genomics-bcftbx Documentation

Release 1.11.1

Peter Briggs

Mar 15, 2022

Getting started

`genomics-bcftbx` is a Python library (`bcftbx`) and set of utility programs and scripts developed to support NGS and genomics-related bioinformatics within the Bioinformatics Core Facility of the Faculty of Biology, Medicine and Health (FBMH) at the University of Manchester (UoM).

CHAPTER 1

Overview

`genomics-bcftbx` provides a Python library and a set of utilities used for NGS and genomics-related bioinformatics tasks.

The `bcftbx` library provides submodules for various tasks, including:

- handling data from Illumina and SOLiD sequencing platforms;
- working with various file formats including Fastq, Fasta, MS
- Excel (`.xls` and `.xlsx`), HTML and tab-delimited (`.tsv`) files;
- running commands on local and cluster systems;
- general filesystem operations, text manipulation and checksumming.

More details on the `bcftbx` library can be found in *bcftbx library reference*.

The library supports a collection of utilities for tasks including:

- handling Illumina and SOLiD sequencing data;
- reporting outputs from bioinformatics software;
- analysing and reporting microarray data;
- performing basic manipulations on Fastq and Fasta files;
- working with MD5 checksumming of files.

More details on the utilities can be found in usage.

CHAPTER 2

Requirements

2.1 Supported Python versions

The package consists predominantly of code written in Python, and the following versions are supported:

- Python 3.6
- Python 3.7
- Python 3.8

2.2 Software dependencies

Some of the utilities also use 3rd-party software packages, including:

- `STAR` https://github.com/alexdobin/STAR

Other utilities perform post-processing of output from the following external software packages:

- `MACS` https://github.com/macs3-project/MACS
- `bowtie2` http://bowtie-bio.sourceforge.net/bowtie2/index.shtml

CHAPTER 3

Installation

The `genomics-bcftbx` package is available from its GitHub respository at

- https://github.com/fls-bioinformatics-core/genomics

Specific versions can be obtained as `tar.gz` archives from:

- https://github.com/fls-bioinformatics-core/genomics/releases

The software is written in Python (see *Supported Python versions* for a list of supported versions).

It is recommended to install the package into a Python `virtualenv`, for example:

```
virtualenv venv.bcftbx
. venv.bcftbx/bin/activate
```

To install a specific version, first download and unpack the source code, e.g.:

```
wget https://github.com/fls-bioinformatics-core/genomics/archive/2.0.0.tar.gz
tar zxf 2.0.0.tar.gz
```

Then install the package using:

```
pip install ./genomics-2.0.0
```

See the *requirements documentation* for details of other 3rd party software that is needed for specific utilities.

CHAPTER 4

Illumina sequencing data

4.1 Overview

This section outlines the general structure of the data from Illumina based sequencers (GA2x, HiSEQ, MiSEQ, NextSeq, MiniSeq, iSeq etc) and the procedures for converting these data into FASTQ format.

A sequencing run performed on one of these sequencer instruments includes image analysis and base calling, and produces data files in either `.bcl` (binary base call) format, or (more commonly), a compressed version `.bcl.gz` (the *primary sequencing data*).

Additional processing is required to convert these BCL data files to Fastq format for subsequent analysis; this processing is referred to as *BCL-to-fastq conversion*.

In the case of multiplexed runs (i.e. runs where multiple samples are sequenced in a single lane or run, now typically the standard way that samples are sequenced) it is also necessary to perform *demultiplexing* of the data, which assigns data from individual samples into distinct Fastq files; this requires an additional control file called a *sample sheet* which specifies which index sequences belong to which sample.

4.2 Primary sequencing data: structure and naming conventions

The directories produced by the runs use a standard naming format of the form:

```
<date_stamp>_<instrument_name>_<run_id>_<flow_cell>
```

for example `120518_ILLUMINA-13AD3FA_00002_FC`.

The components are interpreted as follows:

- `<date_stamp>`: a 6-digit or 8-digit date stamp in year-month-day format (e.g. `120518` is 18th May 2012)
- `<instrument_name>`: name of the Illumina instrument (e.g. `ILLUMINA-13AD3FA`)
- `<run_id>`: id number corresponding to the run (e.g. `00002`)
- `<flow_cell>`: identifier for the flow cell used for the run (e.g. `FC`)

A partial directory structure is shown below:

```
<YYMMDD>_<INSTRUMENT>_<XXXXX>_<FLOWCELL>/
         |
         +-- Data/
         |     |
         |     +------ Intensities/
         |                    |
         +                    +-- .pos files
         |                    |
         |                    +-- config.xml
         +-- RunInfo.xml      |
                              +-- L001(2,3...)/  (lanes)
                              |
                              +-- BaseCalls/
                                     |
                                     +-- config.xml
                                     |
                                     +-- SampleSheet.csv
                                     |
                                     +--L001(2,3...)/  (lanes)
                                           |
                                           +-- C1.1/   (lane and cycle)
                                                 |
                                                 +-- .bcl(.gz) files
                                                 |
                                                 +-- .stats files
```

Key points:

- The `.bcl` or `.bcl.gz` files are located under the `Data/Intensities/BaseCalls/` directory
- The `config.xml` file under the `BaseCalls` directory is implicitly needed for demultiplexing and fastq conversion
- The `SampleSheet` file is only needed if the demultiplexing needs to be performed.

4.3 BCL-to-Fastq conversion software

Over time Illumina have provided a number of different software packages to perform the BCL-to-Fastq process:

- **CASAVA**: included a Perl script `configureBclToFastq.pl` used to generate a `Makefile` which performed BCL-to-Fastq conversion (`.bcl` only). CASAVA is no longer supported;
- **bclToFastq**: just the BCL-to-Fastq conversion components of CASAVA, with support for both `.bcl` and `.bcl.gz` formats). bclToFastq is no longer supported;
- **bcl2fastq**: (version 1.8.*) provided a single `bcl2fastq` program to perform the BCL-to-Fastq conversion; used the same sample sheet file format as CASAVA and bclToFastq; bcl2fastq is no longer supported;
- **bcl2fastq2**: (version 2 and above) updated the `bcl2fastq` program with new options including combining data for the same samples sequenced across multiple lanes into a single Fastq, and introduced a newer sample sheet format (SampleSheet v1), and modified output directory structure and Fastq naming convention. bcl2fastq2 is still commonly used for BCL-to-Fastq conversion;
- **bcl-convert**: replacement for bcl2fastq2, with a new sample sheet format (SampleSheet v2).

4.4 Demultiplexing: sample sheet files

Multiplexed sequencing allows multiple samples to be run per lane, with the samples being identified by distinct index sequences (barcodes) that are attached to the template during sample preparation.

In order to demultiplex the data associated with each sample after sequencing, the index sequences associated with the sample name has to be supplied to the BCL-to-Fastq conversion software via a *sample sheet file*.

There have been three different sample sheet formats:

- **CASAVA format**: comma-separated (CSV) file with one sample description per line. This format is no longer supported;
- **SampleSheet v1**: (also referred to as "Illumina Experimental Manager" or IEM format) introduced with bcl2fastq2 and also supported by bcl-convert. Divided into different sections containing specific data in CSV format;
- **SampleSheet v2**: introduced with bcl-convert and not supported by earlier BCL-to-Fastq conversion software. Similar structure to SampleSheet v1 but with different sections and parameters.

Note: The *prep_sample_sheet.py* utility can convert between CASAVA and SampleSheet v1/IEM formats; it doesn't currently support SampleSheet v2 format.

4.5 Output directory structure and Fastq naming conventions

Since bcl2fastq2, BCL-to-Fastq conversion has resulted in output directory structures of the form:

```
<OUT_DIR>/
    |
    +-- Project_A/
    |       |
    |       +-- *.fastq.gz file(s)
    |
    +-- Project_B/
    |       |
    |       +-- *.fastq.gz files(s)
    :
    |
    +-- Reports/
    |
    +-- Stats/
    |
    +-- Undetermined*.fastq.gz file(s)
```

Note: It is also possible to have additional "sample" subdirectories within each project, grouping together Fastq files belonging to the same sample, if the sample name and sample ID fields in the sample sheet differ.

Within each project, Fastq files are gzipped and use the following naming scheme:

```
<sample_name>_S<sample_index>_L<lane>_<read_id>_001.fastq.gz
```

e.g. `NA10931_S12_L002_R1_001.fastq.gz`

The sample name is the name supplied in the input sample sheet; the sample index is an integer which indicates the order of the sample within the sample sheet (so it is to some extent arbitrary).

Read IDs are `R1`, `R2` etc for data reads, and `I1`, `I2` etc for index reads.

The lane may be omitted if data for the sample has been combined across all lanes into a single Fastq. For example:

```
NA10931_S12_R1_001.fastq.gz
```

The quality scores in the output fastq files are Phred+33 (see http://en.wikipedia.org/wiki/FASTQ_format#Quality under the "Encoding" section).

When demultiplexing it is likely that the software will be unable to assign some of the reads to a specific sample. In this case these reads will be classed as "undetermined" and will be assigned to files directly under the top-level output directory with the name

```
Undetermined_S0_Llane>_<read_id>_001.fastq.gz
```

Note: The undetermined Fastqs always have sample index zero.

4.6 Legacy outputs

For pre-bcl2fastq BCL-to-Fastq conversion the output directory structure would look like:

```
Unaligned/
    |
    +-- Project_A/
    |          |
    |          +- Sample_1/
    |          |      |
    |          |      +-- *.fastq.gz file(s)
    |          |
    |          +- Sample_2/
    |                 |
    |                 +-- *.fastq.gz file(s)
    |
    +-- Project_B/
    |          |
    |          +- Sample_3/
    |                 |
    |                 +-- *.fastq.gz file(s)
    :
    +-- Undetermined_indexes
```

The general naming scheme for fastq output files is:

```
<sample_name>_<barcode_sequence>_L<lane>_R<read_number>_<set_number>.fastq.gz
```

e.g. `NA10931_ATCACG_L002_R1_001.fastq.gz`

For non-multiplex runs (or in the absence of a sample sheet), one sample is assumed per lane and all samples belong to he same project with the sample name being the lane (e.g. `lane1` etc) and the index barcode sequence set to `NoIndex`, for example:

```
lane1_NoIndex_L001_R1_001.fastq.gz
```

When demultiplexing, the "undetermined" reads are assigned to Fastqs in the `Undetermined_indexes` "project".

CHAPTER 5

SOLiD sequencing data

This section outlines the general structure of the data from SOLiD based sequencers.

5.1 Structure of SOLiD run names

For multiplex fragment sequencing the run names will have the form:

```
<instrument-name>_<date-stamp>_FRAG_BC[_2]
```

(For example: `solid0123_20110315_FRAG_BC`).

The components are:

- `<instrument_name>`: name of the SOLiD instrument e.g. `solid0123`
- `<date-stamp>`: a date stamp in year-month-day format e.g. `20110315` is 15th March 2011
- `FRAG`: indicates a fragment library was used
- `BC`: indicates bar-coding was used (note that not all samples in the run might be bar-coded, even if this appears in the name)
- `2`: if this is present then it indicates the data came from flow cell 2; otherwise it's from flow cell 1.

For multiplex paired-end sequencing the run names have the form:

```
<instrument-name>_<date-stamp>_PE_BC
```

Here the `PE` part of the name indicates a paired-end run.

Note: If the run name contains `WFA` then it's a work-flow analysis and not final sequence data.

See also SOLiD 4 System Instrument Operation Quick Reference (PDF) for more information.

5.2 Navigating SOLiD run data directories

5.2.1 Run definition file

Typically the top-level of a SOLiD run data directory should contain the run definition file which has information about the samples and libraries used in the run, including the names that were assigned when the run was set up. For example for a bar-coded sample this might look like:

```
version        userId  runType isMultiplexing  runName runDesc mask    protocol
v1.3   lab_user        FRAGMENT        TRUE    solid0127_20111013_FRAG_BC
↪ 1_spot_mask_sf  SOLiD4 Multiplex
primerSet      baseLength
BC     10
F3     50
sampleName     sampleDesc      spotAssignments primarySetting  library application
↪ secondaryAnalysis       multiplexingSeries      barcodes
DB_SB_JL_pool          1       default primary DB01    SingleTag       sacCer2 BC Kit
↪Module 1-96      "1"
DB_SB_JL_pool          1       default primary DB02    SingleTag       sacCer2 BC Kit
↪Module 1-96      "2"
...
DB_SB_JL_pool          1       default primary SB_DIMB_2       SingleTag       none
↪ BC Kit Module 1-96      "14"
DB_SB_JL_pool          1       default primary SB_DMTA SingleTag       none    BC Kit
↪Module 1-96      "15"
```

Essentially the run definition file consists of a three sections, each delimited by a header line. The last section (with the header line beginning `sampleName...`) has the information on each of the libraries, and can be used to locate the primary data files.

5.2.2 Primary data files (csfasta/qual) for multiplex fragment sequencing

Locating the primary data files within the SOLiD data directories can be quite tedious and confusing. For bar-coded samples the following heuristic can be used:

1. From the top-level of the SOLiD run directory (e.g. `solid0123_20111013_FRAG_BC`) move into the subdirectory with the sample name of interest (e.g. `DB_SB_JL_pool`, from the run definition file in the previous section).
2. Within the sample subdirectory, look for a directory called `results` (which will be a link to one of the other `results...` directories here). Move into the `results` directory.
3. Within the `results` subdirectory, look for a directory called `libraries` and move into this.
4. Within `libraries` you should see subdirectories named for each of the libraries associated with this sample, as they appear in the run definition file (e.g. `DB01`, `DB02`, ..., `SB_DIMB_2`, `SB_DMTA`). Move into the subdirectory for the library of interest.
5. Within the directory for a specific library, there should be one or more subdirectories with names of the form `primary.20111015000420127` (and possibly also `secondary.20111015000420127`). Check each of these subdirectories looking for the one which itself contains three subdirectories `reads`, `rejects` and `reports` (the others will only contain `reads` and `reports`). Move into this directory, and then into the `reads` subdirectory. This is the location of the primary data files (csfasta and qual files).

Typically this results in a path of the form:

```
solid0123_20111013_FRAG_BC/SAMPLE_NAME/results/libraries/LIBRARY_NAME/primary.
↪TIMESTAMP/reads/
```

As a further check, the primary data file names should include `F3` in the name.

5.2.3 Primary data files (csfasta/qual) for multiplex paired-end sequencing

In the case of paired-end sequencing the final data consists of primary data file pairs for both the `F3` and `F5` reads for each library.

Locating the `F3` and `F5` reads uses a similar heuristic to that described above for multiplex fragment sequencing:

1. From the top-level of the SOLiD run directory, move into the subdirectory for the sample name of interest (e.g. `DB_SB_JL_pool`).
2. Look for the `results` directory and move into it.
3. Look for the `libraries` directory and move into it.
4. Within `libraries` there are subdirectories for each of the libraries associated with this sample (e.g. `DB01`, `DB02`, ..., `SB_DIMB_2`, `SB_DMTA`) - move into the one for the library of interest.
5. Here there are one or more subdirectories with names of the form `primary.20111015000420127` etc. Check each of these subdirectories looking for those which contain three subdirectories `reads`, `rejects` and `reports` (not just `reads` and `reports`). There should be two `primary...` directories which match this criterion: in the `reads` directory of one there will be primary data files with `F5-BC` in the name, and in the other files with `F3`.

5.2.4 Automatic location of primary data using analyse_solid_run.py

The heuristics described above are also encoded in the *analyse_solid_run.py* program, which will identify and report the location of the primary data files when without any other arguments i.e.:

```
analyse_solid_run.py solid0123_20111101_FRAG_BC
```

This works for both multiplex fragment and multiplex paired-end sequencing.

CHAPTER 6

Handling sequencing run data

6.1 Illumina sequencing runs

The *prep_sample_sheet.py* utility can be used for editing and clean-up of sample sheet files that are used as input to the Fastq generation process, including converting between different sample sheet formats.

Examples:

1. Read in the sample sheet file `SampleSheet.csv`, update the `SampleProject` and `SampleID` for lanes 1 and 8, and write the updated sample sheet to the file `SampleSheet2.csv`:

```
prep_sample_sheet.py -o SampleSheet2.csv --set-project=1,8:Control \
     --set-id=1:PhiX_10pM --set-id=8:PhiX_12pM SampleSheet.csv
```

2. Automatically fix spaces and duplicated `sampleID`/`sampleProject` combinations and write out to `SampleSheet3.csv`:

```
prep_sample_sheet.py --fix-spaces --fix-duplicates \
     -o SampleSheet3.csv SampleSheet.csv
```

The `bcftbx` library also provides classes and functions for handling Illumina sequencing data:

- *bcftbx.IlluminaData*

See *Illumina sequencing data* for details of the data structures of raw and processed Illumina sequencing runs.

6.2 SOLiD sequencing runs

The *analyse_solid_run.py* utility can be used to report on the primary data from a run of a SOLiD sequencer instrument, and perform various checks and operations those data.

The `bcftbx` library also provides classes and functions for handling SOLiD sequencing data:

- *bcftbx.SolidData*

- *bcftbx.Experiment*

See *SOLiD sequencing data* for details of the directory structure and contents of a SOLiD sequencing run.

CHAPTER 7

NGS utilities

7.1 Reporting ChIP-seq outputs

The *make_macs2_xls.py* utility can be used to convert an output tab-delimited `.XLS` file from `macs2` into an MS Excel spreadsheet (either `.xlsx` or `.xls` format).

Additionally a `.bed` format file can be output, provided that `macs2` was not run with the `--broad` option.

To process output from older versions of `macs` (i.e. 1.4.2 and earlier) the legacy *make_macs_xls.py* utility can be used; however for this version only MS XLS format is supported, and there is no option to output a `.bed` file.

7.2 Reporting RNA-seq outputs

The *bowtie_mapping_stats.py* utility can be used to summarise the mapping statistics produced by `bowtie2` or `bowtie`, and output to an MS Excel spreadsheet file.

The utility reads the `bowtie2` log file and expects this to consist of multiple blocks of text of the form:

```
...
<SAMPLE_NAME>
Time loading reference: 00:00:01
Time loading forward index: 00:00:00
Time loading mirror index: 00:00:02
Seeded quality full-index search: 00:10:20
# reads processed: 39808407
# reads with at least one reported alignment: 2737588 (6.88%)
# reads that failed to align: 33721722 (84.71%)
# reads with alignments suppressed due to -m: 3349097 (8.41%)
Reported 2737588 alignments to 1 output stream(s)
Time searching: 00:10:27
Overall time: 00:10:27
...
```

The sample name will be extracted along with the numbers of reads processed, with at least one reported alignment, that failed to align, and with alignments suppressed and tabulated in the output spreadsheet.

7.3 Determining strandedness of sequencing data

The *fastq_strand.py* utility can be used to determine the strandedness (forward, reverse, or unstranded) of sequencing data in Fastq format, using either a single Fastq file, or an an R1/R2 pair of Fastqs.

Note: The utility is a wrapper for the `STAR` mapper and requires that `STAR` has been installed separately and is available on the `PATH`.

The simplest example checks the strandedness for a single genome:

```
fastq_strand.py R1.fastq.gz R2.fastq.gz -g STARindex/mm10
```

In this example, `STARindex/mm10` is a directory which contains the `STAR` indexes for the `mm10` genome build.

The output is a file called `R1_fastq_strand.txt` which summarises the forward and reverse strandedness percentages:

```
#fastq_strand version: 0.0.1        #Aligner: STAR  #Reads in subset: 1000
#Genome     1st forward     2nd reverse
STARindex/mm10      13.13   93.21
```

To include the count sums for unstranded, 1st read strand aligned and 2nd read strand aligned in the output file, specify the `--counts` option:

```
#fastq_strand version: 0.0.1        #Aligner: STAR  #Reads in subset: 1000
#Genome     1st forward     2nd reverse     Unstranded      1st read strand aligned␣
↪2nd read strand aligned
STARindex/mm10      13.13   93.21   391087  51339   364535
```

Strandedness can be checked for multiple genomes by specifying additional `STAR` indexes on the command line with multiple `-g` flags:

```
fastq_strand.py R1.fastq.gz R2.fastq.gz -g STARindex/hg38 -g STARindex/mm10
```

Alternatively a panel of indexes can be supplied via a configuration file of the form:

```
#Name       STAR index
hg38        /mnt/data/STARindex/hg38
mm10        /mnt/data/STARindex/mm10
```

(NB blank lines and lines starting with a # are ignored). Use the `-c`/`--conf` option to get the strandedness percentages using a configuration file, For example:

```
fastq_strand.py -c model_organisms.conf R1.fastq.gz R2.fastq.gz
```

By default a random subset of 1000 read pairs is used from the input Fastq pair; this can be changed using the `--subset` option. If the subset is set to zero then all reads are used.

The number of threads used to run `STAR` can be set via the `-n` option; to keep all the outputs from `STAR` specify the `--keep-star-output` option.

The strandedness statistics can also be generated for a single Fastq file, by only specifying one file on the command line. For example:

```
fastq_strand.py -c model_organisms.conf R1.fastq.gz
```

7.4 Manage contaminant sequences for FastQC

The *manage_seqs.py* utility can to help create and update files with lists of so-called "contaminant" sequences, for input into the FastQC program (specifically, via FastQC's `--contaminants` option).

For example, to create a new contaminants file using sequences from a FASTA file:

```
manage_seqs.py -o custom_contaminants.txt sequences.fa
```

To append sequences to an existing contaminants file:

```
manage_seqs.py -a custom_contaminants.txt additional_seqs.fa
```

The inputs can be a mixture of FastQC "contaminants" format and/or Fasta format files). The utility also check for redundancy (i.e. sequences with multiple associated names) and contradictions (i.e. names with multiple associated sequences).

7.5 Convert SAM file to SOAP format

The *sam2soap.py* utility converts a SAM file to SOAP format.

CHAPTER 8

Fastq manipulation

8.1 Extract subsets of reads from Fastq files

The *extract_reads.py* utility extracts subsets of reads from each of the supplied Fastq files according to specified criteria (either a random subset of a specified number reads, or readings matching a specified pattern).

Multiple files are assumed to be pairs (e.g. R1/R2 Fastqs) or groups (R1/I1/R2 Fastqs), with the same number of read records. The same subset will be extracted from each file, so that pairing/grouping is preserved.

Note: Input files can be any mixture of Fastq (`.fastq`, `.fq`), or CSFASTA (`.csfasta`) and QUAL (`.qual`) files.

8.2 Split multi-lane Fastq into individual lanes

Given a multi-lane Fastq file (that is, a Fastq file containing reads for several different sequencer lanes), the *split_fastq.py* utility splits that data into multiple output Fastqs where each file only contains reads from a single lane.

8.3 Verify that Fastq files are paired

The *verify_paired.py* utility verifies that two Fastqs form an R1/R2 pair, by checking that read headers for corresponding records from the input Fastq files are in agreement.

CHAPTER 9

Fasta manipulation

9.1 Extract chromosome sequences from FASTA file

The *split_fasta.py* utility extracts the sequences associated with individual chromosomes from one or more FASTA files.

Specifically, for each chromosome `CHROM` found in the input FASTA file, outputs a file called `CHROM.fa` containing just the sequence for that chromosome.

Sequences are identified as belonging to a specific chromosome by a line `>CHROM`.

9.2 Reorder FASTA file into karyotypic order

The *reorder_fasta.py* utility will reorder the chromosome records in a FASTA file into 'karyotypic' order, for example:

```
chr1
chr2
...
chr10
chr11
```

in contrast to standard alphanumeric sorting (e.g. `chr1`, `chr10`, `chr11`, `chr2` etc).

CHAPTER 10

Microarray data

10.1 Probeset annotation

The *annotate_probesets.py* utility can be used to annotate a probeset list based on probe set names.

It requires a tab-delimited file as input, where the first column comprises the probeset names (any other other columns are ignored), and outputs each name to a new tab-delimited file alongside a description of each.

For example input, the following input:

```
...
1769726_at
1769727_s_at
...
```

generates:

```
...
1769726_at  Rank 1: _at : anti-sense target (most probe sets on the array)
1769727_s_at        Warning: _s_at : designates probe sets that share common probes␣
↪among multiple transcripts from different genes
...
```

10.2 Average data for 'best' exons

The *best_exons.py* utility picks the 'top' three exons for each gene symbol from a tab-delimited (TSV) input file containing the exon data, and outputs a single line for that gene symbol with values averaged over the top three.

'Top' or 'best' exons are determined by ranking on either the `log2FoldChange` (the default) or `pValue` (this is set using the `--rank-by` option):

- For `log2FoldChange`, the 'best' exon is the one with the biggest absolute `log2FoldChange`; if this is positive or zero then takes the top three largest fold change value. Otherwise takes the bottom three.

- For `pValue`, the 'best' exon is the one with the smallest value.

Outputs a TSV file with one line per gene symbol plus the average of each data value for the 3 best exons according to the specified criterion. The averages are just the mean of all the values.

10.2.1 Input file format

Tab separated values (TSV) file, with first line optionally being a header line.

By default the program assumes:

- Column 0: probeset name (change using `--probeset-col`)
- Column 1: gene symbol (change using `--gene-symbol-col`)
- Column 12: log2 fold change (change using `--log2-fold-change-col`)
- Column 13: p-value (change using `--p-value-col`)

Column numbering starts from zero.

10.2.2 Output file format

TSV file with one gene symbol per line plus averaged data for the three 'best' exons (according to the specified criterion), and an extra column which has a $\star$ to indicate which gene symbols had 4 or fewer exons associated with them in the input file.

Note that the averages are just the mean of all the values.

10.3 Cross-reference data for two species

The *xrorthologs.py* utility will cross-reference data from two species, given a lookup file that maps probe set IDs from one species onto those onto the other.

The lookup file is a tab-delimited file with one probe set for species #1 per line in the first column, and a comma-separated list of the equivalent probe sets for species 2 in the fourth column (columns two and three are ignored).

For example:

```
...
121_at      7849    18510   1418208_at,1446561_at
1255_g_at   2978    14913   1421061_a
1316_at     7067    21833   1426997_at,1443952_at,1454675_at
1320_at     11099   24000   1419054_a_at,1419055_a_at,1453298_at
1405_i_at   6352    20304   1418126_at
...
```

Data for the two species are supplied via tab-delimited files `SPECIES1` and `SPECIES2`, where the first column in each is a probe set ID (this is the only requirement).

The output consists of two files:

- `SPECIES1_appended.txt`: a copy of `SPECIES1` with the cross-referenced data from `SPECIES2` appended to each line, and
- `SPECIES2_appended.txt`: a copy of `SPECIES2` with the `SPECIES1` data appended.

Where there are multiple matching orthologs to a probe set ID, the data for each match is appended onto a single line on the output.

CHAPTER 11

Non-bioinformatics utilities

11.1 Checking files and directories using MD5 sums

The `md5checker.py` utility provides a way of checking files and directories using MD5 sums; it can generate a set of MD5 sums for a file or the contents of a directory, and then use these to verify the contents of another file, directory or set of files.

Its basic functionality is very much like the standard `md5sum` Linux program (however note that `md5checker.py` should also work on Windows), but it can also compare two directories directly with MD5 sums, without the need for an intermediate checksum file. This function is intended to provide a straightforward way of running MD5 checks for example when copying analysis of data generated in a cluster scratch area to the archive area.

For example: say you have a directory in `$SCRATCH` called `my_work`, which holds the results of various analysis jobs that you've run on the cluster. At some point you decide to copy these results to the data area:

```
cp -a $SCRATCH/my_work /mnt/data/copy_of_my_work
```

Then you run an MD5 sum check on the copy by doing:

```
md5checker.py --diff $SCRATCH/my_work /mnt/data/copy_of_my_work
```

which by default will generate output of the form:

```
Recursively checking files in /scratch/my_work against copies in /mnt/data/copy_of_my_
↪work
important_data.sam: OK
important_data.bam: OK
...
Summary: 147 files checked, 147 okay 0 failed
```

(Note that this differencing mode only considers files that are in `my_work`, so if `copy_of_my_work` contains additional files then these won't be checked or reported.)

Run `md5checker.py -h` to see the other available options.

11.2 Logging details of sequencing runs

The *log_seq_data.sh* script can be used to add and manage entries for sequencing runs, analyses etc to a tab-delimited "logging file".

