occupancy of pores over time. In General, pores break over time, which is a major factor in limiting the total yield over the lifetime of a flowcell.

4.6 Customizing plots

The plots inside `porekit.plots` are designed to work best inside the Jupyter notebook when exploring nanopore data interactively, and showing nanopore data as published notebooks or presentations. This is why they use colors and a wide aspect ratio.

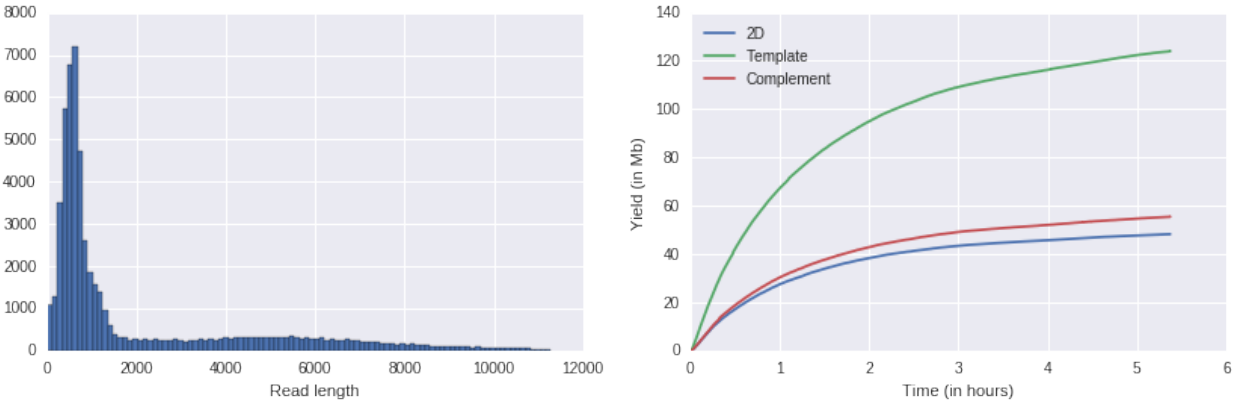
But the plots can be customized somewhat using standard matplotlib. Every plot function returns a figure and an axis object:

```
In [8]: f, ax = porekit.plots.read_length_distribution(df)
        f.suptitle("Hello World");
        f.set_figwidth(6)
```



Sometimes you want to subdivide a figure into multiple plots. You can do it like this:

```
In [9]: f, axes = plt.subplots(1,2)
        f.set_figwidth(14)
        ax1, ax2 = axes
        porekit.plots.read_length_distribution(df, ax=ax1);
        porekit.plots.yield_curves(df, ax=ax2);
```



If you want to go beyond those relatively simple customizations, you may want to just copy and paste some code from `porekit/plots.py` and go from there. The plots are relatively simple overall.

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/akloster/porekit-python/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

5.1.4 Write Documentation

porekit could always use more documentation, whether as part of the official porekit docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/akloster/porekit-python/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *porekit-python* for local development.

1. Fork the *porekit-python* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/porekit-python.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv porekit-python
$ cd porekit-python/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 porekit-python tests
$ python setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

CHAPTER 6

Credits

6.1 Development Lead

- Andreas Klostermann <andreasklostermann@gmail.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

CHAPTER 8

Indices and tables

- genindex
- modindex
- search