# Chanjo Documentation

***Release 3.2.0***

**Robin Andeer**

December 22, 2015

# Contents

# Simple Sambamba integration

There's a new kid on the BAM processing block: Sambamba! You can easily load output into a Chanjo database for further coverage exploration:

```
$ sambamba depth region -L exons.bed -t 10 -t 20 alignment.bam > exons.coverage.bed
$ chanjo load exons.coverage.bed
$ chanjo calculate region 1 861321 871275 --sample ADM980A2
{"completeness_10": 100, "completeness_20": 87.123, "mean_coverage": 47.24234}
```

To learn more about Chanjo and how you can use it to gain a better understanding of sequencing coverage you can do no better than to:

CHAPTER 2

# Contents

## 2.1 Installation

### 2.1.1 Guide

Chanjo targets **Unix** and **Python 2.7+/3.2+**. Since chanjo relies on Sambamba for BAM processing, it's now very simple to install. I do still recommend Miniconda, it's a slim distribution of Python with a very nice package manager.

```
$ pip install chanjo
```

#### Vagrant dev/testing environment

You can also set up a local development environment in a virtual machine through Vagrant. This will handle the install for you automatically through Ansible provisioning! After downloading/cloning the repo:

```
$ vagrant up
$ vagrant ssh
```

### 2.1.2 Sambamba

You will also need a copy of Sambamba which you can simply grab from their **'GitHub repo'_** where they serve up static binaries - just drop the latest in your path and you are good to go!

```
$ wget -P /tmp/ https://github.com/lomereiter/sambamba/releases/download/v0.5.8/sambamba_v0.5.8_linux
$ tar xjfv /tmp/sambamba_v0.5.8_linux.tar.bz2 -C /tmp/
$ mv /tmp/sambamba_v0.5.8 ~/bin/sambamba
$ chmod +x ~/bin/sambamba
```

### 2.1.3 Optional dependecies

If you plan on using MySQL for your SQL database you also need a SQL adapter. My recommendation is 'pymysql'. It's written in pure Python and works on both version 2.x and 3.x.

```
$ pip install pymysql
```

### 2.1.4 Getting the code

Would you like to take part in the development or tweak the code for any other reason? In that case, you should follow these simple steps and download the code from the GitHub repo.

```
$ git clone https://github.com/robinandeer/chanjo.git
$ cd chanjo
$ pip install --editable .
```

## 2.2 Introduction & Demo

### 2.2.1 Concise overview

**Current release**: Radical Red Panda (3.2.0)

The goals of Chanjo are quite simple:

1. Integrate seamlessly with `sambamba depth region` output
2. Break down coverage to intuitive levels of abstraction: exons, transcripts, and genes
3. Enable explorative coverage investigations across samples and genomic regions

### 2.2.2 Demo

The rest of this document will guide you through a short demo that will cover how to use Chanjo from the command line.

#### Demo files

First we need some files to work with. Chanjo comes with some pre-installed demo files we can use.

```
$ chanjo demo && cd chanjo-demo
```

This will create a new folder (`chanjo-demo`) in your current directory and fill it with the example files you need.

**Note:** You can name the new folder anything you like but it *must not already exist*!

#### Setup and configuration

The first task is to create a config file (`chanjo.yaml`) and prepare the database. Chanjo will walk you through setting it up by running:

```
$ chanjo init
$ chanjo db setup
```

**Note:** Chanjo uses project-level config files by default. This means that it will look for a `chanjo.yaml` file in the **current directory** where you execute your command. You can also point to a diffrent config file using the `chanjo -c /path/to/chanjo.yaml` option.

### Defining interesting regions

One important thing to note is that Chanjo considers coverage across exonic regions of the genome. It's perfectly possible to compose your own list of intervals. Just make sure to follow the BED conventions (http://genome.ucsc.edu/FAQ/FAQformat.html#format1). You then add a couple of additional columns that define relationships between exons and transcripts and transcripts and genes:

If an exon belongs to multiple transcripts you define a list of ids and an equal number of gene identifiers to match.