For example, logging the primary data directory for a SOLiD sequencing run to the file `SEQ_DATA.log` with the associated description `Primary data`:

```
log_seq_data.sh SEQ_DATA.log /mnt/data/solid0127_20110914_FRAG_BC "Primary data"
```

Logging an analysis directory associated with an Illumina sequencing run, with no description:

```
log_seq_data.sh SEQ_DATA.log /mnt/data/220314_NB189782_0020_AHBXXXYX_analysis
```

Updating an existing entry to add a description:

```
log_seq_data.sh SEQ_DATA.log -u \
    /mnt/data/220314_NB189782_0020_AHBXXXYX_analysis \
    "Analysis of paired end NextSeq run"
```

Deleting an existing entry:

```
log_seq_data.sh SEQ_DATA.log -d /mnt/data/220314_NB189782_0020_AHBXXXYX_analysis
```

CHAPTER 12

Command reference

Note: This documentation has been auto-generated from the command help

The following utilities are available:

- *xrorthologs.py*

12.1 analyse_solid_run.py

```
usage: analyse_solid_run.py [-h] [--version] [--only] [--report]
                            [--report-paths] [--xls] [--verify] [--layout]
                            [--rsync] [--copy COPY_PATTERN]
                            [--gzip GZIP_PATTERN] [--md5 MD5_PATTERN]
                            [--md5sum] [--no-warnings] [--debug]
                            solid_run_dir [solid_run_dir ...]

Utility for performing various checks and operations on SOLiD run directories.
If a single solid_run_dir is specified then analyse_solid_run.py automatically
finds and operates on all associated directories from the same instrument and
with the same timestamp.

positional arguments:
  solid_run_dir         SOLiD run directory to operate on

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  --only                only operate on the specified solid_run_dir, don't
                        locate associated run directories
  --report              print a report of the SOLiD run
  --report-paths        in report mode, also print full paths to primary data
                        files
  --xls                 write report to Excel spreadsheet
  --verify              do verification checks on SOLiD run directories
  --layout              generate script for laying out analysis directories
  --rsync               generate script for rsyncing data
  --copy COPY_PATTERN   copy primary data files to pwd from specific library
                        where names match COPY_PATTERN, which should be of the
                        form '<sample>/<library>'
  --gzip GZIP_PATTERN   make gzipped copies of primary data files in pwd from
                        specific libraries where names match GZIP_PATTERN,
                        which should be of the form '<sample>/<library>'
  --md5 MD5_PATTERN     calculate md5sums for primary data files from specific
                        libraries where names match MD5_PATTERN, which should
                        be of the form '<sample>/<library>'
  --md5sum              calculate md5sums for all primary data files
                        (equivalent to --md5=*/*)
  --no-warnings         suppress warning messages
  --debug               turn on debugging output (nb overrides --no-warnings)
```

12.2 annotate_probesets.py

```
usage: annotate_probesets.py [-h] [--version] [-o OUT_FILE] IN_FILE

Annotate probeset list based on name: reads in first column of tab-delimited
input file 'probe_set_file' as a list of probeset names and outputs these
names to another tab-delimited file with a description for each. Output file
```

(continues on next page)

(continued from previous page)

```
name can be specified with the -o option, otherwise it will be the input file
name with '_annotated' appended.

positional arguments:
  IN_FILE      input probeset file

optional arguments:
  -h, --help   show this help message and exit
  --version    show program's version number and exit
  -o OUT_FILE  specify output file name
```

12.3 best_exons.py

```
usage: best_exons.py [-h] [--version] [--rank-by {log2_fold_change,p_value}]
                     [--probeset-col PROBESET_COL]
                     [--gene-symbol-col GENE_SYMBOL_COL]
                     [--log2-fold-change-col LOG2_FOLD_CHANGE_COL]
                     [--p-value-col P_VALUE_COL] [--debug]
                     EXONS_IN BEST_EXONS

Read exon and gene symbol data from EXONS_IN and picks the top three exons for
each gene symbol, then outputs averages of the associated values to
BEST_EXONS.

positional arguments:
  EXONS_IN              input file with exon and gene symbol data
  BEST_EXONS            output file averages from top three exons for eachgene
                        symbol

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  --rank-by {log2_fold_change,p_value}
                        select the criterion for ranking the 'best' exons;
                        possible options are: 'log2_fold_change' (default), or
                        'p_value'.
  --probeset-col PROBESET_COL
                        specify column with probeset names (default=0, columns
                        start counting from zero)
  --gene-symbol-col GENE_SYMBOL_COL
                        specify column with gene symbols (default=1, columns
                        start counting from zero)
  --log2-fold-change-col LOG2_FOLD_CHANGE_COL
                        specify column with log2 fold change (default=12,
                        columns start counting from zero)
  --p-value-col P_VALUE_COL
                        specify column with p-value (default=13; columns start
                        counting from zero)
  --debug               Turn on debug output
```

12.4 bowtie_mapping_stats.py

```
usage: bowtie_mapping_stats.py [-h] [--version] [-o xls_file] [-t]
                               BOWTIE_LOG_FILE [BOWTIE_LOG_FILE ...]

Extract mapping statistics for each sample referenced in the input bowtie log
files and summarise the data in an XLS spreadsheet. Handles output from both
Bowtie and Bowtie2.

positional arguments:
  BOWTIE_LOG_FILE  logfile output from Bowtie or Bowtie2

optional arguments:
  -h, --help       show this help message and exit
  --version        show program's version number and exit
  -o xls_file      specify name of the output XLS file (otherwise defaults to
                   'mapping_summary.xls').
  -t               write data to tab-delimited file in addition to the XLS
                   file. The tab file will have the same name as the XLS file,
                   with the extension replaced by .txt
```

12.5 extract_reads.py

```
usage: extract_reads.py [-h] [--version] [-m PATTERN] [-n N] [-s SEED]
                        infile [infile ...]

Extract subsets of reads from each of the supplied files according to
specified criteria (e.g. random, matching a pattern etc). Input files can be
any mixture of FASTQ (.fastq, .fq), CSFASTA (.csfasta) and QUAL (.qual).

positional arguments:
  infile                input FASTQ, CSFASTA, or QUAL file

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -m PATTERN, --match PATTERN
                        extract records that match Python regular expression
                        PATTERN
  -n N                  extract N random reads from the input file(s). If
                        multiple files are supplied (e.g. R1/R2 pair) then the
                        same subsets will be extracted for each. (Optionally a
                        percentage can be supplied instead e.g. '50%' to
                        extract a subset of half the reads.)
  -s SEED, --seed SEED  specify seed for random number generator (used for -n
                        option; using the same seed should produce the same
                        'random' sample of reads)
```

12.6 fastq_strand.py

```
Fastq_strand: version 1.11.1
usage: fastq_strand.py [-h] [--version] [-g GENOMEDIR] [--subset SUBSET]
                       [-o OUTDIR] [-c FILE] [-n N] [--counts]
                       [--keep-star-output]
                       READ1 [READ2]

Generate strandedness statistics for FASTQ or FASTQpair, by running STAR using
one or more genome indexes

positional arguments:
  READ1                 R1 Fastq file
  READ2                 R2 Fastq file

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -g GENOMEDIR, --genome GENOMEDIR
                        path to directory with STAR index for genome to use
                        (use as an alternative to -c/--conf; can be specified
                        multiple times to include additional genomes)
  --subset SUBSET       use a random subset of read pairs from the input
                        Fastqs; set to zero to use all reads (default: 10000)
  -o OUTDIR, --outdir OUTDIR
                        specify directory to write final outputs to (default:
                        current directory)
  -c FILE, --conf FILE  specify delimited 'conf' file with list of NAME and
                        STAR index directory pairs. NB if a conf file is
                        supplied then any indices specifed on the command line
                        will be ignored
  -n N                  number of threads to run STAR with (default: 1)
  --counts              include the count sums for unstranded, 1st read strand
                        aligned and 2nd read strand aligned in the output file
                        (default: only include percentages)
  --keep-star-output    keep the output from STAR (default: delete outputs on
                        completion)
```

12.7 log_seq_data.sh

```
/home/docs/checkouts/readthedocs.org/user_builds/genomics-bcftbx/envs/devel/bin/log_
↪seq_data.sh: line 40: bcftbx.functions.sh: No such file or directory
/home/docs/checkouts/readthedocs.org/user_builds/genomics-bcftbx/envs/devel/bin/log_
↪seq_data.sh: line 41: bcftbx.lock.sh: No such file or directory
Usage:
    log_seq_data.sh <logging_file> [-d|-u] <seq_data_dir> [<description>]
    log_seq_data.sh <logging_file> -c <seq_data_dir> <new_dir> [<description>]
    log_seq_data.sh <logging_file> -i <seq_data_dir>
    log_seq_data.sh <logging_file> -v

Add, update or delete an entry for <seq_data_dir> in <logging_file>, or
verify entries.

<seq_data_dir> can be a primary data directory from a sequencer or a
directory of derived data (e.g. analysis directory)

By default an entry is added for the specified data directory; each
```

(continues on next page)

(continued from previous page)

```
entry is a tab-delimited line with the full path for the data directory
followed by the UNIX timestamp and the optional description text.

If <logging_file> doesn't exist then it will be created; if
<seq_data_dir> is already in the log file then it won't be added again.

Options:

     -d     deletes an existing entry
     -u     update description for an existing entry (or creates a new one
            if an existing entry not found)
     -c     changes an existing entry, updating the directory path and
            (optionally) the description
     -i     print information about an entry
     -v     validates the entries in the logging file.
```

12.8 make_macs_xls.py

```
usage: make_macs_xls.py [-h] [--version] MACS_OUTPUT [XLS_OUT]

Create an XLS spreadsheet from the output of the MACS peak caller.
<MACS_OUTPUT> is the output '.xls' file from MACS; if supplied then <XLS_OUT>
is the name to use for the output file, otherwise it will be called
'XLS_<MACS_OUTPUT>.xls'.

positional arguments:
  MACS_OUTPUT  output .xls file from MACS
  XLS_OUT      output MS XLS file (defaults to 'XLS_<MACS_OUTPUT>.xls').

optional arguments:
  -h, --help   show this help message and exit
  --version    show program's version number and exit
```

12.9 make_macs2_xls.py

```
usage: make_macs2_xls.py [-h] [--version] [-f XLS_FORMAT] [-b]
                         MACS2_XLS [XLS_OUT]

Create an XLS(X) spreadsheet from the output of the MACS2 peak caller.
MACS2_XLS is the output '.xls' file from MACS2; if supplied then XLS_OUT is
the name to use for the output file (otherwise it will be called
'XLS_<MACS2_XLS>.xls(x)').

positional arguments:
  MACS2_XLS             output '.xls' file from MACS2
  XLS_OUT               name to use for the output file (default is
                        'XLS_<MACS2_XLS>.xls(x)')

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
```

(continues on next page)

(continued from previous page)

```
  -f XLS_FORMAT, --format XLS_FORMAT
                        specify the output Excel spreadsheet format; must be
                        one of 'xlsx' or 'xls' (default is 'xlsx')
  -b, --bed             write an additional TSV file with chrom,
                        abs_summit+100 and abs_summit-100 data as the columns.
                        (NB only possible for MACS2 run without --broad)
```

12.10 manage_seqs.py

```
usage: manage_seqs.py [-h] [--version] [-o OUT_FILE] [-a APPEND_FILE]
                      [-d DESCRIPTION]
                      INFILE [INFILE ...]

Read sequences and names from one or more INFILEs (which can be a mixture of
FastQC 'contaminants' format and or Fasta format), check for redundancy (i.e.
sequences with multiple associated names) and contradictions (i.e. names with
multiple associated sequences).

positional arguments:
  INFILE          input sequences

optional arguments:
  -h, --help      show this help message and exit
  --version       show program's version number and exit
  -o OUT_FILE     write all sequences to OUT_FILE in FastQC 'contaminants'
                  format
  -a APPEND_FILE  append sequences to existing APPEND_FILE (not compatible
                  with -o)
  -d DESCRIPTION  supply arbitrary text to write to the header of the output
                  file
```

12.11 md5checker.py

```
usage:
  md5checker.py -d SOURCE_DIR DEST_DIR
  md5checker.py -d FILE1 FILE2
  md5checker.py [ -o CHKSUM_FILE ] DIR
  md5checker.py [ -o CHKSUM_FILE ] FILE
  md5checker.py -c CHKSUM_FILE

Compute and verify MD5 checksums for files and directories.

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -d, --diff            for two directories: check that contents of directory
                        DIR1 are present in DIR2 and have the same MD5 sums;
                        for two files: check that FILE1 and FILE2 have the
                        same MD5 sums
  -c, --check           read MD5 sums from the specified file and check them
  -q, --quiet           suppress output messages and only report failures
```

(continues on next page)

(continued from previous page)

```
Directory comparison (-d, --diff):
  Check that the contents of SOURCE_DIR are present in TARGET_DIR and have
  matching MD5 sums. Note that files that are only present in TARGET_DIR are
  not reported.

File comparison (-d, --diff):
  Check that FILE1 and FILE2 have matching MD5 sums.

Checksum generation:
  MD5 checksums are calcuated for all files in the specified directory, or
  for a single specified file.

  -o CHKSUM_FILE, --output CHKSUM_FILE
                        optionally write computed MD5 sums to CHKSUM_FILE
                        (otherwise the sums are written to stdout). The output
                        format is the same as that used by the Linux 'md5sum'
                        tool.

Checksum verification (-c, --check):
  Check MD5 sums for each of the files listed in the specified CHKSUM_FILE
  relative to the current directory. This option behaves the same as the
  Linux 'md5sum' tool.
```

12.12 prep_sample_sheet.py

```
usage: prep_sample_sheet.py [-h] [--version] [-o SAMPLESHEET_OUT] [-f FMT]
                            [-V] [--fix-spaces] [--fix-duplicates]
                            [--fix-empty-projects] [--set-id SAMPLE_ID]
                            [--set-project SAMPLE_PROJECT]
                            [--reverse-complement-i5] [--ignore-warnings]
                            [--include-lanes LANES] [--set-adapter ADAPTER]
                            [--set-adapter-read2 ADAPTER_READ2]
                            [--truncate-barcodes BARCODE_LEN] [--miseq]
                            SAMPLE_SHEET

Utility to prepare SampleSheet files from Illumina sequencers. Can be used to
view, validate and update or fix information such as sample IDs and project
names before running BCL to FASTQ conversion.

positional arguments:
  SAMPLE_SHEET          input sample sheet file

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -o SAMPLESHEET_OUT    output new sample sheet to SAMPLESHEET_OUT
  -f FMT, --format FMT  specify the format of the output sample sheet written
                        by the -o option; can be either 'CASAVA' or 'IEM'
                        (defaults to the format of the original file)
  -V, --view            view predicted outputs from sample sheet
  --fix-spaces          replace spaces in sample ID and project fields with
                        underscores
  --fix-duplicates      append unique indices to sample IDs where the original
```

(continues on next page)

(continued from previous page)

```
                        ID and project name combination are duplicated
  --fix-empty-projects  create sample project names where these are blank in
                        the original sample sheet
  --set-id SAMPLE_ID    update/set the values in sample ID field; SAMPLE_ID
                        should be of the form '<lanes>:<name>', where <lanes>
                        is a single integer (e.g. 1), a set of integers (e.g.
                        1,3,...), a range (e.g. 1-3), or a combination (e.g.
                        1,3-5,7)
  --set-project SAMPLE_PROJECT
                        update/set values in the sample project field;
                        SAMPLE_PROJECT should be of the form
                        '[<lanes>:]<name>', where the optional <lanes> part
                        can be a single integer (e.g. 1), a set of integers
                        (e.g. 1,3,...), a range (e.g. 1-3), or a combination
                        (e.g. 1,3-5,7). If no lanes are specified then all
                        samples will have their project set to <name>
  --reverse-complement-i5
                        replace i5 index sequences with their reverse
                        complement
  --ignore-warnings     ignore warnings about spaces and duplicated
                        sampleID/sampleProject combinations when writing new
                        samplesheet.csv file
  --include-lanes LANES
                        specify a subset of lanes to include in the output
                        sample sheet; LANES should be single integer (e.g. 1),
                        a list of integers (e.g. 1,3,...), a range (e.g. 1-3)
                        or a combination (e.g. 1,3-5,7). Default is to include
                        all lanes
  --set-adapter ADAPTER
                        set the adapter sequence in the 'Settings' section to
                        ADAPTER
  --set-adapter-read2 ADAPTER_READ2
                        set the adapter sequence for read 2 in the
                        'Settings'section to ADAPTER_READ2

Deprecated options:
  --truncate-barcodes BARCODE_LEN
                        trim barcode sequences in sample sheet to number of
                        bases specified by BARCODE_LEN. Default is to leave
                        barcode sequences unmodified (deprecated; only works
                        for CASAVA-style sample sheets)
  --miseq               convert input MiSEQ sample sheet to CASAVA-compatible
                        format (deprecated; specify -f/--format CASAVA to
                        convert IEM sample sheet to older format)
```

12.13 reorder_fasta.py

```
usage: reorder_fasta.py [-h] [--version] FASTA

Reorder the chromosome records in a FASTA file into karyotypic order.

positional arguments:
  FASTA       FASTA file to reorder
```

(continues on next page)

(continued from previous page)

```
optional arguments:
  -h, --help  show this help message and exit
  --version   show program's version number and exit
```

12.14 sam2soap.py

```
usage: sam2soap.py [-h] [--version] [-o SOAPFILE] [--debug] [SAMFILE]

Convert SAM file to SOAP format - reads from stdin (or SAMFILE, if specified),
and writes output to stdout unless -o option is specified.

positional arguments:
  SAMFILE      SAM file to convert (or stdin if not specified)

optional arguments:
  -h, --help   show this help message and exit
  --version    show program's version number and exit
  -o SOAPFILE  Output SOAP file name
  --debug      Turn on debugging output
```

12.15 split_fasta.py

```
usage: split_fasta.py [-h] [--version] [fasta_file]

Split input FASTA file with multiple sequences into multiple files each
containing sequences for a single chromosome.

positional arguments:
  fasta_file  input FASTA file to split

optional arguments:
  -h, --help  show this help message and exit
  --version   show program's version number and exit
```

12.16 split_fastq.py

```
usage: split_fastq.py [-h] [--version] [-l LANES] FASTQ

Split input Fastq file into multiple output Fastqs where each output only
contains reads from a single lane.

positional arguments:
  FASTQ                 Fastq to split

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -l LANES, --lanes LANES
```

(continues on next page)

(continued from previous page)

```
                        lanes to extract: can be a single integer, a comma-
                        separated list (e.g. 1,3), a range (e.g. 5-7) or a
                        combination (e.g. 1,3,5-7). Default is to extract all
                        lanes in the Fastq
```

12.17 verify_paired.py

```
usage: verify_paired.py [-h] [--version] R1.fastq R2.fastq

Check that read headers for R1 and R2 fastq files are in agreement, and that
the files form an R1/2 pair.

positional arguments:
  R1.fastq    Fastq file with R1 reads
  R2.fastq    Fastq file with R2 reads to check against R1 reads

optional arguments:
  -h, --help  show this help message and exit
  --version   show program's version number and exit
```

12.18 xrorthologs.py

```
usage: xrorthologs.py [-h] [--version] [--debug] LOOKUPFILE SPECIES1 SPECIES2

Cross-reference data from two species given a lookup file that maps probeset
IDs from one species onto those onto the other. LOOKUPFILE is tab-delimited
file with one probe set for species 1 per line in first column and a comma-
separated list of the equivalent probe sets for species 2 in the fourth
column. Data for the two species are in tab-delimited files SPECIES1 and
SPECIES2. Output is two files: SPECIES1_appended.txt (SPECIES1 with the cross-
referenced data from SPECIES2 appended to each line) and SPECIES2_appended.txt
(SPECIES2 with SPECIES1 data appended).

positional arguments:
  LOOKUPFILE  tab-delimited file with one probe set for species 1 per line in
              first column and a comma-separated list of the equivalent probe
              sets for species 2 in the fourth column
  SPECIES1    data for species 1
  SPECIES2    data for species 2

optional arguments:
  -h, --help  show this help message and exit
  --version   show program's version number and exit
  --debug     Turn on debugging output
```

CHAPTER 13

bcftbx library reference

13.1 bcftbx.IlluminaData

Provides classes for extracting data about runs of Illumina-based sequencers (e.g. GA2x or HiSeq) from directory structure, data files and naming conventions.

13.1.1 Core data and run handling classes

class bcftbx.IlluminaData.**IlluminaData**(*illumina_analysis_dir*, *unaligned_dir='Unaligned'*)

Class for examining Illumina data post bcl-to-fastq conversion

Provides the following attributes:

- analysis_dir: top-level directory holding the 'Unaligned' subdirectory with the primary fastq.gz files
- projects: list of IlluminaProject objects (one for each project defined at the fastq creation stage)
- undetermined: IlluminaProject object for the undetermined reads
- unaligned_dir: full path to the 'Unaligned' directory holding the primary fastq.gz files
- paired_end: True if at least one project is paired end, False otherwise
- format: Format of the directory structure layout (either 'casava' or 'bcl2fastq2', or None if the format cannot be determined)
- lanes: List of lane numbers present; if there are no lanes then this will be a list with 'None' as the only value

Provides the following methods:

- get_project(): lookup and return an IlluminaProject object corresponding to the supplied project name

class bcftbx.IlluminaData.**IlluminaProject**(*dirn*)

Class for storing information on a 'project' within an Illumina run

A project is a subset of fastq files from a run of an Illumina sequencer; in the first instance projects are defined within the SampleSheet.csv file which is output by the sequencer.

Note that the "undetermined" fastqs (which hold reads for each lane which couldn't be assigned to a barcode during demultiplexing) is also considered as a project, and can be processed using an IlluminaProject object.

Provides the following attributes:

- name: name of the project
- dirn: (full) path of the directory for the project
- expt_type: the application type for the project e.g. RNA-seq, ChIP-seq (initially set to None; should be explicitly set by the calling subprogram)
- samples: list of IlluminaSample objects for each sample within the project
- paired_end: True if all samples are paired end, False otherwise
- undetermined: True if 'samples' are actually undetermined reads

class bcftbx.IlluminaData.**IlluminaRun**(*illumina_run_dir*, *platform=None*)
Class for examining 'raw' Illumina data directory.

Provides the following properties:

- run_dir: name and full path to the top-level data directory
- basecalls_dir: name and full path to the subdirectory holding bcl files
- sample_sheet_csv: full path of the SampleSheet.csv file
- runinfo_xml: full path of the RunInfo.xml file
- platform: platform e.g. 'miseq'
- bcl_extension: file extension for bcl files (either "bcl" or "bcl.gz")
- lanes: list of (integer) lane numbers in the run
- sample_sheet: SampleSheet instance (if the run has an associated sample sheet file)
- runinfo: IlluminaRunInfo instance (if the run has an associated RunInfo.xml file)

class bcftbx.IlluminaData.**IlluminaRunInfo**(*runinfo_xml*)
Class for examining Illumina RunInfo.xml file

Extracts basic information from a RunInfo.xml file:

- run_id: the run id e.g.'130805_PJ600412T_0012_ABCDEZXDYY'
- run_number: the run number e.g. '0012'
- instrument: the instrument name e.g. 'PJ600412T'
- date: the run date e.g. '130805'
- flowcell: the flowcell id e.g. 'ABCDEZXDYY'
- lane_count: the flowcell lane count e.g. 8
- bases_mask: bases mask string derived from the read information e.g. 'y101,I6,y101'
- reads: a list of Python dictionaries (one per read)

Each dictionary in the 'reads' list has the following keys:

- number: the read number (1,2,3,...)
- num_cycles: the number of cycles in the read e.g. 101

- is_indexed_read: whether the read is an index (i.e. barcode); either 'Y' or 'N'

class bcftbx.IlluminaData.**IlluminaSample**(*dirn*, *fastqs=None*, *name=None*, *prefix='Sample_'*)

Class for storing information on a 'sample' within an Illumina project

A sample is a fastq file generated within an Illumina sequencer run.

Provides the following attributes:

- name: sample name
- dirn: (full) path of the directory for the sample
- fastq: name of the fastq.gz file (without leading directory, join to 'dirn' to get full path)
- paired_end: boolean; indicates whether sample is paired end

13.1.2 Samplesheet handling

class bcftbx.IlluminaData.**SampleSheet**(*sample_sheet=None*, *fp=None*)

Class for handling Illumina sample sheets

This is a general class which tries to handle and convert between older (i.e. 'CASAVA'-style) and newer (IEM-style) sample sheet files for Illumina sequencers, in a transparent manner.

The Experimental Manager (IEM) samplel sheets are text files with data delimited by '[...]' lines e.g. '[Header]', '[Reads]' etc.

The 'Header' section consists of comma-separated key-value pairs e.g. 'Application,HiSeq FASTQ Only'.

The 'Reads' section consists of values (one per line) (possibly number of bases per read?) e.g. '101'.

The 'Settings' section consists of comma-separated key-value pairs e.g. 'Adapter,CTGTCTCTTATACACATCT'.

The 'Manifests' section consists of comma-separated key-filename pairs e.g. 'A,TruSeqAmpliconManifest-1.txt'.

The 'Data' section contains the data about the lanes, samples and barcode indexes. It consists of lines of comma-separated values, with the first line being a 'header', and the remainder being values for each of those fields.

This older style of sample sheet is used by CASAVA and bcl2fastq v1.8.*. It consists of lines of comma-separated values, with the first line being a 'header' and the remainder being values for each of the fields:

- FCID: flow cell ID
- Lane: lane number (integer from 1 to 8)
- SampleID: ID (name) for the sample
- SampleRef: reference used for alignment for the sample
- Index: index sequences (multiple index reads are separated by a hyphen e.g. ACCAGTAA-GGACATGA
- Description: Description of the sample
- Control: Y indicates this lane is a control lane, N means sample
- Recipe: Recipe used during sequencing
- Operator: Name or ID of the operator
- SampleProject: project the sample belongs to

Although the CASAVA-style sample sheet looks much like the IEM 'Data' section, note that it has different fields and field names.

To load data from an IEM-format file:

```
>>> iem = SampleSheet('SampleSheet.csv')
```

To access 'header' items:

```
>>> iem.header_items
['IEMFileVersion','Date',..]
>>> iem.header['IEMFileVersion']
'4'
```

To access 'reads' data:

```
>>> iem.reads
['101','101']
```

To access 'settings' items:

```
>>> iem.settings_items
['ReverseComplement',...]
>>> iem.settings['ReverseComplement']
'0'
```

To access 'manifests' items:

```
>>> iem.manifests_items
['A',...]
>>> iem.manifests['A']
'TruSeqAmpliconManifest-1.txt'
```

To access 'data' (the actual sample sheet information):

```
>>> iem.data.header()
['Lane','Sample_ID',...]
>>> iem.data[0]['Lane']
1
```

etc.

To load data from a CASAVA style sample sheet:

```
>>> casava = SampleSheet('SampleSheet.csv')
```

To access the data use the 'data' property:

```
>>> casava.data.header()
['Lane','SampleID',...]
>>> casava.data[0]['Lane']
1
```

The data in the 'Data' section can be accessed directly from the SampleSheet instance, e.g.

```
>>> iem[0]['Lane']
```

is equivalent to

```
>>> iem.data[0]['Lane']
```

It is also possible to set new values for data items using this notation.

The data lines can be iterated over using:

```
>>> for line in iem:
>>> ...
```

To find the number of lines that are stored:

```
>>> len(iem)
```

To append a new line:

```
>>> new_line = iem.append(...)
```

A number of methods are available to check and fix common problems, specifically:

- detect and replace 'illegal' characters in sample and project names
- detect and fix duplicated sample name, project and lane combinations
- detect blank sample and project names

Data is loaded it is also subjected to some basic cleaning up, including stripping of unnecessary commas and white space. The 'show' method returns a reconstructed version of the original sample sheet after the cleaning operations were performed.

class bcftbx.IlluminaData.**CasavaSampleSheet**(*samplesheet=None*, *fp=None*)
Class for reading and manipulating sample sheet files for CASAVA

This class is a subclass of the SampleSheet class, and provides an additional method ('casava_sample_sheet') to convert to a CASAVA-style sample sheet, suitable for input into bcl2fastq version 1.8.*.

Raises IlluminaDataError exception if the input data doesn't appear to be in the correct format.

class bcftbx.IlluminaData.**IEMSampleSheet**(*sample_sheet=None*, *fp=None*)
Class for handling Experimental Manager format sample sheet

This class is a subclass of the SampleSheet class, and provides an additional method ('casava_sample_sheet') to convert to a CASAVA-style sample sheet, suitable for input into bcl2fastq version 1.8.*.

bcftbx.IlluminaData.**convert_miseq_samplesheet_to_casava**(*samplesheet=None*, *fp=None*)
Convert a Miseq sample sheet file to CASAVA format

Reads the data in a Miseq-format sample sheet file and returns a CasavaSampleSheet object with the equivalent data.

Note: this is now just a wrapper for the more general conversion function 'get_casava_sample_sheet' (which can handle the conversion without knowing a priori what the SampleSheet format is.

Parameters **samplesheet** – name of the Miseq sample sheet file

Returns A populated CasavaSampleSheet object.

bcftbx.IlluminaData.**get_casava_sample_sheet**(*samplesheet=None*, *fp=None*, *FCID_default='FC1'*)
Load data into a 'standard' CASAVA sample sheet CSV file

Reads the data from an Illumina platform sample sheet CSV file and populates and returns a CasavaSampleSheet object which can be used to generate make a SampleSheet suitable for bcl-to-fastq conversion.

The source sample sheet may be in the format output by the Experimental Manager software (needed when running BaseSpace) or may already be in "standard" format for bcl-to-fastq format.

For Experimental Manager format, the sample sheet consists of sections delimited by headers of the form "[Header]", "[Reads]" etc. The information about the sample names and barcodes are in the "[Data]" section, which is essentially a list of CSV format lines with the following fields:

MiSEQ:

Sample_ID,Sample_Name,Sample_Plate,Sample_Well,I7_Index_ID,index, Sample_Project,Description

HiSEQ:

Lane,Sample_ID,Sample_Name,Sample_Plate,Sample_Well,I7_Index_ID, index,Sample_Project,Description

(Note that for dual-indexed runs the fields are e.g.:

Sample_ID,Sample_Name,Sample_Plate,Sample_Well,I7_Index_ID,index, I5_Index_ID,index2,Sample_Project,Description

i.e. there are an additional pair of fields describing the second index)

The conversion maps a subset of these onto fields in the Casava format:

Sample_ID -> SampleID index -> Index Sample_Project -> SampleProject Description -> Description

If no lane information is present in the original file then this is set to 1. The FCID is set to an arbitrary value.

For dual-indexed samples, the Index field is generated by putting together the index and index2 fields.

All other fields are left empty.

