petlx Documentation

Release 1.0.3

Alistair Miles

July 22, 2015

Contents

petlx is a collection of domain-specific and/or experimental extensions to petl, a general purpose Python package for extracting, transforming and loading tables of data.

- Documentation: http://petlx.readthedocs.org/
- Source Code: https://github.com/alimanfoo/petlx
- Download: http://pypi.python.org/pypi/petlx
- Mailing List: http://groups.google.com/group/python-etl

Please feel free to ask questions via the mailing list (python-etl@googlegroups.com).

To report installation problems, bugs or any other issues please email python-etl@googlegroups.com or raise an issue on GitHub.

For an overview of all functions in the package, see the genindex.

Note: Version 1.0 is a new major release of *petlx*. The content of this package is significantly changed. See the *Changes* section below for more information.

CHAPTER 1

Installation

This package is available from the Python Package Index. If you have pip you should be able to do:

```
$ pip install petlx
```

You can also download manually, extract and run `python setup.py install`.

CHAPTER 2

Dependencies

This package has no installation requirements other than the Python core modules.

Some of the functions in this package require installation of third party packages. This is indicated in the relevant parts of the documentation.

CHAPTER 3

Modules

3.1 Biology

3.1.1 GFF3

petlx.bio.gff3.**fromgff3**(*filename*, *region=None*)

Extract feature rows from a GFF3 file, e.g.:

```
>>> import petl as etl
>>> # activate bio extensions
... import petlx.bio
>>> table1 = etl.fromgff3('fixture/sample.gff')
>>> table1.look(truncate=30)
+--------------+---------+---------------+-------+---------+-------+--------+-------+----------
| seqid        | source  | type          | start | end     | score | strand | phase | attributes
+==============+=========+===============+=======+=========+=======+========+=======+==========
| 'apidb|MAL1' | 'ApiDB' | 'supercontig' |     1 |  643292 | '.'   | '+'    | '.'   | {'localiza
+--------------+---------+---------------+-------+---------+-------+--------+-------+----------
| 'apidb|MAL2' | 'ApiDB' | 'supercontig' |     1 |  947102 | '.'   | '+'    | '.'   | {'localiza
+--------------+---------+---------------+-------+---------+-------+--------+-------+----------
| 'apidb|MAL3' | 'ApiDB' | 'supercontig' |     1 | 1060087 | '.'   | '+'    | '.'   | {'localiza
+--------------+---------+---------------+-------+---------+-------+--------+-------+----------
| 'apidb|MAL4' | 'ApiDB' | 'supercontig' |     1 | 1204112 | '.'   | '+'    | '.'   | {'localiza
+--------------+---------+---------------+-------+---------+-------+--------+-------+----------
| 'apidb|MAL5' | 'ApiDB' | 'supercontig' |     1 | 1343552 | '.'   | '+'    | '.'   | {'localiza
+--------------+---------+---------------+-------+---------+-------+--------+-------+----------
...
```

A region query string of the form ‘[seqid]’ or ‘[seqid]:[start]-[end]’ may be given for the *region* argument. If given, requires the GFF3 file to be position sorted, bgzipped and tabix indexed. Requires pysam to be installed. E.g.:

```
>>> # extract from a specific genome region via tabix
... table2 = etl.fromgff3('fixture/sample.sorted.gff.gz',
...                       region='apidb|MAL5:1289593-1289595')
>>> table2.look(truncate=30)
+--------------+---------+---------------+---------+---------+-------+--------+-------+---------
| seqid        | source  | type          | start   | end     | score | strand | phase | attribut
+==============+=========+===============+=========+=========+=======+========+=======+=========
| 'apidb|MAL5' | 'ApiDB' | 'supercontig' |       1 | 1343552 | '.'   | '+'    | '.'   | {'locali
+--------------+---------+---------------+---------+---------+-------+--------+-------+---------
| 'apidb|MAL5' | 'ApiDB' | 'exon'        | 1289594 | 1291685 | '.'   | '+'    | '.'   | {'size':
```

```
+--------------+---------+---------------+---------+---------+-------+--------+--------+---------
| 'apidb|MAL5' | 'ApiDB' | 'gene'        | 1289594 | 1291685 | '.'   | '+'    | '.'    | {'ID': '
+--------------+---------+---------------+---------+---------+-------+--------+--------+---------
| 'apidb|MAL5' | 'ApiDB' | 'rRNA'        | 1289594 | 1291685 | '.'   | '+'    | '.'    | {'ID': '
+--------------+---------+---------------+---------+---------+-------+--------+--------+---------
```