### Linking exons/transcripts/genes

Let's tell Chanjo which exons belong to which transcripts and which transcripts belong to which genes. It's fine to use the output from Sambamba as long as the two columns after "strand" are present in the file.

```
$ chanjo link sample1.coverage.bed
```

### Loading annotations

After running `sambamba depth region` you can take the output and load it into the database. Let's also add a group identifier to indicate that the sample is related to some other samples.

```
$ for file in *.coverage.bed; do echo "${file}"; chanjo load "${file}"; done
```

### Extracting informtion

We now have some information loaded for a few samples and we can now start exploring what coverage looks like! The output will be in the handy JSON lines format.

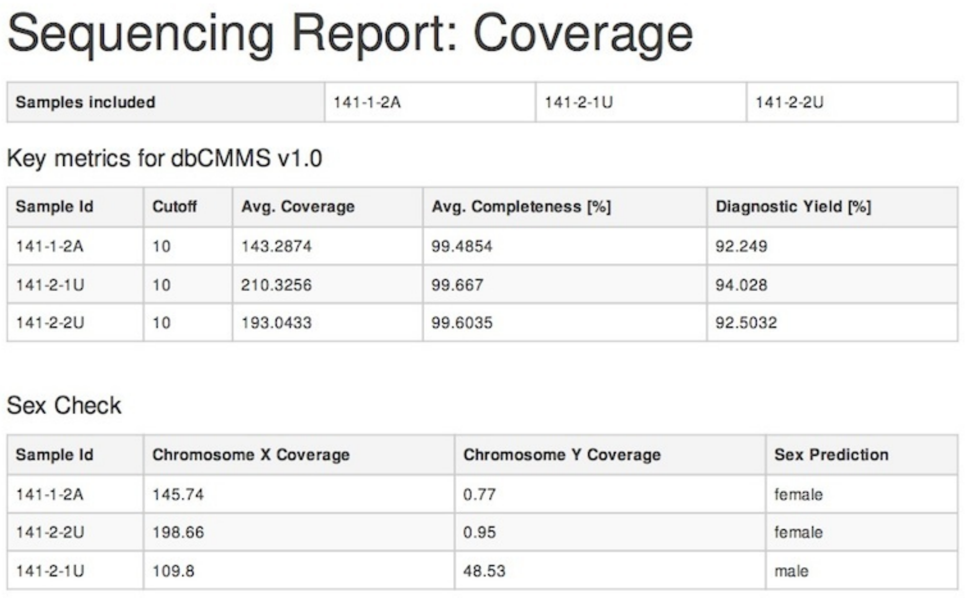
```
$ chanjo calculate mean sample1
{
    "metrics": {
        "completeness_20": 90.38,
        "completeness_10": 90.92,
        "completeness_100": 70.62,
        "mean_coverage": 193.85
    },
    "sample_id": "sample1"
}

$ chanjo calculate gene FAP MUSK
{
    "genes": {
        "MUSK": {
            "completeness_20": 100.0,
            "completeness_10": 100.0,
            "completeness_100": 99.08,
            "mean_coverage": 370.36
        },
        "FAP": {
            "completeness_20": 97.24,
            "completeness_10": 100.0,
            "completeness_100": 50.19,
            "mean_coverage": 151.63
        }
    },
    "sample_id": "sample5"
}
[...]

$ chanjo calculate region 11 619304 619586
{
    "completeness_20": 100.0,
    "completeness_10": 100.0,
    "completeness_100": 100.0,
    "mean_coverage": 258.24
}
$ chanjo calculate region 11 619304 619586 --per exon
{
    "metrics": {
        "completeness_20": 100.0,
        "completeness_10": 100.0,
        "completeness_100": 100.0,
        "mean_coverage": 223.3904
    },
    "exon_id": "11-619305-619389"
}
{
    "metrics": {
        "completeness_20": 100.0,
        "completeness_10": 100.0,
        "completeness_100": 100.0,
        "mean_coverage": 284.23
    },
    "exon_id": "11-619473-619586"
}
```

**Note:** So what is this "completeness"? Well, it's pretty simple; the percentage of bases with at least "sufficient" (say; 10x) coverage.