Parameters

- **samplesheet** – name of the Miseq sample sheet file
- **FCID_default** – name to use for flow cell ID if not present in the source file (optional)

Returns A populated CasavaSampleSheet object.

bcftbx.IlluminaData.**verify_run_against_sample_sheet**(*illumina_data*, *sample_sheet*, *include_sample_dir=False*)

Checks existence of predicted outputs from a sample sheet

Parameters

- **illumina_data** – a populated IlluminaData directory
- **sample_sheet** – path and name of a CSV sample sheet
- **include_sample_dir** – if True then always include a 'sample_name' directory level when checking for bcl2fastq2 outputs

Returns

True if all the predicted outputs from the sample sheet are found, False otherwise.

bcftbx.IlluminaData.**samplesheet_index_sequence**(*line*)

Return the index sequence for a sample sheet line

Parameters **line** (TabDataLine) – line from a SampleSheet instance

Returns barcode sequence, or 'None' if not defined.

Return type String

bcftbx.IlluminaData.**normalise_barcode**(*seq*)

Return normalised version of barcode sequence

This standardises the sequence so that:

- all bases are uppercase
- dual index barcodes have '-' and '+' removed

13.1.3 Utility classes and functions

class bcftbx.IlluminaData.**IlluminaFastq**(*fastq*)

Class for extracting information about Fastq files

Given the name of a Fastq file from CASAVA/Illumina platform, extract data about the sample name, barcode sequence, lane number, read number and set number.

For Fastqs produced by CASAVA and bcl2fastq v1.8, the format of the names follows the general form:

<sample_name>_<barcode_sequence>_L<lane_number>_R<read_number>_<set_number>.fastq.gz

e.g. for

NA10831_ATCACG_L002_R1_001.fastq.gz

sample_name = 'NA10831' barcode_sequence = 'ATCACG' lane_number = 2 read_number = 1 set_number = 1

For Fastqs produced by bcl2fast v2, the format looks like:

<sample_name>_S<sample_number>_L<lane_number>_R<read_number>_<set_number>.fastq.gz

e.g. for

NA10831_S4_L002_R1_001.fastq.gz

sample_name = 'NA10831' sample_number = 4 lane_number = 2 read_number = 1 set_number = 1

Provides the follow attributes:

fastq: the original fastq file name sample_name: name of the sample (leading part of the name) sample_number: number of the same (integer or None, bcl2fastq v2 only) barcode_sequence: barcode sequence (string or None, CASAVA/bcl2fast v1.8 only) lane_number: integer read_number: integer set_number: integer

bcftbx.IlluminaData.**describe_project**(*illumina_project*)

Generate description string for samples in a project

Description string gives the project name and a human-readable summary of the sample names, plus number of samples and whether the data is paired end.

Example output: "Project Control: PhiX_1-2 (2 samples)"

Arguments illumina_project: IlluminaProject instance

Returns Description string.

bcftbx.IlluminaData.**fix_bases_mask**(*bases_mask*, *barcode_sequence*)

Adjust input bases mask to match actual barcode sequence lengths

Updates the bases mask string extracted from RunInfo.xml so that the index read masks correspond to the index barcode sequence lengths given e.g. in the SampleSheet.csv file.

For example: if the bases mask is 'y101,I7,y101' (i.e. assigning 7 cycles to the index read) but the barcode sequence is 'CGATGT' (i.e. only 6 bases) then the adjusted bases mask should be 'y101,I6n,y101'.

Parameters

- **bases_mask** – bases mask string e.g. 'y101,I7,y101','y250,I8,I8,y250'
- **barcode_sequence** – index barcode sequence e.g. 'CGATGT' (single

- **'TAAGGCGA-TAGATCGC'** (*index*),)–

Returns Updated bases mask string.

bcftbx.IlluminaData.**get_unique_fastq_names**(*fastqs*)
Generate mapping of full fastq names to shorter unique names

Given an iterable list of Illumina file fastq names, return a dictionary mapping each name to its shortest unique form within the list.

Parameters **fastqs** – an iterable list of fastq names

Returns Dictionary mapping fastq names to shortest unique versions

bcftbx.IlluminaData.**split_run_name**(*dirname*)
Split an Illumina directory run name into components

Given a directory for an Illumina run, e.g.

140210_M00879_0031_000000000-A69NA

split the name into components and return as a tuple:

(date_stamp,instrument_name,run_number)

e.g.

('140210','M00879','0031')

Note that this function doesn't return the flow cell ID; use the `split_run_name_full` function to also extract the flow cell information.

bcftbx.IlluminaData.**summarise_projects**(*illumina_data*)
Short summary of projects, suitable for logging file

The summary description is a one line summary of the project names along with the number of samples in each, and an indication if the run was paired-ended.

Parameters **illumina_data** – a populated IlluminaData directory

Returns Summary description.

13.1.4 Exception classes

class bcftbx.IlluminaData.**IlluminaDataError**
Base class for errors with Illumina-related code

13.2 bcftbx.SolidData

Provides classes for extracting data about SOLiD runs from directory structure, data files and naming conventions.

Typical usage is to create a new SolidRun instance by pointing it at the top-level output directory produced by the sequencer:

```
>>> solid_run = SolidRun('/path/to/solid0123_20141225_FRAG_BC')
```

This will automatically attempt to collect the data about the run, which can then be accessed via other objects linked through the SolidRun object's properties.

The most useful are:

- SolidRun.run_info: a SolidRunInfo object which holds data extracted from the run name (e.g. instrument, datestamp etc)
- SolidRun.samples: a list of SolidSample objects which hold data about each of the samples in the run.

Each sample in turn holds a list of libraries within that sample (SolidLibrary objects in 'SolidSample.libraries') and a list of projects (SolidProject objects in 'SolidSample.projects'). The 'getLibrary' and 'getProject' methods also provide ways to look up specific libraries or projects.

Projects are groupings of libraries (based on library names) which are assumed to form a single experiment. The libraries within a project can be obtained via the SolidLibrary.projects, or using the 'getLibrary' method.

Finally, SolidLibrary objects hold data about the location of the primary data files. The 'SolidLibrary.csfasta' and 'SolidLibrary.qual' properties hold the locations of the data for the F3 reads, while for paired-end runs the 'SolidLibrary.csfasta_f5' and 'SolidLibrary.qual_f5' properties point to the F5 reads.

(The 'is_paired_end' function can be used to test whether a SolidRun object holds data for a paired-end run.)

13.2.1 SolidRun

class bcftbx.SolidData.**SolidRun**(*solid_run_dir*)
Describe a SOLiD run.

The SolidRun class provides an interface to data about a SOLiD run. It analyses the SOLiD data directory to look for run definitions, statistics files and primary data files.

It uses the same terminology as the SETS interface and the data files produced by the SOLiD instrument, so a run contains 'samples' and each sample contains one or more 'libraries'.

One initialised, access the data about the run via the SolidRun object's properties:

- run_dir: directory with the run data
- run_name: name of the run e.g. solid0123_20130426_FRAG_BC
- run_info: a SolidRunInfo object with data derived from the run name
- run_definition: a SolidRunDefinition object with data extracted from the run_definition.txt file
- samples: a list of SolidSample objects representing the samples in the run

class bcftbx.SolidData.**SolidRunInfo**(*run_name*)
Extract data about a run from the run name

Run names are of the form 'solid0123_20130426_FRAG_BC_2'

This class analyses the name and breaks it down into components that can be accessed as object properties, specifically:

name: the supplied run name instrument: the instrument name e.g. solid0123 datestamp: e.g. 20130426 is_fragment_library: True or False is_barcoded_sample: True or False flow_cell: 1 or 2 date: datestamp reformatted as DD/MM/YY id: the run name without any flow cell identifier

class bcftbx.SolidData.**SolidRunDefinition**(*run_definition_file*)
Class to store data from a SOLiD run definition file

Once the SolidRunDefinition object is populated from a run definition file, use the 'nSamples' method to find out how many 'samples' (actually sample/library pairs) are defined, and the 'fields' method to get a list of column headings for each.

Data can be extracted for each sample using the 'getDataItem' method to look up the value for a particular field on a particular line, e.g.:

```
>>> library = run_defn.getDataItem('library',0)
```

The SolidRunDefinition object also has a number of attributes populated from the header of the run definition file, specifically:

version, userId, runType, isMultiplexing, runName, runDesc, mask and protocol.

The attributes are strings and can be accessed directly from the object, e.g.:

```
>>> version = run_defn.version
>>> isMultiplexing = run_defn.isMultiplexing
```

class bcftbx.SolidData.**SolidBarcodeStatistics**(*barcode_statistics_file*)
Store data from a SOLiD BarcodeStatistics file

class bcftbx.SolidData.**SolidProject**(*name*, *run=None*, *sample=None*)
Class to hold information about a SOLiD 'project'

A SolidProject object holds a collection of libraries which together constitute a 'project'.

The definition of a 'project' is quite loose in this context: essentially it's a grouping of libraries within a sample. Typically the grouping is by the initial letters of the library name e.g. DR for DR1, EP for EP_NCYC2669 - but this determination is made at the application level.

Libraries are added to the project via the addLibrary method. Data about the project can be accessed via the following properties:

name: the project name (supplied on object creation) libraries: a list of libraries in the project

Also has the following methods:

- getSample(): returns the parent SolidSample
- getRun(): returns the parent SolidRun
- isBarcoded(): returns boolean indicating whether the libraries in the sample are barcoded

13.2.2 SolidSample

class bcftbx.SolidData.**SolidSample**(*name*, *parent_run=None*)
Store information about a sample in a SOLiD run.

A sample has a name and contains a set of libraries. The information about the sample can be accessed via the following properties:

- name: the sample name
- libraries: a list of SolidLibrary objects representing the libraries within the sample
- projects: a list of SolidProject objects representing groups of related libraries within the sample
- unassigned: SolidProject object representing the 'unassigned' data
- barcode_stats: a SolidBarcodeStats object with data extracted from the BarcodeStatistics file (or None, if no file was available)
- parent_run: the parent SolidRun object, or None.

The class also provides the following methods:

- addLibrary: to create and append a SolidLibrary object
- getLibrary: fetch an existing SolidLibrary

- getProject: fetch an existing SolidProject

Typically the calling subprogram calls the 'addLibrary' method to add a SolidLibrary object, which it then populates itself.

The SolidSample class automatically creates SolidProject objects based on the library names to group libraries considered to belong to the same experiments.

13.2.3 SolidLibrary

class bcftbx.SolidData.**SolidLibrary**(*name*, *parent_sample=None*)

Store information about a SOLiD library.

The following properties hold data about the library:

- name: the library name
- initials: the experimenter's initials
- prefix: the library name prefix (i.e. name without the trailing numbers)
- index_as_string: the trailing numbers from the name, as a string (preserves any leading zeroes)
- index: the trailing numbers from the name as an integer
- csfasta: full path to the csfasta file for the library (F3 reads)
- qual: full path to qual file for the library (F3 reads)
- csfasta_f5: full path to the F5 read (paired-end runs, otherwise will be None)
- qual_f5: full path to the F5 read (paired-end runs, otherwise will be None)
- primary_data: list of SolidPrimaryData objects for all possible primary data file pairs associated with the library
- parent_sample: parent SolidSample object, or None.

The following methods are also available:

- addPrimaryData: creates a new SolidPrimaryData object and appends to the list in the primary_data property

13.2.4 SolidPrimaryData

class bcftbx.SolidData.**SolidPrimaryData**

Class to store references to primary data files

This is a convenience class for storing references to csfasta/qual file pairs within a SolidLibrary instance.

The class provides the following attributes:

csfasta: full path to csfasta file qual: full path to qual file timestamp: timestamp associated with the file pair type: string indicating 'F3' or 'F5', or None

The following methods are provided:

is_f3: indicates if data is F3 is_f5: indicates if data is F5

13.2.5 Functions

bcftbx.SolidData.**extract_library_timestamp**(*path*)
Extract the timestamp string from a path

Given a path of the form '/path/to/data/.../primary.1234567/...', return the timestamp string attached to the 'primary.XXXXXXX' component of the name.

Parameters **path** – absolute or relative path to arbitrary directory or file in the SOLiD data structure

Returns Timestamp string, or None if no timestamp was identified.

bcftbx.SolidData.**get_primary_data_file_pair**(*dirn*)
Return csfasta/qual file pair from specified directory

Parameters **dirn** – directory to search for csfasta/qual pair

Returns Tuple (csfasta,qual) with full path for each file, or (None,None) if a pair wasn't located.

bcftbx.SolidData.**is_paired_end**(*solid_run*)
Determine if a SolidRun instance is a paired-end run

Parameters **solid_run** – a populated SolidRun instance

Returns True if this is a paired-end run, False otherwise.

bcftbx.SolidData.**match**(*pattern*, *word*)
Check if a word matches pattern

Implements a very simple pattern matching algorithm, which allows only exact matches or glob-like strings (i.e. using a trailing '*' to indicate a wildcard).

For example: 'ABC*' matches 'ABC', 'ABC1', 'ABCDEFG' etc, while 'ABC' only matches itself.

Parameters

- **pattern** – simple glob-like pattern
- **word** – string to test against 'pattern'

Returns True if 'word' is a match to 'pattern', False otherwise.

bcftbx.SolidData.**slide_layout**(*nsamples*)
Description of the slide layout based on number of samples

Parameters **nsamples** – number of samples in the run

Returns A string describing the slide layout for the run based on the number of samples in the run, e.g. "Whole slide", "Quads", "Octets" etc. Returns None if the number of samples doesn't map to a recognised layout.

13.3 bcftbx.Experiment

Experiment.py

The Experiment module provides two classes: the Experiment class defines a single experiment (essentially a collection of one or more related primary data sets) from a SOLiD run; the ExperimentList class is a collection of experiments which are typically part of the same SOLiD run.

class bcftbx.Experiment.**Experiment**
Class defining an experiment from a SOLiD run.

An 'experiment' is a collection of related data.

copy()
Return a new Experiment instance which is a copy of this one.

describe()
Describe the experiment as a set of command line options

dirname(*top_dir=None*)
Return directory name for experiment

The directory name is the supplied name plus the experiment type joined by an underscore, unless no type was specified (in which case it is just the experiment name).

If top_dir is also supplied then this will be prepended to the returned directory name.

class bcftbx.Experiment.**ExperimentList**(*solid_run_dir=None*)
Container for a collection of Experiments

Experiments are created and added to the ExperimentList by calling the addExperiment method, which returns a new Experiment object.

The calling subprogram then populates the Experiment properties as appropriate.

Once all Experiments are defined the analysis directory can be constructed by calling the buildAnalysisDirs method, which creates directories and symbolic links to primary data according to the definition of each experiment.

addDuplicateExperiment(*expt*)
Duplicate an existing Experiment and add to the list

Parameters **expt** – an existing Experiment object

Returns New Experiment object with the same data as the input

addExperiment(*name*)
Create a new Experiment and add to the list

Parameters **name** – the name of the new experiment

Returns New Experiment object with name already set

buildAnalysisDirs(*top_dir=None*, *dry_run=False*, *link_type='relative'*, *naming_scheme='partial'*)
Construct and populate analysis directories for the experiments

For each defined experiment, create the required analysis directories and populate with links to the primary data files.

Parameters

- **top_dir** – if set then create the analysis directories as subdirs of the specified directory; otherwise operate in cwd
- **dry_run** – if True then only report the mkdir, ln etc operations that would be performed. Default is False (do perform the operations).
- **link_type** – type of link to use when linking to primary data, one of 'relative' or 'absolute'.
- **naming_scheme** – naming scheme to use for links to primary data, one of 'full' (same names as primary data files), 'partial' (cut-down version of the full name which excludes sample names - the default), or 'minimal' (just the library name).

getLastExperiment()
Return the last Experiment added to the list

class bcftbx.Experiment.**LinkNames**(*scheme*)

Class to construct names for links to primary data files

The LinkNames class encodes a set of naming schemes that are used to construct names for the links in the analysis directories that point to the primary CFASTA and QUAL data files.

The schemes are:

full: link name is the same as the source file, e.g. solid0123_20111014_FRAG_BC_AB_CD_EF_pool_F3_CD_PQ5.csfa

partial: link name consists of the instrument name, datestamp and library name, e.g. solid0123_20111014_CD_PQ5.csfasta

minimal: link name consists of just the library name, e.g. CD_PQ5.csfasta

For paired-end data, the 'partial' and 'minimal' names have '_F3' and '_F5' appended as appropriate (full names already have this distinction).

Example usage:

To get the link names using the minimal scheme for the F3 reads ('library' is a SolidLibrary object):

```
>>> csfasta_lnk,qual_lnk = LinkNames('minimal').names(library)
```

To get names for the F5 reads using the partial scheme:

```
>>> csfasta_lnk,qual_lnk = LinkNames('partial').names(library,F5=True)
```

names(*library*, *F5=False*)

Get names for links to the primary data in a library

Returns a tuple of link names:

(csfasta_link_name,qual_link_name)

derived from the data in the library plus the naming scheme specified when the LinkNames object was created.

Parameters

- **library** – SolidLibrary object
- **F5** – if True then indicates that names should be returned for linking to the F5 reads (default is F3 reads)

13.4 bcftbx.FASTQFile

A set of classes for reading through FASTQ files and manipulating the data within them:

- FastqIterator: enables looping through all read records in FASTQ file
- FastqRead: provides access to a single FASTQ read record
- SequenceIdentifier: provides access to sequence identifier info in a read
- FastqAttributes: provides access to gross attributes of FASTQ file

Additionally there are a few utility functions:

- get_fastq_file_handle: return a file handled opened for reading a FASTQ file
- nreads: return the number of reads in a FASTQ file
- fastqs_are_pair: check whether two FASTQs form an R1/R2 pair

Information on the FASTQ file format: http://en.wikipedia.org/wiki/FASTQ_format

class bcftbx.FASTQFile.**FastqAttributes**(*fastq_file=None*, *fp=None*)
Class to provide access to gross attributes of a FASTQ file

Given a FASTQ file (can be uncompressed or gzipped), enables various attributes to be queried via the following properties:

nreads: number of reads in the FASTQ file fsize: size of the file (in bytes)

fsize
Return size of the FASTQ file (bytes)

nreads
Return number of reads in the FASTQ file

class bcftbx.FASTQFile.**FastqIterator**(*fastq_file=None*, *fp=None*, *bufsize=102400*)
Class to loop over all records in a FASTQ file, returning a FastqRead object for each record.

Example looping over all reads:

```
>>> for read in FastqIterator(fastq_file):
>>>     print(read)
```

Input FASTQ can be in gzipped format; FASTQ data can also be supplied as a file-like object opened for reading, for example:

```
>>> fp = io.open(fastq_file,'rt')
>>> for read in FastqIterator(fp=fp):
>>>     print(read)
>>> fp.close()
```

class bcftbx.FASTQFile.**FastqRead**(*seqid_line=None*, *seq_line=None*, *optid_line=None*, *quality_line=None*)
Class to store a FASTQ record with information about a read

Provides the following properties for accessing the read data:

- seqid: the "sequence identifier" information (first line of the read record) as a SequenceIdentifier object
- sequence: the raw sequence (second line of the record)
- optid: the optional sequence identifier line (third line of the record)
- quality: the quality values (fourth line of the record)

Additional properties:

- raw_seqid: the original sequence identifier string supplied when the object was created
- seqlen: length of the sequence
- maxquality: maximum quality value (in character representation)
- minquality: minimum quality value (in character representation)
- is_colorspace: returns True if the read looks like a colorspace read, False otherwise

---

Note: Quality scores can only be obtained from character representations once the encoding scheme is known.

---

class bcftbx.FASTQFile.**SequenceIdentifier**(*seqid*)
Class to store/manipulate sequence identifier information from a FASTQ record

Provides access to the data items in the sequence identifier line of a FASTQ record.

format
Identify the format of the sequence identifier

Returns 'illumina18', 'illumina' or None

Return type String

is_pair_of(*seqid*)
Check if this forms a pair with another SequenceIdentifier

bcftbx.FASTQFile.**fastqs_are_pair**(*fastq1=None*, *fastq2=None*, *verbose=True*, *fp1=None*, *fp2=None*)
Check that two FASTQs form an R1/R2 pair

Parameters

- **fastq1** – first FASTQ
- **fastq2** – second FASTQ

Returns True if each read in fastq1 forms an R1/R2 pair with the equivalent read (i.e. in the same position) in fastq2, otherwise False if any do not form an R1/R2 (or if there are more reads in one than than the other).

bcftbx.FASTQFile.**get_fastq_file_handle**(*fastq*, *mode='rt'*)
Return a file handle opened for reading for a FASTQ file

Deals with both compressed (gzipped) and uncompressed FASTQ files.

Parameters

- **fastq** – name (including path, if required) of FASTQ file. The file can be gzipped (must have '.gz' extension)
- **mode** – optional mode for file opening (defaults to 'rt')

Returns File handle that can be used for read operations.

bcftbx.FASTQFile.**nreads**(*fastq=None*, *fp=None*)
Return number of reads in a FASTQ file

Performs a simple-minded read count, by counting the number of lines in the file and dividing by 4.

The FASTQ file can be specified either as a file name (using the 'fastq' argument) or as a file-like object opened for line reading (using the 'fp' argument).

This function can handle gzipped FASTQ files supplied via the 'fastq' argument.

Line counting uses a variant of the "buf count" method outlined here: http://stackoverflow.com/a/850962/579925

Parameters

- **fastq** – fastq(.gz) file
- **fp** – open file descriptor for fastq file

Returns Number of reads

13.5 bcftbx.JobRunner

Classes for starting, stopping and managing jobs.

Class BaseJobRunner is a template with methods that need to be implemented by subclasses. The subclasses implemented here are:

- SimpleJobRunner: run jobs (e.g. scripts) on a local file system.
- GEJobRunner : run jobs using Grid Engine (GE) i.e. qsub, qdel etc

A single JobRunner instance can be used to start and manage multiple processes.

Each job is started by invoking the 'run' method of the runner. This returns an id string which is then used in calls to the 'isRunning', 'terminate' etc methods to check on and control the job.

The runner's 'list' method returns a list of running job ids.

Simple usage example:

```
>>> # Create a JobRunner instance
>>> runner = SimpleJobRunner()
>>> # Start a job using the runner and collect its id
>>> job_id = runner.run('Example',None,'myscript.sh')
>>> # Wait for job to complete
>>> import time
>>> while runner.isRunning(job_id):
>>>     time.sleep(10)
>>> # Get the names of the output files
>>> log,err = (runner.logFile(job_id),runner.errFile(job_id))
```

Processes run using a job runner inherit the environment where the runner is created and executed.

Additionally runners set an 'BCFTBX_RUNNER_NSLOTS' environment variable, which is set to the number of slots (aka CPUs/cores/threads) available to processes executed by the runner. For both 'SimpleJobRunner' and 'GEJobRunner', this defaults to one (i.e. serial jobs); the 'nslots' option can be used when instantiating 'SimpleJobRunner' objects to specify more cores, for example:

```
>>> multicore_runner = SimpleJobRunner(nslots=4)
```

For 'GEJobRunner' instances the number of cores is set by specifying '-pe smp.pe' as part of the 'ge_extra_args' option, for example:

```
>>> multicore_runner = GEJobRunner(extra_ge_args=('-pe','smp.pe','4'))
```

class bcftbx.JobRunner.**BaseJobRunner**

Base class for implementing job runners

This class can be used as a template for implementing custom job runners. The idea is that the runners wrap the specifics of interacting with an underlying job control system and thus provide a generic interface to be used by higher level classes.

A job runner needs to implement the following methods:

- run : starts a job running
- terminate : kills a running job
- list : lists the running job ids
- logFile : returns the name of the log file for a job
- errFile : returns the name of the error file for a job
- exit_status: returns the exit status for the command (or None if the job is still running)

Optionally it can also implement the methods:

- errorState: indicates if running job is in an "error state"
- isRunning : checks if a specific job is running

if the default implementations are not sufficient.

errFile(*job_id*)
Return name of error file relative to working directory

errorState(*job_id*)
Check if the job is in an error state

Return True if the job is deemed to be in an 'error state', False otherwise.

exit_status(*job_id*)
Return the exit status code for the command

Return the exit status code from the command that was run by the specified job, or None if the job hasn't exited yet.

isRunning(*job_id*)
Check if a job is running

Returns True if job is still running, False if not

list()
Return a list of running job_ids

logFile(*job_id*)
Return name of log file relative to working directory

log_dir
Return the current log directory setting

run(*name*, *working_dir*, *script*, *args*)
Start a job running

Parameters

- **name** – Name to give the job
- **working_dir** – Directory to run the job in
- **script** – Script file to run
- **args** – List of arguments to supply to the script

Returns Returns a job id, or None if the job failed to start

set_log_dir(*log_dir*)
(Re)set the directory to write log files to

terminate(*job_id*)
Terminate a job

Returns True if termination was successful, False otherwise

class bcftbx.JobRunner.**GEJobRunner**(*queue=None*, *log_dir=None*, *ge_extra_args=None*, *poll_interval=5.0*, *timeout=30.0*)
Class implementing job runner for Grid Engine

GEJobRunner submits jobs to a Grid Engine cluster using the 'qsub' command, determines the status of jobs using 'qstat' and terminates then using 'qdel'.

Additionally the runner can be configured for a specific GE queue on initialisation.

Each GEJobRunner instance creates a temporary directory which it uses for internal admin; this will be removed at program exit via 'atexit'.

errFile (*job_id*)
Return the error file name for a job

The name should be '<name>.e<job_id>'

errorState (*job_id*)
Check if the job is in an error state

Return True if the job is deemed to be in an 'error state' (i.e. qstat returns the state as 'E..'), False otherwise.

exit_status (*job_id*)
Return exit status from command run by a job

If the job is still running then returns 'None'.

ge_extra_args
Return the extra GE arguments

list ()
Get list of job ids which are queued or running

logFile (*job_id*)
Return the log file name for a job

The name should be '<name>.o<job_id>'

name (*job_id*)
Return the name for a job

nslots
Return the number of associated slots

This is extracted from the 'ge_extra_args' property, by looking for qsub options of the form '-pe smp.pe N' (in which case 'nslots' will be N).

queue (*job_id*)
Fetch the job queue name

Returns the queue as reported by qstat, or None if not found.

run (*name*, *working_dir*, *script*, *args*)
Submit a script or command to the cluster via 'qsub'

Parameters

- **name** – Name to give the job
- **working_dir** – Directory to run the job in
- **script** – Script file to run
- **args** – List of arguments to supply to the script

Returns Job id for submitted job, or 'None' if job failed to start.

terminate (*job_id*)
Remove a job from the GE queue using 'qdel'

class bcftbx.JobRunner.**ResourceLock**
Class for managing in-process locks on 'resources'

A 'resource' is identified by an arbitrary string.

Example usage: create a new ResourceLock instance and check if a resource is locked:

```
>>> r = ResourceLock()
>>> r.is_locked("resource1")
False
```

Try to acquire the lock on the resource:

```
>>> lock = r.acquire("resource1")
>>> r.is_locked("resource1")
True
```

Release the lock on the resource:

```
>>> r.release(lock)
>>> r.is_locked("resource1")
False
```

acquire(*resource_name*, *timeout=None*)
Attempt to acquire the lock on a resource

Parameters

- **resource_name** (*str*) – name of the resource to acquire the lock name for
- **timeout** (*float*) – optional, specifies a timeout period after which failure to acquire the lock raises an exception.

Returns lock name.

Return type String

is_locked(*resource_name*)
Check if a resource is locked

Parameters **resource_name** (*str*) – name of the resource to check the lock for

Returns

True if resource is locked, False if not.

Return type Boolean

release(*lock*)
Release a lock on a resource

Parameters **lock** (*str*) – lock to release.

class bcftbx.JobRunner.**SimpleJobRunner**(*log_dir=None*, *join_logs=False*, *nslots=1*)
Class implementing job runner for local system

SimpleJobRunner starts jobs as processes on a local system; the status of jobs is determined using the Linux 'ps eu' command, and jobs are terminated using 'kill -9'.

errFile(*job_id*)
Return the error file name for a job

exit_status(*job_id*)
Return exit status from command run by a job

list()
Return a list of running job_ids

logFile(*job_id*)
Return the log file name for a job

name (*job_id*)
Return the name for a job

nslots
Return the number of associated slots

run (*name*, *working_dir*, *script*, *args*)
Run a command and return the PID (=job id)

Parameters

- **name** – Name to give the job
- **working_dir** – Directory to run the job in
- **script** – Script file to run
- **args** – List of arguments to supply to the script

Returns Job id for submitted job, or 'None' if job failed to start.

terminate (*job_id*)
Kill a running job using 'kill -9'

bcftbx.JobRunner.**fetch_runner** (*definition*)
Return job runner instance based on a definition string

Given a definition string, returns an appropriate runner instance.

Definitions are of the form:

```
RunnerName[(args)]
```

RunnerName can be 'SimpleJobRunner' or 'GEJobRunner'. If '(args)' are also supplied then:

- for SimpleJobRunners, this can be a list of optional arguments separated by spaces:
  - 'nslots=N' (where N is an integer; sets a non-default number of slots
  - 'join_logs=BOOLEAN' (where BOOLEAN can be 'True', 'true','y','False','false','n'; sets whether stdout and stderr should be written to the same file)
- for GEJobRunners, this is a set of arbitrary 'qsub' options that will be used on job submission.

13.6 bcftbx.Pipeline

Classes for running scripts iteratively over a collection of data files.