3.1.2 Tabix (pysam)

Note: The pysam package is required, e.g.:

```
$ pip install pysam
```

petlx.bio.tabix.**fromtabix**(*filename*, *reference=None*, *start=None*, *stop=None*, *region=None*, *header=None*)

Extract rows from a tabix indexed file, e.g.:

```
>>> import petl as etl
>>> # activate bio extensions
... import petlx.bio
>>> table1 = etl.fromtabix('fixture/test.bed.gz',
...                        region='Pf3D7_02_v3')
>>> table1
+---------------+----------+----------+-----------------------------+
| #chrom        | start    | end      | region                      |
+===============+==========+==========+=============================+
| 'Pf3D7_02_v3' | '0'      | '23100'  | 'SubtelomericRepeat'        |
+---------------+----------+----------+-----------------------------+
| 'Pf3D7_02_v3' | '23100'  | '105800' | 'SubtelomericHypervariable' |
+---------------+----------+----------+-----------------------------+
| 'Pf3D7_02_v3' | '105800' | '447300' | 'Core'                      |
+---------------+----------+----------+-----------------------------+
| 'Pf3D7_02_v3' | '447300' | '450450' | 'Centromere'                |
+---------------+----------+----------+-----------------------------+
| 'Pf3D7_02_v3' | '450450' | '862500' | 'Core'                      |
+---------------+----------+----------+-----------------------------+
...

>>> table2 = etl.fromtabix('fixture/test.bed.gz',
...                        region='Pf3D7_02_v3:110000-120000')
>>> table2
+---------------+----------+----------+--------+
| #chrom        | start    | end      | region |
+===============+==========+==========+========+
| 'Pf3D7_02_v3' | '105800' | '447300' | 'Core' |
+---------------+----------+----------+--------+
```

3.1.3 Variant call format (PyVCF)

Note: The pyvcf package is required, e.g.:

```
$ pip install pyvcf
```

petlx.bio.vcf.**fromvcf**(*filename*, *chrom=None*, *start=None*, *stop=None*, *samples=True*)

Returns a table providing access to data from a variant call file (VCF). E.g.:

```
>>> import petl as etl
>>> # activate bio extensions
... import petlx.bio
>>> table1 = etl.fromvcf('fixture/sample.vcf')
>>> table1.look(truncate=20)
+-------+---------+-------------+-----+--------+------+---------+----------------------+--------
| CHROM | POS     | ID          | REF | ALT    | QUAL | FILTER  | INFO                 | NA00001
+=======+=========+=============+=====+========+======+=========+======================+========
| '19'  |     111 | None        | 'A' | [C]    |  9.6 | None    | {}                   | Call(sa
+-------+---------+-------------+-----+--------+------+---------+----------------------+--------
| '19'  |     112 | None        | 'A' | [G]    |   10 | None    | {}                   | Call(sa
+-------+---------+-------------+-----+--------+------+---------+----------------------+--------
| '20'  |   14370 | 'rs6054257' | 'G' | [A]    |   29 | []      | {'DP': 14, 'H2': Tru | Call(sa
+-------+---------+-------------+-----+--------+------+---------+----------------------+--------
| '20'  |   17330 | None        | 'T' | [A]    |    3 | ['q10'] | {'DP': 11, 'NS': 3,  | Call(sa
+-------+---------+-------------+-----+--------+------+---------+----------------------+--------
| '20'  | 1110696 | 'rs6040355' | 'A' | [G, T] |   67 | []      | {'DP': 10, 'AA': 'T' | Call(sa
+-------+---------+-------------+-----+--------+------+---------+----------------------+--------
...
```

petlx.bio.vcf.**vcfunpackinfo**(*table*, **keys*)

Unpack the INFO field into separate fields. E.g.:

```
>>> import petl as etl
>>> # activate bio extensions
... import petlx.bio
>>> table1 = (
...     etl
...     .fromvcf('fixture/sample.vcf', samples=None)
...     .vcfunpackinfo()
... )
>>> table1
+-------+---------+-------------+-----+--------+------+---------+------+------+----------------+
| CHROM | POS     | ID          | REF | ALT    | QUAL | FILTER  | AA   | AC   | AF             |
+=======+=========+=============+=====+========+======+=========+======+======+================+
| '19'  |     111 | None        | 'A' | [C]    |  9.6 | None    | None | None | None           |
+-------+---------+-------------+-----+--------+------+---------+------+------+----------------+
| '19'  |     112 | None        | 'A' | [G]    |   10 | None    | None | None | None           |
+-------+---------+-------------+-----+--------+------+---------+------+------+----------------+
| '20'  |   14370 | 'rs6054257' | 'G' | [A]    |   29 | []      | None | None | [0.5]          |
+-------+---------+-------------+-----+--------+------+---------+------+------+----------------+
| '20'  |   17330 | None        | 'T' | [A]    |    3 | ['q10'] | None | None | [0.017]        |
+-------+---------+-------------+-----+--------+------+---------+------+------+----------------+
| '20'  | 1110696 | 'rs6040355' | 'A' | [G, T] |   67 | []      | 'T'  | None | [0.333, 0.667] |
+-------+---------+-------------+-----+--------+------+---------+------+------+----------------+
...
```

petlx.bio.vcf.**vcfmeltsamples**(*table*, **samples*)

