# tardis.py

Contents

[tardis.py 1](#_Toc423705968)

[Description 2](#_Toc423705969)

[Availability 2](#_Toc423705970)

[Usage 2](#_Toc423705971)

[Passing run parameters via Command-line 3](#_Toc423705972)

[Passing run parameters via .tardishrc configuration file 4](#_Toc423705973)

[Conditioning Directives 4](#_Toc423705974)

[Input conditioning 4](#_Toc423705975)

[Output conditioning 5](#_Toc423705976)

[Product conditioning 6](#_Toc423705977)

[Throughput conditioning 6](#_Toc423705978)

[Wait conditioning 6](#_Toc423705979)

[Filtering and editing records 7](#_Toc423705980)

[Conditioning Examples 7](#_Toc423705981)

[No conditioning 7](#_Toc423705982)

[Executing a custom fastq processing script 7](#_Toc423705983)

[Executing a blast of a random sample of sequences from a compressed fastq file 7](#_Toc423705984)

[Executing a blast of a random sample of sequences from a compressed fastq file, generating XML output for import into MEGAN 8](#_Toc423705985)

[Executing a blast with customised tabular output – escaping quotes 8](#_Toc423705986)

[Running hmmscan, using a custom job template 8](#_Toc423705987)

[Quickly Sub-sampling a fasta file using a list of sequence id’s 9](#_Toc423705988)

[Product conditioning 9](#_Toc423705989)

[Product conditioning with output basename specified by a command line arg 10](#_Toc423705990)

[Product conditioning with an output folder specified by a command line arg 11](#_Toc423705991)

[Random sampling of (paired ) fastq files 12](#_Toc423705992)

[Several outputs, including stdout redirect 12](#_Toc423705993)

[Executing a short chain of commands – throughput conditioning 13](#_Toc423705994)

[List file processing 14](#_Toc423705995)

[Piping blast results to a protein translation utility (referring to an input file twice, and avoiding intermediate files) 14](#_Toc423705996)

[Wait conditioning - using tardis to launch a job on a cluster and wait for the output 15](#_Toc423705997)

[Using tardis in a make-file to implement a simple sequence processing pipeline 15](#_Toc423705998)

[Filtering records on the fly 16](#_Toc423705999)

[Implementation 17](#_Toc423706000)

[Overview 17](#_Toc423706001)

[Related Utilities 17](#_Toc423706002)

[tardis\_analyze.py 17](#_Toc423706003)

[summarise\_text.py 19](#_Toc423706004)

[Appendix 1 19](#_Toc423706005)

[Condor job file template example 19](#_Toc423706006)

[Appendix 2 20](#_Toc423706007)

[Galaxy wrapper modification example (fastx\_trimmer.xml) 20](#_Toc423706008)

## Description

tardis.py *conditions* a command for execution on a cluster. *Conditioning* a command means re-writing it to refer to conditioned input and output files. Conditioning an input file, means splitting it into chunks (and may include optional inline filtering and editing of records, before they are written to chunks) - each chunk is a conditioned input file, and yields a conditioned output file when the conditioned command is executed on the cluster. Once all of the conditioned commands have completed, the conditioned output is *unconditioned*, which means that it is merged together to yield the output file that was referred to by the original unconditioned command.

To indicate how a command should be conditioned, it is marked up with conditioning directives. For example, a command which runs a script to process a fastq input file and generate a fastq output file, such as

Perl split\_pairEnds.pl L01\_BWA\_Trimmed.fastq L01\_BWA\_Trimmed\_pair1.fastq

might be marked up as

Perl split\_pairEnds.pl *\_condition\_fastq\_input\_*L01\_BWA\_Trimmed.fastq *\_condition\_fastq\_output\_*L01\_BWA\_Trimmed\_pair1.fastq

and run on the cluster using tardis

*tardis.py –w –c 100000* Perl split\_pairEnds.pl *\_condition\_fastq\_input\_*L01\_BWA\_Trimmed.fastq *\_condition\_fastq\_output\_*L01\_BWA\_Trimmed\_pair1.fastq

tardis has been developed using a condor based cluster.

## Availability

Tardis is available from <https://bitbucket.org/agr-bifo/tardis>

## Usage

### Passing run parameters via Command-line

tardis.py [-w] [-c Chunksize] [-s SampleRate] [-from record\_number] [-to record\_number] [-d workingRootPath] [-k] [-v] any command (with optional conditioning directives)

-w Run the command as part of a workflow. After launching all of the jobs, tardis waits for all outputs, which are then collated and merged into a single output file, as specified by the output file path in the original command; all of the temporary input files (for example chunks of uncompressed fastq) are deleted provided all prior steps completed without error (if there was an error they are left there to assist with debugging). Without this option, the program exits immediately after launching all of the jobs, and output is left un-collated in the scratch folder created by this script, and no cleanup is done.

-c ChunkSize When conditioning the input file(s), split into files each containing Chunksize logical records. (A logical record for a sequence file is a complete sequence. For a text file it is a line of text). (If the -s option is used to sample the inputs, the chunksize relates to the full un-sampled file – so the same chunk-size can be used whether random sampling or not. For example a chunksize of 1,000,000 is specified in combination with a sampling rate of .0001, then each chunk would contain 100 sequences – i.e. you should not adjust the chunk-size, for the sampling rate. Note that to avoid a race-condition that could be caused by a very small chunk-size resulting in launching a very large number of jobs, tardis will throw an exception if the chunk-size used would result in launching more than MAX\_DIMENSION jobs (currently 5000) )

-from record\_number When conditioning the input file(s), only use records from the input file after or including record\_number (where that is logical record number – e.g. in a fastq file, start from record number N means start from sequence N). By combining this option with -to, you can process slices of a file. Note that this option has no affect when processing a list-file.

-to record\_number When conditioning the input file(s), only use records up to and including the record record\_number (where that is logical record number – e.g. in a fastq file, process up to record number N means process up to and including sequence N). By combining this option with -from, you can process slices of a file. Note that this option has no affect when processing a list-file.

-s SampleRate Rather than process the entire input file(s), a random sample of the records is processed. SampleRate is the probability that a given record will be sampled. For example -s .001 will result in roughly 1 in every 1000 logical records being sampled. When the -s option is specified, tardis does not clean up the conditioned input and output – e.g. all of the uncompressed fastq sample fragments would be retained. These are retained to assist with the Q/C work that is normally associated with a sampled run. Paired fastq input files are sampled in lock-step, provided the paired fastq conditioning directive is used for both files.

-d workingRootPath create the tardis working folder under workingRootPath. If no working root is specified, a default location is used.

-v validate the run by doing a dry run. This means that the chunks, job scripts and job files etc. are all generated but the jobs are not launched. The user can start then kill (CTRL-C) the run, inspect the script and job files that were generated to check that their command has been conditioned as envisaged.

-k keep the conditioned input and output - i.e. the input and output fragments. Normally in workflow mode these are deleted after the output is successfully "unconditioned" - i.e. joined back together

-t filename optionally supply a condor job template - tardis will read the contents of filename and use this as the job template.

-hpctype value optionally indicate the hpc environment. Currently the only supported values are :

default which results in tardis attempting to set up and launch condor jobs;

local which results in each job being launched by tardis itself on the local machine, using the native python sub-process API. The maximum number of processes it will run at a time is controlled by a global variable in the script MAX\_PROCESSES, which is initially 10.

If the –hpctype option is not used, tardis assumes default and will try to set up and launch condor jobs.

-batonfile value if you supply a "baton file" name, tardis will write the process exit code to this file after all processing has completed. This can be useful to preserve synchronous execution of a workflow, even if tardis is started in the background - the workflow can test the existence of the batonfile - if it exists then the corresponding tardis processing step has completed - i.e. this is another way of a tardis based step in a workflow "passing the baton" to the next step.