### 2.2.3 What's next?

The SQL schema has been designed to be a powerful tool on it's own for studying coverage. It let's you quickly aggregate metrics across multiple samples and can be used as a general coverage API for accompanying tools.

One example of such a tool is Chanjo-Report, a coverage report generator for Chanjo output. A report could look something like this (click for the full PDF):



Sequencing Report: Coverage

| Samples included | 141-1-2A | 141-2-1U | 141-2-2U |
|---|---|---|---|

Key metrics for dbCMMS v1.0

| Sample Id | Cutoff | Avg. Coverage | Avg. Completeness [%] | Diagnostic Yield [%] |
|---|---|---|---|---|
| 141-1-2A | 10 | 143.2874 | 99.4854 | 92.249 |
| 141-2-1U | 10 | 210.3256 | 99.667 | 94.028 |
| 141-2-2U | 10 | 193.0433 | 99.6035 | 92.5032 |

Sex Check

| Sample Id | Chromosome X Coverage | Chromosome Y Coverage | Sex Prediction |
|---|---|---|---|
| 141-1-2A | 145.74 | 0.77 | female |
| 141-2-2U | 198.66 | 0.95 | female |
| 141-2-1U | 109.8 | 48.53 | male |

## 2.3 Interface Walkthrough

### 2.3.1 Table of Contents

The page provides an extended look into each of the six subcommands that make up the command line interface of Chanjo. In accordance with UNIX philosophy, they each try to do only one thing really well.

1. *chanjo*
2. *init*
3. *load*
4. *link*
5. *calculate*
6. *db*

### 2.3.2 chanjo

The base command doesn't do anything on it's own. However, there are a few global options accessible at this level. For example, to log debug information to a file called `./chanjo.log` use this command:

```
$ chanjo -v --log ./chanjo.log [SUBCOMMAND HERE]
```

This is also where you can define what database to connect to using the `-d/--database` option. Chanjo will otherwise use the database defined in the config. To learn more about the global Chanjo options, run `chanjo --help`.

### 2.3.3 chanjo init

Walks you through the setup of a Chanjo config file and optionally initialized a new database. With a config you won't have to worry about missing to specify default options on the command line.

The format of the config is `.yaml`.

```
$ chanjo init
```

The generated config file will be stored in the current working directory. This is also where Chanjo will automatically look for it. If you want to share a config file between projects it's possible to point to a global file with the `--config` option.

### 2.3.4 chanjo load

Chanjo takes advantage of the power behind SQL databases by providing a way to store coverage annotations in SQL. You load coverage annotations from `sambamba depth region` output. It's possible to pipe directly to this command:

```
$ sambamba depth region -L exons.bed alignment.bam | chanjo load
```

Each line is added independently so it doesn't really matter if the file is sorted.

Most of the information is already stored in the BED output file from Sambamba but to link multiple samples into logical related groups you can future specify a group identifier when calling the load command:

```
$ chanjo load --group group1 exons.coverage.bed
```

### 2.3.5 chanjo link

Another main benefit of Chanjo is the ability to get coverage for related genomic elements: exons, transcripts, and genes. The "link" only need to be run ones (at any time) and accepts a similar BED file to "*load*".

```
$ chanjo link exons.coverage.bed
```

Each line is added independently so it doesn't really matter if the file is sorted.

### 2.3.6 chanjo calculate

This is where the exiting exploration happens! Chanjo exposes a few powerful ways to investigate coverage ones it's been loaded into the databse.

#### mean

Extract basic coverage stats across all exons for samples.

```
$ chanjo calculate mean
{"complateness_10": 78.93, "mean_coverage": 114.21, "sample_id": "sample1"}
{"complateness_10": 45.92, "mean_coverage": 78.42, "sample_id": "sample2"}
```

#### gene

Extract metrics for particular genes. This requries that the exons have been linked using the "*link*" command to the related transcripts and genes. It should be noted that this information is only an approximation since we don't take overlapping exons into consideration.

```
$ chanjo calculate gene ADK
{"ADK": {"complateness_10": 78.93, "mean_coverage": 114.21}, "sample_id": "sample1"}
```

The calculation is based on simple averages across all exons related to the gene.