Melt the samples columns. E.g.:

```
>>> import petl as etl
>>> # activate bio extensions
... import petlx.bio
>>> table1 = (
...     etl
...     .fromvcf('fixture/sample.vcf')
...     .vcfmeltsamples()
... )
>>> table1
```

```
+-------+-----+------+-----+-----+------+--------+------+----------+-----------------------------
| CHROM | POS | ID   | REF | ALT | QUAL | FILTER | INFO | SAMPLE   | CALL
+=======+=====+======+=====+=====+======+========+======+==========+=============================
| '19'  | 111 | None | 'A' | [C] |  9.6 | None   | {}   | 'NA00001' | Call(sample=NA00001, CallD
+-------+-----+------+-----+-----+------+--------+------+----------+-----------------------------
| '19'  | 111 | None | 'A' | [C] |  9.6 | None   | {}   | 'NA00002' | Call(sample=NA00002, CallD
+-------+-----+------+-----+-----+------+--------+------+----------+-----------------------------
| '19'  | 111 | None | 'A' | [C] |  9.6 | None   | {}   | 'NA00003' | Call(sample=NA00003, CallD
+-------+-----+------+-----+-----+------+--------+------+----------+-----------------------------
| '19'  | 112 | None | 'A' | [G] |   10 | None   | {}   | 'NA00001' | Call(sample=NA00001, CallD
+-------+-----+------+-----+-----+------+--------+------+----------+-----------------------------
| '19'  | 112 | None | 'A' | [G] |   10 | None   | {}   | 'NA00002' | Call(sample=NA00002, CallD
+-------+-----+------+-----+-----+------+--------+------+----------+-----------------------------
...
```

petlx.bio.vcf.**vcfunpackcall**(*table*, **keys*)

Unpack the call column. E.g.:

```
>>> import petl as etl
>>> # activate bio extensions
... import petlx.bio
>>> table1 = (
...     etl
...     .fromvcf('fixture/sample.vcf')
...     .vcfmeltsamples()
...     .vcfunpackcall()
... )
>>> table1
+-------+-----+------+-----+-----+------+--------+------+----------+------+------+-------+-----
| CHROM | POS | ID   | REF | ALT | QUAL | FILTER | INFO | SAMPLE   | DP   | GQ   | GT    | HQ
+=======+=====+======+=====+=====+======+========+======+==========+======+======+=======+=====
| '19'  | 111 | None | 'A' | [C] |  9.6 | None   | {}   | 'NA00001' | None | None | '0|0' | [10,
+-------+-----+------+-----+-----+------+--------+------+----------+------+------+-------+-----
| '19'  | 111 | None | 'A' | [C] |  9.6 | None   | {}   | 'NA00002' | None | None | '0|0' | [10,
+-------+-----+------+-----+-----+------+--------+------+----------+------+------+-------+-----
| '19'  | 111 | None | 'A' | [C] |  9.6 | None   | {}   | 'NA00003' | None | None | '0/1' | [3,
+-------+-----+------+-----+-----+------+--------+------+----------+------+------+-------+-----
| '19'  | 112 | None | 'A' | [G] |   10 | None   | {}   | 'NA00001' | None | None | '0|0' | [10,
+-------+-----+------+-----+-----+------+--------+------+----------+------+------+-------+-----
| '19'  | 112 | None | 'A' | [G] |   10 | None   | {}   | 'NA00002' | None | None | '0|0' | [10,
+-------+-----+------+-----+-----+------+--------+------+----------+------+------+-------+-----
...
```

3.2 Branching Pipelines

3.2.1 Introduction

This module provides some functions for setting up branching data transformation pipelines.

