NeatSeq-Flow: A Lightweight Software For Efficient Execution Of High Throughput Sequencing Workflows

Menachem Sklarz[1,*], Michal Gordon[1] and Vered Chalifa-Caspi[1,*]

[1]Bioinformatics Core Facility, National Institute for Biotechnology in the Negev, Ben-Gurion University of the Negev, 84105, Beer-Sheva, Israel,

* Corresponding authors

Supplementary information

Abstract: Bioinformatics workflows (WFs) in general, and those involving next generation sequencing (NGS) data in particular, typically involve executing a sequence of programs on raw sequence files from as many as thousands of samples. Management of these WFs is laborious and error-prone. We have developed NeatSeq-Flow, a python package that manages WF creation for execution on computer clusters. NeatSeq-Flow creates shell scripts as well as a directory structure for storing analysis results, error messages, and execution logs. The user maintains full control over the execution of the WF, while the computer cluster enforces sequential execution and parallelization. NeatSeq-Flow also supplies tools for version tracking, documentation and execution logging.

Users may add modules for open source, commercial or custom programs not included in the basic package using basic python code (see template in Fig. S5). Detailed instructions for module creation can be found at (http://neatseq-flow.readthedocs.io/en/latest/). It is our hope that the community of users will contribute additional modules to the public.

Availability: https://github.com/bioinfo-core-BGU/neatseq-flow
Documentation: http://neatseq-flow.readthedocs.io/en/latest/
Additional info: http://in.bgu.ac.il/en/bioinfo/Pages/software/neatseq-flow.aspx

Contents

Example workflow

Following is an example of a basic WF (see figure S1 for the parameter file and figure S2 for the sample file). The purpose of the example WF is to perform quality testing and trimming on a set of fastq sequence files, align the sequences to a reference genome and create bigwig files for display in the UCSC genome browser (Kent et al., 2002). Additionally, the example WF creates a report on the quality of the reads and on the mapping of the reads to the reference.

All NeatSeq-Flow WFs begin with the merge module, which copies the raw files from their original locations to the data directory, decompresses them, if necessary, and merges split files into a single file (per direction, in case of paired end reads). The fatsqc_html and trimmo modules are included for quality testing with FastQC (http://www.bioinformatics.babraham.ac.uk/projects/fastqc/) and trimming with trimmomatic (Bolger et al., 2014), respectively.

The sequences are then aligned to a reference genome with bowtie2 (bowtie2_mapper module) and bowtie (bowtie_mapper module). The SAM files produced from each of these mapping strategies are sorted and compressed with samtools, converted to bedgraph format with genomeCoverageBed from bedtools (Quinlan and Hall, 2010) and to bigwig format with bedGraphToBigWig (Kent et al., 2010).

Finally, the results of FastQC and mapping steps are graphically presented in a single MultiQC (http://multiqc.info/) report.

A graphical representation of the WF is shown in figure S3. Note the two parallel mapping branches and the convergence of the fastQC and samtools steps in the MultiQC step, which makes the MultiQC step create a report containing data from both fastQC and both mapping branches.

The structure of the WF and the directories and scripts produced are summarized in supplementary figure S6 below.