-h print usage and exit.

### Passing run parameters via .tardishrc configuration file

You can also pass run-time parameters by specifying these in a .tardishrc file. Tardis will search for a configuration file in the current working folder, and also in /etc/tardis

Example:

|  |
| --- |
| [tardis\_engine] # filter sequences – minimum length 200 seqlength\_min=200  # take 1 in 1000 random sample of the input  samplerate=.001  # job template to use job\_template\_name=condor\_job  #..or specify a filename, containing a custom job definition jobtemplatefile=/somewhere/my\_template.txt  #specify a shell template that uses bash shell\_template\_name=condor\_bash\_shell  # you can supply a list of valid command patterns to restrict which commands # the engine will execute – e.g. if tardis is running as a server  valid\_command\_patterns=cat awk [t]\*blast[nxp] bwa bowtie flexbar  # by default tardis creates uniquely named working folders under the  # root folder specified by the –d argument. In some cases you might  # want to specify the working folder name yourself. To do that set # workdir\_is\_rootdir to True – tardis will then use the named  # folder specified by the –d argument as its working folder workdir\_is\_rootdir=False  # by default tardis uses a fast C based program (kseq\_split) to condition # fasta and fastq files. To use the slower biopython based method  #( for example if you want to include a sequence length filter) # use the following configuration [tardis\_engine] fast\_sequence\_input\_conditioning=False |

## Conditioning Directives

### Input conditioning

Input conditioning directives are prefixed to command arguments corresponding to input filenames. Input files that are prefixed with conditioning directives will

* If necessary be uncompressed (e.g. fastq.gz 🡪 fastq)
* If necessary be sampled randomly
* If necessary be format-converted (e.g. fastq 🡪 fasta) as implied by the directive
* Split into chunks

and the original command will be rewritten to refer to the conditioned files containing the conditioned data.

The following input conditioning directives are available

\_condition\_fastq\_input\_ optionally uncompress and then split a fastq file into fastq fragments

\_condition\_fasta\_input\_ optionally uncompress and then split a fasta file into fasta fragments

\_condition\_fastq2fasta\_input\_ optionally uncompress and then split a fastq file into fasta fragments

\_condition\_paired\_fastq\_input\_ optionally uncompress and then split one of two pair-end matched fastq files into pair-end matched fastq fragments, enforcing integrity of pair-end matching by name

\_condition\_text\_input\_ optionally uncompress and then split a text file

### Output conditioning

Output conditioning directives are prefixed to command arguments corresponding to output filenames. Output filenames that are prefixed with conditioning directives will be re-written into the conditioned command as conditioned filenames, so that conditioned input files yield output that is written to conditioned output filenames. Conditioned output filenames are obtained by adding chunk numbers to the original output filename.

At the conclusion of the run – i.e. when all of the conditioned commands have finished executing on the cluster – tardis will “uncondition” the output, which means joining all of the output fragments together, to yield the single output file referred to by the original command. This is also usually then automatically compressed.

The following output conditioning directives are available:

\_condition\_fastq\_output\_

\_condition\_fasta\_output\_

\_condition\_uncompressedfastq\_output\_ as above but don’t compress the concatenated output

\_condition\_uncompressedfasta\_output\_ as above but don’t compress the concatenated output

\_condition\_text\_output\_ generate conditioned output filenames corresponding to the prefixed output file. At the end of the run concatenate the output fragments into the prefixed filename and compress. (All of these directives result in simple text concatenation of the output and could be used interchangeably at the moment)

\_condition\_uncompressedtext\_output\_ as above but don’t compress the concatenated output

\_condition\_sam\_output\_ generate conditioned output filenames corresponding to the prefixed output file. At the end of the run compress each SAM output file chunk to sorted (by sequence name) BAM and merge all of the resulting BAM fragments into a single output BAM file

\_condition\_uncompressedsam\_output\_ as for \_condition\_sam\_output\_ but final output is SAM not BAM

\_condition\_headlesssam\_output\_ as for \_condition\_sam\_output\_ but final output does not include the SAM header. Note that the unconditioned SAM output generated by the command when run on each chunk should include the SAM header, as this is used to help check the integrity of the job. For example the header of each SAM output chunk is compared, and an error flagged if they are not identical (excluding the @PG record, which may be different for different chunks).

\_condition\_pdf\_output\_ generate conditioned output filenames corresponding to the prefixed output file. At the end of the run combine the pdf fragments (using ghostscript), and compress the final results

\_condition\_blastxml\_output\_ generate conditioned output filenames corresponding to the prefixed output file. At the end of the run combine the blast XML fragments into a single valid blast XML output

\_condition\_output\_ generate conditioned output command line tokens, but don’t attempt to uncondition any output. These tokens can be referenced by product conditioning mark-up – see the flexbar example below.

\_condition\_wait\_output\_ this causes tardis to wait for the requested command to complete, and check that the output exists, but no further processing is done. This option can be used for example if you simply want to use tardis to launch a command on a cluster and wait for the job to finish.

### Product conditioning

Some applications do not let you specify output filename(s), or generate output that you can redirect, and write their output to files with auto-generated names. These types of output are here termed “products” and are handled by product conditioning directives. Product conditioning directives are appended as new arguments to the command to be executed, rather than appended to output filenames as are output conditioning directives. Product conditioning directives specify how any products of the command are to be unconditioned - i.e. collated and merged – at the end of the run. (). Product conditioning directives include sub-expressions which are used by tardis to find the conditioned products, and to determine the name of the final target file into which the conditioned products from each chunk are to be merged.

There is a product conditioning directive corresponding to each output conditioning directive.

See examples of product conditioning below

### Throughput conditioning

Throughput conditioning directives are used where a chain of commands is conditioned, and output from one command is input to the next. Output that is used as input to a subsequent command in a chain is here termed throughput.

There is a throughput conditioning directive corresponding to each output conditioning directive.

See examples of throughput conditioning below

### Wait conditioning

Wait conditioning directives are used where you do not necessarily want to split up your input, but rather you want to launch a command on a cluster and then wait for it to complete – for example as part of a workflow

See examples of wait conditioning below.

### Filtering and editing records

There are often situations where you want to filter or edit records before they are processed. For example reject short records (filtering); reformatting fastq sequence names for compatibility with parsing requirements of a tool (editing).

Rather than create filtered or edited copies of the original data files, you can configure tardis to filter and edit data as part of conditioning.

See examples of filtering and editing as part of conditioning, below

## Conditioning Examples

Each example shows first the original command, then the command marked up with conditioning directives and typical tardis options.

### No conditioning

|  |
| --- |
| ls -l /dataset/resequencing/data |
| tardis.py ls -l /dataset/resequencing/data |

(This could be used as a convenient way to run a command on a single node of the cluster)

### Executing a custom fastq processing script

|  |
| --- |
| perl split\_pairEnds\_3\_files.pl 1049L\_both\_with\_no\_primers\_or\_adapters\_BWA\_Trimmed.fastq 1049L\_both\_with\_no\_primers\_or\_adapters\_BWA\_Trimmed\_pair1.fastq 1049L\_both\_with\_no\_primers\_or\_adapters\_BWA\_Trimmed\_pair2.fastq 1049L\_both\_with\_no\_primers\_or\_adapters\_BWA\_Trimmed\_single.fastq |
| *tardis.py -w -c 100000 -d /dataset/RNA-Seq/scratch* perl split\_pairEnds\_3\_files.pl *\_condition\_fastq\_input\_*1049L\_both\_with\_no\_primers\_or\_adapters\_BWA\_Trimmed.fastq *\_condition\_fastq\_output\_*1049L\_both\_with\_no\_primers\_or\_adapters\_BWA\_Trimmed\_pair1.fastq *\_condition\_fastq\_output\_*1049L\_both\_with\_no\_primers\_or\_adapters\_BWA\_Trimmed\_pair2.fastq *\_condition\_fastq\_output\_*1049L\_both\_with\_no\_primers\_or\_adapters\_BWA\_Trimmed\_single.fastq |