The essential classes are:

- Job: wrapper for setting up, submitting and monitoring running scripts
- PipelineRunner: queue and run script multiple times on standard set of inputs
- SolidPipelineRunner: subclass of PipelineRunner specifically for running on SOLiD data (i.e. pairs of csfasta/qual files)

There are also some useful methods:

- GetSolidDataFiles: collect csfasta/qual file pairs from a specific directory
- GetSolidPairedEndFiles: collect csfasta/qual file pairs for paired end data
- GetFastqFiles: collect fastq files from a specific directory

- GetFastqGzFiles: collect gzipped fastq files

The PipelineRunners depend on the JobRunner instances (created from classes in the JobRunner module) to interface with the job management system. So typical usage might look like:

```
>>> import JobRunner
>>> import Pipeline
>>> runner = JobRunner.GEJobRunner() # to use Grid Engine
>>> pipeline = Pipeline.PipelineRunner(runner)
>>> pipeline.queueJob(...)
>>> pipeline.run()
```

13.6.1 Classes

class bcftbx.Pipeline.**Job**(*runner*, *name*, *dirn*, *script*, *args*, *label=None*, *group=None*)

Wrapper class for setting up, submitting and monitoring running scripts

Set up a job by creating a Job instance specifying the name, working directory, script file to execute, and arguments to be supplied to the script.

The job is started by invoking the 'start' method; its status can be checked with the 'isRunning' method, and terminated and restarted using the 'terminate' and 'restart' methods respectively.

Information about the job can also be accessed via its properties. The following properties record the original parameters supplied on instantiation:

name working_dir script args label group_label

Additional information is set once the job has started or stopped running:

job_id The id number for the running job returned by the JobRunner log The log file for the job (relative to working_dir) err The error log file for the job start_time The start time (seconds since the epoch) end_time The end time (seconds since the epoch) exit_status The exit code from the command that was run (integer, or None)

The Job class uses a JobRunner instance (which supplies the necessary methods for starting, stopping and monitoring) for low-level job interactions.

class bcftbx.Pipeline.**PipelineRunner**(*runner*, *max_concurrent_jobs=4*, *poll_interval=30*, *jobCompletionHandler=None*, *groupCompletionHandler=None*)

Class to run and manage multiple concurrent jobs.

PipelineRunner enables multiple jobs to be queued via the 'queueJob' method. The pipeline is then started using the 'run' method - this starts each job up to a a specified maximum of concurrent jobs, and then monitors their progress. As jobs finish, pending jobs are started until all jobs have completed.

Example usage:

```
>>> p = PipelineRunner()
>>> p.queueJob('/home/foo','foo.sh','bar.in')
... Queue more jobs ...
>>> p.run()
```

By default the pipeline runs in 'blocking' mode, i.e. 'run' doesn't return until all jobs have been submitted and have completed; see the 'run' method for details of how to operate the pipeline in non-blocking mode.

The invoking subprogram can also specify functions that will be called when a job completes ('jobCompletionHandler'), and when a group completes ('groupCompletionHandler'). These can perform any specific actions that are required such as sending notification email, setting file ownerships and permissions etc.

class bcftbx.Pipeline.**SolidPipelineRunner**(*runner*, *script*, *max_concurrent_jobs=4*, *poll_interval=30*)

Class to run and manage multiple jobs for Solid data pipelines

Subclass of PipelineRunner specifically for dealing with scripts that take Solid data (i.e. csfasta/qual file pairs).

Defines the addDir method in addition to all methods already defined in the base class; use this method one or more times to specify directories with data to run the script on. The SOLiD data file pairs in each specified directory will be located automatically.

For example:

solid_pipeline = SolidPipelineRunner('qc.sh') solid_pipeline.addDir('/path/to/datadir') solid_pipeline.run()

13.6.2 Functions

bcftbx.Pipeline.**GetSolidDataFiles**(*dirn*, *pattern=None*, *file_list=None*)

Return list of csfasta/qual file pairs in target directory

Note that files with names ending in '_T_F3' will be rejected as these are assumed to come from the preprocess filtering stage.

Optionally also specify a regular expression pattern that file names must also match in order to be included.

Parameters

- **dirn** – name/path of directory to look for files in
- **pattern** – optional, regular expression pattern to filter names with
- **file_list** – optional, a list of file names to use instead of fetching a list of files from the specified directory

Returns List of tuples consisting of two csfasta-qual file pairs (F3 and F5).

bcftbx.Pipeline.**GetSolidPairedEndFiles**(*dirn*, *pattern=None*, *file_list=None*)

Return list of csfasta/qual file pairs for paired end data

Optionally also specify a regular expression pattern that file names must also match in order to be included.

Parameters

- **dirn** – name/path of directory to look for files in
- **pattern** – optional, regular expression pattern to filter names with
- **file_list** – optional, a list of file names to use instead of fetching a list of files from the specified directory

Returns List of csfasta-qual pair tuples.

bcftbx.Pipeline.**GetFastqFiles**(*dirn*, *pattern=None*, *file_list=None*)

Return list of fastq files in target directory

Optionally also specify a regular expression pattern that file names must also match in order to be included.

Parameters

- **dirn** – name/path of directory to look for files in
- **pattern** – optional, regular expression pattern to filter names with
- **file_list** – optional, a list of file names to use instead of fetching a list of files from the specified directory

Returns List of file-pair tuples.

bcftbx.Pipeline.**GetFastqGzFiles**(*dirn*, *pattern=None*, *file_list=None*)
Return list of fastq.gz files in target directory

Optionally also specify a regular expression pattern that file names must also match in order to be included.

Parameters

- **dirn** – name/path of directory to look for files in
- **pattern** – optional, regular expression pattern to filter names with
- **file_list** – optional, a list of file names to use instead of fetching a list of files from the specified directory

Returns List of file-pair tuples.

13.7 bcftbx.Md5sum

Md5sum

Classes and functions for performing various MD5 checksum operations.

The code function is the 'md5sum' function, which computes the MD5 hash for a file and is based on examples at:

http://www.python.org/getit/releases/2.0.1/md5sum.py

and

http://stackoverflow.com/questions/1131220/get-md5-hash-of-a-files-without-open-it-in-python

Usage:

```
>>> import Md5sum
>>> Md5Sum.md5sum("myfile.txt")
... eacc9c036025f0e64fb724cacaadd8b4
```

This module implements two methods for generating the md5 digest of a file: the first uses a method based on the hashlib module, while the second (used as a fallback for pre-2.5 Python) uses the now deprecated md5 module. Note however that the md5sum function determines itself which method to use.

There is also a high-level class 'Md5Checker' which implements various class methods for running MD5 checks across all files in a directory, and a wrapper class 'Md5Reporter' which

class bcftbx.Md5sum.**Md5CheckReporter**(*results=None*, *verbose=False*, *fp=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)
Provides a generic reporting class for Md5Checker methods

Typical usage modes are either:

```
>>> r = Md5CheckReporter()
>>> for f,s in Md5Checker.md5cmp_dirs(d1,d2):
...     r.add_result(f,s)
```

or more concisely:

```
>>> r = Md5CheckReporter(Md5Checker.md5cmp_dirs(d1,d2))
```

Use the 'summary' method to generate a summary of all the checks.

Use the 'status' method to get a single indicator of success or failure which is consistent with UNIX-style return codes.

To find out how many results were processed in total, how many failed etc use the following properties:

- n_files : total number of results examined
- n_ok : number that passed MD5 checks (MD5_OK)
- n_failed : number that failed due to different MD5 sums (MD5_FAILED)
- n_missing: number that failed due to a missing target file (MISSING_TARGET)
- n_errors : number that had errors calculating their MD5 sums (MD5 ERROR)

add_result (*f*, *status*)
Add a result to the reporter

Takes a file and an Md5Checker status code and adds it to the results.

If the status code indicates a failed check then the file name is added to a list corresponding to the nature of the failure (e.g. MD5 sums didn't match, target was missing etc).

n_errors
Number of files with errors checking MD5 sums

n_failed
Number of failed MD5 sum checks

n_files
Total number of files checked

n_missing
Number of missing files

n_ok
Number of passed MD5 sum checks

status
Return status code

Returns 0 if all files that were checked passed the MD5 check, or 1 if at least one file failed the check for whatever reason.

summary ()
Write a summary of the results

Writes a summary of the number of files checked, how many passed or failed MD5 checks and so on, to the specified output stream.

class bcftbx.Md5sum.**Md5Checker**
Provides static methods for performing checks using MD5 sums

The Md5Checker class is a collection of static methods that can be used for performing checks using MD5 sums.

It also provides a set of constants to

classmethod compute_md5sums (*d*, *links=0*)
Calculate MD5 sums for all files in directory

Given a directory, traverses the structure underneath (including subdirectories) and yields the path and MD5 sum for each file that is found.

The 'links' option determines how symbolic links are handled, see the 'walk' function for details.

Parameters

- **dirn** – name of the top-level directory
- **links** – (optional) specify how symbolic links are handled

Returns Yields a tuple (f,md5) where f is the path of a file relative to the top-level directory, and md5 is the calculated MD5 sum.

classmethod md5_walk(*dirn*, *links=0*)
Calculate MD5 sums for all files in directory

Given a directory, traverses the structure underneath (including subdirectories) and yields the path and MD5 sum for each file that is found.

The 'links' option determines how symbolic links are handled, see the 'walk' function for details.

Parameters

- **dirn** – name of the top-level directory
- **links** – (optional) specify how symbolic links are handled

Returns Yields a tuple (f,md5) where f is the path of a file relative to the top-level directory, and md5 is the calculated MD5 sum.

classmethod md5cmp_dirs(*d1*, *d2*, *links=0*)
Compares the contents of one directory with another using MD5 sums

Given two directory names 'd1' and 'd2', compares the MD5 sum of each file found in 'd1' against that of the equivalent file in 'd2', and yields the result as an Md5checker constant for each file pair, i.e.:

MD5_OK: if MD5 sums match; MD5_FAILED: if MD5 sums differ.

If the equivalent file doesn't exist then yields MISSING_TARGET.

If one or both MD5 sums cannot be computed then yields MD5_ERROR.

How symbolic links are handled depends on the setting of the 'links' option:

FOLLOW_LINKS: (default) MD5 sums are computed and compared for the targets of symbolic links. Broken links are treated as if the file was missing.

IGNORE_LINKS: MD5 sums are not computed or compared if either file is a symbolic link, and links to directories are not followed.

Parameters

- **d1** – 'reference' directory
- **d2** – 'target' directory to be compared with the reference
- **links** – (optional) specify how symbolic links are handled.

Returns Yields a tuple (f,status) where f is the relative path of the file pair being compared, and status is the Md5Checker constant representing the outcome of the comparison.

classmethod md5cmp_files(*f1*, *f2*)
Compares the MD5 sums of two files

Given two file names, attempts to compute and compare their MD5 sums.

If the MD5s match then returns MD5_OK, if they don't match then returns MD5_FAILED.

If one or both MD5 sums cannot be computed then returns MD5_ERROR.

Note that if either file is a link then MD5 sums will be computed for the link target(s), if they exist and can be accessed.

Parameters

- **f1** – name and path for reference file

- **f2** – name and path for file to be checked

Returns Md5Checker constant representing the outcome of the comparison.

classmethod verify_md5sums(*filen=None*, *fp=None*)
Verify md5sums from a file

Given a file (or a file-like object opened for reading), reads each line and attemps to interpret as an md5sum line i.e. of the form

<md5 sum> <path/to/file>

e.g.

66b201ae074c36ae9bffec7fb74ff03a md5checker.py

It then attempts to verify the MD5 sum against the file located on the file system, and yields the result as an Md5checker constant for each file line i.e.:

MD5_OK: if MD5 sums match; MD5_FAILED: if MD5 sums differ.

If the file cannot be found then it yields MISSING_TARGET; if there is a problem computing the MD5 sum then it yields MD5_ERROR.

Parameters

- **filen** – name of the file containing md5sum output
- **fp** – file-like object opened for reading, with md5sum output

Returns Yields a tuple (f,status) where f is the path of the file being verified (as it appears in the file), and status is the Md5Checker constant representing the outcome.

classmethod walk(*dirn*, *links=0*)
Traverse all files found in a directory structure

Given a directory, traverses the structure underneath (including subdirectories) and yields the path for each file that is found.

How symbolic links are handled depends on the setting of the 'links' option:

FOLLOW_LINKS: symbolic links to files are treated as files; links to directories are followed.

IGNORE_LINKS: symbolic links to files are ignored; links to directories are not followed.

Parameters

- **dirn** – name of the top-level directory
- **links** – (optional) specify how symbolic links are handled

Returns Yields the name and full path for each file under 'dirn'.

bcftbx.Md5sum.**md5sum**(*f*)
Return md5sum digest for a file or stream

This implements the md5sum checksum generation using both the hashlib module.

Parameters **f** – name of the file to generate the checksum from, or a file-like object opened for reading in binary mode.

Returns Md5sum digest for the named file.

13.8 bcftbx.platforms

platforms.py

Utilities and data to identify NGS sequencer platforms

bcftbx.platforms.**get_sequencer_platform**(*sequencer_name*)
Attempt to determine platform from sequencer name

Checks the supplied sequencer name against the patterns in PLATFORMS and returns the first match (or None if no match is found).

Parameters **sequencer_name** – sequencer name (can include a leading directory path)

Returns Matching sequencer platform, or None.

bcftbx.platforms.**list_platforms**()
Return list of known platform names

13.9 bcftbx.TabFile

Classes for working with generic tab-delimited data.

The TabFile module provides a TabFile class, which represents a tab-delimited data file, and a TabDataLine class, which represents a line of data.

13.9.1 Creating a TabFile

TabFile objects can be initialised from existing files:

```
>>> data = TabFile('data.txt')
```

or an 'empty' TabFile can be created if no file name is specified.

Lines starting with '#' are ignored.

13.9.2 Accessing Data within a TabFile

Within a TabFile object each line of data is represented by a TabDataLine object. Lines of data are referenced using index notation, with the first line of data being index zero:

```
>>> line = data[0]
>>> line = data[i]
```

Note that the index is not the same as the line number from the source file, (if one was specified) - this can be obtained from the 'lineno' method of each line:

```
>>> line_number = line.lineno()
```

len() gives the total number of lines of data in the TabFile object:

```
>>> len(data)
```

It is possible to iterate over the data lines in the object:

```
>>> for line in data:
>>>     ... do something with line ...
```

By default columns of data in the file are referenced by index notation, with the first column being index zero:

```
>>> line = data[0]
>>> value = line[0]
```

If column headers are specified then these can also be used to reference columns of data:

```
>>> data = TabFile('data.txt',column_names=['ex','why','zed'])
>>> line = data[0]
>>> ex = line['ex']
>>> line['why'] = 3.454
```

Headers can also be read from the first line of an input file:

```
>>> data = TabFile('data.txt',first_line_is_header=True)
```

A list of the column names can be fetched using the 'header' method:

```
>>> print(data.header())
```

Use the 'str' built-in to get the line as a tab-delimited string:

```
>>> str(line)
```

13.9.3 Adding and Removing Data

New lines can be added to the TabFile object via the 'append' and 'insert' methods:

```
>>> data.append()  # No data i.e. empty line
>>> data.append(data=[1,2,3]) # Provide data values as a list
>>> data.append(tabdata='1	2	3') # Provide values as tab-delimited string
>>> data.insert(1,data=[5,6,7]) # Inserts line of data at index 1
```

Type conversion is automatically performed when data values are assigned:

```
>>> line = data.append(data=['1',2,'3.4','pjb'])
>>> line[0]
1
>>> line[2]
3.4
>>> line[3]
'pjb'
```

Lines can also be removed using the 'del' built-in:

```
>>> del(data[0]) # Deletes first data line
```

New columns can be added using the 'appendColumn' method e.g.:

```
>>> data.appendColumn('new_col') # Creates a new empty column
```

13.9.4 Filtering Data

The 'lookup' method returns a set of data lines where a key matches a specific value:

```
>>> data = TabFile('data.txt',column_names=['chr','start','end'])
>>> chrom = data.lookup('chr','chrX')
```

Within a single data line the 'subset' method returns a list of values for a set of column indices or column names:

```
>>> data = TabFile(column_names=['chr','start','end','strand'])
>>> data.append(data=['chr1',123456,234567,'+'])
>>> data[0].subset('chr1','start')
['chr1',123456]
```

13.9.5 Sorting Data

The 'sort' method offers a simple way of sorting the data lines within a TabFile. The simplest example is sorting on a specific column:

```
>>> data.sort(lambda line: line['start'])
```

See the method documentation for more detail on using the 'sort' method.

13.9.6 Manipulating Data: whole column operations

The 'transformColumn' and 'computeColumn' methods provide a way to update all the values in a column with a single method call. In each case the calling subprogram must supply a function object which is used to update the values in a specific column.

The function supplied to 'transformColumn' must take a single argument which is the current value of the column in that line. For example: define a function to increment a supplied value by 1:

```
>>> def addOne(x):
>>> ...     return x+1
```

Then use this to add one to all values in the column 'start':

```
>>> data.transformColumn('start',addOne)
```

Alternatively a lambda can be used to avoid defining a new function:

```
>>> data.transformColumn('start',lambda x: x+1)
```

The function supplied to 'computeColumn' must take a single argument which is the current line (i.e. a TabDataLine object) and return a new value for the specified column. For example:

```
>>> def calculateMidpoint(line):
>>> ...   return (line['start'] + line['stop'])/2.0
>>> data.computeColumn('midpoint',calculateMidpoint)
```

Again a lambda expression can be used instead:

```
>>> data.computeColumn('midpoint',lambda line: line['stop'] - line['start'])
```

13.9.7 Writing to File

Use the TabFile's 'write' method to output the content to a file:

```
>>> data.write('newfile.txt') # Writes all the data to newfile.txt
```

It's also possible to reorder the columns before writing out using the 'reorderColumns' method.

13.9.8 Specifying Delimiters

It's possible to use a different field delimiter than tabs, by explicitly specifying the value of the 'delimiter' argument when creating a new TabFile object, for example for a comma-delimited file:

```
>>> data = TabFile('data.txt',delimiter=',')
```

13.9.9 TabFileIterator: iterating through a tab-delimited file

The `TabFileIterator` provides a light-weight alternative to `TabFile` in situations where it is only necessary to iterate through each line in a tab-delimited file:

```
>>> for line in TabFileIterator(filen='data.tsv'):
...   print(line)
```

Each line is returned as a `TabDataLine` instance, so the methods available that class can be used on the data.

class bcftbx.TabFile.**TabDataLine**(*line=None*, *column_names=None*, *delimiter='t'*, *lineno=None*, *convert=True*, *allow_underscores_in_numeric_literals=False*)

Class to store a line of data from a tab-delimited file

Values can be accessed by integer index or by column names (if set), e.g.

line = TabDataLine("1 2 3",('first','second','third'))

allows the 2nd column of data to accessed either via line[1] or line['second'].

Values can also be changed, e.g.

line['second'] = new_value

Values are automatically converted to integer or float types as appropriate.

Subsets of data can be created using the 'subset' method.

Line numbers can also be set by the creating subprogram, and queried via the 'lineno' method.

It is possible to use a different field delimiter than tabs, by explicitly specifying the value of the 'delimiter' argument, e.g. for a comma-delimited line:

line = TabDataLine("1,2,3",delimiter=',')

Check if a line is empty:

if not line: print("Blank line")

append(**values*)

Append values to the data line

Should only be used when creating new data lines.

appendColumn(*key*, *value*)
Append keyed values to the data line

This adds a new value along with a header name (i.e. key)

convert_to_str(*value*)
Convert value to string

convert_to_type(*value*)
Internal: convert a value to the correct type

Used to coerce input values into integers or floats if appropriate before storage in the TabDataLine object.

convert_to_type_pep515(*value*)
Internal: convert a value to the correct type

Used to coerce input values into integers or floats if appropriate before storage in the TabDataLine object.

The conversion honors PEP 515 so numerical values can also contain underscore characters.

delimiter(*new_delimiter=None*)
Set and get the delimiter for the line

If 'new_delimiter' is not None then the field delimiter for the line will be updated to the supplied value. This affects how lines are represented via the __repr__ built-in.

Returns the current value of the delimiter.

lineno()
Return the line number associated with the line

NB The line number is set by the class or function which created the TabDataLine object, it is not guaranteed by the TabDataLine class itself.

subset(**keys*)
Return a subset of data items

This method creates a new TabDataLine instance with a subset of data specified by the 'keys' argument, e.g.

new_line = line.subset(2,1)

returns an instance with only the 2nd and 3rd data values in reverse order.

To access the items in a subset using index notation, use the same keys as those specified when the subset was created. For example, for

s = line.subset("two","nine")

use s["two"] and s["nine"] to access the data; while for

s = line.subset(2,9)

use s[2] and s[9].

Parameters **keys** – one or more keys specifying columns to include in the subset. Keys can be column indices, column names, or a mixture, and the same column can be referenced multiple times.

class bcftbx.TabFile.**TabFile**(*filen=None*, *fp=None*, *column_names=None*, *skip_first_line=False*, *first_line_is_header=False*, *tab_data_line=<class 'bcftbx.TabFile.TabDataLine'>*, *delimiter='t'*, *convert=True*, *allow_underscores_in_numeric_literals=False*, *keep_commented_lines=False*)
Class to get data from a tab-delimited file

Loads data from the specified file into a data structure than can then be queried on a per line and per item basis.

Data lines are represented by data line objects which must be TabDataLine-like.

Example usage:

data = TabFile(myfile) # load initial data

print('%s' % len(data)) # report number of lines of data

print('%s' % data.header()) # report header (i.e. column names)

for line in data: ... # loop over lines of data

myline = data[0] # fetch first line of data

append(*data=None*, *tabdata=None*, *tabdataline=None*)
Create and append a new data line

Creates a new data line object and appends it to the end of the list of lines.

Optionally the 'data' or 'tabdata' arguments can specify data items which will be used to populate the new line; alternatively 'tabdataline' can provide a TabDataLine-based object to be appended.

If none of these are specified then a default blank TabDataLine-based object is created, appended and returned.

Parameters

- **data** – (optional) a list of data items
- **tabdata** – (optional) a string of tab-delimited data items
- **tabdataline** – (optional) a TabDataLine-based object

Returns Appended data line object.

appendColumn(*name*, *fill_value=''*)
Append a new (empty) column

Parameters

- **name** – name for the new column
- **fill_value** – optional, value to insert into all rows in the new column

computeColumn(*column_name*, *compute_func*)
Compute and store values in a new column

For each line of data the computation function will be invoked with the line as the sole argument, and the result will be stored in a new column with the specified name.

Parameters

- **column_name** – name or index of column to write transformation result to
- **compute_func** – callable object that will be invoked to perform the computation

filename()
Return the file name associated with the TabFile

header()
Return list of column names

If no column names were set then this will be an empty list.

indexByLineNumber(*n*)
Return index of a data line given the file line number

Given the line number n for a line in the original file, returns the index required to access the data for that line in the TabFile object.

If no matching line is found then raises an IndexError.

insert(*i*, *data=None*, *tabdata=None*, *tabdataline=None*)
Create and insert a new data line at a specified index

Creates a new data line object and inserts it into the list of lines at the specified index position 'i' (nb NOT a line number).

Optionally the 'data' or 'tabdata' arguments can specify data items which will be used to populate the new line; alternatively 'tabdataline' can provide a TabDataLine-based object to be inserted.

Parameters

- **i** – index position to insert the line at
- **data** – (optional) a list of data items
- **tabdata** – (optional) a string of tab-delimited data items
- **tabdataline** – (optional) a TabDataLine-based object

Returns New inserted data line object.

lookup(*key*, *value*)
Return lines where the key matches the specified value

nColumns()
Return the number of columns in the file

If the file had a header then this will be the number of header columns; otherwise it will be the number of columns found in the first line of data

reorderColumns(*new_columns*)
Rearrange the columns in the file

Parameters **new_columns** – list of column names or indices in the new order

Returns New TabFile object

sort(*sort_func*, *reverse=False*)
Sort data using arbitrary function

Performs an in-place sort based on the suppled sort_func.

sort_func should be a function object which takes a data line object as input and returns a single numerical value; the data lines will be sorted in ascending order of these values (or descending order if reverse is set to True).

To sort on the value of a specific column use e.g.

```
>>> tabfile.sort(lambda line: line['col'])
```

Parameters

- **sort_func** – function object taking a data line object as input and returning a single numerical value
- **reverse** – (optional) Boolean, either False (default) to sort in ascending order, or True to sort in descending order

transformColumn (*column_name*, *transform_func*)
Apply arbitrary function to a column

For each line of data the transformation function will be invoked with the value of the named column, with the result being written back to that column (overwriting the existing value).

Parameters

- **column_name** – name of column to write transformation result to
- **transform_func** – callable object that will be invoked to perform the transformation

transpose ()
Transpose the contents of the file

Returns New TabFile object

write (*filen=None*, *fp=None*, *include_header=False*, *no_hash=False*, *delimiter=None*)
Write the TabFile data to an output file

One of either the 'filen' or 'fp' arguments must be given, specifying the file name or stream to write the TabFile data to.

Parameters

- **filen** – (optional) name of file to write to; ignored if fp is also specified
- **fp** – (optional) a file-like object opened for writing; used in preference to filen if set to a non-null value Note that the calling program must close the stream in these cases.
- **include_header** – (optional) if set to True, the first line will be a 'header' line
- **no_hash** – (optional) if set to True and include_header is also True then don't put a hash character '#' at the start of the header line in the output file.
- **delimiter** – (optional) delimiter to use when writing data values to file (defaults to the delimiter specified on input)

class bcftbx.TabFile.**TabFileIterator** (*filen=None*, *fp=None*, *column_names=None*)
Iterate through lines in a tab-delimited file

Class to loop over all lines in a TSV file, returning a TabDataLine object for each record.

13.10 bcftbx.simple_xls and bcftbx.Spreadsheet

13.10.1 simple_xls

Simple spreadsheet module intended to provide a nicer programmatic interface to Excel spreadsheet generation.

It is currently built on top of SpreadSheet.py, which itself uses the xlwt, xlrd and xlutils modules. In future the relevant parts may be rewritten to remove the dependence on Spreadsheet.py and call the appropriate xl* classes and functions directly.

Example usage

Start by making a workbook, represented by an XLSWorkBook object:

```
>>> wb = XLSWorkBook("Test")
```

Then add worksheets to this:

```
>>> wb.add_work_sheet('test')
>>> wb.add_work_sheet('data',"My Data")
```

Worksheets have an id and an optional title. Ids must be unique and can be used to fetch the XLSWorkSheet object that represent the worksheet:

```
>>> data = wb.worksheet['data']
```

Cells can be addressed directly using various notations:

```
>>> data['A1'] = "Column 1"
>>> data['A']['1'] = "Updated value"
>>> data['AZ']['3'] = "Another value"
```

The extent of the sheet is defined by the outermost populated rows and columns

```
>>> data.last_column # outermost populated column
>>> data.last_row    # outermost populated row
```

There are various other methods for returning the next row or column; see the documentation for the XLSWorkSheet class.

Data can be added cell-wise (i.e. referencing individual cells as above), row-wise, column-wise and block-wise.

Column-wise operations include inserting a column (shifting columns above it along one to make space):

```
>>> data.insert_column('B',data=['hello','goodbye','whatev'])
```

Append a column (writing data to the first empty column at the end of the sheet):

```
>>> data.append_column(data=['hello','goodbye','whatev'])
```

Write data to a column, overwriting any existing values:

```
>>> data.write_column(data=['hello','goodbye','whatev'])
```

Data can be specified as a list, text or as a single value which is repeated for each cell (i.e. a "fill" value).

Similar row-wise operations also exist:

```
>>> data.insert_row(4,data=['Dozy','Beaky','Mick','Titch'])
>>> data.append_row(data=['Dozy','Beaky','Mick','Titch'])
>>> data.write_row(4,data=['Dozy','Beaky','Mick','Titch'])
```

Block-wise data can be added via a tab and newline-delimited string:

```
>>> data.insert_block_data("This        is              some
        random
        data")
>>> data.insert_block_data("This        is              some
        MORE    random
        data",
...                         col='M',row=7)
```

Formulae can be specified by prefixing a '=' symbol to the start of the cell contents, e.g.:

```
>>> data['A3'] = '=A1+A2'
```

'?' and '#' are special characters that can be used to indicate 'current row' and 'current column' respectively, e.g.:

```
>>> data.fill_column('A','=B?+C?') # evaluates to 'B1+C1' (A1), 'B2+C2' (A2) etc
```

Styling and formatting information can be associated with a cell, either when adding column, row or block data or by using the 'set_style' method. In each case the styling information is passed via an XLSStyle object, e.g.

```
>>> data.set_style(XLSStyle(number_format=NumberFormats.PERCENTAGE),'A3')
```

The workbook can be saved to file:

```
>>> wb.save_as_xls('test.xls')
```

Alternatively the contents of a sheet (or a subset) can be rendered as text:

```
>>> data.render_as_text(include_columns_and_rows=True,
...                     eval_formulae=True,
...                     include_styles=True)
>>> data.render_as_text(start='B1',end='C6',include_columns_and_rows=True)
```

class bcftbx.simple_xls.**CellIndex**(*idx*)
Convenience class for handling XLS-style cell indices

The CellIndex class provides a way of handling XLS-style cell indices i.e. 'A1', 'BZ112' etc.

Given a putative cell index it extracts the column and row which can then be accessed via the 'column' and 'row' attributes respectively.