Figure S1. NeatSeq-Flow example parameter-file

The parameter file used by NeatSeq-Flow is in YAML format. It includes two main blocks: Global_params and Step_params. See examples below and package documentation for a complete description of the expected format.

```
#################### Parameter file

############## Global definitions:

Global_params:
    ###### QSUB options
    ## It is required that you define a queue on which to run default scripts.
    ## You can override this option in the per-module definitions below
    Qsub_q: queue.q
    ## You can limit the scripts to specific nodes in the queue. Make sure all the nodes are accessible to the
queue defined in Qsub_q!
    ## The node list can be either comma separated or on different Qsub_nodes lines, or both:
    Qsub_nodes: [node1,node2]
    ## You can pass additional default qsub parameters as follows. These will be added to all scripts:
    Qsub_opts: -V -cwd
    ## Default_wait defines the time to wait between steps to ensure the job scheduler registers the jobs
correctly and hence keeps them running in correct order:
    Default_wait: 10


Step_params:

##################################
############### Module definitions:
# Module instances are defined in YAML format, along the following guidelines:
# 1. The instance name (one indent)
# 2. Within each instance, define the following compulsory elements:
#   2a. 'module': The module to use to create the scripts
#   2b. 'base': The base instance from which to take input files ('merge' is the only exception. Since it is
first, you do not define a base).
#   2c. 'script_path': The full path to the program to be executed.
# 3. Additional program arguments can be passed to the program within a 'redirects' group. Make sure you keep
the leading '-' and '--' symbols in the parameter name (see examples below)
# 4. Default qsub parameters can be overriden with parameters within a 'qsub_params' group.
#   4a. Default queue and nodes can be overriden with 'node' and 'queue' params within the 'qsub_params' group.
# 5. Some modules require transferring additional information. See module documentation for these options.



######################################### merge parameters
#
    merge1:
        module: merge
        script_path:         gzip -cd            # When raw files are fastq.gz
        # script_path:         cat                 # When raw files are unzipped fastq
        # script_path:         dsrc d -s           # When raw files are fastq.dsrc2


######################################### trimmo
#
# Sample parameters for regular trimmomatic implementation:
    trim1:
        module:                   trimmo
        base:                     merge1
        script_path:      java -jar trimmomatic-0.32.jar
        todo:                     ILLUMINACLIP:TruSeq3-SE.fa:2:30:10 LEADING:20 TRAILING:20 SLIDINGWINDOW:4:15
MINLEN:36
        qsub_params:
            node:         node1
            -pe:     shared 20
            -threads:     20




######################################### fastqc_html
#
# Parameters for fastqc_html function running on the merge results
    fqc_merge1:
        module:                   fastqc_html
```

```
        base:                     merge1
        script_path:      /path/to/FastQC/fastqc
        qsub_params:
            -pe          shared 15
        redirects:
            --threads     15


# Parameters for fastqc_html running on the trimmo results
    fqc_trim1:
        module:                   fastqc_html
        base:                     trim1
        script_path:      /path/to/FastQC/fastqc
        qsub_params:
            -pe          shared 15
        redirects:
            --threads     15


######################################### bowtie2_mapper
# Sample parameters for bowtie1_mapper
    bwt2_1:
        module:                   bowtie2_mapper
        base:                     trim1
        script_path:              /path/to/bowtie2
        get_map_log:      # Send the log data from stderr to a log file in data/...
        ## The following defines a reference genome INCLUDING THE .fa. MAKE SURE TO BUILD AN INDEX WITH bowtie-
build BEFORE THE ANALYSIS
        ## This is important for downstream steps which might require the reference genome. DON'T NEGLECT!!!
        ref_genome:               /path/to/ref_genome.fna
        qsub_params:
            -pe: shared 20
        redirects:
            -q:
            -p:                                     20
            -x:                           /path/to/ bowtie2_index/hg19.bt2ind


######################################### samtools
# Parameters for Samtools function
# after running mapper, convert sam to bam, sort and index it.
# You can also request removal of old sam and unsorted bam files.
    sam_bwt2_1:
        module:                   samtools
        base:                     bwt2_1
        script_path:      /path/to/samtools/bin/samtools
        qsub_params:
            -pe: shared 20
        view:                     -buh  -q 30 -@ 20 -F 4        # Here you can list samtools view parameters
        sort:                     -@ 20                       # Here you can list samtools sort parameters
        index:
        flagstat:
        stats:          --remove-dups
        idxstats:
        del_unsorted:   # Remove unsorted BAM file after sorting
        del_sam:        # Remove SAM file after conversion


######################################### genomeCoverageBed
## Uses the BAM file to create a bedgraph using the genomeCoverageBed program
    genCovBed_bwt2_1:
        module:      genomeCoverageBed
        base:      sam_bwt2_1
        script_path:   /path/to/bedtools/bin/genomeCoverageBed
        redirects:
            -g:       /path/to/ref_genome/ref_genome.chrom.sizes


######################################### UCSC_BW_wig
## Converts a bedgrapg (bdg) file into bigwig (bw) and wiggle (wig) formats with the bedGraphToBigWig and
bigWigToWig scripts of kentUtils, respectively.
## Operates on the existing bdg file in the sample, created by genomeCoverageBed
    UCSCmapfiles_bwt2_1:
        module:        UCSC_BW_wig
        base:        genCovBed_bwt2_1
        script_path:   /path/to/kentUtils/bin   # This is the bin path, unlike the regular script_path!
        genome:                     /path/to/ref_genome/ref_genome.chrom.sizes
## If you want to pass params to one of these scripts, do it as follows rather than using redirect parameters
        bedGraphToBigWig_params:     -blockSize 10 -itemsPerSlot 20
        bigWigToWig_params:          -chrom X1 -start X2 -end X3
```

```
######################################### bowtie1_mapper

    bwt1:
        module:     bowtie1_mapper
        base:                               trim1
        script_path:                        /path/to/bowtie
## The following defines a reference genome INCLUDING THE .fa. MAKE SURE TO BUILD AN INDEX WITH bowtie-build
BEFORE THE ANALYSIS
## This is important for downstream steps which might require the reference genome. DON'T NEGLECT!!!
        ref_genome:                         /path/to/ref_genome.fna
        ebwt:             /path/to/bowtie1_index/ hg19.bt1ind
        qsub_params:
            -pe:        shared 20
        redirects:
            -p                    20


##################################################################### samtools
# Parameters for Samtools function
# after run mapper, convert sam - to - bam, sort and index it
    sam_bwt1_1:
        module:                   samtools
        base:                     bwt1
        script_path:      /path/to/samtools/bin/samtools
        qsub_params:
            -pe: shared 20
        view:                     -buh  -q 30 -@ 20 -F 4        # Here you can list samtools view parameters
        sort:                     -@ 20                       # Here you can list samtools sort parameters
        index:
        flagstat:
        stats:          --remove-dups
        idxstats:
        del_unsorted:   # Remove unsorted BAM file after sorting
        del_sam:        # Remove SAM file after conversion


######################################### genomeCoverageBed
## Uses the BAM file to create a bedgraph using the genomeCoverageBed program
    genCovBed_bwt1_1:
        module:      genomeCoverageBed
        base:      sam_bwt1_1
        script_path:   /path/to/bedtools/bin/genomeCoverageBed
        redirects:
            -g:       /path/to/ref_genome/ref_genome.chrom.sizes


######################################### UCSC_BW_wig
## Converts a bedgrapg (bdg) file into bigwig (bw) and wiggle (wig) formats with the bedGraphToBigWig and
bigWigToWig scripts of kentUtils, respectively.
## Operates on the existing bdg file in the sample, created by genomeCoverageBed
    UCSCmapfiles_bwt2_1:
        module:        UCSC_BW_wig
        base:        genCovBed_bwt1_1
        script_path:   /path/to/kentUtils/bin   # This is the bin path, unlike the regular script_path!
        genome:                     /path/to/ref_genome/ref_genome.chrom.sizes
## If you want to pass params to one of these scripts, do it as follows rather than using redirect parameters
        bedGraphToBigWig_params:     -blockSize 10 -itemsPerSlot 20
        bigWigToWig_params:          -chrom X1 -start X2 -end X3



    QC_and_map_MultQC:
        module:     Multiqc
        # Note that an instance can be based on more than one instance:
        base:
            - fqc_merge1
            - fqc_trim1
            - sam_bwt2
            - sam_bwt1
        script_path: /path/to/multiqc
```