### Executing a blast of a random sample of sequences from a compressed fastq file

|  |
| --- |
| first would need to uncompress and sample the input file manually – then execute something like  blastn -query L01.fastq.sampled.fa -num\_threads 2 -db nt -evalue 1.0e-10 -max\_target\_seqs 1 -outfmt 7 -out contamination\_check.txt |
| tardis handles uncompression and sampling the original compressed file  *tardis.py -w -c 1000000 –s .0001* blastn –query *\_condition\_fastq2fasta\_input\_*L01.fastq.gz -num\_threads 2 -db nt -evalue 1.0e-10 -max\_target\_seqs 1 -outfmt 7 -out *\_condition\_text\_output\_*contamination\_check.txt |

### Executing a blast of a random sample of sequences from a compressed fastq file, generating XML output for import into MEGAN

|  |
| --- |
| first would need to uncompress and sample the input file manually – then execute something like  blastn –query /dataset/JHI\_High\_Low\_Sequencing\_Data/L01\_filtered\_sampled.fastq -query\_gencode 11 -db nt -outfmt 5 -evalue .00001 -out \_condition\_blastxml\_output\_L01\_sample.xml |
| tardis handles uncompression, sampling the original compressed file and merging the XML chunks  *tardis.py -c 1500000 –s .0001 -w blastn -query \_condition\_fastq2fasta\_input\_*/dataset/JHI\_High\_Low\_Sequencing\_Data/L01\_filtered.fastq.gz -query\_gencode 11 -db nt -outfmt 5 -evalue .00001 -out *\_condition\_blastxml\_output\_*/dataset/JHI\_High\_Low\_Sequencing\_Data/scratch/ L01\_sample.xml |

### Executing a blast with customised tabular output – escaping quotes

|  |
| --- |
| This uses a custom outfmt string  blastx –query /dataset/Deer\_Transcriptomics\_Resources/scratch/deerTranscript/deerTranscript\_and\_mRNA-seq\_contigs.fa -num\_threads 4 -db uniprot\_sprot.fa -evalue 1.0e-6 -outfmt "7 qseqid qstart qend qframe sacc stitle" -out deer.out |
| Note the need to escape the quotes  *tardis.py -w -c 1000 -d /dataset/Deer\_Transcriptomics\_Resources/scratch/tardis blastx -query \_condition\_fasta\_input\_/dataset/Deer\_Transcriptomics\_Resources/scratch/deerTranscript/deerTranscript\_and\_mRNA-seq\_contigs.fa -num\_threads 4 -db uniprot\_sprot.fa -evalue 1.0e-6 -outfmt \"7 qseqid qstart qend qframe sacc stitle\" -out \_condition\_text\_output\_deer.out* |

### Running hmmscan, using a custom job template

|  |
| --- |
| Command has two outputs  hmmscan -o vel98.21.hmm.tout --tblout vel98.21.tblout --noali Pfam-A.hmm /dataset/Sand\_Scarab\_Gut\_Transcriptome/active/rRNA/vel98.21.prot |
| Multiple outputs are fine. Note again that we can start with compressed sequence data. Here we include a custom job template (for example we may want to exclude a specific machine for some reason)  *tardis.py -c 50 -d /dataset/Sand\_Scarab\_Gut\_Transcriptome/scratch -w -t custom\_job\_template.dat* hmmscan -o *\_condition\_text\_output\_*vel98.21.hmm.tout --tblout *\_condition\_text\_output\_*vel98.21.tblout --noali Pfam-A.hmm *\_condition\_fasta\_input\_*/dataset/Sand\_Scarab\_Gut\_Transcriptome/active/rRNA/vel98.21.prot.gz |

### Quickly Sub-sampling a fasta file using a list of sequence id’s

|  |
| --- |
| The original command uses a simple bio-python script to filter a fasta file, to output only those sequences whose id’s are in a list file.  cat allseqs.fasta | fasta\_yank.py gg\_13\_5\_taxonomy.nonarchaea.names > selectedseqs.fasta  Where fasta\_yank.py is simply :  #!/usr/bin/python2.6 # filters a fasta file (from standard input), pulling out sequences by name #  from Bio import SeqIOimport sys usage = "usage : cat some\_file.fa | fasta\_yank.py file\_of\_names "  if len(sys.argv) != 2:  print usage  sys.exit(1)  SeqIO.write ((r for r in SeqIO.parse(sys.stdin, "fasta") if r.name in [name.strip() for name in open(sys.argv[1],"r")]) , sys.stdout, "fasta")  ….and gg\_13\_5\_taxonomy.nonarchaea.names just contains sequence ids – e.g.  228054 844608 178780 198479 187280 etc |
| The original is very slow. The following marked up command splits the list file (not the sequence file), and launches jobs to filter the entire sequence file through chunks of the list file. This is much faster.  *tardis.py -w -c 100 -d /dataset/scratch/tmp cat allseqs.fasta* *\| fasta\_yank.py \_condition\_text\_input\_gg\_13\_5\_taxonomy.nonarchaea.names \> \_condition\_fasta\_output\_selectedseqs.fasta* |

### Product conditioning

|  |
| --- |
| This script writes 3 output files, with names constructed from the input file, as well as writing some stats to stdout  /usr/bin/DynamicTrim.pl -probcutoff 0.05 AD1AAFACXX\_L001\_R2\_001.fastq |
| There are no DynamicTrim command-line options available to specify the filenames or redirect the output, so output conditioning is not possible. Product conditioning directives include sub-expressions which are used by tardis to find and collate the conditioned output. Here the sub-expressions indicate that the conditioned product filenames are obtained by appending the indicated suffices to the conditioned input filenames, and that these are to be collated into the filename that appears after the comma in the sub-expression. The conditioning directive prefixed to the redirected stdout means that the stdout of each chunk will be written to conditioned filenames AD1AAFACXX\_L001\_R2\_001\_trimming\_stdout.1, AD1AAFACXX\_L001\_R2\_001\_trimming\_stdout.2, etc, and then these will be concatenated to yield a final stdout file AD1AAFACXX\_L001\_R2\_001\_trimming\_stdout.gz (merged output is compressed by default). Note that the shell redirect character needs to be escaped, so that it is passed through to tardis to be included in the command conditioning ( rather than interpreted by the shell being used to run the tardis command).  *tardis.py -w -c 500000* /usr/bin/DynamicTrim.pl -probcutoff 0.05 \_condition\_fastq\_input\_AD1AAFACXX\_L001\_R2\_001.fastq \>\_condition\_text\_output\_ AD1AAFACXX\_L001\_R2\_001\_trimming\_stdout *\_condition\_uncompressedfastq\_product\_.trimmed,AD1AAFACXX\_L001\_R2\_001\_trimmed.fastq \_condition\_uncompressedtext\_product\_.trimmed\_segments,AD1AAFACXX\_L001\_R2\_001\_segments.txt \_condition\_uncompressedpdf\_product\_.trimmed\_segments.hist.pdf, AD1AAFACXX\_L001\_R2\_001\_segments\_histogram.pdf* |