The 'is_full' property reports whether the supplied index is actually a 'full' index with both column and row specifiers. If it is just a column or just a row then only the appropriate 'column' or 'row' attributes will be set.

is_full
Return True if index has both column and row information

class bcftbx.simple_xls.**ColumnRange**(*i*, *j=None*, *include_end=True*, *reverse=False*)
Iterator for a range of column indices

Range-style iterator for iterating over alphabetical column indices, e.g.

```
>>> for c in ColumnRange('A','Z'):
...   print(c)
```

next()
Implements Iterator subclass 'next' method (Python 2 only)

class bcftbx.simple_xls.**Limits**
Limits for XLS files (kept for backwards compatibility)

class bcftbx.simple_xls.**XLSColumn**(*column_index*, *parent=None*)
Class representing a column in a XLSWorkSheet

An XLSColumn object provides access to data in a column from a XLSWorkSheet object. Typically one can be returned by doing something like:

```
>>> colA = ws['A']
```

and individual cell values then accessed by row number alone, e.g.:

```
>>> value = colA['1']
>>> colA['2'] = "New value"
```

full_index(*row*)
Return the full index for a cell in the column

Given a row index, returns the index of the cell that this addresses within the column (e.g. if the column is 'A' then row 2 addresses cell 'A2').

class bcftbx.simple_xls.**XLSLimits**
Limits for XLS files

class bcftbx.simple_xls.**XLSStyle**(*bold=False*, *color=None*, *bgcolor=None*, *wrap=False*, *border=None*, *number_format=None*, *font_size=None*, *centre=False*, *shrink_to_fit=False*)
Class representing a set of styling and formatting data

An XLSStyle object represents a collection of data used for styling and formatting cell values on output to an Excel file.

The style attributes can be set on instantiation, or queried and modified afterwards.

The attributes are:

bold: whether text is bold or not (boolean) color: text color (name) bgcolor: background color (name) wrap: whether text in a cell should wrap (boolean) border: style of cell border (thick, medium, thin etc) number_format: a format code from the NumbersFormat class font_size: font size in points (integer) centre: whether text is centred in the cell (boolean) shrink_to_fit: whether to shrink cell to fit the contents.

The 'name' property can be used to generate a name for the style based on the attributes that have been set, for example:

```
>>> XLSStyle(bold=true).name
... '__bold__'
```

excel_number_format
Return an Excel-style equivalent of the stored number format

Returns an Excel-style number format, or None if the format isn't set or is unrecognised.

name
Return a name based on the attributes

style(*item*)
Wrap 'item' with <style...>...</style> tags

Given a string (or object that can be rendered as a string) return the string representation surrounded by <style...> </style> tags, where the tag attributes describe the style information stored in the XLSStyle object:

font=bold color=(color) bgcolor=(color) wrap border=(border) number_format=(format) font_size=(size) centre shrink_to_fit

class bcftbx.simple_xls.**XLSWorkBook**(*title=None*)
Class for creating an Excel (xls) spreadsheet

An XLSWorkBook instance provides an interface to creating an Excel spreadsheet.

It consists of a collection of XLSWorkSheet objects, each of which represents a sheet in the workbook.

Sheets are created and appended using the add_work_sheet method:

```
>>> xls = XLSWorkBook()
>>> sheet = xls('example')
```

Sheets are kept in the 'worksheet' property and can be acquired by name:

```
>>> sheet = xls.worksheet['example']
```

Once the worksheet(s) have been populated an XLS file can be created using the 'save_as_xls' method:

```
>>> xls.save_as_xls('example.xls')
```

add_work_sheet (*name*, *title=None*)
Create and append a new worksheet

Creates a new XLSWorkSheet object and appends it to the workbook.

Parameters

- **name** – unique name for the worksheet
- **title** – optional, title for the worksheet - defaults to the name.

Returns New XLSWorkSheet object.

save_as_xls (*filen*)
Output the workbook contents to an Excel-format file

Parameters **filen** – name of the file to write the workbook to.

save_as_xlsx (*filen*)
Output the workbook contents to an XLSX-format file

Parameters **filen** – name of the file to write the workbook to.

class bcftbx.simple_xls.**XLSWorkSheet** (*title*)
Class for creating sheets within an XLS workbook.

XLSWorkSheet objects represent a sheet within an Excel workbook.

Cells are addressed within the sheet using Excel notation i.e. <column><row> (columns start at index 'A' and rows at '1', examples are 'A1' or 'D19'):

```
>>> ws = XLSWorkSheet('example')
>>> ws['A1'] = 'some data'
>>> value = ws['A1']
```

If there is no data stored for the cell then 'None' is returned. Any cell can addressed without errors.

Data can also be added column-wise, row-wise or as a "block" of tab- and new-line delimited data:

```
>>> ws.insert_column_data('B',[1,2,3])
>>> ws.insert_row_data(4,['x','y','z'])
>>> ws.insert_block_data("This\tis\nthe\tdata")
```

A column can be "filled" with a single repeating value:

```
>>> ws.fill_column('D','single value')
```

The extent of the sheet can be determined from the 'last_column' and last_row' properties; the 'next_column' and 'next_row' properties report the next empty column and row respectively.

Cells can contain Excel-style formula by adding an equals sign to the start of the value. Typically formulae reference other cells and perform mathematical operations on them, e.g.:

```
>>> ws['E11'] = "=A1+A2"
```

Wildcard characters can be used which will be automatically translated into the cell column ('#') or row ('?'), for example:

```
>>> ws['F46'] = "=#47+#48"
```

will be transformed to "=F47+F48".

Styles can be applied to cells, using either the 'set_style' method or via the 'style' argument of some methods, to associate an XLSStyle object. Associated XLSStyle objects can be retrieved using the 'get_style' method.

The value of an individual cell can be 'rendered' for output using the 'render_cell' method:

```
>>> print(ws.render_cell('F46'))
```

All or part of the sheet can be rendered as a tab- and newline-delimited string by using the 'render_as_text' method:

```
>>> print(ws.render_as_text())
```

append_column(*data=None*, *text=None*, *fill=None*, *from_row=None*, *style=None*)
Create a new column at the end of the sheet

Appends a new column at the end of the worksheet i.e. in the first available empty column.

By default the appended column is empty, however data can be specified to populate the column.

Parameters

- **data** – optional, list of data items to populate the inserted column
- **text** – optional, tab-delimited string of text to be used to populate the inserted column
- **fill** – optional, single data item to be repeated to fill the inserted column
- **from_row** – optional, if specified then inserted column is populated from that row onwards
- **style** – optional, an XLSStyle object to associate with the data being inserted

Returns The index of the appended column.

append_row(*data=None*, *text=None*, *fill=None*, *from_column=None*, *style=None*)
Create a new row at the end of the sheet

Appends a new row at the end of the worksheet i.e. in the first available empty row.

By default the appended row is empty, however data can be specified to populate the row.

Parameters

- **data** – optional, list of data items to populate the inserted row
- **text** – optional, newline-delimited string of text to be used to populate the inserted row
- **fill** – optional, single data item to be repeated to fill the inserted row
- **from_row** – optional, if specified then inserted row is populated from that column onwards
- **style** – optional, an XLSStyle object to associate with the data being inserted

Returns The index of the inserted row.

column_is_empty(*col*)
Determine whether a column is empty

Returns False if any cells in the column are populated, otherwise returns True.

columnof(*s*, *row=1*)
Return column index for cell which matches string

Return index of first column where the content matches the specified string 's'.

Parameters

- **s** – string to search for
- **row** – row to search in (defaults to 1)

Returns Column index of first matching cell. Raises LookUpError if no match is found.

fill_column(*column*, *item*, *start=None*, *end=None*, *style=None*)
Fill a column with a single repeated data item

A single data item is inserted into all rows in the specified column which have at least one data item already in any column in the worksheet. A different range of rows can be specified via the 'start' and 'end' arguments.

*** THIS METHOD IS DEPRECATED ***

Consider using insert_column, append_column or write_data.

Parameters

- **column** – index of column to insert the item into (e.g. 'A','MZ')
- **item** – data item to be repeated
- **start** – (optional) first row to insert data into
- **end** – (optional) last row to insert data into
- **style** – (optional) XLSStyle object to be associated with each cell that has data inserted into it

get_style(*idx*)
Return the style information associated with a cell

Returns an XLSStyle object associated with the specific cell.

If no style was previously associated then return a new XLSStyle object.

Parameters **idx** – cell index e.g 'A1'

Returns XLSStyle object.

insert_block_data(*data*, *col=None*, *row=None*, *style=None*)
Insert data items from a block of text

Data items are supplied via a block of tab- and newline-delimited text. Each tab-delimited item is inserted into the next column in a row; newlines indicate that subsequent items are inserted into the next row.

By default items are inserted starting from cell 'A1'; a different starting cell can be explicitly specified via the 'col' and 'row' arguments.

Parameters

- **data** – block of tab- and newline-delimited data
- **col** – (optional) first column to insert data into
- **row** – (optional) first row to insert data into

- **style** – (optional) XLSStyle object to be associated with each cell that has data inserted into it

insert_column(*position*, *data=None*, *text=None*, *fill=None*, *from_row=None*, *style=None*)
Create a new column at the specified column position

Inserts a new column at the specified column position, pushing up the column currently at that position plus all higher positioned columns.

By default the inserted column is empty, however data can be specified to populate the column.

Parameters

- **position** – column index specifying position to insert the column at
- **data** – optional, list of data items to populate the inserted column
- **text** – optional, tab-delimited string of text to be used to populate the inserted column
- **fill** – optional, single data item to be repeated to fill the inserted column
- **from_row** – optional, if specified then inserted column is populated from that row onwards
- **style** – optional, an XLSStyle object to associate with the data being inserted

Returns The index of the inserted column.

insert_column_data(*col*, *data*, *start=None*, *style=None*)
Insert list of data into a column

Data items are supplied as a list, with each item in the list being inserted into the next row in the column.

By default items are inserted starting from row 1, unless a starting row is explicitly specified via the 'start' argument.

*** THIS METHOD IS DEPRECATED ***

Consider using insert_column, append_column or write_data.

Parameters

- **col** – index of column to insert the data into (e.g. 'A','MZ')
- **data** – list of data items
- **start** – (optional) first row to insert data into
- **style** – (optional) XLSStyle object to be associated with each cell that has data inserted into it

insert_row(*position*, *data=None*, *text=None*, *fill=None*, *from_column=None*, *style=None*)
Create a new row at the specified row position

Inserts a new row at the specified row position, pushing up the row currently at that position plus all higher positioned row.

By default the inserted row is empty, however data can be specified to populate the column.

Parameters

- **position** – row index specifying position to insert the row at
- **data** – optional, list of data items to populate the inserted row
- **text** – optional, newline-delimited string of text to be used to populate the inserted row
- **fill** – optional, single data item to be repeated to fill the inserted row

- **from_row** – optional, if specified then inserted row is populated from that column onwards
- **style** – optional, an XLSStyle object to associate with the data being inserted

Returns The index of the inserted row.

insert_row_data(*row*, *data*, *start=None*, *style=None*)
Insert list of data into a row

Data items are supplied as a list, with each item in the list being inserted into the next column in the row.

By default items are inserted starting from column 'A', unless a starting column is explicitly specified via the 'start' argument.

*** THIS METHOD IS DEPRECATED ***

Consider using insert_row, append_row or write_row.

Parameters

- **row** – index of row to insert the data into (e.g. 1, 112)
- **data** – list of data items
- **start** – (optional) first column to insert data into
- **style** – (optional) XLSStyle object to be associated with each cell that has data inserted into it

last_column
Return index of last column with data

last_row
Return index of last row with data

next_column
Index of first empty column after highest index with data

next_row
Index of first empty row after highest index with data

render_as_text(*include_columns_and_rows=False*, *include_styles=False*, *eval_formulae=False*, *apply_format=False*, *start=None*, *end=None*)
Text representation of all or part of the worksheet

All or part of the sheet can be rendered as a tab- and newline-delimited string.

Parameters

- **include_columns_and_rows** – (optional) if True then also output a header row of column indices, and a column of row indices (default is to not output columns and rows).
- **include_styles** – (optional) if True then also render the styling information associated with the cell (default is not to apply styling).
- **apply_format** – (optional) if True then format numbers according to the formatting information associated with the cell (default is not to apply formatting).
- **eval_formulae** – (optional) if True then if the cell contains a formula, attempt to evaluate it and return the result. Otherwise return the formula itself (this is the default)
- **start** – (optional) specify the top-lefthand most cell index to start rendering from (default is 'A1').

- **end** – (optional) specify the bottom-righthand most cell index to finish rendering at (default is the cell corresponding to the highest column and row indices. Note that this cell may be empty.)

Returns String containing the rendered sheet or sheet subset, with items within a row separated by tabs, and rows separated by newlines.

render_cell(*idx*, *eval_formulae=False*, *apply_format=False*)
Text representation of value stored in a cell

Create a text representation of a cell's contents. If the cell contains a formula then '?'s will be replaced with the row index and '#'s with the column index. Optionally the formula can also be evaluated, and any style information associated with the cell can also be rendered.

Parameters

- **idx** – cell index e.g. 'A1'
- **eval_formulae** – (optional) if True then if the cell contains a formula, attempt to evaluate it and return the result. Otherwise return the formula itself (this is the default)
- **apply_format** – (optional) if True then format numbers according to the formatting information associated with the cell (default is not to apply formatting).

Returns String representing the cell contents.

row_is_empty(*row*)
Determine whether a row is empty

Returns False if any cells in the row are populated, otherwise returns True.

rowof(*s*, *column='A'*)
Return row index for cell which matches string

Return index of first row where the content matches the specified string 's'.

Parameters

- **s** – string to search for
- **column** – column to search in (defaults to 'A')

Returns Row index of first matching cell. Raises LookUpError if no match is found.

set_style(*cell_style*, *start*, *end=None*)
Associate style information with one or more cells

Associates a specified XLSStyle object with a single cell, or with a range of cells (if a second cell index is supplied).

The style associated with a cell can be fetched using the 'get_style' method.

Parameters

- **cell_style** – XLSStyle object
- **start** – cell index e.g. 'A1'
- **end** – (optional) second cell index; together with 'start' this defines a range of cells to associate the style with.

write_column(*col*, *data=None*, *text=None*, *fill=None*, *from_row=None*, *style=None*)
Write data to rows in a column

Data can be specified as a list, a newline-delimited string, or as a single repeated data item.

Parameters

- **data** – optional, list of data items to populate the inserted column
- **text** – optional, newline-delimited string of text to be used to populate the inserted column
- **fill** – optional, single data item to be repeated to fill the inserted column
- **from_row** – optional, if specified then inserted column is populated from that row onwards
- **style** – optional, an XLSStyle object to associate with the data being inserted

write_row(*row*, *data=None*, *text=None*, *fill=None*, *from_column=None*, *style=None*)
Write data to rows in a column

Data can be specified as a list, a tab-delimited string, or as a single repeated data item.

Parameters

- **row** – row index specifying which row
- **data** – optional, list of data items to populate the inserted row
- **text** – optional, tab-delimited string of text to be used to populate the inserted row
- **from_column** – optional, if specified then inserted row is populated from that column onwards
- **style** – optional, an XLSStyle object to associate with the data being inserted

class bcftbx.simple_xls.**XLSXLimits**
Limits for XLSX files

bcftbx.simple_xls.**cell**(*col*, *row*)
Return XLS cell index for column and row

E.g. cell('A',3) returns 'A3'

bcftbx.simple_xls.**cmp_column_indices**(*x*, *y*)
Comparision function for column indices

x and y are XLS-style column indices e.g. 'A', 'B', 'AA' etc.

Returns -1 if x is a column index less than y, 1 if it is greater than y, and 0 if it's equal.

bcftbx.simple_xls.**column_index_to_integer**(*col*)
Convert XLS-style column index into equivalent integer

Given a column index e.g. 'A', 'BZ' etc, converts it to the integer equivalent using zero-based counting system (so 'A' is equivalent to zero, 'B' to 1 etc).

bcftbx.simple_xls.**column_integer_to_index**(*idx*)
Convert integer column index to XLS-style equivalent

Given an integer index, converts it to the XLS-style equivalent e.g. 'A', 'BZ' etc, using a zero-based counting system (so zero is equivalent to 'A', 1 to 'B' etc).

bcftbx.simple_xls.**convert_to_number**(*s*)
Convert a number to float or int as appropriate

Raises ValueError if neither conversion is possible.

bcftbx.simple_xls.**eval_formula**(*item*, *worksheet*)
Evaluate a formula using the contents of a worksheet

Given an item, attempts an Excel-style evaluation.

If the item doesn't start with '=' then it is returned as-is. Otherwise the function attempts to evaluate the formula, including looking up (and if necessary also evaluating) the contents of any cells that are referenced.

Note: The implementation of the evaluation is very simplistic and cannot handle complex formulae or functions, it can only deal with basic mathematical operations (i.e. +, −, ⋆ and /)

---

bcftbx.simple_xls.**format_value**(*value*, *number_format=None*)
Format a cell value based on the specified number format

bcftbx.simple_xls.**incr_col**(*col*, *incr=1*)
Return column index incremented by specific number of positions

Parameters

- **col** – index of column to be incremented
- **incr** – optional, number of cells to shift by. Can be negative to go backwards. Defaults to 1 i.e. next column along.

bcftbx.simple_xls.**is_float**(*s*)
Test if a number is a float

bcftbx.simple_xls.**is_int**(*s*)
Test if a number is an integer

13.10.2 Spreadsheet

Provides classes for writing data to an Excel spreadsheet, using the 3rd party modules xlrd, xlwt and xlutils.

The basic classes are 'Workbook' (representing an XLS spreadsheet) and 'Worksheet' (representing a sheet within a workbook). There is also a 'Spreadsheet' class which is built on top of the other two classes and offers a simplified interface to writing line-by-line XLS spreadsheets.

Simple usage examples

1. Writing a new XLS spreadsheet using the Workbook class

```
>>> wb = Workbook()
>>> ws = wb.addSheet('test1')
>>> ws.addText("Hello   Goodbye
Goodbye Hello")
>>> wb.save('test2.xls')
```

2. Appending to an existing XLS spreadsheet using the Workbook class

```
>>> wb = Workbook('test2.xls')
>>> ws = wb.getSheet('test1')
>>> ws.addText("Some more data for you")
>>> ws = wb.addSheet('test2')
>>> ws.addText("<style font=bold bgcolor=gray25>Hahahah</style>")
>>> wb.save('test3.xls')
```

3. Creating or appending to an XLS spreadsheet using the Spreadsheet class

```
>>> wb = Spreadsheet('test.xls','test')
>>> wb.addTitleRow(['File','Total reads','Unmapped reads'])
>>> wb.addEmptyRow()
>>> wb.addRow(['DR_1',875897,713425])
>>> wb.write()
```

Module constants

MAX_LEN_WORKSHEET_TITLE: maximum length allowed by xlwt for worksheet titles
MAX_LEN_WORKSHEET_CELL_VALUE: maximum number of characters allowed for cell value
MAX_NUMBER_ROWS_PER_WORKSHEET: maximum number of rows allowed per worksheet by xlwt

Dependencies

The Spreadsheet module depends on the xlwt, xlrd and xlutils libraries which can be found at:

- http://pypi.python.org/pypi/xlwt/0.7.2
- http://pypi.python.org/pypi/xlrd/0.7.1
- http://pypi.python.org/pypi/xlutils/1.4.1

Note that xlutils also needs functools: http://pypi.python.org/pypi/functools

but if you're using Python<2.5 then you need a backported version of functools, try:

https://github.com/dln/pycassa/blob/90736f8146c1cac8287f66e8c8b64cb80e011513/pycassa/py25_functools.py

class bcftbx.Spreadsheet.**Spreadsheet**(*name*, *title*)
Class for creating and writing a spreadsheet.

This creates a very simple single-sheet workbook.

addEmptyRow(*color=None*)
Add an empty row to the spreadsheet.

Inserts an empty row into the next position in the spreadsheet.

Parameters **color** – optional background color for the empty row

Returns Integer index of (empty) row just written

addRow(*data*, *set_widths=False*, *bold=False*, *wrap=False*, *bg_color=''*)
Add a row of data to the spreadsheet.

Parameters

- **data** – list of data items to be added.
- **set_widths** – (optional) Boolean; if True then set the column width to the length of the cell contents for each cell in the new row
- **bold** – (optional) use bold font for cells
- **wrap** – (optional) wrap the cell content
- **bg_color** – (optional) set the background color for the cell

Returns Integer index of row just written

addTitleRow(*headers*)
Add a title row to the spreadsheet.

The title row will have the font style set to bold for all cells.

Parameters **headers** – list of titles to be added.

Returns Integer index of row just written

write()
Write the spreadsheet to file.

class bcftbx.Spreadsheet.**Styles**
Class for creating and caching EasyXfStyle objects.

XLS files have a limit of 4,000 styles, so cache and reuse EasyXfStyle objects to avoid exceeding this limit.

getXfStyle(*bold=False*, *wrap=False*, *color=None*, *bg_color=None*, *border_style=None*, *num_format_str=None*, *font_size=None*, *centre=False*, *shrink_to_fit=False*)
Return EasyXf object to apply styles to spreadsheet cells.

Parameters

- **bold** – indicate whether font should be bold face
- **wrap** – indicate whether text should wrap in the cell
- **color** – set text colo(u)r
- **bg_color** – set colo(u)r for cell background.
- **border_style** – set line type for cell borders (thin, medium, thick, etc)
- **font_size** – font size (in points)
- **centre** – centre the cell content horizontally
- **shrink_to_fit** – shrink cell to fit contents

Note that colours must be a valid name as recognised by xlwt.

class bcftbx.Spreadsheet.**Workbook**(*xls_name=''*)
Class for writing data to an XLS spreadsheet.

A Workbook represents an XLS spreadsheet, which conists of sheets (represented by Worksheet instances).

addSheet(*title*, *xlrd_sheet=None*, *xlrd_index=None*)
Add a new sheet to the spreadsheet.

Parameters

- **title** – title for the sheet
- **xlrd_sheet** – (optional) an xlrd sheet from an existing XLS workbook.

getSheet(*title*)
Retrieve a sheet from the spreadsheet.

save(*xls_name*)
Finish adding data and write the spreadsheet to disk.

Note that for a spreadsheet based on an existing XLS file, this doesn't have to be the same name.

Parameters **xls_name** – the file name to write the spreadsheet to. Note that if a file already exists with this name then it will be overwritten.

class bcftbx.Spreadsheet.**Worksheet**(*workbook*, *title*, *xlrd_index=None*, *xlrd_sheet=None*)

Class for writing to a sheet in an XLS spreadsheet.

A Worksheet object represents a sheet in an XLS spreadsheet.

Data can be inserted into the worksheet in a variety of ways:

- addTabData: a Python list of tab-delimited lines; each line forms a line in the output XLS, with each field forming a column.
- addText: a string representing arbitrary text, with newlines delimiting lines and tabs (if any) in each line delimiting fields.

Each can be called multiple times in any order on the same spreadsheet before it is saved, and the data will be appended.

For new Worksheet objects (i.e. those which weren't read from a pre-existing XLS file), it is also possible to insert new columns:

- insertColumn: if a single value is specified then all columns are filled with that value; alternatively a list of values can be supplied which are written one-per-row.

Formulae can be specified using a variation on Excel's '=' notation, e.g.

=A1+B2

adds the values from cells A1 and B2 in the final spreadsheet.

Formulae are written directly as supplied unless they contain special characters '?' (indicates the current line number) or '#' (indicates the current column).

Using '?' allows simple row-wise formulae to be added, e.g.

=A?+B?

will be converted to substitute the row index (e.g. '=A1+B1' for row 1, '=A2+B2' for row 2 etc).

Using '#' allows simple column-wise formulae to be added, e.g.

=#1-#2

will be converted to substitue the column id (e.g. '=A1-A2' for column A, '=B1-B2' for column B etc).

Note that the substitution occurs when the spreadsheet is saved.

Individual items can have basic styles applied to them by wrapping them in <style ...>...</style> tags. Within the leading style tag the following attributes can be specified:

font=bold (sets bold face) color=<color> (sets the text colour) bgcolor=<color> (sets the background colour) border=<style> (sets the cell border style to 'thin', 'medium', 'thick' etc) wrap (specifies that text should wrap) number_format=<format_string> (specifies how to display numbers, see below) font_height=<height> (sets font size in points) centre (specifies that text should be centred) shrink_to_fit (specifies that cells should shrink to fit their contents)

For example <style font=bold bgcolor=gray25>...</style>

Note that styles can also be applied to formulae.

The 'number_format' style attribute allows the calling program to specify how numbers should be displayed, for example:

number_format=0.00 (displays values to 2 decimal places) number_format=0.0% (displays values as percentages to 1 decimal place) number_format=#,### (displays values with , as the delimiter for thousands)

The spreadsheet data is held internally as a list of rows, with each row represented by a tab-delimited string.

addTabData (*rows*)
Write a list of tab-delimited data rows to the sheet.

Given a list of rows with tab-separated data items, append the data to the worksheet.

Parameters **data** – Python list representing rows of tab-separated data items

addText (*text*)
Append and populate rows from text.

Given some arbitrary text as a string, the data it contains will be appended to the worksheet using newlines to indicate multiple rows and tabs to delimit data items.

This method is useful for turning tab-delimited data read from a CSV-type file into a spreadsheet.

Parameters **text** – a string representing the data to add: rows are delimited by newlines, and items by tabs

column_id_from_index (*i*)
Get XLS column id from index of column

cindex is the zero-based column index (an integer); this method returns the matching XLS column identifier (i.e. 'A', 'B', 'AA', 'BA' etc).

freezePanes (*row=None*, *column=None*)
Split panes and mark as frozen

'row' and 'column' are integer indices specifying the cell which defines the pane to be marked as frozen

getColumnId (*name*)
Lookup XLS column id from name of column.

If there is no data, or if the name isn't in the header row of the data, then an exception is raised.

Returns the column identifier (i.e. 'A', 'B' etc) for the column with the matching name.

insertColumn (*position*, *insert_items=None*, *title=None*)
Insert a new column into the spreadsheet.

This inserts a new column into each row of data, at the specified positional index (starting from 0).

Note: at present columns can only be inserted into worksheets that have been created from scratch via Worksheet class (i.e. cannot insert into an existing worksheet read in from a file).

Parameters

- **position** – positional index for the column to be inserted at (0=A, 1=B etc)
- **title** – (optional) value to be written to the first row (i.e. a column title)
- **insert_items** – value(s) to be inserted; either a single item, or a list of items. Each item can be blank, a constant value, or a formula.

save ()
Write the new data to the spreadsheet.

setCellValue (*row*, *col*, *value*)
Set the value of a cell

Given row and column coordinates (using integer indices starting from zero for both), replace the existing value with a new one.

The new value can include style information.

Parameters

- **row** – integer row index (starting at zero)

- **col** – integer column index (starting at zero, i.e. 0=A, 1=B etc)
- **value** – new value to be written into the cell

13.11 bcftbx.cmdparse

Provides a CommandParser class for handling command lines of the form:

```
PROG COMMAND OPTION ARGS
```

where different sets of options can be defined based on the initial command.

The CommandParser can support arbitrary 'subparser backends' which are created to parse the ARGS list for each defined COMMAND. The default subparser is the 'argparse.ArgumentParser' class, but this can be swapped for arbitrary subparser when the CommandParser is created.

In addition to the core CommandParser class, there are a number of supporting functions that can be used with any argparse-based parser instance, to add the following 'standard' options:

- –nprocessors
- –runner
- –no-save
- –dry-run
- –debug

class bcftbx.cmdparse.**CommandParser**(*description=None*, *version=None*, *subparser=None*)

Class defining multiple command line parsers

This parser can process command lines of the form

PROG CMD OPTIONS ARGS

where different sets of options can be defined based on the major command ('CMD') supplied at the start of the line.

Usage:

Create a simple CommandParser which uses argparse.ArgumentParser as the default subparser backend using:

```
>>> p = CommandParser()
```

Alternatively, specify argparse.ArgumentParser as the subparser using:

```
>>> p = CommandParser(subparser=argparser.ArgumentParser)
```

Add a 'setup' command:

```
>>> p.add_command('setup',usage='%prog setup OPTIONS ARGS')
```

Add options to the 'setup' command using the appropriate methods of the subparser (e.g. 'add_argument' for an ArgumentParser instance).

For example:

```
>>> p.parser_for('info').add_argument('-f',...)
```

To process a command line, use the 'parse_args' method, for example for an OptionParser-based subparser:

```
>>> cmd,options,args = p.parse_args()
```

Note that the exact form of the returned values depends on on the subparser instance; it will be the same as that returned by the 'parse_args' method of the subparser.

add_command(*cmd*, *help=None*, ***args*)
Add a major command to the CommandParser

Adds a command, and creates and returns an initial subparser instance for it.

Parameters

- **cmd** – the command to be added
- **help** – (optional) help text for the command

Other arguments are passed to the subparser instance when it is created i.e. 'usage', 'version', 'description'.

If 'version' isn't specified then the version supplied to the CommandParser object will be used.

Returns Subparser instance object for the command.

error(*message*)
Exit with error message

handle_generic_commands(*cmd*)
Process 'generic' commands e.g. 'help'

list_commands()
Return the list of commands

parse_args(*argv=None*)
Process a command line

Parses a command line (either those supplied to the calling subprogram e.g. via the Python interpreter, or as a list).

Once the command is identified from the first argument, the remainder of the arguments are passed to the 'parse_args' method of the appropriate subparser for that command.

This method returns a tuple, with the first value being the command, and the rest of the values being those returned from the 'parse_args' method of the subparser.

Parameters **argv** – (optional) a list consisting of a command line. If not supplied then defaults to sys.argv[1:].