Figure S2. NeatSeq-Flow example sample-file

```
#################### Sample file

#SampleID  Type       Path
Sample1    Single     Sample1a.fastq
Sample1    Single     Sample1b.fastq
Sample2    Single     Sample2a.fastq
Sample2    Single     Sample2b.fastq
```

Figure S3. Tree structure of an example NeatSeq-Flow workflow

Each circle represents a step in the workflow. The color represents the module, the name of which also appears in the circles in brackets. In this case, there are two instances of the fastqc_html module, one testing the original fastq files and one testing the files produced by the trimmo module. After the trimmo step, there are two main branches, which are identical but for the fact that one branch (on the left) uses bowtie (bowtie1_mapper) for aligning the sequences to the genome while the other branch uses bowtie2 (bowtie2_mapper module).

Results from QC steps and mapping steps are summarized in a graphical report produced by MultiQC (QC_and_map_MultQC step). To achieve this, QC_and_map_MultQC is based on all steps which produce input for it, namely both fastqc_html steps and both samtools steps. This is represented by the convergence of the 4 steps into the Multiqc step.

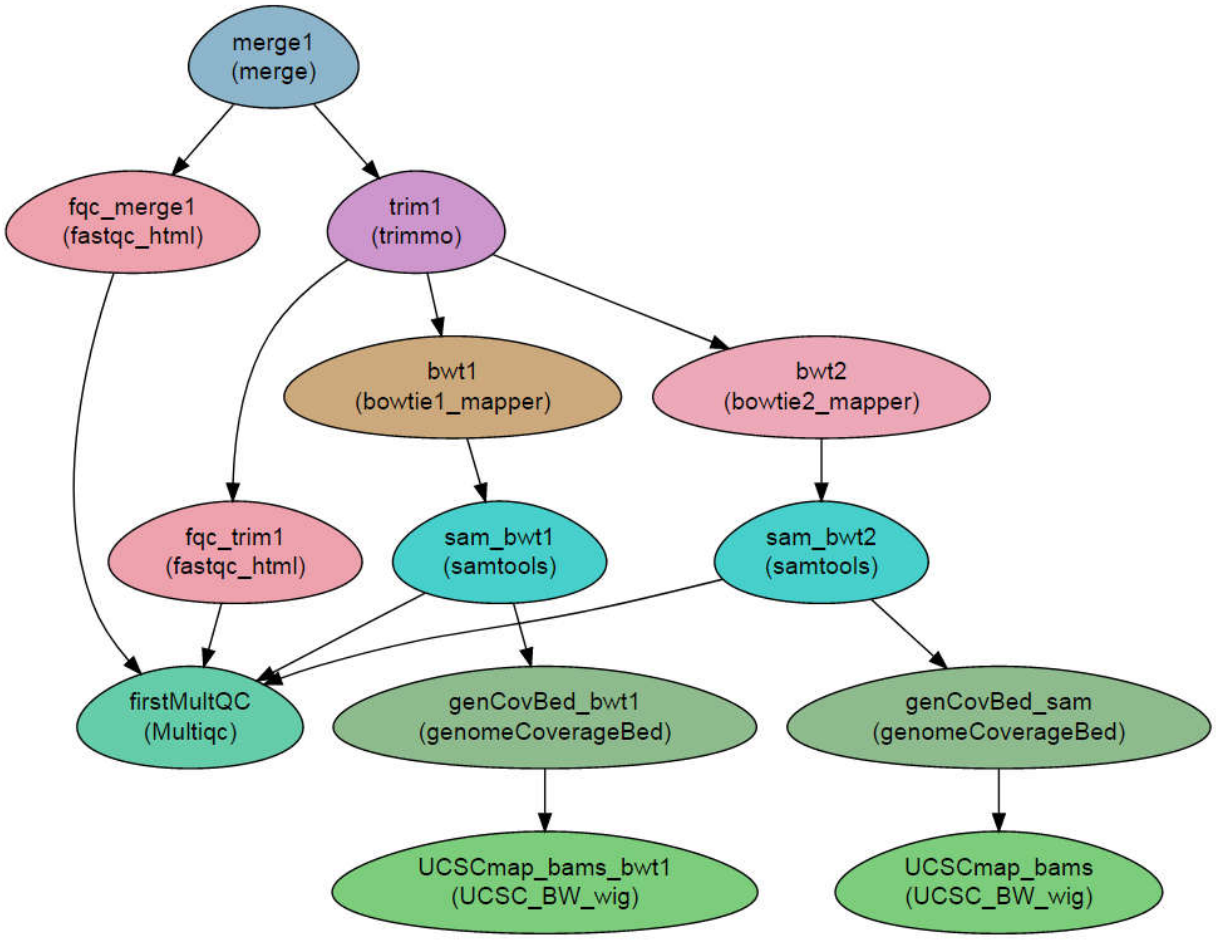


Table S4. Modules currently included in NeatSeq-Flow

Group	Module name	Program	Description
Preparation	merge	-	Copies the raw files, decompresses zipped files and concatenates multiple files.
	fastqc_html	fastqc	Runs the quality checking software FastQC on all fastq files
	Trimmo	trimmomatic	Trims the reads by quality
Read alignment	bowtie1_mapper	bowtie	Maps fastq files to genomes with bowtie