|  |
| --- |
| Another example of product conditioning  /usr/bin/bismark -q --phred33-quals -n 1 -l 50 -I 50 -X 350 --path\_to\_bowtie /usr/bin/ /bifo/active/Run938\_5/Rachael/oarv3\_conversion -1 /bifo/active/Run938\_5/Rachael/C279BACXX\_NZGL00461\_423\_250\_NoIndex\_L003\_R1.fastq\_Qtrimmed -2 /bifo/active/Run938\_5/Rachael/C279BACXX\_NZGL00461\_423\_250\_NoIndex\_L003\_R2.fastq\_Qtrimmed |
| In this case bismark would generate output filenames itself, as follows  C279BACXX\_NZGL00461\_423\_250\_NoIndex\_L003\_R1.fastq\_Qtrimmed\_bismark\_pe.sam C279BACXX\_NZGL00461\_423\_250\_NoIndex\_L003\_R1.fastq\_Qtrimmed\_Bismark\_paired-end\_mapping\_report.txt  Product conditioning directives are used, so that tardis can find the output files for each chunk and join them back together to yield target filenames. As in the previous example, the product conditioning is of the form \_condition\_outputtype\_product\_***expected suffix,output product file name***. This directs tardis to look for output products from each input chunk named *input\_chunk\_name.****expected suffix***, and join these together to produce ***output product file name.*** Where, as in this case, there is more than one input file, then (for this product conditioning directive), tardis expects the output product file names to be based on the chunks arising from the first input file encountered in the command .  *tardis.py -w -c 500000* /usr/bin/bismark -q --phred33-quals -n 1 -l 50 -I 50 -X 350 --path\_to\_bowtie /usr/bin/ /bifo/active/Run938\_5/Rachael/oarv3\_conversion -1 *\_condition\_paired\_fastq\_input\_*/bifo/active/Run938\_5/Rachael/C279BACXX\_NZGL00461\_423\_250\_NoIndex\_L003\_R1.fastq\_Qtrimmed -2 *\_condition\_paired\_fastq\_input\_*/bifo/active/Run938\_5/Rachael/C279BACXX\_NZGL00461\_423\_250\_NoIndex\_L003\_R2.fastq\_Qtrimmed *\_condition\_text\_product\_\_Bismark\_paired-end\_mapping\_report.txt,* *C279BACXX\_NZGL00461\_423\_250\_NoIndex\_L003\_Bismark\_paired-end\_mapping\_report.txt* *\_condition\_uncompressedsam\_product\_\_bismark\_pe.sam,* *C279BACXX\_NZGL00461\_423\_250\_NoIndex\_L003\_bismark\_pe.sam*  - so that the final unconditioned (i.e. joined-back-together) output from this command will be  C279BACXX\_NZGL00461\_423\_250\_NoIndex\_L003\_Bismark\_paired-end\_mapping\_report.txt C279BACXX\_NZGL00461\_423\_250\_NoIndex\_L003\_bismark\_pe.sam |

### Product conditioning with output basename specified by a command line arg

|  |
| --- |
| This utility (used to trim sequence reads) allows you to specify an output base-name which is then used to construct a number of output file names all with that base-name, but with different suffixes. For example  flexbar --threads 4 --reads /home/bloggs/fbproject/reads1.fastq --reads2 /home/bloggs/fbproject /reads2.fastq --format sanger --max-uncalled 2 --min-read-length 35 --pre-trim-phred 20 --/home/bloggs/adapters/adapters.txt --adapter-trim-end ANY --adapter-min-overlap 4 --adapter-threshold 3 --single-reads --length-dist --target flexbartag > fbrun.stdout  this tells the utility to use flexbartag as the basename. For these options, the utility will thus generate the following output files :  flexbartag\_1.fastq flexbartag\_1\_single.fastq flexbartag\_1.fastq.lengthdist flexbartag\_2.fastq flexbartag\_2\_single.fastq flexbartag\_2.fastq.lengthdist fbrun.stdout |
| The conditioning directive \_condition\_output\_somelabel , will cause conditioned command-line arguments somelabel.1 somelabel.2 etc to appear in each conditioned command, however no output processing will be invoked. These conditioned labels can then be referenced by product conditioning directives like  \_condition\_text\_product\_{}.suffix,finalfile.dat  When used together with the \_condition\_output\_ directive, this directive means that at the end of the run tardis will expect to find conditioned output files somelabel.1.suffix, somelabel.2.suffix etc, and will join these together to yield a final unconditioned output file finalfile.dat  The marked-up version of the above command is thus for example  tardis.py -w -c 1000000 -d /data/working/myproject flexbar --threads 4 --reads \_condition\_paired\_fastq\_input\_/home/bloggs/fbproject/reads1.fastq --reads2 \_condition\_paired\_fastq\_input\_/home/bloggs/fbproject/reads2.fastq --format sanger --max-uncalled 2 --min-read-length 35 --pre-trim-phred 20 --adapters /home/bloggs/adapters/adapters.txt --adapter-trim-end ANY --adapter-min-overlap 4 --adapter-threshold 3 --single-reads --length-dist --target \_condition\_output\_flexbartag \> \_condition\_uncompressedtext\_output\_/home/bloggs/fbproject/output/fbrun.stdout \_condition\_fastq\_product\_{}\_1.fastq, /home/bloggs/fbproject/output/reads\_1.fastq \_condition\_fastq\_product\_{}\_2.fastq, /home/bloggs/fbproject/output/reads\_2.fastq \_condition\_fastq\_product\_{}\_1\_single.fastq, /home/bloggs/fbproject/output/reads\_1\_single.fastq \_condition\_fastq\_product\_{}\_2\_single.fastq, /home/bloggs/fbproject/output/reads\_2\_single.fastq \_condition\_uncompressedtext\_product\_{}\_1.fastq.lengthdist, /home/bloggs/fbproject/output/reads\_1.fastq.lengthdist \_condition\_uncompressedtext\_product\_{}\_2.fastq.lengthdist, /home/bloggs/fbproject/output/reads\_2.fastq.lengthdist |

### Product conditioning with an output folder specified by a command line arg

|  |
| --- |
| Tophat2 allows you to specify an output folder, into which it writes a number of output files (if you don’t specify a folder, it uses a default of “tophat\_out”). Example:  tophat2 -o th2temp --num-threads 4 -r 300 --mate-std-dev=20 /dataset/datacache/oar31/bowtie2\_index/oar31 dataset\_9767.fastq dataset\_9768.fastq  (where oar31 is the location of a reference genome, and the two fastq arguments are paired-end files)  This will write the following output files to the sub-folder th2temp:  accepted\_hits.sam align\_summary.txt deletions.bed insertions.bed junctions.bed logs prep\_reads.info unmapped.bam |
| As above, the conditioning directive \_condition\_output\_somelabel, will cause conditioned command-line arguments somelabel.1 somelabel.2 etc to appear in each conditioned command, however no output processing will be invoked. These conditioned labels can then be referenced by product conditioning directives like  \_condition\_text\_product\_{}/outputfile.dat,finalfile.dat  When used together with the \_condition\_output\_ directive, this directive means that at the end of the run tardis will expect to find conditioned sub-folder names somelabel.1, somelabel.2 etc, each containing a file outputfile.dat, and will join these together to yield a final unconditioned output file finalfile.dat  The marked up version of the above command is thus for example  tardis.py -w -c 1000000 -d /data/working/myproject tophat2 -o \_condition\_output\_th2temp --num-threads 4 -r 300 --mate-std-dev=20 /dataset/datacache/oar31/bowtie2\_index/oar31 \_condition\_paired\_fastq\_input\_dataset\_9767.fastq \_condition\_paired\_fastq\_input\_dataset\_9768.fastq \_condition\_bam\_product\_{}/accepted\_hits.bam,accepted\_hits.bam \_condition\_uncompressedtext\_product\_{}/deletions.bed,deletions.bed \_condition\_uncompressedtext\_product\_{}/insertions.bed,insertions.bed \_condition\_uncompressedtext\_product\_{}/junctions.bed,junctions.bed \_condition\_uncompressedtext\_product\_{}/prep\_reads.info,prep\_reads.info \_condition\_bam\_product\_{}/unmapped.bam,unmapped.bam  If you want to generate SAM output, use the tophat2 --no-convert-bam option, in conjunction with the tardis uncompressed SAM conditioning directive :  tardis.py -w -c 1000000 -d /data/working/myproject tophat2 --no-convert-bam -o \_condition\_output\_th2temp --num-threads 4 -r 300 --mate-std-dev=20 /dataset/datacache/oar31/bowtie2\_index/oar31 \_condition\_paired\_fastq\_input\_dataset\_9767.fastq \_condition\_paired\_fastq\_input\_dataset\_9768.fastq \_condition\_uncompressedsam\_product\_{}/accepted\_hits.sam,accepted\_hits.sam \_condition\_uncompressedtext\_product\_{}/deletions.bed,deletions.bed \_condition\_uncompressedtext\_product\_{}/insertions.bed,insertions.bed \_condition\_uncompressedtext\_product\_{}/junctions.bed,junctions.bed \_condition\_uncompressedtext\_product\_{}/prep\_reads.info,prep\_reads.info \_condition\_bam\_product\_{}/unmapped.bam,unmapped.bam  (Note that tophat2 appears to always generate BAM format for the unmapped reads even when --no-convert-bam is specified - hence we have still used the tardis BAM conditioning directive to handle the unmapped reads output) |