#### region

Metrics can also be extracted for a continous interval of exons. This enables some interesting opportunities for exploration. The base command reports average metrics for all included exons across all samples:

```
$ chanjo calculate region 1 122544 185545
{"complateness_10": 50.45, "mean_coverage": 56.12}
```

We can split this up into each individual exon as well for more detail:

```
$ chanjo calculate region 1 122544 185545 --per exon
{"complateness_10": 10.12, "mean_coverage": 12.00, "exon_id": "exon1"}
{"complateness_10": 90.76, "mean_coverage": 114.98, "exon_id": "exon2"}
```

We can of course also filter the results down to individual samples as well:

```
$ chanjo calculate region 1 122544 185545 --per exon --sample sample1
{"complateness_10": 23.56, "mean_coverage": 34.05, "exon_id": "exon1"}
{"complateness_10": 91.86, "mean_coverage": 157.02, "exon_id": "exon2"}
```

### 2.3.7 chanjo db

Enables you to quickly perform housekeeping tasks on the database.

#### setup

Set up and tear down a Chanjo database.

#### remove

Remove all traces of a sample from the database.

```
$ chanjo db remove sample1
```

### 2.3.8 Closing words

The command line interface is really just a bunch of shortcuts that simplifies the use of Chanjo in a UNIX environment. To customize your particular use of Chanjo you would probably want to look into the API Reference.

## 2.4 Public API

This is the *public* API documentation. The functions and classes listed below *exclusively* make up the code scope which is guaranteed to stay consistent.

If you plan to make use of the Public API it would be a good idea to also check out the Developer's Guide that coveres some additional implementation details.

Chanjo exclusively uses unicode strings throughout the interface. It therefore important to always specify 'utf-8' encoding when e.g. reading files from the OS. In Python 2:

```
>>> import io
>>> handle = io.open('./LICENSE', encoding='utf-8')
>>> next(handle)
u'The MIT License (MIT)\n'
```

### 2.4.1 Chanjo coverage store module

The central API for Chanjo SQL databases. Built on SQLAlchemy. From here you have access to contents of the database (models) and can access query interface that SQLAlchemy exposes.

**class** `chanjo.store.`**`Store`**(*uri=None*, *debug=False*)

SQLAlchemy-based database object.

Bundles functionality required to setup and interact with various related genomic interval elements.

Changed in version 2.1.0: Lazy-loadable, all "init" arguments optional.

#### Examples

```
>>> chanjo_db = Store('data/elements.sqlite3')
>>> chanjo_db.set_up()
```

**Note:** For testing pourposes use `:memory:` as the `path` argument to set up in-memory (temporary) database.

**Parameters**

- **`uri`** (*Optional[str]*) – path/URI to the database to connect to
- **`debug`** (*Optional[bool]*) – whether to output logging information

**`uri`**
*str* – path/URI to the database to connect to

**`engine`**
*class* – SQLAlchemy engine, defines what database to use

**`session`**
*class* – SQLAlchemy ORM session, manages persistance

**`query`**
*method* – SQLAlchemy ORM query builder method

**`classes`**
*dict* – bound ORM classes

**add**(*elements*)
Add one or more new elements and commit the changes. Chainable.

**Parameters** **elements** (*orm/list*) – new ORM object instance or list of such

**Returns** `self` for chainability

**Return type** *Store*

**connect**(*db_uri*, *debug=False*)
Configure connection to a SQL database.

New in version 2.1.0.

**Parameters**

- **db_uri** (*str*) – path/URI to the database to connect to
- **debug** (*Optional[bool]*) – whether to output logging information

**create**(*class_id*, **args*, ***kwargs*)
Create a new instance of an ORM element.

If attributes are supplied as a tuple they must be in the correct order. Supplying a *dict* doesn't require the attributes to be in any particular order.