	bowtie2_mapper	bowtie2	Maps fastq files to genomes with bowtie2
	bwa_mapper	Bwa	Maps fastq files to genomes with bwa
	bowtie1_builder	bowtie-build	Builds a bowtie index for fasta files
	bowtie2_builder	bowtie2-build	Builds a bowtie2 index for fasta files
	bwa_builder	bwa index	Builds a bwa index for fasta files
	samtools		Runs various samtools on the SAM file produced by alignment modules.
	genomeCoverageBed	bedtools genomecov	Computes BEDGRAPH summaries of feature coverage for a given genome.
	makeblastdb	makeblastdb	Creates a BLAST database from project or sample fasta files.
BLAST	blast	Any program from the BLAST family.	Runs any type of blast using project or sample fasta files as query or database.
	primer_search	primer_search_wrapper.R	Runs a primer search wrapper available from here
Assembly	spades_assembl	SPAdes	Assembles reads with SPAdes.
	quast	quast.py	Produces quast report on an assembly.
	trinity	Trinity assembler	Assemble sample or project reads. Used mainly for assembling transcriptomes.
	add_trinity_tags	-	Adds tags required by trinity to read names.
UCSC	UCSC_BW_wig	bedGraphToBigWig & bigWigToWig	Creates bigwig and wig coverage files compatible with UCSC
ChIP-seq	macs2_callpeak	callpeak	Runs callpeak on the BAM files. Note: Requires defining sample:control pairs in the sample file.
Reporting	Multiqc	MultiQC	Creates a report for various file formats: FastQC, bowtie2 log, samtools stats and others.
IGV and UCSC	IGV_count	igvtools count	Converts BAM or SAM files into TDF files for viewing in IGV
	IGV_toTDF	igvtools toTDF	Converts wiggle files into TDF files for viewing in IGV