### Product conditioning using a regular expression to specify output

|  |
| --- |
| Tophat2 allows you to specify an output folder, into which it writes a number of output files (if you don’t specify a folder, it uses a default of “tophat\_out”). Example:  tophat2 -o th2temp --num-threads 4 -r 300 --mate-std-dev=20 /dataset/datacache/oar31/bowtie2\_index/oar31 dataset\_9767.fastq dataset\_9768.fastq  (where oar31 is the location of a reference genome, and the two fastq arguments are paired-end files)  This will write the following output files to the sub-folder th2temp:  accepted\_hits.sam align\_summary.txt deletions.bed insertions.bed junctions.bed logs prep\_reads.info unmapped.bam |
| tardis.py -w -c 200000 -d /dataset/AG\_1000\_sheep/scratch/general\_processing\_062015/NZCMPF100016647269 -batonfile /dataset/AG\_1000\_sheep/scratch/general\_processing\_062015/NZCMPF100016647269/C6KHFANXX-1418-36-05-01\_ATTCCT\_L005\_R1\_001.baton /dataset/hiseq/active/bin/quadtrim -d sheep \_condition\_paired\_fastq\_input\_/dataset/BLGsheep/archive/NZGL01418\_2/Raw/C6KHFANXX-1418-36-05-01\_ATTCCT\_L005\_R1\_001.fastq.gz \_condition\_paired\_fastq\_input\_/dataset/BLGsheep/archive/NZGL01418\_2/Raw/C6KHFANXX-1418-36-05-01\_ATTCCT\_L005\_R2\_001.fastq.gz '\_condition\_uncompressedfastq\_product\_\S\*?\_R1\_\S\*?\d{5}-pass.fq,/dataset/AG\_1000\_sheep/scratch/general\_processing\_062015/NZCMPF100016647269/C6KHFANXX-1418-36-05-01\_ATTCCT\_L005\_R1\_001.fastq.quadtrim' '\_condition\_uncompressedfastq\_product\_\S\*?\_R2\_\S\*?\d{5}-pass.fq,/dataset/AG\_1000\_sheep/scratch/general\_processing\_062015/NZCMPF100016647269/C6KHFANXX-1418-36-05-01\_ATTCCT\_L005\_R2\_001.fastq.quadtrim' '\_condition\_uncompressedfastq\_product\_\S\*?\_RX\_\S\*?\d{5}-singleton.fq,/dataset/AG\_1000\_sheep/scratch/general\_processing\_062015/NZCMPF100016647269/C6KHFANXX-1418-36-05-01\_ATTCCT\_L005\_RX\_001.singlefastq.quadtrim' \> \_condition\_text\_output\_/dataset/AG\_1000\_sheep/scratch/general\_processing\_062015/NZCMPF100016647269/C6KHFANXX-1418-36-05-01\_ATTCCT\_L005\_001.stdout |

### Random sampling of (paired ) fastq files

|  |
| --- |
|  |
| We want to take a “pair-aware” random sample of paired end sequence data, with both mates of every pair sampled extracted. Use the –s option to specify the sample rate (one in ten in this example)  tardis.py -s .1 -d */bifo/scratch/sampling* cat \_condition\_paired\_fastq\_input\_C7N0GANXX-1804-04-4-1\_L004\_R1.fastq \> \_condition\_fastq\_output\_sample\_R1.fastq \; cat \_condition\_paired\_fastq\_input\_C7N0GANXX-1804-04-4-1\_L004\_R2.fastq \> \_condition\_fastq\_output\_sample\_R2.fastq |

### Several outputs, including stdout redirect

|  |
| --- |
| cutadapt -m 40 -f fastq -b AATGATACGGCGACCACCGAGATCTACACTCTTTCCCTACACGACGCTCTTCCGATCT -b GATCGGAAGAGCACACGTCTGAACTCCAGTCACATCACGATCTCGTATGCCGTCTTCTGCTTG -b GATCGGAAGAGCACACGTCTGAACTCCAGTCACCGATGTATCTCGTATGCCGTCTTCTGCTTG -b GATCGGAAGAGCACACGTCTGAACTCCAGTCACTTAGGCATCTCGTATGCCGTCTTCTGCTTG -b GATCGGAAGAGCACACGTCTGAACTCCAGTCACTGACCAATCTCGTATGCCGTCTTCTGCTTG -b GATCGGAAGAGCACACGTCTGAACTCCAGTCACACAGTGATCTCGTATGCCGTCTTCTGCTTG -b GATCGGAAGAGCACACGTCTGAACTCCAGTCACGCCAATATCTCGTATGCCGTCTTCTGCTTG -b GATCGGAAGAGCACACGTCTGAACTCCAGTCACCAGATCATCTCGTATGCCGTCTTCTGCTTG -b GATCGGAAGAGCACACGTCTGAACTCCAGTCACACTTGAATCTCGTATGCCGTCTTCTGCTTG -b GATCGGAAGAGCACACGTCTGAACTCCAGTCACGATCAGATCTCGTATGCCGTCTTCTGCTTG -b GATCGGAAGAGCACACGTCTGAACTCCAGTCACTAGCTTATCTCGTATGCCGTCTTCTGCTTG -b GATCGGAAGAGCACACGTCTGAACTCCAGTCACGGCTACATCTCGTATGCCGTCTTCTGCTTG -b GATCGGAAGAGCACACGTCTGAACTCCAGTCACCTTGTAATCTCGTATGCCGTCTTCTGCTTG 5kb\_example.fastq --too-short-output=tooshort.fastq --untrimmed-output=untrimmed.fastq > 5kb\_example.cutadapt.fastq |
| *tardis.py -w -c 1000* cutadapt -m 40 -f fastq -b AATGATACGGCGACCACCGAGATCTACACTCTTTCCCTACACGACGCTCTTCCGATCT -b GATCGGAAGAGCACACGTCTGAACTCCAGTCACATCACGATCTCGTATGCCGTCTTCTGCTTG -b GATCGGAAGAGCACACGTCTGAACTCCAGTCACCGATGTATCTCGTATGCCGTCTTCTGCTTG -b GATCGGAAGAGCACACGTCTGAACTCCAGTCACTTAGGCATCTCGTATGCCGTCTTCTGCTTG -b GATCGGAAGAGCACACGTCTGAACTCCAGTCACTGACCAATCTCGTATGCCGTCTTCTGCTTG -b GATCGGAAGAGCACACGTCTGAACTCCAGTCACACAGTGATCTCGTATGCCGTCTTCTGCTTG -b GATCGGAAGAGCACACGTCTGAACTCCAGTCACGCCAATATCTCGTATGCCGTCTTCTGCTTG -b GATCGGAAGAGCACACGTCTGAACTCCAGTCACCAGATCATCTCGTATGCCGTCTTCTGCTTG -b GATCGGAAGAGCACACGTCTGAACTCCAGTCACACTTGAATCTCGTATGCCGTCTTCTGCTTG -b GATCGGAAGAGCACACGTCTGAACTCCAGTCACGATCAGATCTCGTATGCCGTCTTCTGCTTG -b GATCGGAAGAGCACACGTCTGAACTCCAGTCACTAGCTTATCTCGTATGCCGTCTTCTGCTTG -b GATCGGAAGAGCACACGTCTGAACTCCAGTCACGGCTACATCTCGTATGCCGTCTTCTGCTTG -b GATCGGAAGAGCACACGTCTGAACTCCAGTCACCTTGTAATCTCGTATGCCGTCTTCTGCTTG *\_condition\_fastq\_input\_*5kb\_example.fastq.gz --too-short-output=*\_condition\_uncompressedfastq\_output\_*tooshort.fastq --untrimmed-output=*\_condition\_fastq\_output\_*untrimmed.fastq *\*> *\_condition\_uncompressedfastq\_output\_*5kb\_example.cutadapt.fastq |