Returns A tuple of (cmd,...), where 'cmd' is the command, and '...' represents the values returned from the 'parse_args' method of the subparser. For example, using the default OptionParser backend returns (cmd,options,arguments), where 'options' and 'arguments' are the options and arguments as returned by OptionParser.parse_args; using ArgumentParser as a backend returns (cmd,arguments).

parser_for(*cmd*)
Return OptionParser for specified command

Returns The OptionParser object for the specified command.

print_available_commands()
Pretty-print available commands

Returns a 'pretty-printed' string for all options and commands, with standard whitespace formatting.

print_command(*cmd*, *message=None*)
Print a line for a single command

Returns a 'pretty-printed' line for the specified command and text, with standard whitespace formatting.

bcftbx.cmdparse.**add_arg**(*p*, **args*, ***kwds*)
Add an argument or option to a parser

Given an arbitrary parser instance, adds a new option or argument using the appropriate method call and passing the supplied arguments and keywords.

For example, if the parser is an instance of argparse.ArgumentParser, then the 'add_argument' method will be invoked to add a new argument to the parser.

Parameters

- **p** (*Object*) – parser instance
- **args** (*List*) – list of argument values to pass directly to the argument-addition method
- **kwds** (*mapping*) – keyword-value mapping to pass directly to the argument-addition method

bcftbx.cmdparse.**add_debug_option**(*parser*)
Add a '–debug' option to a parser

Given a parser instance 'parser' (either OptionParser or ArgumentParser), add a '–debug' option.

The value of this option can be accessed via the 'debug' attribute of the parser options.

Returns the input parser object.

bcftbx.cmdparse.**add_dry_run_option**(*parser*)
Add a '–dry-run' option to a parser

Given a parser instance 'parser' (either OptionParser or ArgumentParser), add a '–dry-run' option.

The value of this option can be accessed via the 'dry_run' attribute of the parser options.

Returns the input parser object.

bcftbx.cmdparse.**add_no_save_option**(*parser*)
Add a '–no-save' option to a parser

Given a parser instance 'parser' (either OptionParser or ArgumentParser), add a '–no-save' option.

The value of this option can be accessed via the 'no_save' attribute of the parser options.

Returns the input parser object.

bcftbx.cmdparse.**add_nprocessors_option**(*parser*, *default_nprocessors*, *default_display=None*)
Add a '–nprocessors' option to a parser

Given a parser instance 'parser' (either OptionParser or ArgumentParser), add a '–nprocessors' option.

The value of this option can be accessed via the 'nprocessors' attribute of the parser options.

If 'default_display' is not None then this value will be shown in the help text, rather than the value supplied for the default.

Returns the input parser object.

bcftbx.cmdparse.**add_runner_option**(*parser*)
Add a '–runner' option to a parser

Given a parser instance 'parser' (either OptionParser or ArgumentParser), add a '–runner' option.

The value of this option can be accessed via the 'runner' attribute of the parser options (use the 'fetch_runner' function to return a JobRunner object from the supplied value).

Returns the input parser object.

13.12 bcftbx.qc

13.12.1 bcftbx.qc.report

Utilities for generating reports for NGS QC pipeline runs.

class bcftbx.qc.report.**IlluminaQCReporter**(*dirn*, *data_format=None*, *qc_dir='qc'*, *regex_pattern=None*, *version=None*)

Class for reporting QC run on Illumina data

IlluminaQCReporter assembles the data associated with a QC run for a set of Illumina data and generates a HTML document which summarises the results for quick review.

report()

Write the HTML report

Writes a HTML document 'qc_report.html' to the top-level analysis directory.

zip()

Make a zip file containing the report and the images

Generate the 'qc_report.html' file and make a zip file 'qc_report.<run>.<name>.zip' which contains the report plus the associated image files, which can be unpacked elsewhere for viewing.

Returns Name of the zip file with the report.

class bcftbx.qc.report.**IlluminaQCSample**(*name*, *qc_dir*, *fastq=None*)

Class for holding QC data for an Illumina sample

An Illumina QC run typically consists of contamination screens and output from FastQC.

is_empty

Return True if the sample has no reads, False otherwise

report(*html*)

Write HTML report for this sample

verify()

Check QC products for this sample

Checks that fastq_screens and FastQC files were found. Returns True if the QC products are present and False otherwise.

class bcftbx.qc.report.**QCReporter**(*dirn*, *data_format=None*, *qc_dir='qc'*, *regex_pattern=None*, *version=None*)

Base class for reporting QC runs

This is a general class for reporting runs of the FLS NGS QC pipelines. QC reporters specific to particular pipelines should be subclassed from QCReporter and need to implement the 'report' method to generate the HTML output.

addSample(*sample*)

Add a QCSample class or subclass to the sample list

data_format

Return the format for the primary data files

dirn
Return top-level directory containing data

getPrimaryDataFiles()
Return list of primary data file sets

Returns a list of primary data file names; use the 'primary_data_dir' property to get the directory where the files are actually located.

html
Return HTMLPageWriter instance for the report

name
Return name of experiment

primary_data_dir
Return location of primary data files

qc_dir
Return directory holding QC outputs

report()
Generate a HTML report

This method must be implemented by the subclass.

report_base_name
Return the base name for the report

report_name
Return the full name for the report

run
Return name of run

samples
Return list of samples

verify()
Check that the QC outputs are correct

Returns True if the QC appears to have run successfully, False if not.

zip()
Make a zip file containing the report and the images

Generate the 'qc_report.html' file and make a zip file 'qc_report.<run>.<name>.zip' which contains the report plus the associated image files etc. The archive can then be unpacked elsewhere for viewing.

Returns Name of the zip file with the report.

exception bcftbx.qc.report.**QCReporterError**
Base class for errors with QCReporter-related code

class bcftbx.qc.report.**QCSample**(*name*, *qc_dir*)
Base class for reporting QC for a single sample

This is a general class for reporting the QC outputs associated with a single sample. It attempts to find all possible associated QC products for the given sample name.

Specific pipelines should subclass QCSample and implement the 'report' method, which can call the 'report_*' methods to produce HTML code specific to the pipeline in question.

addBoxplot(*boxplot*)
Associate a boxplot with the sample

Parameters **boxplot** – boxplot file name

addFastQC(*fastqc_dir*)
Associate a FastQC output directory with the sample

addProgramInfo(*programs*)
Collect program information from 'programs' file

addScreen(*screen*)
Associate a fastq_screen with the sample

Parameters **screen** – fastq_screen file name

boxplots()
Return list of boxplots for a sample

fastqc
Return name of FastQC run dir

programs
Return data on programs

report()
Generate a HTML report

This method must be implemented by the subclass.

report_boxplots(*html*, *paired_end=False*, *inline_pngs=True*)
Write HTML code reporting the boxplots

Parameters

- **html** – HTMLPageWriter instance to add the generated HTML to
- **inline_pngs** – if set True then embed the PNG images as base64 encoded data; otherwise link to the original image file

report_fastqc(*html*, *inline_pngs=True*)
Write HTML code reporting the results from FastQC

Parameters **html** – HTMLPageWriter instance to add the generated HTML to

report_programs(*html*)
Write HTML code reporting the program information

report_screens(*html*, *inline_pngs=True*)
Write HTML code reporting the fastq screens

Parameters

- **html** – HTMLPageWriter instance to add the generated HTML to
- **inline_pngs** – if set True then embed the PNG images as base64 encoded data; otherwise link to the original image file

screens()
Return list of screens for a sample

verify()
Verify expected QC products for the sample

This method must be implemented by the subclass. It should return True if the QC appears to have run successfully for the sample, False if not.

zip_includes()
Return list of files and directories to archive

class bcftbx.qc.report.**SolidQCReporter**(*dirn*, *data_format=None*, *qc_dir='qc'*, *regex_pattern=None*, *version=None*)

Class for reporting QC run on SOLiD data

SolidQCReporter assembles the data associated with a QC run for a set of SOLiD data and generates a HTML document which summarises the results for quick review.

report()

Write the HTML report

Writes a HTML document 'qc_report.html' to the top-level analysis directory.

verify()

Verify that SOLiD QC completed successfully for all samples

Returns True if the QC appears to have run successfully, False if not.

class bcftbx.qc.report.**SolidQCSample**(*name*, *qc_dir*, *paired_end*)

Class for holding QC data for a SOLiD sample

A SOLiD QC run typically consists of filtered and unfiltered boxplots, quality filtering stats, and contamination screens.

report(*html*)

Write HTML report for this sample

verify()

Check QC products for this sample

Checks that fastq_screens and boxplots were found. Returns True if the QC products are present and False otherwise.

bcftbx.qc.report.**add_dir_to_zip**(*z*, *dirn*, *zip_top_dir=None*)

Recursively add a directory and its contents to a zip archive

z is a zipfile.ZipFile object already opened for writing; this function adds all files in directory dirn and its subdirectories to z.

If zip_top_dir is not None then this is prepended to the file name written to the zip archive.

bcftbx.qc.report.**count_reads**(*csfasta_file*)

Count the number of reads in a CSFASTA file

Returns number of reads, or None

bcftbx.qc.report.**is_boxplot**(*name*, *f*)

Return True if f is a qc_boxplot associated with sample

'name' can be a file name, or a file 'root' i.e. filename with all trailing extensions removed.

bcftbx.qc.report.**is_fastq_screen**(*name*, *f*)

Return True if f is a fastq_screen file associated with name

'name' can be a file name, or a file 'root' i.e. filename with all trailing extensions removed.

bcftbx.qc.report.**is_fastqc**(*name*, *f*)

Return True if f is a FastQC file associated with name

'name' can be a file name, or a file 'root' i.e. filename with all trailing extensions removed.

bcftbx.qc.report.**is_program_info**(*name*, *f*)

Return True if f is a 'program info' file associated with name

'name' can be a file name, or a file 'root' i.e. filename with all trailing extensions removed.

bcftbx.qc.report.**split_sample_name**(*name*)
Split name into leading part plus trailing number

Returns (start,number)

bcftbx.qc.report.**strip_ngs_extensions**(*name*)
Remove fastq, fastq, csfasta or qual extensions from name

13.13 bcftbx.htmlpagewriter

htmlpagewriter

Provides HTMLPageWriter class which provides a simple programmatic interface for generating HTML files.

class bcftbx.htmlpagewriter.**HTMLPageWriter**(*title=''*)
Generic HTML generation class

HTMLPageWriter provides basic operations for writing HTML files.

Example usage:

```
>>> p = HTMLPageWriter("Example page")
>>> p.add("This is some text")
>>> p.write("example.html")
```

add(*content*)
Add content to page body

Note that the supplied content is added to the HTML document as-is; no escaping is performed so the content can include arbitrary HTML tags. Note also that no validation is performed.

Parameters **content** – text to add to the HTML document body

addCSSRule(*css_rule*)
Add CSS rule

Defines a CSS rule that will be inlined into a "style" tag in the HTML head when the document is written out.

The rule text is added as-is, e.g.:

```
>>> p = HTMLPageWriter("Example page")
>>> p.addCSSRule("body { color: blue; }")
```

No checking or validation is performed.

Parameters **css_rule** – text defining CSS rule

addJavaScript(*javascript*)
Add JavaScript

Defines a line of Javascript code that will be inlined into a "script" tag in the HTML head when the document is written out.

The code is added as-is, no checking or validation is performed.

Parameters **javascript** – Javascript code

write(*filen=None*, *fp=None*)
Write the HTML document to file

Generates a HTML document based on the content, styles etc that have been defined by calls to the object's methods.

Can supply either a filename or a file-like object opened for writing.

Parameters

- **filen** – name of the file to write the document to.
- **fp** – file-like object opened for writing; if this is supplied then filen argument will be ignored even if it is not None.

class bcftbx.htmlpagewriter.**PNGBase64Encoder**

Utility class to encode PNG file into a base64 string

Base64 encoded PNGs can be embedded in HTML <img> tags.

To use:

```
>>> p = PNGBase64Encoder.encodePNG("image.png")
```

encodePNG(*pngfile*)

Return base64 string encoding a PNG file.

13.14 bcftbx.utils

utils

Utility classes and functions shared between BCF codes.

General utility classes:

AttributeDictionary OrderedDictionary

File reading utilities:

getlines

File system wrappers and utilities:

PathInfo mkdir mkdirs mklink chmod touch format_file_size commonprefix is_gzipped_file rootname find_program get_current_user get_user_from_uid get_uid_from_user get_group_from_gid get_gid_from_group get_hostname walk list_dirs strip_ext

Symbolic link handling:

Symlink links

Sample name utilities:

extract_initials extract_prefix extract_index_as_string extract_index pretty_print_names name_matches

File manipulations:

concatenate_fastq_files

Text manipulations:

split_into_lines

Command line parsing utilities:

parse_named_lanes parse_lanes

13.14.1 General utility classes

class bcftbx.utils.**AttributeDictionary**(***args*)
Dictionary-like object with items accessible as attributes

AttributeDict provides a dictionary-like object where the value of items can also be accessed as attributes of the object.

For example:

```
>>> d = AttributeDict()
>>> d['salutation'] = "hello"
>>> d.salutation
... "hello"
```

Attributes can only be assigned by using dictionary item assignment notation i.e. d['key'] = value. d.key = value doesn't work.

If the attribute doesn't match a stored item then an AttributeError exception is raised.

len(d) returns the number of stored items.

The AttributeDict behaves like a dictionary for iterations, for example:

```
>>> for attr in d:
>>>     print("%s = %s" % (attr,d[attr]))
```

class bcftbx.utils.**OrderedDictionary**
Augumented dictionary which keeps keys in order

OrderedDictionary provides an augmented Python dictionary class which keeps the dictionary keys in the order they are added to the object.

Items are added, modified and removed as with a standard dictionary e.g.:

```
>>> d[key] = value
>>> value = d[key]
>>> del(d[key])
```

The 'keys()' method returns the OrderedDictionary's keys in the correct order.

13.14.2 File handling utilities

bcftbx.utils.**getlines**(*filen*)
Fetch lines from a file and return them one by one

This generator function tries to implement an efficient method of reading lines sequentially from a text file, by minimising the number of reads from the file and performing the line splitting in memory. It attempts to replicate the idiom:

```
>>> for line in io.open(filen):
>>> ...
```

using:

```
>>> for line in getlines(filen):
>>> ...
```

The file can be gzipped; this function should handle this invisibly provided that the file extension is '.gz'.

Parameters **filen** (*str*) – path of the file to read lines from

Yields *String* –

next line of text from the file, with any newline character removed.

13.14.3 File system wrappers and utilities

class bcftbx.utils.**PathInfo**(*path*, *basedir=None*)

Collect and report information on a file

The PathInfo class provides an interface to getting general information on a path, which may point to a file, directory, link or non-existent location.

The properties provide information on whether the path is readable (i.e. accessible) by the current user, whether it is readable by members of the same group, who is the owner and what group does it belong to, when was it last modified etc.

chown(*user=None*, *group=None*)

Change associated owner and group

'user' and 'group' must be supplied as UID/GID numbers (or None to leave the current values unchanged).

*** Note that chown will fail attempting to change the owner if the current process is not owned by root ***

This is actually a wrapper to the os.lchmod function, so it doesn't follow symbolic links.

datetime

Return last modification time as datetime object

deepest_accessible_parent

Return longest accessible directory that leads to path

Tries to find the longest parent directory above path which is accessible by the current user.

If it's not possible to find a parent that is accessible then raise an exception.

exists

Return True if the path refers to an existing location

Note that this is a wrapper to os.path.lexists so it reports the existence of symbolic links rather than their targets.

gid

Return associated GID (group ID)

Attempts to return the GID (group ID) number associated with the path.

If the GID can't be found then returns None.

group

Return associated group name

Attempts to return the group name associated with the path. If the name can't be found then tries to return the GID instead.

If neither pieces of information can be found then returns None.

is_dir

Return True if path refers to a directory

is_executable

Return True if path refers to an executable file

is_file
Return True if path refers to a file

is_group_readable
Return True if the path exists and is group-readable

Paths may be reported as unreadable for various reasons, e.g. the target doesn't exist, or doesn't have permission for this user to read it, or if part of the path doesn't allow the user to read the file.

is_group_writable
Return True if the path exists and is group-writable

Paths may be reported as unwritable for various reasons, e.g. the target doesn't exist, or doesn't have permission for this user to write to it, or if part of the path doesn't allow the user to read the file.

is_link
Return True if path refers to a symbolic link

is_readable
Return True if the path exists and is readable by the owner

Paths may be reported as unreadable for various reasons, e.g. the target doesn't exist, or doesn't have permission for this user to read it, or if part of the path doesn't allow the user to read the file.

mtime
Return last modification timestamp for path

path
Return the filesystem path

relpath(*dirn*)
Return part of path relative to a directory

Wrapper for os.path.relpath(...).

resolve_link_via_parent
If path or parent directory is a link then return actual path

Resolves and returns the 'real' path for a path where either it or one of its parent directories is a symbolic link.

It will resolve multiple levels of symlinks to generate a path that is free of links (nb it is possible that the resolved path will not be an existing file or directory).

If there are no links in the directory tree then returns the full path of the input.

uid
Return associated UID (user ID)

Attempts to return the UID (user ID) number associated with the path.

If the UID can't be found then returns None.

user
Return associated user name

Attempts to return the user name associated with the path. If the name can't be found then tries to return the UID instead.

If neither pieces of information can be found then returns None.

bcftbx.utils.**mkdir**(*dirn*, *mode=None*, *recursive=False*)
Make a directory

Parameters

- **dirn** – the path of the directory to be created
- **mode** – (optional) a mode specifier to be applied to the new directory once it has been created e.g. 0775 or 0664
- **recursive** – (optional) if True then also create any intermediate parent directories if they don't already exist

bcftbx.utils.**mklink**(*target*, *link_name*, *relative=False*)
Make a symbolic link

Parameters

- **target** – the file or directory to link to
- **link_name** – name of the link
- **relative** – if True then make a relative link (if possible); otherwise link to the target as given (default)

bcftbx.utils.**chmod**(*target*, *mode*)
Change mode of file or directory

This a wrapper for the os.chmod function, with the addition that it doesn't follow symbolic links.

For symbolic links it attempts to use the os.lchmod function instead, as this operates on the link itself and not the link target. If os.lchmod is not available then links are ignored.

Parameters

- **target** – file or directory to apply new mode to
- **mode** – a valid mode specifier e.g. 0775 or 0664

bcftbx.utils.**touch**(*filename*)
Create new empty file, or update modification time if already exists

Parameters **filename** – name of the file to create (can include leading path)

bcftbx.utils.**format_file_size**(*fsize*, *units=None*)
Format a file size from bytes to human-readable form

Takes a file size in bytes and returns a human-readable string, e.g. 4.0K, 186M, 1.5G.

Alternatively specify the required units via the 'units' arguments.

Parameters

- **fsize** – size in bytes
- **units** – (optional) specify output in kb ('K'), Mb ('M'), Gb ('G') or Tb ('T')

Returns Human-readable version of file size.

bcftbx.utils.**commonprefix**(*path1*, *path2*)
Determine common prefix path for path1 and path2

Use this in preference to os.path.commonprefix as the version in os.path compares the two paths in a character-wise fashion and so can give counter-intuitive matches; this version compares path components which seems more sensible.

For example: for two paths /mnt/dir1/file and /mnt/dir2/file, os.path.commonprefix will return /mnt/dir, whereas this function will return /mnt.

Parameters

- **path1** – first path in comparison

- **path2** – second path in comparison

Returns Leading part of path which is common to both input paths.

bcftbx.utils.**is_gzipped_file**(*filename*)
Check if a file has a .gz extension

Parameters **filename** – name of the file to be tested (can include leading path)

Returns True if filename has trailing .gz extension, False if not.

bcftbx.utils.**rootname**(*name*)
Remove all extensions from name

Parameters **name** – name of a file

Returns Leading part of name up to first dot, i.e. name without any trailing extensions.

bcftbx.utils.**find_program**(*name*)
Find a program on the PATH

Search the current PATH for the specified program name and return the full path, or None if not found.

bcftbx.utils.**get_current_user**()
Return name of the current user

Looks up user name for the current user; returns None if no matching name can be found.

bcftbx.utils.**get_user_from_uid**(*uid*)
Return user name from UID

Looks up user name matching the supplied UID; returns None if no matching name can be found.

bcftbx.utils.**get_uid_from_user**(*user*)
Return UID from user name

Looks up UID matching the supplied user name; returns None if no matching name can be found.

NB returned UID will be an integer.

bcftbx.utils.**get_group_from_gid**(*gid*)
Return group name from GID

Looks up group name matching the supplied GID; returns None if no matching name can be found.

bcftbx.utils.**get_gid_from_group**(*group*)
Return GID from group name

Looks up GID matching the supplied group name; returns None if no matching name can be found.

NB returned GID will be an integer.

bcftbx.utils.**walk**(*dirn*, *include_dirs=True*, *pattern=None*)
Traverse the directory, subdirectories and files

Essentially this 'walk' function is a convenience wrapper for the 'os.walk' function.

Parameters

- **dirn** – top-level directory to start traversal from
- **include_dirs** – if True then yield directories as well as files (default)
- **pattern** – if not None then specifies a regular expression pattern which restricts the set of yielded files and directories to a subset of those which match the pattern

bcftbx.utils.**list_dirs**(*parent*, *matches=None*, *startswith=None*)
Return list of subdirectories relative to 'parent'

Parameters

- **parent** – directory to list subdirectories of
- **matches** – if not None then only include subdirectories that exactly match the supplied string
- **startswith** – if not None then then return subset of subdirectories that start with the supplied string

Returns List of subdirectories (relative to the parent dir).

bcftbx.utils.**strip_ext**(*name*, *ext=None*)

Strip extension from file name

Given a file name or path, remove the extension (including the dot) and return just the leading part of the name.

If an extension is explicitly specified then only remove the extension if it matches.

Extension can be multipart e.g. 'fastq.gz' and can include a leading dot e.g. '.gz' or 'gz'.

Parameters **name** – name of a file

Returns Leading part of name excluding specified extension, or first extension i.e. to last dot.

13.14.4 Symbolic link handling

class bcftbx.utils.**Symlink**(*path*)

Class for interrogating and modifying symbolic links

The Symlink class provides an interface for getting information about a symbolic link.

To create a new Symlink instance do e.g.:

```
>>> l = Symlink('my_link.lnk')
```

Information about the link can be obtained via the various properties:

- target = returns the link target
- is_absolute = reports if the target represents an absolute link
- is_broken = reports if the target doesn't exist

There are also methods:

- resolve_target() = returns the normalise absolute path to the target
- update_target() = updates the target to a new location

is_absolute

Return True if the link target is an absolute link

is_broken

Return True if the link target doesn't exist i.e. link is broken

resolve_target()

Return the normalised absolute path to the link target

target

Return the target of the symlink

update_target(*new_target*)

Replace the current link target with new_target

Parameters **new_target** – path to replace the existing target with

bcftbx.utils.**links**(*dirn*)
Traverse and return all symbolic links in under a directory

Given a starting directory, traverses the structure underneath and yields the path for each symlink that is found.

Parameters **dirn** – name of the top-level directory

Returns Yields the name and full path for each symbolic link under 'dirn'.

13.14.5 Sample name utilities

bcftbx.utils.**extract_initials**(*name*)
Return leading initials from the library or sample name

Conventionaly the experimenter's initials are the leading characters of the name e.g. 'DR' for 'DR1', 'EP' for 'EP_NCYC2669', 'CW' for 'CW_TI' etc

Parameters **name** – the name of a sample or library

Returns The leading initials from the name.

bcftbx.utils.**extract_prefix**(*name*)
Return the library or sample name prefix

Parameters **name** – the name of a sample or library

Returns The prefix consisting of the name with trailing numbers removed, e.g. 'LD_C' for 'LD_C1'

bcftbx.utils.**extract_index_as_string**(*name*)
Return the library or sample name index as a string

Parameters **name** – the name of a sample or library

Returns The index, consisting of the trailing numbers from the name. It is returned as a string to preserve leading zeroes, e.g. '1' for 'LD_C1', '07' for 'DR07' etc

bcftbx.utils.**extract_index**(*name*)
Return the library or sample name index as an integer

Parameters **name** – the name of a sample or library

Returns The index as an integer, or None if the index cannot be converted to integer format.

bcftbx.utils.**pretty_print_names**(*name_list*)
Given a list of library or sample names, format for pretty printing.

Parameters **name_list** – a list or tuple of library or sample names

Returns

String with a condensed description of the library names, for example:

['DR1', 'DR2', 'DR3', DR4'] -> 'DR1-4'

bcftbx.utils.**name_matches**(*name*, *pattern*)
Simple wildcard matching of project and sample names

Matching options are:

- exact match of a single name e.g. pattern 'PJB' matches 'PJB'
- match start of a name using trailing '*' e.g. pattern 'PJ*' matches 'PJB','PJBriggs' etc

- match using multiple patterns by separating with comma e.g. pattern 'PJB,IJD' matches 'PJB' or 'IJD'. Subpatterns can include trailing '*' character to match more names.

Arguments name: text to match against pattern pattern: simple 'glob'-like pattern to match against

Returns True if name matches pattern; False otherwise.

13.14.6 File manipulations

bcftbx.utils.**concatenate_fastq_files**(*merged_fastq*, *fastq_files*, *bufsize=10240*, *overwrite=False*, *verbose=True*)

Create a single FASTQ file by concatenating one or more FASTQs

Given a list or tuple of FASTQ files (which can be compressed or uncompressed or a combination), creates a single output FASTQ by concatenating the contents.

Parameters

- **merged_fastq** – name of output FASTQ file (mustn't exist beforehand)
- **fastq_files** – list of FASTQ files to concatenate
- **bufsize** – (optional) size of buffer to use for copying data
- **overwrite** – (optional) if True then overwrite the output file if it already exists (otherwise raise OSError); default is False
- **verbose** – (optional) if True then report operations to stdout, otherwise operate quietly

13.14.7 Text manipulations

bcftbx.utils.**split_into_lines**(*text*, *char_limit*, *delimiters='\t\n'*, *sympathetic=False*)

Split a string into multiple lines with maximum length

Splits a string into multiple lines on one or more delimiters (defaults to the whitespace characters i.e. ' ',tab and newline), such that each line is no longer than a specified length.

For example:

```
>>> split_into_lines("This is some text to split",10)
['This is','some text','to split']
```

If it's not possible to split part of the text to a suitable length then the line is split "unsympathetically" at the line length, e.g.

```
>>> split_into_lines("This is supercalifragilicous text",10)
['This is','supercalif','ragilicous','text']
```

Set the 'sympathetic' flag to True to include a hyphen to indicate that a word has been broken, e.g.

```
>>> split_into_lines("This is supercalifragilicous text",10,
...                  sympathetic=True)
['This is','supercali-','fragilico-','us text']
```

To use an alternative set of delimiter characters, set the 'delimiters' argument, e.g.

```
>>> split_into_lines("This: is some text",10,delimiters=':')
['This',' is some t','ext']
```

Parameters

- **text** – string of text to be split into lines
- **char_limit** – maximum length for any given line
- **delimiters** – optional, specify a set of non-default delimiter characters (defaults to whitespace)
- **sympathetic** – optional, if True then add hyphen to indicate when a word has been broken

Returns List of lines (i.e. strings).

13.15 bcftbx.ngsutils

ngsutils

Utility classes and functions specific to NGS applications.

Extracting reads from Fastq, cfasta and qual files:

- getreads: fetch reads one-by-one from Fastq, cfasta or qual file
- getreads_subset: fetch subset of reads specified by index
- getreads_regexp: fetch subset of reads matching regular expression

13.15.1 Extracting reads from Fastq, cfasta and qual files

bcftbx.ngsutils.**getreads**(*filen*)

Return Fastq, csfasta or qual file reads one-by-one

This generator function iterates through a sequence file (Fastq, csfasta or qual), and yields read records one at a time. The read records are returned as lists of lines.

The file can be gzipped; this function should handle this invisibly provided that the file extension is '.gz'.

Lines starting with '#' at the start of the file will be treated as comments and ignored. Lines starting with '#' which occur in the body of the file (i.e. after one or more lines of data) will be treated as data.

Example usage:

```
>>> for r in getreads('illumina_R1.fq'):
>>> ... print(r)
```

Parameters **filen** (*str*) – path of the file to fetch reads from

Yields *List* –

next read record from the file, as a list of lines.

bcftbx.ngsutils.**getreads_subset**(*filen*, *indices*)

Fetch subset of reads from Fastq, csfasta or qual file

This generator function iterates through a sequence file (Fastq, csfasta or qual), and yields a subset of the read records which are referenced by the supplied iterable indices.

The subset compromises of reads at the index positions specified by the list of indices, with index 0 being the first read in the file. Each read is returned as a list of lines.

The file can be gzipped; this function should handle this invisibly provided that the file extension is '.gz'.

Example usage (returns 1st, 3rd and 5th reads only):

```
>>> for r in getreads_subset('illumina_R1.fq',(0,2,4)):
>>> ... print(r)
```

Parameters

- **filen** (*str*) – path of the file to fetch reads from
- **indices** (*list*) – list of read indices to return

Yields *List* –

next read record from the file, as a list of lines.

bcftbx.ngsutils.**getreads_regex**(*filen*, *pattern*)

Fetch matching reads from Fastq, csfasta or qual file

This generator function iterates through a sequence file (Fastq, csfasta or qual), and yields a subset of read records. Each read is returned as a list of lines.

The subset compromises of reads which match the supplied regular expression.

The file can be gzipped; this function should handle this invisibly provided that the file extension is '.gz'.