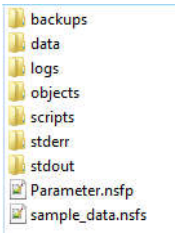
Figure S5. NeatSeq-Flow module template

```
import os
import sys
from PLC_step import Step,AssertionExcept


__author__ = "Author"

class Step_MODULENAME(Step):


    def step_specific_init(self):
        self.shell = "bash"      # Can be set to "bash" by inheriting instances

        # Various assertions
        if CONDITION:
            raise AssertionExcept("ERROR MESSAGE\n")

    def step_sample_initiation(self):
        """ A place to do initiation stages following setting of sample_data
        """


        # Testing a condition on each sample
        # Useful for making sure all samples include the input files required by the module.
        for sample in self.sample_data["samples"]:      # Getting list of samples out of samples_hash
            if (CONDITION ON sample_data):
                raise AssertionExcept("ERROR MESSAGE\n")

    def create_spec_wrapping_up_script(self):
        """ Define self.script to add a script to be executed after all other scripts have terminated
        """
        pass


    def build_scripts(self):
        """ This is the actual script building function
            Most, if not all, editing should be done here
        """

        # Loop over list of samples out of samples_hash and create script for each sample
        for sample in self.sample_data["samples"]:

            # Make a dir for the current sample:
            sample_dir = self.make_folder_for_sample(sample)

            # Name of specific script:
            self.spec_script_name = "_".join([self.step,self.name,sample])
            self.script = ""

            # This line should be left before every new script. It sees to local issues.
            # Use the dir it returns as the base_dir for this step.
            use_dir = self.local_start(sample_dir)

            # Define location and prefix for output files:
            # You can replace _MODULE_SUFFIX with anything you like.
            output_prefix = use_dir + sample + "_MODULE_SUFFIX"

            # Get constant part of script:
            # Adds lines for environmental variables, script path and redirected parameters
            self.script += self.get_script_const()
            # Specifically add input and output files:
            # This changes per module. The input files MUST BE taken from the sample_data dictionary!
            self.script += "%s \\\n\t" % self.sample_data[sample]["fasta"]["nucl"]
            self.script += "%s \n\n" % output_prefix

            # Put the output file/s in the sample_data dictionary
            # If output file is standard format, put in suitable slot.
            # If not, you can invent a slot for it, in a sensible way.
            self.sample_data[sample][...][...] = output_prefix
```

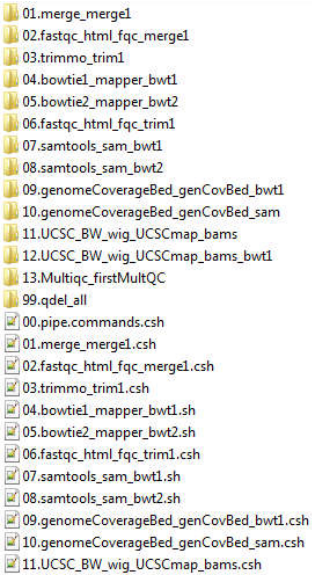
```
            self.sample_data[sample][...][...] = ...
            # Mark file for md5 stamping in log files:
            # Repeat for each file created by the module that you wish to stamp
            self.stamp_file(self.sample_data[sample][...][...])

            # Move all files from temporary local dir to permanent base_dir
            self.local_finish(use_dir,sample_dir)

            # Required line. Leave as is.
            self.create_low_level_script()


    def make_sample_file_index(self):
        """ Make file containing samples and target file names.
            see blast module for implementation.
        """
        pass
```

Figure S6. Description of NeatSeq-Flow output directory structure

1. The main directory structure.
   The directories are elaborated on below.

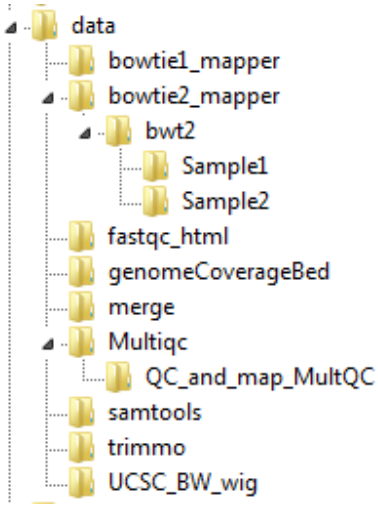
   

2. The **scripts** directory.
   The `00.pipe.commands.csh` scripts executes the entire workflow
   The scripts beginning `01.merge…` etc. execute entire steps.
   The actual scripts running each step per sample or on the entire project are contained in the equivalent directories `01.merge…` etc.