### Executing a short chain of commands – throughput conditioning

|  |
| --- |
| Originally 3 separate commands :  bwa aln protein\_mRNAs.fa Lane2\_1\_Trimmed.fastq.gz > Lane2\_1\_Trimmed\_vs\_protein\_mRNAs.sai  bwa aln protein\_mRNAs.fa Lane2\_2\_Trimmed.fastq.gz > Lane2\_2\_Trimmed\_vs\_protein\_mRNAs.sai  bwa sampe protein\_mRNAs.fa Lane2\_1\_Trimmed\_vs\_protein\_mRNAs.sai Lane2\_2\_Trimmed\_vs\_protein\_mRNAs.sai Lane2\_1\_Trimmed.fastq.gz Lane2\_2\_Trimmed.fastq.gz > Lane2\_Trimmed\_vs\_protein\_mRNAs.sam |
| These can be chained together and conditioned as follows, using the \_condition\_throughput\_ directive to condition the intermediate products. (Note that shell special characters ; and > that are to be passed through to the final conditioned command that gets executed on the cluster need to be escaped).  *tardis.py -w -c 500000* bwa aln protein\_mRNAs.fa *\_condition\_paired\_fastq\_input\_* Lane2\_1\_Trimmed.fastq.gz *\> \_condition\_throughput\_*Lane2\_1\_Trimmed\_vs\_protein\_mRNAs.sai *\;* bwa aln protein\_mRNAs.fa *\_condition\_paired\_fastq\_input\_*Lane2\_2\_Trimmed.fastq.gz *\> \_condition\_throughput\_*Lane2\_2\_Trimmed\_vs\_protein\_mRNAs.sai *\;* bwa sampe protein\_mRNAs.fa *\_condition\_throughput\_*Lane2\_1\_Trimmed\_vs\_protein\_mRNAs.sai *\_condition\_throughput\_*Lane2\_2\_Trimmed\_vs\_protein\_mRNAs.sai *\_condition\_paired\_fastq\_input\_*Lane2\_1\_Trimmed.fastq.gz *\_condition\_paired\_fastq\_input\_*Lane2\_1\_Trimmed.fastq.gz *\> \_condition\_sam\_output\_*Lane2\_Trimmed\_vs\_protein\_mRNAs.bam |

### List file processing

|  |
| --- |
| A common use case, is the need to process a large number of files through some pipeline.  For example 132 paired end files were received from the sequencing provider :  A04\_NoIndex\_L001\_R1\_001.fastq.gz ... A04\_NoIndex\_L001\_R1\_066.fastq.gz  A04\_NoIndex\_L001\_R2\_001.fastq.gz ... A04\_NoIndex\_L001\_R2\_066.fastq.gz  Processing these files in this form can be time-consuming and inefficient, for example   * We can merge these part files to make a single pair of input files. This merge is I/O intensive and slow,   and will double the disk footprint of this data at least for a time (and possibly for the duration of the project on the  system, since it would not be advisable to delete the originals post-merge in case of error) :  ./merge\_files.sh # yields \*\_all.fastq  /usr/bin/DynamicTrim.pl -probcutoff 0.05 A04\_NoIndex\_L001\_R1\_all.fastq  /usr/bin/DynamicTrim.pl -probcutoff 0.05 A04\_NoIndex\_L001\_R2\_all.fastq   * We can process each file individually. This leads to a large number of input and output files to manage, which can be time-consuming and error prone.   /usr/bin/DynamicTrim.pl -probcutoff 0.05 A04\_NoIndex\_L001\_R1\_001.fastq  /usr/bin/DynamicTrim.pl -probcutoff 0.05 A04\_NoIndex\_L001\_R1\_002.fastq  /usr/bin/DynamicTrim.pl -probcutoff 0.05 A04\_NoIndex\_L001\_R1\_003.fastq  etc |
| tardis supports the transparent use of list files – i.e. files that contain the names of the actual files to be processed. A list file name can be used anywhere in a marked-up command, where a data file name would be used. Tardis automatically recognises files as being list files using the following rule :   1. The file name must end with .list (not case sensitive) 2. All non-trivial records in the file must correspond to either absolute or relative file paths on the system, to files that exist. (Relative file paths are relative to the folder containing the list file)   Using list files, all 132 files can be processed with just two commands, and we avoid the time and cost of having to merge the input files – they remain compressed in place. However the results will be as if the files had been merged – i.e. we will end up with ( in this case) two large trimmed files (together with the stats and histograms products).  # prepare list file  ls A04\_NoIndex\_L001\_R1\_\*.fastq.gz > A04\_NoIndex\_L001\_R1.list  # execute the same command as usual – but with the list filename instead of a fastq(.gz) data file name  *tardis.py -w -c 1500000* /usr/bin/DynamicTrim.pl -probcutoff 0.05 *\_condition\_fastq\_input\_*A04\_NoIndex\_L001\_R1.list *\_condition\_uncompressedfastq\_product\_.trimmed,A04\_NoIndex\_L001\_R1.trimmed \_condition\_uncompressedtext\_product\_.trimmed\_segments,A04\_NoIndex\_L001\_R1.trimmed\_segments \_condition\_uncompressedpdf\_product\_.trimmed\_segments.hist.pdf,A04\_NoIndex\_L001\_R1.trimmed\_segments.hist.pdf* |

### Piping blast results to a protein translation utility (referring to an input file twice, and avoiding intermediate files)