**Parameters**

- **class_id** (*str*) – choice between "superblock", "block", "interval"
- ***args** (*tuple*) – list the element attributes in the *correct order*
- ****kwargs** (*dict*) – element attributes in whatever order you like

**Returns** new ORM instance object

**Return type** orm

**dialect**
Return database dialect name used for the current connection.

Dynamic attribute.

**Returns** name of dialect used for database connection

**Return type** str

**find**(*klass_id*, *query=None*, *attrs=None*)
Fetch one or more elements based on the query.

If the 'query' parameter is a string `find()` will fetch one element; just like *get*. If query is a list it will match element ids to items in that list and return a list of elements. If 'query' is `None` all elements of that class will be returned.

**Parameters**

- **klass_id** (*str*) – type of element to find
- **query** (*str/list, optional*) – element Id(s)
- **attrs** (*list, optional*) – list of columns to fetch

**Returns** element(s) from the database

**Return type** object/list

**get** (*typ*, *type_id*)

Fetch a specific element or ORM class.

Calls itself recursively when asked to fetch an element.

**Parameters**

- **typ** (*str*) – element key or 'class'
- **type_id** (*str*) – element id or ORM model id

**Returns** element or ORM class

**Return type** model

### Examples

```
>>> gene = db.get('gene', 'GIT1')
```

**get_or_create** (*model*, ***kwargs*)

Get or create a record in the database.

**save** ()

Manually persist changes made to various elements. Chainable.

Changed in version 2.1.2: Flush session before commit.

**Returns** `self` for chainability

**Return type** *Store*

**set_up** ()

Initialize a new database with the default tables and columns.

**Returns** self

**Return type** *Store*

**tear_down** ()

Tear down a database (tables and columns).

**Returns** self

**Return type** *Store*

## 2.5 Developer's Guide

The developer guide is directed to fellow coders. You should read this if:

- you want to contribute to the development of Chanjo
- develop a Chanjo plugin that hooks into one of the entry points

### 2.5.1 Contributing

Currently the best resource on this topic is available at GitHub, in the CONTRIBUTING.md file.

### 2.5.2 Installation/dev environment

Check out the installation guide to learn how you can set up a Vagrant environment which is ready to start development in no time!

### 2.5.3 Develop a plugin

Chanjo exposes a couple of plugin interfaces using setuptools entry points.

When publishing a new Chanjo plugin you should register with the corresponding entry point in your `setup.py` script. It might look something like this:

```
setup(
  name='Chanjo-PluginName',
  ...
  entry_points={
    'chanjo.subcommands.3': [
      'plugin_name = chanjo_plugin.cli:main',
    ],
  },
  ...
)
```

The setup above would register a new subcommand to the command line interface as `chanjo plugin`.

When you write a Chanjo plugin name it something like "Chanjo-MyPlugin" to make it easy to find using `pip search`.

**Note:** It's absolutly OK for plugins to to depend on Chanjo itself or any Chanjo dependencies.

### 2.5.4 New subcommand

Setuptools entry point: `chanjo.subcommands.3`

You can write a plugin that will show up as an additional subcommand when you type `chanjo` on the command line.

The implementation should use the Click command line framework. As long as you stick to Click, you can do pretty much whatever you want. Let's your imagination run free! The only requirement is that it should tie into some form of Chanjo functionality like generating a report from a populated SQL database.

### 2.5.5 License

The MIT License (MIT)

Copyright (c) 2014, Robin Andeer

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

## 2.6 Release notes

**Current release**: Radical Red Panda (3.2.0)

# Change Log All notable changes to this project will be documented in this file. This project adheres to [Semantic Versioning](http://semver.org/).