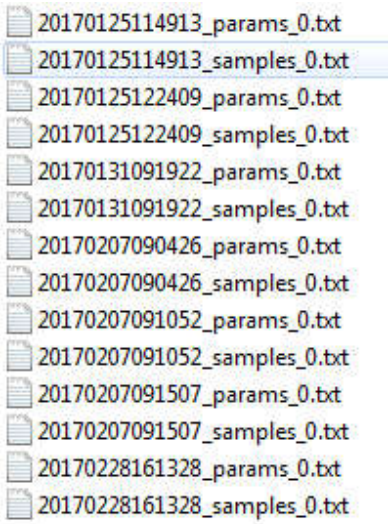
   

3. The **data** directory
   In the data directory, the analysis outputs are organized by module, by module instance and by sample. Below is the data directory for the example, showing the tree organization for the

bowtie2_mapper and Multiqc modules.

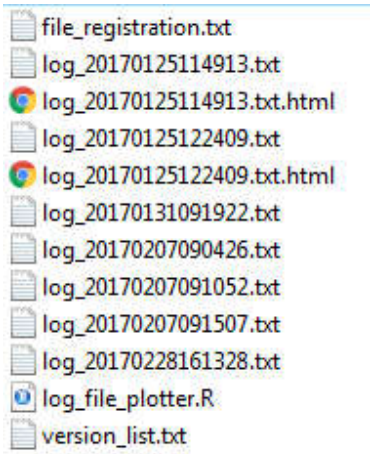


4. The **backup** directory contains a history of workflow sample and parameter files.

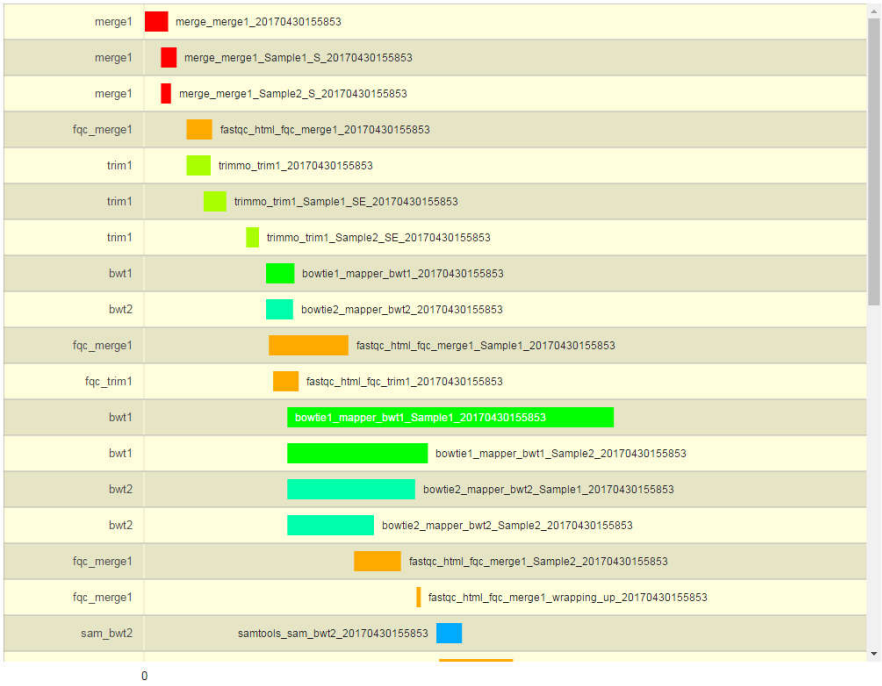


5. The **logs** directory contains various logging files:
    a. version_list. A list of all the versions of the workflow with equivalent comments
    b. file_registration. A list of files produced, including md5 signatures, and the script and workflow version that produced them
    c. `log_file_plotter.R`. An R script for producing a plot of the execution times. (Run with Rscript and receives a single argument – a log file to plot)

d. `log_<workflow_ID>.txt`. Log of the execution times of the script per workflow version ID.