|  |
| --- |
| The big\_blast\_translate.py script uses blastx results to determine the translation frame of input sequences and translates them. It is designed for processing very large files (for example entire lanes of RNASeq data), so avoids loading blast hits into memory. To avoid large intermediate files we can pipe blast output to the utility. When a command refers to the same input file more than once as in this example, tardis only splits the file once, and each reference to that input in the conditioned command is bound with the same chunk name).  In this example, we want to process a 3% random sample of all lanes of an RNAseq experiment through this utility, so that we would need to sample each lane, translate to fasta and then pipe through the utility, i.e. something like…  *# …sample 3% of each lane… # …fastq-to-fasta each sample… blastx -query lane1\_sampled\_formatted.seq -num\_threads 4 -db uniprot\_sprot.fa -evalue 1.0e-6 -outfmt "7 qseqid qstart qend qframe stitle" | big\_blast\_translate.py –o lane1\_sampled\_translated.seq –b - -f "GN\=\S+" lane1\_sampled\_formatted.seq  blastx -query lane2\_sampled\_formatted.seq -num\_threads 4 -db uniprot\_sprot.fa -evalue 1.0e-6 -outfmt "7 qseqid qstart qend qframe stitle" | big\_blast\_translate.py –o lane2\_sampled\_translated.seq –b - -f "GN\=\S+" lane2\_sampled\_formatted.seq blastx -query lane3\_sampled\_formatted.seq -num\_threads 4 -db uniprot\_sprot.fa -evalue 1.0e-6 -outfmt "7 qseqid qstart qend qframe stitle" | big\_blast\_translate.py –o lane3\_sampled\_translated.seq –b - -f "GN\=\S+" lane3\_sampled\_formatted.seq . etc .*  (the –f argument to the script passes in a regexp filter used to filter translations – in this case we want to output only those seqs for which we did manage to obtain a consensus frame, via hits to genes) |
| This can be launched using a single command, using a list-file to process all lanes, fastq2fasta conditioning to do format conversion on the fly. Note how we need to escape the pipe symbol, and also the quoted –f argument. If you are unsure whether the escaping is correct, use the –v option to generate conditioned commands which are not launched, and examine the conditioned commands generated.  *tardis.py -w -c 500000 -s .03 -d my\_scratch\_dir blastx -query \_condition\_fastq2fasta\_input\_AllLanes.list -num\_threads 4 -db uniprot\_sprot.fa -outfmt \"7 qseqid qstart qend qframe sacc stitle\" \| big\_blast\_translate.py -b - -f \"GN\\=\\S+\" -o \_condition\_text\_output\_AllLanesSampled.genes.translated \_condition\_fastq2fasta\_input\_ AllLanes.list* |

### Wait conditioning - using tardis to launch a job on a cluster and wait for the output

|  |
| --- |
| The original is run as a direct command in a shell script as part of a pipeline  *bwa mem -t 4 -M BWA\_reference mysample.R1.fastq.flex.gz mysample.R2.fastq.flex.gz | samtools view -bS - > mysample.bam* |
| tardis can be used to launch the job on a cluster and wait for the output (i.e. no input splitting or output processing is done – tardis is just handling the admin of launching the job on the cluster, and synchronously waiting for the job to finish).  *tardis.py -w -d my\_work\_dir bwa mem -t 4 -M BWA\_reference mysample.R1.fastq.flex.gz mysample.R2.fastq.flex.gz \| samtools view -bS - \> \_condition\_wait\_output\_mysample.bam* |

|  |
| --- |
| In this case we want to use tardis to launch a bam indexing job onto the cluster, but wait for it to complete , as this is part of a pipeline. |
| The bam indexing command does not include the name of an output file, so in this case you wait for the (by)product of the |

### Using tardis in a make-file to implement a simple sequence processing pipeline

|  |
| --- |
| This simple pipeline processes paired fastq files through fastqc, flexbar and bwa. Here the flexbar and bwa steps are parallelised to run on the cluster using tardis, but the fastqc step is not. This make-file is based on a naming convention for the paired end files for the ABCProject, for example “LaneName\_1.fastq.gz”, “LaneName\_2.fastq.gz” - it would need to be tweaked for other patterns (or generalised). |
| *# #ABCProjectworkflow version 0.1 #*  *# reference* [*http://www.gnu.org/software/make/manual/make.html*](http://www.gnu.org/software/make/manual/make.html) *# example :  # make -f ABC1.mk -d --no-builtin-rules /dataset/scratch/ABCProject/laneA.bam #*  *datadir = /dataset/scratch/ABCProject workingdir = /dataset/scratch/ABCProject*  *#RUN\_TARDIS=echo tardis.py # this is for a dry run. Change to next lines for the real run #RUN\_FASTQC=echo fastqc # this is for a dry run. Change to next lines for the real run #*  *RUN\_TARDIS=tardis.py # this is for a dry run. Change to next line for the real run RUN\_FASTQC=fastqc # this is for a dry run. Change to next line for the real run*  *# variables for tardis and other apps TARDIS\_chunksize=100000 TARDIS\_workdir=/dataset/scratch/tardis BWA\_reference=/dataset/active/reference\_genome.fa*  *############################################### # how to make an ABC bam file. Made from  # flexbar output from both pairs, and indirectly also  # FastQC output  # Note that we only mention 1 of the pairs in the rule - but process both ###############################################*  *%.bam: %\_1\_fastqc.zip %\_1.fastq.flex.gz   $(RUN\_TARDIS) -w -c $(TARDIS\_chunksize) -d $(TARDIS\_workdir) bwa aln -t 8 $(BWA\_reference) \_condition\_paired\_fastq\_input\_$\*\_1.fastq.flex.gz \> \_condition\_throughput\_$\*\_1.sai \; bwa aln -t 8 $(BWA\_reference) \_condition\_paired\_fastq\_input\_$\*\_2.fastq.flex.gz \> \_condition\_throughput\_$\*\_2.sai \; bwa sampe $(BWA\_reference) \_condition\_throughput\_$\*\_1.sai \_condition\_throughput\_$\*\_2.sai \_condition\_paired\_fastq\_input\_$\*\_1.fastq.flex.gz \_condition\_paired\_fastq\_input\_$\*\_2.fastq.flex.gz \> \_condition\_sam\_output\_$\**  *############################################### # how to make flexbar intermediate files  # Note that we only mention 1 of the pairs in the rule - but process both ###############################################*  *%\_1.fastq.flex.gz:*  *$(RUN\_TARDIS) -w -c $(TARDIS\_chunksize) -d $(TARDIS\_workdir) flexbar --threads 4 --reads \_condition\_paired\_fastq\_input\_$\*\_1.fastq.gz --reads2 \_condition\_paired\_fastq\_input\_$\*\_2.fastq.gz --format sanger --max-uncalled 2 --min-read-length 35 --pre-trim-phred 20 --adapters /home/mccullocha/galaxy/flexbar\_test/adapters.txt --adapter-trim-end ANY --adapter-min-overlap 4 --adapter-threshold 3 --single-reads --target \_condition\_output\_flexbartag \_condition\_fastq\_product\_{}\_1.fastq,$\*\_1.fastq.flex \_condition\_fastq\_product\_{}\_2.fastq,$\*\_2.fastq.flex \_condition\_fastq\_product\_{}\_1\_single.fastq,$\*\_1\_single.fastq.flex \_condition\_fastq\_product\_{}\_2\_single.fastq,$\*\_2\_single.fastq.flex \> \_condition\_text\_output\_$\*.flex.stdout*  *############################################### # how to make fastqc files (not parallelised) ############################################### %\_1\_fastqc.zip:  $(RUN\_FASTQC) $\*\_1.fastq.gz -o $(workingdir) ; $(RUN\_FASTQC) $\*\_2.fastq.gz -o $(workingdir)*  *# # keep the flexbar and fastqc output .PRECIOUS: %\_1\_fastqc.zip %\_1.fastq.flex.gz %\_2.fastq.flex.gz* |

### Filtering records on the fly

|  |
| --- |
| You write your own filter function and specify it in the .tardishrc configuration file as in the example below. At runtime tardis compiles your function and uses it to filter records before they are written out to chunks. (Your function has to have the name “record\_filter\_func” or it will not be used) |
| [tardis\_engine]  seqlength\_min=200  def record\_filter\_func(seqrecord):  # need to filter id like @M02810:27:000000000-AAJE2:1:1101:9707:1735/2  # to  # id like @M02810:27:000000000-AAJE2:1:1101:9707:1735 2:N:0:21  # so that these are parsed OK by bwa paired alignment  # and both names to @M02810:27:000000000-AAJE2:1:1101:9707:1735  import re  seqrecord.name = re.sub("\/1$","",seqrecord.name)  seqrecord.name = re.sub("\/2$","",seqrecord.name)  seqrecord.id = re.sub("\/2$", " 2:N:0:21",seqrecord.id)  seqrecord.id = re.sub("\/1$", " 1:N:0:21",seqrecord.id)  return seqrecord |

## Implementation

### Overview



Figure 1 - Class Diagram

## Related Utilities

### tardis\_analyze.py

tardis\_analyze analyses a tardis run folder and optionally reports statistics and job repair information. The main use case currently is to look for “awol” chunks (that did not report an exit status, and hence may be incomplete), and also chunks that exited with an error code. tardis\_analyze outputs commands that can be used to manually rerun the jobs that appear to be incomplete, or failed.