## [3.2.0] - 2015-12-22 ### Changed - diagnostic yield function now accepts "exon_ids" explicitly and requires "sample_id" to be given

## [3.1.1] - 2015-11-19 ### Fixed - fixed bug in SQL relationship between gene and transcript

## [3.1.0] - 2015-11-16 ### Added - *sex* subcommand for guessing sex from BAM alignment, see #158

## [3.0.2] - 2015-11-04 ### Fixed - chanjo API: restrict converter queries to distinct/unique rows

## [3.0.1] - 2015-10-26 ### Fixed - import from root init, use root logger

## [3.0.0] - 2015-10-19 Code name: "Radical Red Panda"

This is a major new release. Please refer to the documentation for more details on what has been updated.

### Added - Add functionality to run sambamba from chanjo - Add calulate command to get basic statistics from database - Add link command to specifically link genomic elements - Add db command to interface and perform maintainance on database

### Removed - Support for older versions of Chanjo

### Fixed - Changed way of logging - Added proper logstream handler

## [2.3.2] - 2015-03-21 ### Fixed - Refactor `EntryPointsCLI` to allow for external subcommands - Updated documentation to reflect Chanjo 2.x CLI

## [2.3.1] - 2015-03-05 ### Changed - use custom "multi command class" to load dynamic entry point plugins

### Fixed - some pylint improvements

## [2.3.0] ### Added - New logging module accessible from the command line (-vvv) - Add SQL schema drawing to "Code walkthrough"

## [2.2.1] ### Fixed - Fix incorrectly references ID field in join statement (block_stats)

## [2.2.0] ### Added - Read sample ID from BAM header - Validate BED format in "annotate" - Enable getting config values from "chanjo.toml" (`chanjo config annotate.cutoff`)

### Fixed - Fix issue with hardlinks in Vagrant shared folders (setup.py) - Change Travis CI setup using official guidelines - Fix typography in docs - Use io.open instead of codecs.open - Use .sqlite3 extension for SQLite databases - Better error message when overwriting existing databases

## [2.1.3] ### Fixed - Fix misstake in "import" subcommand so it's finally working!

## [2.1.2] ### Fixed - Fix bug in "import" where the program didn't flush the session before committing. - Change "build_interval_data" to only create model without adding to the session. - Use "scoped_session" with "sessionmaker". - Flush session before each commit call in `chanjo.Store.save()`

## [2.1.1] ### Fixed - Fix interval assertion that didn't allow intervals to start and end on the same chromosomal position.

## [2.1.0] ### Added - Add lazy loading of ‘‘chanjo.Store‘ through new ‘‘chanjo.Store.connect‘ method - Much improved documentation of changes between releases

### Fixed - Fix case where "demo" subcommand fails (‘‘__package__‘ not set)

## [2.0.2] ### Fixed - Rename misspelled method (non-breaking): ‘‘chanjo.Store.tare_down‘ to ‘‘chanjo.Store.tear_down‘

### Fixed - Fix some CSS selectors in theme - Reorder API references in API docs

## [2.0.1] ### Fixed - Fixes broken symlinked demo/fixture files - Adds validation to check that stdin isn't empty - Fixes link to logo on front page - Adds John Kern as collaborator - Adds link to Master's Thesis paper for reference in README - Adds more FAQ

## [2.0.0] Code name: "Wistful Weasel"

Being a major release, expect previous scripts written for Chanjo 1.x to be incompatible with Chanjo 2.x.

### Added - New built-in "demo" subcommand in the CLI - New public setuptools entry point for Chanjo plug-ins (CLI subcommands) - New of cial public Python API (stable until 3.x release). Read more in the new [API documentation][api-docs]. - New "sex-checker" bonus command to guess gender from BAM alignment.