Example usage:

```
>>> for r in getreads_regexp('illumina_R1.fq',"2102:3130"):
>>> ... print(r)
```

Parameters

- **filen** (*str*) – path of the file to fetch reads from
- **pattern** (*list*) – Python regular expression pattern

Yields *List* –

next read record from the file, as a list of lines.

CHAPTER 14

Version History and Changes

14.1 Version 1.11.1 (2021-06-07)

- bcftbx/mock: fix mock HISEQ sample sheet data https://github.com/fls-bioinformatics-core/genomics/pull/181
- illumina2cluster/verify_paired.py: fix broken `--version` option https://github.com/fls-bioinformatics-core/genomics/pull/180

14.2 Version 1.11.0 (2020-09-16)

- bcftbx/TabFile: 'TabFile' can keep commented lines from the input file by specifying 'keep_commented_lines=True' https://github.com/fls-bioinformatics-core/genomics/pull/178
- bcftbx/TabFile: 'TabFile.appendColumn' method accepts new 'fill_value' argument, to provide a default value to put into all rows in the new column https://github.com/fls-bioinformatics-core/genomics/pull/176
- bcftbx/mock: mock 'RunInfoXml' updated to set the flowcell ID more consistently with real-life examples https://github.com/fls-bioinformatics-core/genomics/pull/177

14.3 Version 1.10.0 (2020-09-16)

- bcftbx/IlluminaData: relax the platform identification mechanism in 'IlluminaRun'; Illumina-like run directories will be identified as generic 'illumina' platform if no explicit platform is supplied or can be identified from the instrument name. Exceptions are now only raised for run directories which do not appear to come from an Illumina sequencer https://github.com/fls-bioinformatics-core/genomics/pull/174
- bcftbx/IlluminaData: add new properties to 'IlluminaRun' instances: 'sample_sheet' ('SampleSheet' instance) and 'runinfo' ('IlluminaRunInfo' instance) https://github.com/fls-bioinformatics-core/genomics/pull/173
- bcftbx/IlluminaData: add new properties to 'IlluminaRunInfo' instances: 'instrument', 'date', 'flowcell' and 'lane_count' (extracted from 'RunInfo.xml' file) https://github.com/fls-bioinformatics-core/genomics/pull/172

- bcftbx/IlluminaData: 'SampleSheet' class ignores trailing empty lines present in the input sample sheet file https://github.com/fls-bioinformatics-core/genomics/pull/171
- bcftbx/JobRunner: 'SimpleJobRunner' reports status of 'join_logs' in '__repr__'; 'fetch_runner' handles 'join_logs' when setting up 'SimpleJobRunner' https://github.com/fls-bioinformatics-core/genomics/pull/170
- QC-pipeline: 'fastq_screen.sh' updated to handle FastqScreen v0.13 and v0.14 https://github.com/fls-bioinformatics-core/genomics/pull/168

14.4 Version 1.9.1 (2020-06-09)

- bcftbx: fix unclosed files and related bugs that were producing 'ResourceWarnings' under Python 3 tests https://github.com/fls-bioinformatics-core/genomics/pull/163
- bcftbx/JobRunner: improvements to thread safety of the 'SimpleJobRunner' class when handling job completion and cleanup https://github.com/fls-bioinformatics-core/genomics/pull/166

14.5 Version 1.9.0 (2020-05-20)

- bcftbx/JobRunner: enable available number of CPUS (aka slots, cores, threads) to be set and accessed within the 'SimpleJobRunner' and 'GEJobRunner' classes https://github.com/fls-bioinformatics-core/genomics/pull/152
- bcftbx/mock: update 'MockIlluminaData' class to enable forcing of creation of sample-level subdirectories when generating mock data https://github.com/fls-bioinformatics-core/genomics/pull/161
- bcftbx/IlluminaData: update 'SampleSheetPredictor' to handle prediction of index reads, and to handle arbitrary reads https://github.com/fls-bioinformatics-core/genomics/pull/160
- bcftbx/htmlpagewriter: remove unused imports https://github.com/fls-bioinformatics-core/genomics/pull/158
- Extend the list of supported Python versions to include 3.6 and 3.8; update the licence to Academic Free License AFL 3.0 https://github.com/fls-bioinformatics-core/genomics/pull/157
- config/qc.setup.sample: updated to allow user-defined environment variables to take precedence over values defined in the setup file https://github.com/fls-bioinformatics-core/genomics/pull/156

14.6 Version 1.8.3 (2020-02-27)

- bcftbx: remove internal version numbers from modules which still had them https://github.com/fls-bioinformatics-core/genomics/pull/155
- bcftbx/htmlpagewriter: update 'PNGBase64Encoder' for Python 3 compatibility https://github.com/fls-bioinformatics-core/genomics/pull/154
- bcftbx/IlluminaData: 'SampleSheetPredictor' updated to handle blank lane numbers in input samplesheet https://github.com/fls-bioinformatics-core/genomics/pull/153

14.7 Version 1.8.2 (2020-02-17)

- bcftbx/IlluminaData: fix error in call to 'digits' method in 'split_run_name_full' https://github.com/fls-bioinformatics-core/genomics/pull/149

- NGS-general/extract_reads.py: fix bug with handling gzipped files under Python 2, and broken `--version` option under Python 3 https://github.com/fls-bioinformatics-core/genomics/pull/150
- bcftbx/FASTQFile: fix bugs with reading Fastqs from disk under Python 3 https://github.com/fls-bioinformatics-core/genomics/pull/151

14.8 Version 1.8.1 (2019-11-20)

- bcftbx/IlluminaData: fix to *SampleSheet* class to handle cases when header lines have a 'key' without a comma delimiter or value (thanks Ryan Golhar @golharam) https://github.com/fls-bioinformatics-core/genomics/pull/148

14.9 Version 1.8.0 (2019-09-27)

- Updates for compatibility with Python 2.7 and 3.7
  - https://github.com/fls-bioinformatics-core/genomics/pull/146
  - https://github.com/fls-bioinformatics-core/genomics/pull/145
  - https://github.com/fls-bioinformatics-core/genomics/pull/144
  - https://github.com/fls-bioinformatics-core/genomics/pull/143
  - https://github.com/fls-bioinformatics-core/genomics/pull/141
  - https://github.com/fls-bioinformatics-core/genomics/pull/139
  - https://github.com/fls-bioinformatics-core/genomics/pull/138
  - https://github.com/fls-bioinformatics-core/genomics/pull/137
  - https://github.com/fls-bioinformatics-core/genomics/pull/136
  - https://github.com/fls-bioinformatics-core/genomics/pull/135
  - https://github.com/fls-bioinformatics-core/genomics/pull/134
  - https://github.com/fls-bioinformatics-core/genomics/pull/133
  - https://github.com/fls-bioinformatics-core/genomics/pull/132
  - https://github.com/fls-bioinformatics-core/genomics/pull/131
  - https://github.com/fls-bioinformatics-core/genomics/pull/130
  - https://github.com/fls-bioinformatics-core/genomics/pull/128
  - https://github.com/fls-bioinformatics-core/genomics/pull/127
  - https://github.com/fls-bioinformatics-core/genomics/pull/126
  - https://github.com/fls-bioinformatics-core/genomics/pull/125
  - https://github.com/fls-bioinformatics-core/genomics/pull/124
  - https://github.com/fls-bioinformatics-core/genomics/pull/121
  - https://github.com/fls-bioinformatics-core/genomics/pull/120
  - https://github.com/fls-bioinformatics-core/genomics/pull/119
  - https://github.com/fls-bioinformatics-core/genomics/pull/118

  - https://github.com/fls-bioinformatics-core/genomics/pull/117
  - https://github.com/fls-bioinformatics-core/genomics/pull/116
  - https://github.com/fls-bioinformatics-core/genomics/pull/115
  - https://github.com/fls-bioinformatics-core/genomics/pull/114
  - https://github.com/fls-bioinformatics-core/genomics/pull/113
  - https://github.com/fls-bioinformatics-core/genomics/pull/112
  - https://github.com/fls-bioinformatics-core/genomics/pull/110
  - https://github.com/fls-bioinformatics-core/genomics/pull/109
  - https://github.com/fls-bioinformatics-core/genomics/pull/108
  - https://github.com/fls-bioinformatics-core/genomics/pull/107
  - https://github.com/fls-bioinformatics-core/genomics/pull/106

14.10 Version 1.7.0 (2019-07-04)

- bcftbx/cmdparse: updated to use *argparse* as the default subparser https://github.com/fls-bioinformatics-core/genomics/pull/99
- bcftbx: switch to using Python3-compatible *print* function instead of *print* statement https://github.com/fls-bioinformatics-core/genomics/pull/100
- bcftbx: fix Python syntax for raising and capturing exceptions https://github.com/fls-bioinformatics-core/genomics/pull/101
- bcftbx/JobRunner: remove the *DRMAAJobRunner* class https://github.com/fls-bioinformatics-core/genomics/pull/102
- illumina2cluster/prep_sample_sheet.py: fix to bug with conflicting *-v* options introduced in previous version https://github.com/fls-bioinformatics-core/genomics/pull/105

14.11 Version 1.6.0 (2019-06-10)

- Command line utilities: updated to use *argparse* for processing command line arguments https://github.com/fls-bioinformatics-core/genomics/pull/96
- bcftbx: Python classes updated to ensure they all inherit from *object* https://github.com/fls-bioinformatics-core/genomics/pull/95
- bcftbx/mock: *MockIlluminaData* updated to handle arbitrary reads (e.g. *R1*,'R2','I1') when creating Fastqs https://github.com/fls-bioinformatics-core/genomics/pull/97

14.12 Version 1.5.5 (2019-04-30)

- bcftbx/JobRunner: stability improvements and bug fixes to GEJobRunner https://github.com/fls-bioinformatics-core/genomics/pull/88 https://github.com/fls-bioinformatics-core/genomics/pull/90 https://github.com/fls-bioinformatics-core/genomics/pull/91

14.13 Version 1.5.4 (2019-02-21)

- bcftbx/IlluminaData: fix to SampleSheet class to handle samplesheet files which contain *[Manifests]* section https://github.com/fls-bioinformatics-core/genomics/pull/87

14.14 Version 1.5.3 (2019-01-31)

- bcftbx/JobRunner: fixes to GEJobRunner to deal with race conditions on job finalization https://github.com/fls-bioinformatics-core/genomics/pull/85

14.15 Version 1.5.2 (2018-09-28)

- QC-pipeline/fastq_strand.py:
  - version 0.0.4: fixes cases when *STAR* fails to map any reads https://github.com/fls-bioinformatics-core/genomics/pull/81
- QC-pipeline/illumina_qc.sh:
  - version 1.3.3: fixes bug setting permissions when using *–no-screens* option https://github.com/fls-bioinformatics-core/genomics/pull/82
- bcftbx/JobRunner: updates to *GEJobRunner* to improve thread safety https://github.com/fls-bioinformatics-core/genomics/pull/80

14.16 Version 1.5.1 (2018-09-13)

- bcftbx/IlluminaData:
  - add *iSeq* to the list of known platforms
  - enable handling of run names with four-digit year in the datestamp https://github.com/fls-bioinformatics-core/genomics/pull/79
  - drop module-level version number

14.17 Version 1.5.0 (2018-08-22)

- bcftbx/JobRunner: substantial overhaul of *GEJobRunner* to reduce footprint when running on compute cluster e.g. removed calls to *qacct* and reduced calls to *qstat*.
  - https://github.com/fls-bioinformatics-core/genomics/pull/73
  - https://github.com/fls-bioinformatics-core/genomics/pull/76
- NGS-general/split_fastq.py: new utility that splits a Fastq file or R1/R2 pair based on the lanes present in the file(s); can be used to reverse the merging of Fastq files when *bcl2fastq* is run with *–no-lane-splitting*
  - https://github.com/fls-bioinformatics-core/genomics/pull/77
- QC-pipeline/fastq_strand.py:
  - version 0.0.3

  - removes existing output files on startup
  - only write final outputs on success
  - always remove temporary working directories on completion (even if program failed)
  - https://github.com/fls-bioinformatics-core/genomics/pull/72
- bcftbx/utils: reimplement *AttributeDictionary* class so it can be pickled
  - https://github.com/fls-bioinformatics-core/genomics/pull/78

14.18 Version 1.4.0 (2018-07-03)

- ChIP-seq/make_macs2_xls.py
  - version 0.5.0: add '-b'/'–bed' option to output additional TSV file with { chrom, abs_summit+/-100 } columns
- QC-pipeline/fastq_strand.py:
  - version 0.0.2:
  - can be run on a single Fastq (as well as pairs)
  - changes to command line if specifying STAR indexes directly: now needs '-g'/'–genome' option for this
- QC-pipeline/illumina_qc.sh:
  - version 1.3.2: new '–no-screens' option suppresses running of 'fastq_screen'

14.19 Version 1.3.2 (2018-05-14)

- bcftbx/JobRunner: update *GEJobRunner* to sanitize the supplied job name for use internally (before submission to Grid Engine); the supplied name is still used for communicating with external processes

14.20 Version 1.3.1 (2018-04-19)

- bcftbx/JobRunner: fix *GEJobRunner* to wrap script arguments in double quotes if they contain whitespace

14.21 Version 1.3.0 (2018-03-29)

- QC-pipeline/fastq_strand.py: new utility program which runs the STAR aligner to generate statistics on the strandedness of Fastq R1/R2 file pairs
- bcftbx/IlluminaData: fix the *fix_bases_mask* function to correctly handle empty barcode sequences

14.22 Version 1.2.0 (2018-03-29)

- NGS-general/reorder_fasta.py: new utility program to reorder chromosomes into karyotypical order in a FASTA file

- bcftbx/IlluminaData: new function *split_run_name_full*, which also extracts the datestamp, instrument name, flow cell ID and prefix from the run name
- bcftbx/IlluminaData: allow platform to be specified explicitly when creating *IlluminaRun* objects (for when platform cannot be extracted from the data directory name)

14.23 Version 1.1.0 (2018-01-24)

- bcftbx/cmdparse: major update to enable *argparse* to used as an alternative to *optparse* when parsing subcommands (thanks to Mohit Agrawal *@mohit2agrawal*)
- bcftbx/IlluminaData:
  - Enable *SampleSheet* class to handle quoted header values with commas in IEM-format sample sheets
  - Update *SampleSheetPredictor* to handle missing (blank) projects; fix bugs with the *set* method and update documentation.
- bcftbx/JobRunner: trap for attempt to delete a a missing/already deleted job in *SimpleJobRunner.list()*

14.24 Version 1.0.4 (2017-10-05)

- bcftbx/utils:
  - *mkdir* function supports new *recursive* option (creates any intermediate directories that are required)
  - New *mkdirs* function creates intermediate directories automatically (wraps *mkdir*)
- bcftbx/IlluminaData: samplesheet prediction and validation allows invoking subprogram to force insertion of 'sample' directory level even if *bcl2fastq* wouldn't normally produce one (needed for 10xGenomics *cellranger mkfastq* output)
- bcftbx/ngsutils: new library module with file reading and Fastq read extraction functions taken from *NGS-general/extract_reads.py* utility
- NGS-general/extract_reads.py: read extraction functions moved into new *bcftbx.ngsutils* module

14.25 Version 1.0.3 (2017-08-31)

- QC-pipeline/illumina_qc.sh:
  - version 1.3.1
  - reduce the default subset size for *fastq_screen* to 10000
  - can now handle Fastqs with *.fq[.gz]* extension
  - new option *–qc_dir* (specify target QC output directory
  - checks that required programs are on the path at start up
- QC-pipeline/fastq_screen.sh:
  - reduce the default subset size to 10000
  - can now handle Fastqs with *.fq[.gz]* extension
  - new option *–qc_dir* (specify target QC output directory

- bcftbx/Pipeline: *GetFastq[Gz]Files* now also detects *.fq[.gz]* files
- bcftbx/qc/report: 'strip_ngs_extensions' now also handles *.fq[.gz]* files

14.26 Version 1.0.2 (2017-05-12)

- bcftbx/FASTQFile: *FastqIterator* & *FastqRead* updated to handle reads with zero-length sequences
- bcftbx/JobRunner: *GEJobRunner* skips *qacct* call when job is terminated.
- bcftbx/IlluminaData: *IlluminaFastq* updated to handle "index read" (i.e. I1/I2) Fastq file names

14.27 Version 1.0.1 (2017-03-31)

- bcftbx/htmlpagewriter: fix bug writing closing *</head>* tag to HTML documents
- illumina2cluster/prep_sample_sheet.py: move the lane/name parsing functions into *utils* library
- QC-pipeline/fastq_screen.sh: explicitly specify *fastq_screen –force* option to overwrite existing outputs

14.28 Version 1.0.0 (2017-02-23)

- bcftbx/FASTQFile:
  - ***FastqRead* now supports equality operator (==)** to check if two reads are the same.
  - *nreads* function updated to implicitly handle gzipped FASTQs.
- bcftbx/IlluminaData: *duplicated_names* function handles duplicates in IEM samplesheets which don't have an *index* column.
- QC-pipeline/fastq_screen.sh:
  - updated to support *fastq_screen* versions 0.9.*
  - trap for unsupported *–color* option for later versions of *fastq_screen* (0.6.0+)
  - trap for broken *–subset* option in versions 0.6.0-2 of *fastq_screen*

14.29 Version 0.99.15 (2016-10-07)

- bcftbx/IlluminaData: fix bug in *SampleSheetPredictor* class which generated incorrect sample indexes for *bcl2fastq2* output when the sample sheet contained lanes out of order (e.g. 2 appearing before 1).
- bcftbx/IlluminaData: new function *list_missing_fastqs* (returns list of Fastqs predicted from sample sheet which are missing from the output of *CASAVA* or *bcl2fastq*); update *verify_run_against_sample_sheet* to wrap this (functionality should be unchanged).

14.30 Version 0.99.14 (2016-08-31)

- bcftbx/IlluminaData: new class *SampleSheetPredictor* (and supporting classes) for improved prediction of sample sheet outputs; new function *cmp_sample_names* added (use for sorting sample names)
- illumina2cluster/prep_sample_sheet.py 0.4.0: update prediction of outputs and add automatic pagination when run in a terminal window
- QC-pipeline/fastq_screen.sh: updated to handle *fastq_screen* 0.6.* and 0.7.0.
- bcftbx/JobRunner: update *SimpleJobRunner* and *GEJobRunner* classes to capture exit code from the underlying jobs (via *exit_status* property)
- bcftbx/Pipeline: update *Job* class to add new *update* method (checks job status and updates internals) and expose the exit code from the underlying job (as returned via the job runner) via *exit_code* property
- bcftbx/simple_xls: new *save_as_xlsx* method added to *XLSWorkBook* class, to enable output to XLSX format Excel files; new *freeze_panes* function added to *XLSWorkSheet* class
- ChIP-seq/make_macs2_xls.py: default output is now XLSX (use *–format* option to switch back to XLS)

14.31 Version 0.99.13 (2016-08-16)

- bcftbx/IlluminaData: updates to *IlluminaData* and *IlluminaFastq* classes to handle 'non-canonical' FASTQ file names (i.e. names which don't conform to Illumina naming scheme)
- bcftbx/IlluminaData: new function *samplesheet_index_sequence* (extracts barcodes from lines from *SampleSheet* objects)
- Add *HISeq4000* and *MiniSeq* to known platforms in *bcftbx/IlluminaData* and *bcftbx/platforms*.

14.32 Version 0.99.12 (2016-06-30)

- bcftbx/IlluminaData: new 'cycles' property for IlluminaRun class; update SampleSheet class to handle missing '[Data]' section in input file; improvements to IlluminaData class for handling bcl2fastq v2.* outputs.

14.33 Version 0.99.11 (2016-06-09)

- QC-pipeline/fastq_screen.sh: updated to handle output from *fastq_screen* v0.5.2.
- QC-pipeline/prep_sample_sheet.py 0.3.1: new options –set-adapter and –set-adapter-read2 allow updating of adapter sequences specified in IEM sample sheets.
- bcftbx/IlluminaData: new *sample_name_column* property added to the *SampleSheet* class.

14.34 Version 0.99.10 (2016-06-02)

- QC-pipeline/fastq_screen.sh & illumina_qc.sh: new –subset option allows explicit specification of subset size to be passed to fastq_screen (default is still 1000000, use 0 to use all reads as per fastq_screen 0.5.+)

14.35 Version 0.99.9 (2016-05-23)

- bcftbx/utils: fix pretty_print_names function, which was broken if consective sample name prefixes differed but their indices were consecutive.

14.36 Version 0.99.8 (2016-04-05)

- bcftbx/IlluminaData: fixes for IlluminaRun when the target directory doesn't exist; fixes for prediction and verification of IlluminaData against sample sheets for bcl2fastq v2 outputs using –no-lane-splitting option.
- bcftbx/mock: new module with classes for creating "mock" Illumina directories for testing (moved from the unit tests).

14.37 Version 0.99.7 (2016-04-01)

- bcftbx/IlluminaData: fixes for "illegal" name and ID detection and mitigation in IEM samplesheets; fixes to handle of outputs from bcl2fastq v2 in special cases when 'Sample_ID's and 'Sample_Name's are not consistent.

14.38 Version 0.99.6 (2016-01-19)

- Updates for handling sequencing data from NextSeq and bcl2fastq v2:
- bcftbx/IlluminaData: new generic SampleSheet class handles both IEM- and CASAVA-style sample sheets transparently; CasavaSampleSheet and IEMSampleSheet classes reimplemented as wrappers for SampleSheet.
- bcftbx/IlluminaData: IlluminaRun class updated to handle NextSeq output.
- bcftbx/IlluminaData: IlluminaData, IlluminaProject, IlluminaSample and IlluminaFastq classes updated to handle outputs from bcl2fastq v2.
- prep_sample_sheet.py: handles both IEM and CASAVA style sample sheets; use -f/–format option to convert one to the other.

14.39 Version 0.99.5 (2016-01-04)

- extract_reads.py: updated to use a more efficient method for reading data from input files.
- bcftbx/FASTQFile: FastqIterator updated to use a more efficient method for reading data from FASTQ files.
- bcftbx/qc/report: updated to handle special case for Illumina data where the input FASTQ is empty (i.e. has no reads) so there are no QC outputs.

14.40 Version 0.99.4 (2015-11-19)

- changed package name to 'genomics-bcftbx' in setup.py.

14.41 Version 0.99.3 (2015-09-25)

- fetch_fasta.sh: fix bug when MD5 sum failed (e.g. if file was missing)
- extract_reads.py: updated to handle gzipped input files.

14.42 Version 0.99.2 (2015-08-05)

- Porting to Ubuntu: update Python scripts to use '#!/usr/bin/env python' and shell scripts to use '#!/bin/bash'
- bcftbx/TabFile: add switch to TabFile class to prevent type conversions when reading in data
- bcftbx/utils: new function 'get_hostname'.
- NGS-general/split_fasta.py: fixes to handle comments in sequence definition lines.

14.43 Version 0.99.1 (2015-04-16)

- First version which is installable via setup.py
- Significant rearrangement of various scripts and programs
- First version of sphinx-based documentation added
- First version of test scripts for SOLiD and Illumina QC scripts

14.44 Version 2015-02-12

- QC-pipeline/illumina_qc.sh
  - Version 1.2.2
  - Add –threads option (pass number of threads to use to fastq_screen and fastqc)
- QC-pipeline/fastq_screen.sh
  - Add –threads option (pass number of threads to use to fastq_screen command)

14.45 Version 2014-12-10

- utils/cmpdirs.py
  - Version 0.0.1
  - Version 0.0.2
  - Version 0.0.3
  - New program to recursively compare the contents of one directory against another.

14.46 Version 2014-12-04

- build-indexes/make_seq_alignments.sh
  - New script to create sequence alignment (.nib) files from a Fasta file.

14.47 Version 2014-12-03

- utils/symlink_checker.py
  - version 1.1.1
  - Add 'genomics' top-level directory to search path for Python modules.

14.48 Version 2014-10-31

- QC-pipeline/illumina_qc.sh
  - version 1.2.0
  - Default behaviour is not *not* to decompress fastq files, unless new '–ungzip-fastqs' option is specified (and existing option '–no-gzip-fastqs' now does nothing).
  - version 1.2.1
  - Added –version option.

14.49 Version 2014-10-14

- bcftbx/cmdparse.py
  - version 1.0.0
  - New module for creating 'command parsers', for processing command lines of the form 'PROG CMD OPTIONS ARGS'.
- bcftbx/JobRunner.py
  - version 1.1.0
  - New function 'fetch_runner', returns appropriate job runner instance matching text description (used for specifying job runners on command line or in config files).

14.50 Version 2014-10-10

- bcftbx/utils.py
  - version 1.5.0
  - New function 'list_dirs', gets subdirectories of specified parent directory.
- bcftbx/Solid.py
  - Updated 'SolidRun' class to handle cases where the run definition file is missing.

14.51 Version 2014-10-09

- bcftbx/Md5sum.py
  - version 1.1.0
  - **'md5sum' function updated to handle either file name,** or a file-like object opened for reading.
- bcftbx/utils.py
  - version 1.4.8
  - New function 'get_current_user', gets name of user running the program.

14.52 Version 2014-10-08

- bcftbx/utils.py
  - version 1.4.7
  - New property 'resolve_link_via_parent' for PathInfo class, gets 'real' path from one that includes symbolic links at any level.

14.53 Version 2014-09-01

- bcftbx/qc/report.py
  - version 0.99.1
  - relocated QC reporting classes and functions from the qcreporter.py program into a new module in the bcftbx package.
- bcftbx
  - version 0.99.0
  - add a single version for the whole package, accessible using the 'bcftbx.get_version()' function.
- utils/md5checker.py
  - version 0.3.2
  - move unit tests into separate test module & remove –test option.

14.54 Version 2014-08-21

- bcftbx
  - Substantial update: Python library modules from 'share' relocated to 'bcftbx' and turned into a Python package.
  - 'bcf_utils.py' also renamed to 'bcftbx/utils.py'.
  - Python applications also updated to reflect the changes.
- microarray/best_exons.py
  - version 1.2.1

  - new program: averages data for 'best' exons for each gene symbol in a file.

14.55 Version 2014-08-15

- share/JobRunner.py
  - version 1.0.5
  - new 'ge_extract_args' property for GEJobRunner.

14.56 Version 2014-08-11

- share/Md5sum.py
  - version 1.0.1
  - fixed compute_md5sums function to handle broken links

14.57 Version 2014-06-16

- QC-pipeline/illumina_qc.sh
  - version 1.1.1
  - Need to specify the –extract option to work with FastQC
    0.11.2 (should be backwardsly compatible with 0.10.1).
- share/IlluminaData.py
  - version 1.1.5
  - 'get_casava_sample_sheet' needs to handle leading & trailing spaces in barcode sequences.
- share/bcf_utils.py
  - version 1.4.5
  - New function 'walk' traverses directory tree (wrapper for os.walk function).

14.58 Version 2014-06-04

- share/IlluminaData.py
  - version 1.1.4
  - Fix_bases_mask updated to handle situation when a single index sequence is supplied for dual index data.
- illumina2cluster/report_barcodes.py
  - version 0.0.2
  - Make reporting cutoff apply only to exact matches.

14.59 Version 2014-06-02

- illumina2cluster/prep_sample_sheet.py
  - version 0.2.1
  - New options –include-lanes and –truncate-barcodes allow selection of subset of lanes, and barcode sequences to be cut down.

14.60 Version 2014-05-22

- illumina2cluster/report_barcodes.py
  - New program: examine barcode sequences from one or more FASTQ files and report the most prevalent.

14.61 Version 2014-05-15

- utils/manage_seqs.py
  - New program: utility to handle sets of named sequences; intended to help manage custom 'contaminants' files for input into the Brabaham 'FastQC' program.

14.62 Version 2014-05-07

- QC-pipeline/illumina_qc.sh
  - version 1.1.0
  - Optionally use a non-default list of contaminants for FastQC (if specified in the qc.setup file)
  - Create and set a local tmp directory for Java when running FastQC.
  - New –no-gunzip option suppresses creation of uncompressed fastq files.
- share/bcf_utils.py
  - version 1.4.4
  - New functions for getting user and group names and ID numbers from the system.
  - New 'PathInfo' class for getting information about file system paths.
  - Moved symbolic link handling classes and functions in from utils/symlink_checker.py program.
  - 'format_file_sizes' function updated to format to specific units, and able to handle terabyte sizes.
  - new function 'find_program'.
- share/htmlpagewriter.py
  - version 1.0.0
  - New module: HTML page generation functionality relocated from the QC-pipeline/qcreporter.py utility.
- share/IlluminaData.py
  - version 1.1.3

    - Move 'describe_project', 'summarise_projects' and 'verify_run_against_sample_sheet' functions from illumina2cluster/analyse_illumina_run.py into this module.
- share/JobRunner.py
    - version 1.0.4
    - fix broken 'terminate' method for SimpleJobRunner.
    - move set/get of log directory into the BaseJobRunner class.
- share/Md5sum.py
    - Moved Md5Checker and Md5Reporter classes from utils/md5checker.py program.
- share/Pipeline.py
    - version 0.1.3
    - add 'runner' property to Job class (to access associated JobRunner instance).
- share/platforms.py
    - added additional platforms and new function 'list_platforms'
- utils/md5checker.py
    - version 0.3.0
    - substantial refactoring of code to add unit tests; core functions and classes moved to the share/Md5sym.py module.
- utils/symlink_checker.py
    - version 1.1.0
    - refactored to add unit tests and move core functions and classes to share/bcf_utils.
- utils/uncompress_fastqz.sh
    - New utility script for uncompressing fastq files.