You only need to consider running tardis\_analyze, if either tardis does not finish, or it exits with an error code. (There are normally three possible outcomes for a tardis run:

1. All chunk jobs exit with return status 0. Tardis collates all of the chunk outputs into your final joined-together-file, and itself then exits with code 0.

2. All jobs finish, but some finish with a non-zero exit code. tardis does not collate the outputs, because the result would be incomplete and possibly misleading. tardis itself exits with a non-zero code, so that the caller knows something went wrong.

3. Some jobs either do not finish, or “go awol” – i.e. no exit code is written to the chunk log file. In that case tardis will wait “forever” for the jobs it is expecting.

- only in situations (2) and (3) is there any need to consider running tardis\_analyze.py – if tardis finishes and returns exit code 0, then “everything worked”, and there is no need to analyse the tardis working folder.

Note that in situation (3), if you compile a redo script using tardis\_analyze and run it, and the original tardis process is still running and waiting, then as noted below, you can signal the waiting tardis master that the redo chunks have completed by uncommenting the lines indicated. Tardis will see all of the status codes written to the log files and will then wrap up the job by collating all of the chunks into your final joined-together-output (assuming all the codes were 0)

Usage:

tardis\_analyze.py [-l path\_to\_log\_folder ] [-x repair|statistics]

|  |
| --- |
| Example:  tardis\_analyze.py -l /dataset/reseq\_wf\_dev/scratch/temp/inbfop03.agresearch.co.nz/tardis\_xtlzI2 -x repair |
| Output:  tardis\_analyze.py using {'action': 'repair', 'logfolder': '/dataset/reseq\_wf\_dev/scratch/temp/inbfop03.agresearch.co.nz/tardis\_xtlzI2'}  Analyzed 5000 log files.  349 jobs need repair  Summary of jobs by status (reported by condor):  {0: 4651, None: 293, 2: 56}  Suggested repair code is below, split into (up to ) 10 batches. You can for example write this to 10 batch-files, then execute all of them concurrently (assuming you have capacity). (Also included are commented out commands that will append a normal termination code to each log file - if your tardis master job is still running, this will then pickup the repaired output(it is polling for the termination line in each log file), and if all outputs are picked up, will complete the job and merge the output. These commands can be uncommented and executed after checking that the repairs ran without error, and completed)  \*\*\*\* Batch 1 \*\*\*\*  /dataset/reseq\_wf\_dev/scratch/temp/inbfop03.agresearch.co.nz/tardis\_xtlzI2/run97.sh # (return code = None)  #echo "JOB REPAIR RESULTS : Normal termination (return value 0)" >> /dataset/reseq\_wf\_dev/scratch/temp/inbfop03.agresearch.co.nz/tardis\_xtlzI2/run97.log  /dataset/reseq\_wf\_dev/scratch/temp/inbfop03.agresearch.co.nz/tardis\_xtlzI2/run258.sh # (return code = None)  #echo "JOB REPAIR RESULTS : Normal termination (return value 0)" >> /dataset/reseq\_wf\_dev/scratch/temp/inbfop03.agresearch.co.nz/tardis\_xtlzI2/run258.log  /dataset/reseq\_wf\_dev/scratch/temp/inbfop03.agresearch.co.nz/tardis\_xtlzI2/run583.sh # (return code = 2)  #echo "JOB REPAIR RESULTS : Normal termination (return value 0)" >> /dataset/reseq\_wf\_dev/scratch/temp/inbfop03.agresearch.co.nz/tardis\_xtlzI2/run583.log  .  .  .  etc |
|  |

|  |
| --- |
| Example – analyse all of the tardis working folders that were generated in the last week for a certain project  find /dataset/Sheep\_GHG/scratch -name "tardis\*" -type d -mtime -7 -exec tardis\_analyze.py -l {} -x repair \; > Weeks\_analysis.log |
| Output:  As above, but concatenated for all of the working folders found |

### summarise\_text.py

Often the non-sequence concatenated text output of a tardis run such as logging or statistics needs further processing to obtain totals. summarise\_text.py is a simple utility that can be used to obtain summary totals.

Usage:

summarise\_text.py [-h] -p PATTERN [-t {add}] [-l] filename

## Appendix 1

### Condor job file template example

Executable = $script

Universe = vanilla

error = $script.err.$(Cluster).$(Process)

output = $script.out.$(Cluster).$(Process)

log = $log

#

# additional requirements - e.g. dynamically exclude specific machines

Requirements = (Machine != "fawlty.myorg.org") && (Machine != "basil.myorg.org")

Queue

## Appendix 2

### Galaxy wrapper modification example (fastx\_trimmer.xml)

|  |
| --- |
| Original templating :  <command>zcat -f '$input' | fastx\_trimmer -v -f $first -l $last -o $output  #if $input.ext == "fastqsanger":  -Q 33  #end if  </command>  <inputs>  <param format="fasta,fastqsolexa,fastqsanger" name="input" type="data" label="Library to clip" />  <param name="first" size="4" type="integer" value="1">  <label>First base to keep</label>  </param>  <param name="last" size="4" type="integer" value="21">  <label>Last base to keep</label>  </param>  </inputs> |
| With tardis related mark-up, and including the additional input fields  <command>  #import string  #if str($use\_hpc) == "yes":  #if str($use\_hpc\_samplerate) != "":  ## not strictly necessary to set up the command as an array  ## of args and then join them as we do here , but sometimes works  ## better  #set $prog = ["tardis.py -w -c ", str($use\_hpc\_chunksize) , " -s ", str($use\_hpc\_samplerate) , " -d /dataset/galaxy\_scratch2011/scratch/tardis zcat -f \_condition\_fastq\_input\_", str($input), " \\| fastx\_trimmer -v -f ",str($first), " -l ", str($last), " -o \_condition\_uncompressedfastq\_output\_", str($output)]  #else:  #set $prog = ["tardis.py -w -c ", str($use\_hpc\_chunksize) , " -d /dataset/galaxy\_scratch2011/scratch/tardis zcat -f \_condition\_fastq\_input\_", str($input), " \\| fastx\_trimmer -v -f ", str($first), " -l ", str($last), " -o \_condition\_uncompressedfastq\_output\_", str($output)]  #end if  #set $prog = string.join($prog, "")  #else  #set $prog = ["zcat -f ", str($input), " | fastx\_trimmer -v -f ",str($first), " -l ", str($last), " -o ", str($output)]  #set $prog = string.join($prog, "")  #end if  $prog  #if $input.ext == "fastqsanger":  -Q 33  #end if  </command>  <inputs>  <param format="fasta,fastqsolexa,fastqsanger" name="input" type="data" label="Library to clip" />  <param name="first" size="4" type="integer" value="1">  <label>First base to keep</label>  </param>  <param name="last" size="4" type="integer" value="21">  <label>Last base to keep</label>  </param>  <!—add HPC switch -->  <param name="use\_hpc" type="boolean" label="Use HPC cluster" truevalue="yes" falsevalue="no" checked="False" help="the input file will be split, and chunks will be processed on the compute farm. The chunk outputs will be pasted together before entry into your history"/>  <param name="use\_hpc\_chunksize" type="integer" label="Chunk size" value="1500000" min="100000" help="the number of sequences each split should contain"/>  <param name="use\_hpc\_samplerate" type="float" optional="True" label="sample rate (leave empty to process the complete file)" value="" min=".000001" max="1.0" help="for example entering .001 will randomly sample and process roughly 1 in every 1000 records"/>  </inputs> |