The general pattern is to define the pipeline, connecting components together via the `pipe()` method call, then pushing data through the pipeline via the `push()` method call at the top of the pipeline. E.g.:

```
>>> from petl import fromcsv
>>> source = fromcsv('fruit.csv')
>>> from petlx.push import *
>>> p = partition('fruit')
```

```
>>> p.pipe('orange', tocsv('oranges.csv'))
>>> p.pipe('banana', tocsv('bananas.csv'))
>>> p.push(source)
```

The pipe operator can also be used to connect components in the pipeline, by analogy with the use of the pipe character in unix/linux shells, e.g.:

```
>>> from petl import fromcsv
>>> source = fromcsv('fruit.csv')
>>> from petlx.push import *
>>> p = partition('fruit')
>>> p | ('orange', tocsv('oranges.csv')
>>> p | ('banana', tocsv('bananas.csv')
>>> p.push(source)
```

3.2.2 Push Functions

petlx.push.**partition**(*discriminator*)

Partition rows based on values of a field or results of applying a function on the row. E.g.:

```
>>> from petlx.push import partition, tocsv
>>> p = partition('fruit')
>>> p.pipe('orange', tocsv('oranges.csv'))
>>> p.pipe('banana', tocsv('bananas.csv'))
>>> p.push(sometable)
```

In the example above, rows where the value of the 'fruit' field equals 'orange' are piped to the 'oranges.csv' file, and rows where the 'fruit' field equals 'banana' are piped to the 'bananas.csv' file.

petlx.push.**sort**(*key=None*, *reverse=False*, *buffersize=None*)

Sort rows based on some key field or fields. E.g.:

```
>>> from petlx.push import sort, tocsv
>>> p = sort('foo')
>>> p.pipe(tocsv('sorted_by_foo.csv'))
>>> p.push(sometable)
```

petlx.push.**duplicates**(*key*)

Report rows with duplicate key values. E.g.:

```
>>> from petlx.push import duplicates, tocsv
>>> p = duplicates('foo')
>>> p.pipe(tocsv('foo_dups.csv'))
>>> p.pipe('remainder', tocsv('foo_uniq.csv'))
>>> p.push(sometable)
```

N.B., assumes data are already sorted by the given key.

petlx.push.**unique**(*key*)

Report rows with unique key values. E.g.:

```
>>> from petlx.push import unique, tocsv
>>> p = unique('foo')
>>> p.pipe(tocsv('foo_uniq.csv'))
>>> p.pipe('remainder', tocsv('foo_dups.csv'))
>>> p.push(sometable)
```

N.B., assumes data are already sorted by the given key. See also *duplicates()*.

petlx.push.**diff**()
Find rows that differ between two tables. E.g.:

```
>>> from petlx.push import diff, tocsv
>>> p = diff()
>>> p.pipe('+', tocsv('added.csv'))
>>> p.pipe('-', tocsv('subtracted.csv'))
>>> p.pipe(tocsv('common.csv'))
>>> p.push(sometable, someothertable)
```

petlx.push.**tocsv**(*filename*, *dialect='excel'*, ***kwargs*)
Push rows to a CSV file. E.g.:

```
>>> from petlx.push import tocsv
>>> p = tocsv('example.csv')
>>> p.push(sometable)
```

petlx.push.**totsv**(*filename*, *dialect='excel-tab'*, ***kwargs*)
Push rows to a tab-delimited file. E.g.:

```
>>> from petlx.push import totsv
>>> p = totsv('example.tsv')
>>> p.push(sometable)
```

petlx.push.**topickle**(*filename*, *protocol=-1*)
Push rows to a pickle file. E.g.:

```
>>> from petlx.push import topickle
>>> p = topickle('example.pickle')
>>> p.push(sometable)
```

CHAPTER 4

Changes

4.1 Version 1.0

Version 1.0 is a new major release of both `petlx` and `petl`. This package has been completely reorganised, and several areas of functionality have been migrated to petl. The major changes are described below.

- The *petlx.xls* module has been migrated to *petl.io.xls*
- The *petlx.xlsx* module has been migrated to *petl.io.xlsx*
- The *petlx.array* module has been migrated to *petl.io.numpy*
- The *petlx.dataframe* module has been migrated to *petl.io.pandas*
- The *petlx.hdf5* module has been migrated to *petl.io.pytables*
- The *petlx.index* module has been migrated to *petl.io.whoosh*
- The *petlx.interval* module has been migrated to *petl.transform.intervals*
- The *display()* and *displayall()* functions from the *petlx.ipython* module have been migrated to *petl.util.vis*
- The *petlx.tabix* module has been renamed to *petlx.bio.tabix*
- The *petlx.gff3* module has been renamed to *petlx.bio.gff3*
- The *petlx.vcf* module has been renamed to *petlx.bio.vcf*

Please email python-etl@googlegroups.com if you have any questions.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

p

D

F

P

S

T

U

V