14.63 Version 2014-04-17

- ChIP-seq/make_macs2_xls.py
    - version 0.3.2
    - Only sort output on fold enrichment
    - Handle output from –broad option of MACS2
    - Split data over multiple sheets if row limit is exceeded (approx 64k records)
    - Prevent reported command line being truncated if maximum cell size is exceeded (approx 250 characters)
    - Refactored internals to make more robust, added unit tests and switched to use simple_xls module for spreadsheet generation.

14.64 Version 2014-04-10

- RNA-seq/bowtie_mapping_stats.py
    - version 1.1.5

  - Updated to handle paired-end output from Bowtie2

14.65 Version 2014-04-09

- share/simple_xls.py
  - version 0.0.7
  - New methods for inserting and appending columns and rows, which better mimic operations that would be used within a graphical spreadsheet program.
  - Significant updates to handling internal book-keeping to improve performance.

14.66 Version 2014-04-04

- RNA-seq/bowtie_mapping_stats.py
  - version 1.1.3
  - Updated, now works with output from both Bowtie and Bowtie2
- share/simple_xls.py
  - version 0.0.3
  - New module intended to provide a nicer programmatic interface to Excel spreadsheet generation (built on top of Spreadsheet.py).

14.67 Version 2014-02-11

- share/JobRunner.py
  - version 1.0.2
  - SimpleJobRunner: 'join_dirs' option joins stderr to stdout
  - GEJobRunner: jobs in 't' (transferring) and 'qw' (queued-waiting) states counted as "running"
  - GEJobRunner: arbitrary qsub arguments can be specified via 'ge_extra_args' option
- share/SpreadSheet.py
  - version 0.1.8: add support for additional style options ('font_height', 'centre', 'shrink_to_fit')
- share/bcf_utils.py
  - version 1.0.3
  - New function 'find_program' (locate file on PATH)
  - New function 'name_matches' (simple pattern matching for project and sample names, moved from analyse_illumina_data.py)
  - New class 'AttributeDictionary'
  - New class 'OrderedDictionary'
  - New function 'touch' (creates new empty file)
- QC-pipeline/illumina_qc.sh

  - Gunzip fastq.gz files via temporary name, to avoid partial fastqs left behind if script terminates prematurely
  - Write program version information to 'qc' subdirectory
- QC-pipeline/fastq_screen.sh
  - Clean up existing files from previous incomplete run
- QC-pipeline/qcreporter.py
  - version 0.1.1
  - QCSample: 'fastqc' method made into a property
- share/Pipeline.py
  - version 0.1.2
  - Job class: add 'wait' method (waits for job to complete)
  - PipelineRunner: 'max_concurrent_jobs' now applies only to pipeline instance (i.e. not across all pipelines)
  - PipelineRunner: implemented __del__ method to clean up running pipeline instance (i.e. terminate running jobs)
- share/IlluminaData.py
  - version 1.1.2
  - New function 'fix_bases_mask' (adjust bases mask to match actual barcode sequence lengths, for bclToFastq)
- ChIP-seq/make_macs_xls.sh
  - Removed (redundant wrapper script to make_macs_xls.py)
- Unit tests
  - Python unit tests moved into separate files in 'share'

14.68 Version 2013-11-18

- build-indexes/fetch_fasta.sh
  - Neurospora crassa (Ncrassa) updated to June 25th 2013 version.
- build-indexes/bowtie2_build_indexes.sh
  - New: wrapper script to build bowtie2 indexes from a fasta file.
- build-indexes/build_indexes.sh
  - remove bfast indexes & add bowtie2.

14.69 Version 2013-11-15

- build-indexes/fetch_fasta.sh
  - various builds renamed to longer & more accurate names: * hg18 -> hg18_random_chrM * hg19 -> hg19_GRCh37_random_chrM * mm9 -> mm9_random_chrM_chrUn * mm10 -> mm10_random_chrM_chrUn * dm3 -> dm3_het_chrM_chrU * ecoli -> e_coli * dicty -> dictyostelium * chlamyR -> Creinhardtii169

  - updates to broken download URLs and checksums for PhiX, sacBay, ws200 and ws201 genome builds.
  - UniVec updated to build #7.1.

14.70 Version 2013-11-13

- build-indexes/fetch_fasta.sh
  - updated to include sacCer1, sacCer3 and mm10 sequences.
  - updated URL for C. reinhardtii.
  - fixed minor bug in 'fetch_url' function.

14.71 Version 2013-09-11

- share/IlluminaData.py
  - version 1.1.1: update get_casava_sample_sheet function to handle "Experimental Manager"-type sample sheet files when there are no barcode indexes.
- share/JobRunner.py
  - version 1.0.1: fix and standardise handling of log and error files for SimpleJobRunner and GEJobRunner classes; also added minimal unit tests for these classes.

14.72 Version 2013-09-09

- share/FASTQFile.py
  - version 0.3.0: attempt to improve performance of SequenceIdentifier class (use string parsing instead of regular expressions), and added new method 'is_pair_of' (can be used to check if another SequenceIdentifier forms an R1/2 pair with this one). FastqRead class has new attribute 'raw_seqid' (returns original sequence id header supplied on instantiation). New function 'fastqs_are_pair' checks that corresponding read headers match between two FASTQ files.
- illumina2cluster/verify_paired.py
  - version 1.0.0: new utility to check that two fastq files form an R1/R2 pair.
- illumina2cluster/analyse_illumina_run.py
  - version 0.1.11: updated implementation of –merge-fastqs option.
- illumina2cluster/check_paired_fastqs.py
  - Removed: replaced by 'verify_paired.py'.
- share/JobRunner.py
  - version 1.0.1: updates to SimpleJobRunner and GEJobRunner classes (store names associated with each job, and enable lookup via 'name' method; ensure stored log directory is an absolute path, and that log and error file names can be retrieved correctly even if log dir is subsequently changed).

14.73 Version 2013-09-06

- illumina2cluster/analyse_illumina_run.py
  - version 0.1.9: improvements to reporting options when using –summary and –list options.
  - version 0.1.10: fix bug for runs that don't have undetermined indices.
- share/IlluminaData.py
  - version 1.0.2: new method 'fastq_subset' for IlluminaSample (returns subset of fastq files based on read number).

14.74 Version 2013-08-22

- share/bcf_utils.py:
  - version 1.0.1: added new function 'concatenate_fastq_files' (concatenates a list of fastq files).
  - version 1.0.2: updated 'concatenate_fastq_files' to improve performance, and added tests.
- illumina2cluster/analyse_illumina_run.py
  - version 0.1.8: new option –merge-fastqs, creates concatenated fastq files for each sample.
- share/IlluminaData.py
  - version 1.0.1: new property 'full_name' for IlluminaData, (returns name suitable for analysis subdirectory); new function 'get_unique_fastq_names' (generates mapping of full Illumina-style fastq file names to shortest unique version).
- illumina2cluster/build_illumina_analysis_dir.py
  - version 1.0.1: move analysis directory creation code from __main__ to new 'create_analysis_dir' function.
  - version 1.0.2: remove redundant functions and switch to versions in bcf_utils module.

14.75 Version 2013-08-21

- share/bcf_utils.py
  - added baseline version number (1.0.0)
- illumina2cluster/build_illumina_analysis_dir.py
  - added baseline version number (1.0.0)

14.76 Version 2013-08-20

- share/IlluminaData.py, JobRunner.py
  - added version numbers (baseline 1.0.0)
- share/FASTQFile.py
  - version 0.2.6: fix sequence length returned for colorspace reads by FastqRead.seqlen
  - version 0.2.5: added is_colorspace property to FastqRead

14.77 Version 2013-08-19

- illumina2cluster/prep_sample_sheet.py:
  - version 0.2.0: –miseq option is deprecated as it's no longer necessary; sample sheet conversion is performed automatically if required.
- illumina2cluster/IlluminaData.py:
  - new function 'get_casava_sample_sheet' produces a CasavaSampleSheet object from sample sheet CSV file regardless of format. 'convert_miseq_samplesheet_to_casava' is deprecated as it is now just a wrapper to the more genral function.
- share/FASTQFile.py
  - version 0.2.4: added new properties to FastqRead: seqlen (return sequence length), maxquality and minquality (max and min encoded quality scores).

14.78 Version 2013-08-14

- share/FASTQFile.py
  - version 0.2.3: new FastqAttributes class provides access to "gross" attributes of FASTQ file (e.g. read count, file size).
- share/JobRunner.py
  - SimpleJobRunner and GEJobRunner classes allow destination directory for log files to be specified explicitly, and to be changed after instantiation via new 'log_dir' methods.
  - GEJobRunner class has new 'queue' method allowing GE queue to be changed after instantiation.

14.79 Version 2013-08-08

- illumina2cluster/analyse_illumina_run.py
  - version 0.1.7: –summary option generates a one-line description of projects and numbers of samples, suitable for logging file entries.

14.80 Version 2013-08-05

- share/IlluminaData.py
  - new classes IlluminaRun (extracts data from a directory with the "raw" data from a sequencer run) and IlluminRunInfo (extracts data from a RunInfo.xml file).
- share/platforms.py
  - new Python module with utilities and data to identify NGS sequencer platforms
- illumina2cluster/rsync_seq_data.py
  - version 0.0.5: moved sequencer platform identification code to share/platforms.py

  - version 0.0.4: new options –no-log (write rsync ouput directly to stdout) and –exclude (specify rsync filter patterns to exclude files from transfer); explicitly handle keyboard interrupt (i.e. ctrl-C) during rsync operation.

14.81 Version 2013-08-01

- illumina2cluster/rsync_seq_data.py
  - version 0.0.3: added new hiseq sequencer pattern to PLATFORMS.

14.82 Version 2013-07-26

- illumina2cluster/rsync_seq_data.py
  - version 0.0.2: add –mirror option, runs rsync with –delete-after option to remove files from target directory which are no longer present in the source.
- share/Spreadsheet.py
  - version 0.1.7: fixed bug which meant formulae generation failed for columns after 'Z' (i.e. 'AA', 'AB' etc).

14.83 Version 2013-07-19

- ChIP-seq/make_macs2_xls.py
  - modified version of make_macs_xls.py to convert XLS output files from MACS 2.0.10 (contributed by Ian Donaldson).

14.84 Version 2013-07-15

- illumina2cluster/rsync_seq_data.sh
  - removed, replaced by rsync_seq_data.py.
- illumina2cluster/rsync_seq_data.py
  - version 0.0.1: new program for rsync'ing sequencing data to the appropriate location in the archive.
- utils/cluster_load.py
  - new utility for reporting current Grid Engine utilisation by wrapping the qstat program.

14.85 Version 2013-05-21

- illumina2cluster/auto_process_illumina.sh
  - version 0.2.4: use multiple cores for bcl-to-fastq conversion.
- share/IlluminaData.py

  - IlluminaSample class no longer raises an exception if no fastq files are found, so IlluminaData objects can be populated from an incomplete CASAVA run.

- illumina2cluster/build_illumina_analysis_dir.py
  - automatically determine the set of shortest unique link names to use for fastqs in each project.

14.86 Version 2013-05-20

- illumina2cluster/bclToFastq.sh
  - New option –nprocessors allows specification of number of cores to utilise when performing bcl to Fastq conversion.

14.87 Version 2013-05-17

- illumina2cluster/auto_process_illumina.sh
  - version 0.2.3: fix bug with extracting the exit code from the CASAVA/bcl2fastq step.
- share/FASTQFile.py
  - version 0.2.1: implement more efficient line counting in nreads function.
- illumina2cluster/analyse_illumina_run.py
  - version 0.1.4: print results from –stats option in real time.

14.88 Version 2013-05-15

- illumina2cluster/auto_process_illumina.sh
  - version 0.2.2: fix automatic determination of number of allowed mismatches from the bases mask, to deal with e.g. 'I6n'

14.89 Version 2013-05-02

- illumina2cluster/auto_process_illumina.sh
  - version 0.2.1: write log files to "logs" subdirectory.

14.90 Version 2013-05-01

- illumina2cluster/auto_process_illumina.sh
  - version 0.2.0: updated to work with multiple sample sheets.

14.91 Version 2013-04-25

- illumina2cluster/auto_process_illumina.sh
  - version 0.1.0: significant updates to improve robustness, automatically acquire mismatches and generate statistics report.
- ilumina2cluster/analyse_illumina_run.py
  - version 0.1.2: also report file sizes as well as number of reads for Fastq files using –stats option.
- share/bcf_utils.py
  - new function "format_file_size" (converts file size supplied in bytes into human-readable form e.g. 4.0K, 186.0M, 1.6G).

14.92 Version 2013-04-24

- share/bcf_utils.py
  - fix bug in extract_index (failed for names ending with 0 e.g. 'PJB0').

14.93 Version 2013-04-23

- ilumina2cluster/analyse_illumina_run.py
  - version 0.1.1: added –stats option (reports number of reads for each FASTQ file generated by CASAVA's bcl-to-FASTQ conversion).
- share/IlluminaData.py
  - IlluminaData class has new property "undetermined" (allows access to undetermined reads produced by demultiplexing).
  - IlluminaProject.prettyPrintSamples() no longer includes info on paired endedness of the data in the project.

14.94 Version 2013-04-22

- illumina2cluster/auto_process_illumina.sh
  - new script to automate processing of sequencing data from Illumina platforms.

14.95 Version 2013-04-16

- QC-pipeline/run_qc_pipeline.py
  - fix bug with –queue option which meant queue specification was not being honoured by the program.

14.96 Version 2013-04-11

- illumina2cluster/analyse_illumina_run.py
  - version 0.1.0: new option –verify=SAMPLE_SHEET, verifies outputs against those predicted by the named sample sheet.
- share/IlluminaData.py
  - CasavaSampleSheet class:
    1. In "duplicated_names" method, now considers index and lane number as well as SampleID and SampleProject in determining uniqueness.
    2. New method "predict_output", returns a data structure describing the expected project/sample/base file name hierarchy that would be created using the sample sheet.
    3. Added 'paired_end' attribute to the IlluminaData and IlluminaProject classes.
- illumina2cluster/prep_sample_sheet.py
  - version 0.1.0: renamed from 'update_sample_sheet.py'
  - version 0.1.1: print predicted outputs for the input sample sheet.
- illumina2cluster/update_sample_sheet.py
  - renamed to 'prep_sample_sheet.py'
- illumina2cluster/demultiplex_undetermined_fastq.py
  - new program: reassign reads with undetermined index sequences (i.e. barcodes) from the FASTQ files in the 'Undetermined_indices' output directory from CASAVA.

14.97 Version 2013-04-10

- QC-pipeline/qcreporter.py
  - version 0.1.0: added version number, and write this to report header along with date and time of report generation.
  - put the per-base quality boxplot from FastQC into the top-level report.
- share/IlluminaData.py
  - CasavaSampleSheet class: automatically remove double quotes from around sample sheet values upon reading.

14.98 Version 2013-04-09

- share/FASTQFile.py
  - version 0.2.0: added tests, new function "nreads" (counts reads in FASTQ), and enabled FastqIterator to read data from an open file-like object.

14.99 Version 2013-04-08

- share/IlluminaData.py
  - updated IlluminaProject class: allow "Undetermined_indices" dir to also be treated as a "project" within the class framework.
- illumina2cluster/analyse_illumina_run.py
  - added –copy option, to copy specific FASTQ files to pwd.

14.100 Version 2013-04-05

- QC-pipeline/qcreporter.py
  - new –regexp option allows selection of a subset of samples based on regular expression pattern matching e.g. –regexp=SY[1-4]?_trim

14.101 Version 2013-03-13

- share/JobRunner.py
  - update GEJobRunner and DRMAAJobRunner classes to deal with suspended jobs.
- share/FASTQFile.py
  - version 0.1.2: update FastqRead class to operate in a more efficient "lazy" fashion.

14.102 Version 2013-03-07

- utils/fastq_sniffer.py
  - new utility to identify likely FASTQ file format, quality encoding and equivalent Galaxy data type.

14.103 Version 2013-02-19

- utils/extract_reads.py
  - version 0.1.3: fix bug handling fastq files, was confused by quality lines beginning with '#' character.

14.104 Version 2013-02-18

- illumina2cluster/update_sample_sheet.py
  - fix bug in –set-id option which misidentified lanes by their number.

14.105 Version 2013-01-29

- illumina2cluster/update_sample_sheet.py
  - new option –miseq indicates input sample sheet is in MiSeq format, (which will be converted to CASAVA format on output).
- share/IlluminaData.py
  - update convert_miseq_samplesheet_to_casava to handle paired-end MiSeq sample sheet.
  - add new attribute "paired_end" to IlluminaSample objects, to indicate whether the sample has paired end data.
- illumina2cluster/build_illumina_analysis_dir.py
  - deal correctly with linking to paired end Fastq files.

14.106 Version 2013-01-25

- share/IlluminaData.py
  - fix bug in convert_miseq_samplesheet_to_casava (always wrote empty sample sheet).

14.107 Version 2013-01-24

- share/FASTQFile.py
  - version 0.1.0: "casava" format now renamed to "illumina18", for consistency with FASTQ information at http://en.wikipedia.org/wiki/FASTQ_format
  - version 0.1.1: fixed failure to read Illumina 1.8+ files that are missing barcode sequences in the identifier string.

14.108 Version 2013-01-23

- share/IlluminaData.py
  - new class CasavaSampleSheet for handling sample sheet files for input into CASAVA.
  - new function convert_miseq_samplesheet_to_casava for creating CASAVA style sample sheet from one from a MiSEQ sequencer.
- illumina2cluster/update_sample_sheet.py
  - updated to use the CasavaSampleSheet class from IlluminaData.py.

14.109 Version 2013-01-22

- share/FASTQFile.py
  - version 0.0.2: enable FastqIterator to operate on gzipped FASTQ input.

14.110 Version 2013-01-21

- utils/split_fasta.py
  - version 0.1.0: substantial rewrite to enable the core functionality to be unit tested.
- utils/extract_reads.py
  - version 0.1.2: cosmetic updates to comments etc only.

14.111 Version 2013-01-18

- utils/split_fasta.py
  - new utility for splitting Fasta file into individual chromosomes.

14.112 Version 2013-01-14

- QC-pipeline/qcreporter.py
  - new option –verify: reports if all expected outputs from the QC pipeline exist for each sample, to check that the pipeline ran to completion.

14.113 Version 2013-01-10

- QC-pipeline/fastq_stats.sh
  - fix bug in sorting stats file, now header lines should always sort to the top of the file.
- illumina2cluster/analyse_illumina_run.py
  - first version of reporting utility for Illumina data, similar to the "analyse_solid_run.py" in solid2cluster.
- illumina2cluster/build_illumina_analysis_dir.py
  - moved –list and –report functions to new analyse_illumina_data.py utility.
- solid2cluster/analyse_solid_run.py
  - only print paths to primary data files if –report-paths option is specified
  - print timestamps for primary data files along with sample names
  - –quiet option renamed to –no-warnings

14.114 Version 2013-01-09

- illumina2cluster/build_illumina_analysis_dir.py
  - moved classes for handling Illumina data to IlluminaData.py, and take other utility functions from bcf_utils.py
- share/Experiment.py
  - moved utility functions to bcf_utils.py module

- share/IlluminaData.py
  - new Python module containing classes for handling Illumina-based sequencing data, extracted from build_illumina_analysis_dir.py.
- share/bcf_utils.py
  - new Python module containing common utility functions shared between sequencing data modules, extracted from Experiment.py.

14.115 Version 2013-01-07

- illumina2cluster/build_illumina_analysis_dir.py
  - add –report option to pretty print sample names within each project.

14.116 Version 2012-12-06

- NGS-general/boxplotps2png.sh
  - utility to generate PNGs from PS boxplots generated by qc_boxplotter.
- QC-pipeline/qcreporter.py
  - updated to deal with reporting QC for older SOLiD runs which predate filtering (so there are just boxplots and fastq_screens).

14.117 Version 2012-11-27

- QC-pipeline/qcreporter.py
  - added –qc_dir option to specify a non-default QC directory.

14.118 Version 2012-11-26

- illumina2cluster/rsync_seq_data.sh
  - utility script wrapping rsync command for copying arbitrary sequence data directories.
- illumina2cluster/update_sample_sheet.py
  - check for empty sampleID and SampleProject names.
- QC-pipeline/illumina_qc.sh
  - add –nogroup option to FastQC invocation.
  - remove ".fastq" from output log file names when running with fastq.gz input files.
- illumina2cluster/build_illumina_analysis_dirs.py
  - make relative (rather than absolute) symbolic links to source fastq files when building analysis directories.

14.119 Version 2012-11-16

- utils/fastq_edit.py
  - version 0.0.2: added –stats option to generate simple statistics about input FASTQ file.

14.120 Version 2012-11-13

- illumina2cluster/bclToFastq.sh
  - added –nmismatches options (passes number of allowed mismatches to the underlying configureBclToFastq.pl script in CASAVA).

14.121 Version 42012-11-01

- utils/symlink_checker.py
  - new utility for checking and updating (broken) symbolic links.
- QC-pipeline/qcreporter.py
  - added –format option (explicitly specify format of base input files if necessary) and updated automatic platform and data type detection.
- share/Spreadsheet.py
  - version 0.1.6: Workbook class issues warning when appending to an existing XLS file (previously warned when creating a new file)

14.122 Version 2012-10-31

- illumina2cluster/update_sample_sheet.py
  - new option –fix-duplicates automatically deals with duplicated SampleID/SampleProject combinations; using –fix-duplicates and –fix-spaces together should deal with most sample sheet problems without requiring further intervention.

14.123 Version 2012-10-18

- solid2cluster/analyse_solid_run.py
  - –layout option now defaults to 'absolute' links to primary data in generated script.
- solid2cluster/build_analysis_dir.py
  - default is now to make absolute links to primary data files

14.124 Version 2012-10-16

- illumina2cluster/update_sample_sheet.py
  - added –ignore-warnings option (forces output sample sheet file to be written out even if there are errors)

14.125 Version 2012-10-15

- illumina2cluster/bclToFastq.sh
  - added –use-bases-mask option (passes mask specification to the underlying configureBclToFastq.pl script in CASAVA).
- illumina2cluster/build_illumina_analysis_dir.py
  - added new options –keep-names (preserve the full names of the source fastq files when creating links) and –merge-replicates (create merged fastq files for each set of replicates detected).

14.126 Version 2012-10-03

- QC-pipeline/run_qc_pipeline.py
  - added –regexp option to allow filtering of input file names.
- QC-pipeline/solid_qc.sh, illumina_qc.sh
  - write data about underlying QC programs (including versions) to <sample>.programs output files.
- QC-pipeline/qcreporter.py
  - report QC program information from <sample>.programs files (if available).
  - output ZIP file has run/sample-specific top-level directory; HTML report file name restored to 'qc_report.html'.

14.127 Version 2012-10-01

- QC-pipeline/qcreporter.py
  - fixed bug for correctly allocating screens to samples
  - added –platform option to explicitly specify platform type
  - output HTML and ZIP file names now of the form qc_report.<run>.<name>
- solid2cluster/build_analysis_dir.py, illumina2cluster/build_illumina_analysis_dir.py
  - create empty "ScriptCode" subdirectories for each analysis directory, for bioinformaticians to store project-specific scripts and code etc.

14.128 Version 2012-09-28

- utils/md5checker.py
  - version 0.2.3: explicitly report if either of the inputs doesn't exist in -d/–diff mode.

- solid2cluster/log_solid_run.sh
  - renamed to log_seq_data.sh
- illumina2cluster/build_illumina_analysis_dir.py
  - fix bug that resulted in broken links being generated.

14.129 Version 2012-09-24

- solid2clusteranalyse_solid_run.py
  - new option –md5=... generates checksums for specified primary data files (offering more fine-grained control than –md5sum option).

14.130 Version 2012-09-18

- solid2cluster/analyse_solid_run.py
  - new option –gzip=... creates compressed versions of specified primary data files for transfer.
- share/TabFile.py
  - version 0.2.6: TabFile.append and TabFile.insert methods updated to allow arbitrary TabDateLine objects to be added to the TabFile object.

14.131 Version 2012-09-17

- share/SolidData.py
  - add SolidRun.verify method to check run integrity
- solid2cluster/analyse_solid_run.py
  - use SolidRun.verify method to check SOLiD runs

14.132 Version 2012-09-13

- illumina2cluster/update_sample_sheet.py
  - added checks for duplicated SampleID/SampleProject combinations & spaces in names, and refuse to write new SampleSheet containing either of these features.
  - new option –fix-spaces will automatically replace spaces with underscores in SampleID and SampleProject fields.
- illumina2cluster/build_illumina_analysis_dir.py
  - updated to allow for possibility of more than one fastq.gz file per sample directory
  - new option –unaligned=... allows alternative name to be specified for the "Unaligned" subdirectory holding fastq.gz files.
- share/TabFile.py

  - version 0.2.5: implement __nonzero__ built-in for TabDataLine to enable easy test for whether a line is blank.

14.133 Version 2012-09-11

- utils/md5checker.py
  - version 0.2.2: added unit tests (run using –test option); fixed exit code for -d/–diff mode if broken or missing files are encountered.

14.134 Version 2012-08-30

- utils/md5checker.py
  - version 0.2.1: -d/–diff mode now compares files in pairwise fashion; reports "missing" files as part of the total number of files checked; also reports "broken" source files which cannot be checksummed.

14.135 Version 2012-08-24

- share/SolidData.py
  - updates to SolidLibrary allows access to all primary data associated with a sample/library, via new SolidLibrary.primary_data property (which holds a list of SolidPrimaryData objects referencing CSFASTA QUAL file pairs plus timestamp information).
  - added basic support for locating 'unassigned' read files for each sample: each SolidSample object has an associated unassigned SolidLibrary.

14.136 Version 2012-08-23

- share/SolidData.py
  - SolidRun class updated to handle situations where SOLiD run directory names differ from the run names (e.g. because the directory has been renamed)
  - New function 'list_run_directories' gets matching SOLiD run directory names
- solid2cluster/analyse_solid_run.py
  - new option –copy can be used to copy selected primary data files from a run (useful if preparing data for transfer)
- illumina2cluster/build_illumina_analysis_dirs.py
  - new utility to query/build analysis directories for Illumina GA2 sequencing data post bcl-to-fastq conversion

14.137 Version 2012-08-15

- illumina2cluster/update_sample_sheet.py

  - new utility for editing Illumina GA2 SampleSheet.csv files before running bcl to fastq conversion

14.138 Version 2012-08-07

- ChIP-seq/make_macs_xls.py
  - version 0.1.0: fixed to handle output from MACS 1.4.2 (backwards compatible with output from other version of MACS)

14.139 Version 2012-08-03

- QC-pipeline/qcreporter.py
  - new utility to generate HTML reports for SOLiD and Illumina QC script runs

14.140 Version 2012-07-27

- shared/TabFile.py
  - version 0.2.4: allow TabFile.computeColumn() to reference destination columns by integer indices as well as by column name

14.141 Version 2012-07-24

- shared/TabFile.py
  - version 0.2.3: TabFile can now handle user-defined delimiters (not just tabs) for reading and writing; new TabFile.transpose() method converts columns to rows

14.142 Version 2012-07-05

- utils/md5checker.py
  - version 0.1.2: explicitly report missing files separately from checksum failures

14.143 Version 2012-07-02

- RNA-seq/bowtie_mapping_stats.py
  - version 0.1.6: for multiple input files, add the filename to the sample number in the output file

14.144 Version 2012-06-29

- illumina2cluster/bclToFastq.sh
  - Bcl to Fastq conversion wrapper script for Illumina sequencing data
- QC-pipeline
  - new script illumina_qc.sh implements QC pipeline for Illumina data
  - qc.sh renamed to solid_qc.sh

14.145 Version 2012-06-25

- share/TabFile.py
  - version 0.2.1: TabDataLine now preserves the type of non-numeric data items (previously they were automatically converted to strings)

14.146 Version 2012-06-22

- utils/md5checker.py
  - version 0.1.1: reports 'bad' MD5 sum lines; can now handle file names containing whitespace

14.147 Version 2012-06-13

- build-indexes/bowtie_build_indexes.sh
  - added –cs and –nt options (build only color- or nucleotide space indexes)
- build-indexes/fetch_fasta.sh
  - updated UniVec for build 7.0 (Dec. 5 2011)

14.148 Version 2012-06-01

- QC-pipeline/qc.sh
  - updated to run in either 'single end' mode (operate on one F3 or F5 csfasta/qual pair) or 'paired end' mode (operate on F3 csfasta/qual pair plus csfasta/qual F5 pair)
- QC-pipeline/cleanup_qc.sh
  - utility to clean up all QC products from current directory

14.149 Version 2012-05-17

- NGS-general/remove_mispairs.py
  - Python implementation of remove_mispairs.pl works with non-interleaved any fastq

14.150 Version 2012-05-10

- NGS-general
  - New utilities from Ian Donaldson:
  - remove_mispairs.pl: remove "singleton" reads from paired end fastq
  - separate_paired_fastq.pl: separate F3 and F5 reads from fastq
  - trim_fastq.pl: trim down sequences in fastq file from 5' end

14.151 Version 2012-05-09

- microarray/xrothologs.py
  - cross-reference data for two species using probe set lookup

14.152 Version 2012-05-08

- RNA-seq/bowtie_mapping_stats.py
  - summarise statistics from bowtie output into XLS spreadsheet

14.153 Version 2012-05-03

- utils/sam2soap.py
  - first version of SAM to SOAP converter

Python Module Index

b

Index

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z