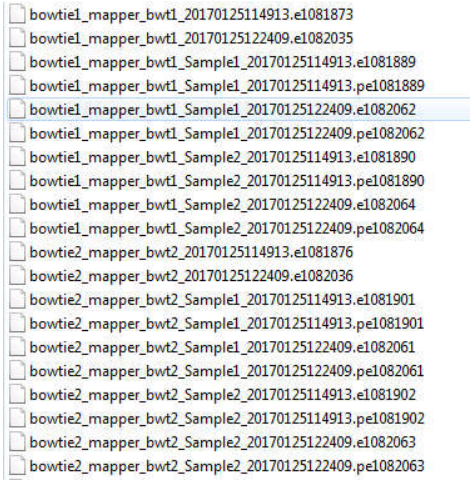


e. `log_<workflow_ID>.txt.html`. Graphical representation of the progress of the WF execution, as produced by the `log_file_plotter.R` script.

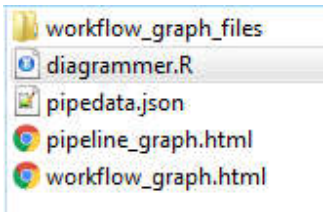


6. The **stderr** and **stdout** directories store the script standard error and outputs, respectively. These are stored in files containing the module name, module instance, sample name, workflow

ID and cluster job ID.



7. The **objects** directory contains various files describing the workflow: An SVG diagram, an R script - diagrammer.R – for producing a DiagrammeR diagram of the workflow, and pipedata.json, containing all the workflow data in JSON format, for uploading to JSON compliant databases etc. (workflow_graph.html is the output from executing diagrammer.R).
   The `diagrammer.R` script requires installing the 'DiagrammeR' and 'htmlwidgets' packages.



References

Bolger,A.M., Lohse,M., *et al.* (2014) Trimmomatic: a flexible trimmer for Illumina sequence data. *Bioinformatics*, 30, 2114-2120.

Hatakeyama,M., Opitz,L., *et al.* (2016) SUSHI: an exquisite recipe for fully documented, reproducible and reusable NGS data analysis. *BMC Bioinformatics*, 17, 228.

Kent,W.J., Sugnet,C.W., *et al.* (2002) The Human Genome Browser at UCSC. *Genome Res.*, 12, 996-1006.

Kent,W.J., Zweig,A.S., *et al.* (2010) BigWig and BigBed: enabling browsing of large distributed datasets. *Bioinformatics*, 26, 2204-2207.

Köster,J. and Rahmann,S. (2012) Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics*, 28, 2520-2522.

Langmead,B. and Salzberg,S.L. (2012) Fast gapped-read alignment with Bowtie 2. *Nature methods*, 9, 357-359.

Leipzig,J. (2016) A review of bioinformatic pipeline frameworks. *Briefings in Bioinformatics*, bbw020.

Li,H., Handsaker,B., *et al.* (2009) The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, 25, 2078-2079.

Linke,B., Giegerich,R., *et al.* (2011) Conveyor: a workflow engine for bioinformatic analyses. *Bioinformatics*, 27, 903-911.

Quinlan,A.R. and Hall,I.M. (2010) BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, 26, 841-842.

Sadedin,S.P., Pope,B., *et al.* (2012) Bpipe: a tool for running and managing bioinformatics pipelines. *Bioinformatics*, 28, 1525-1526.

Stocker,G., Rieder,D., *et al.* (2004) ClusterControl: a web interface for distributing and monitoring bioinformatics applications on a Linux cluster. *Bioinformatics*, 20, 805-807.