- **Command line interface updates**
  - `--out` option removed across CLI. Use `>|` to redirect STDOUT instead.
  - `--prepend` is now known as ``--prefix`
  - `--db` and `--dialect` must be supplied directly after "chanjo" on the command line (not after the subcommand). Like: `chanjo --db ./coverage.sqlite import annotations.bed`.
  - `--extend-by` is now `--extendby`
- Config file format has changed from JSON to [TOML][toml]. It's a more readable format (think INI) that also supports comments!
- Improves BED-format compliance. Chanjo will now expect the "score" field to be in position 5 (and strand in position 6). The Chanjo specific fields start from position 7.
- Major internal code restructuring. Essentially everything is built as plugins/self-contained components. Since no official Python API existed pre Chanjo 2, I won't go into any details here.
- Improves documentation.
- Last but not least, Chanjo will now code name releases according to animals in the Musteloidea superfamily :)
- Introduces a new compat module to better support Python 2+3.
- Trades command line framework from "docopt" to "click" to build more flexible nested commands.
- Adds a first hand *BaseInterval* object to unify handling of intervals inside Chanjo.
- BamFile no-longer requires numpy as a hard dependency. You still likely want to keep it though for performance reasons.

## [1.0.0] Code name: "Rebel Raccoon"

First and current stable version of Chanjo.

## [0.6.0] ### Added - BREAKING: changes group_id field to string instead of int. - Exposes the threshold option to the CLI for optimizing BAM-file reading with SAMTools, fixes #58

## [0.5.0] ### Fixed - UPDATE: Small updates to the command line interface - UPDATE: New tests for new functions ### Added - NEW: MySQL support added - CHANGE: A lot of internal restructuring from classes to functions -

IMPROV ENT: New structure seems to significantly improve speed **Documentation** - UPDATE: New documentation covering new features/structure

## [0.4.0] - NEW: Table with Sample meta-data - UPDATE: CLI creates sample entries - UPDATE: SQL structure in docs - UPDATE: Updated tests - UPDATE: included test data (MANIFEST.in) - more on this later...

## [0.3.0] - NEW: API - annotate: splice sites option - NEW: CLI - annotate: splice sites option - UPDATE: Much improved documentation - UPDATE: Modern setuptools only installation - UPDATE: New cleaner banner - NEW: travis integration

## [0.2.0] New CLI!

- New Command Line: “chanjo” replaces “chanjo-autopilot”
- Ability to save a temporary JSON file when running Chanjo in parallel (avoids writing to SQLite in several instances)
- New command line option: peaking into a database
- New command line option: building a new SQLite database skeleton
- New command line option: import temporary JSON files
- New command line option: reading coverage from any interval from BAM-file
- Many small bugfixes and minor improvements
- New dependency: path.py

[api-docs]: https://chanjo.readthedocs.org/en/latest/api.html [toml]: https://github.com/toml-lang/toml

## 2.7 FAQ

### 2.7.1 Why doesn’t Chanjo work on my remote server?

Chanjo complains when trying to import annotations to my SQLite database.

To get to the point it has to do with the NFS filesystem (Network FileSystem or something). Some more complex SQL queries simply aren’t compatible with NFS. So what can you do?

1. Switch to a MySQL database which should work fine.
2. Only process the data to JSON-files with the “–json” flag. Then on a different computer you could import the annotations.

I’m afraid I can’t offer any other solutions at this point.

### 2.7.2 Error: (2006, ‘MySQL server has gone away’)

You need to change a setting in the “my.cnf” file. More information here.

- Known limitations
- Parallelization (postpone import) -> Usage examples
- NFS file systems -> Usage examples
- Troubleshooting

### 2.7.3 I can’t overwrite exising files using Chanjo!

As of the 2.0 release, Chanjo now completely relies on UNIX style output redirection for writing to files. You might, *wisely*, be using `set -o noclobber` in your .bashrc. This raises an error in UNIX if you try to overwrite existing files by output redirection.

The way to force overwrite is by using a special syntax:

```
$ echo two >| afile
$ cat afile
two
```

### 2.7.4 How does Chanjo handle genes in pseudoautosomal regions (X+Y)?

A few genes are present on both sex chromosomes (X+Y). Becuase the chromosomes are treated as separate entities, Chanjo also treat these genes as separate entities. To keep them separated in the SQL database, the default “ccds” converter adapter will prefix their names by “X-” and “Y-” respectively.

**Note:** It’s imporant that all converter adapters find some consistent way of handling elements in these tricky regions.

CHAPTER 3

# made by

# Index