NeatSeq-Flow: A Lightweight High Throughput Sequencing Workflow Platform for Non-Programmers and Programmers alike.

Menachem Sklarz[1,*], Liron Levin[1], Michal Gordon[1], 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

Abstract

Nowadays, it has become almost a necessity for many biologists to execute bioinformatics workflows (WFs) as part of their research. However, most WF-management software packages require for their operation at least some programming expertise. Here we describe NeatSeq-Flow, a platform that enables users with no programming knowledge to design and execute complex high throughput sequencing WFs. This is achieved by using a compendium of pre-built modules as well as a generic module, both do not require programming expertise. Nonetheless, NeatSeq-Flow retains the flexibility to generate sophisticated WF modules using templates and only basic Python programming abilities. NeatSeq-Flow is designed to enable easy sharing of WFs and modules by conceptually separating modules, WF design, sample information and execution. Moreover, NeatSeq-Flow works hand in hand with CONDA environments for easy installation of the WF's analysis programs in one go. NeatSeq-Flow enables efficient WF execution on computer clusters by parallelizing on both samples and WF steps. NeatSeq-Flow operates by shell-script generation; thus it allows full transparency of the WF process. NeatSeq-Flow offers real-time WF execution monitoring, detailed documentation and self-sustaining WF backups for reproducibility. All of these features make NeatSeq-Flow an easy-to-use WF platform while not compromising for flexibility, reproducibility, transparency and efficiency.

Availability: http://neatseq-flow.readthedocs.io/en/latest/

Contact: sklarz@bgu.ac.il

Introduction

Modern biological experiments involving High Throughput Sequencing (HTS) produce large amounts of data, which scientists must analyze in order to reach the kernel of information of interest. Usually, analysis of the data is composed of several operations, each of which consists of calling a program with inputs, receiving the outputs and passing them on to the next step. Often, the analysis is parallelized on multiple processing units (CPUs) or cluster nodes, thus saving execution time. The bioinformatician will typically write short shell scripts that execute the different operations and send them sequentially to a computer cluster job scheduler for execution on distributed nodes.

Creating and executing these script-based workflows (WFs) is time consuming and error prone, especially when considering projects with hundreds or thousands of samples, or when the same analysis has to be repeated with different combinations of programs and parameters.

To address these and other issues, many commendable efforts have been made to create platforms for automating execution of such WFs (for examples, Refs. 1-6).

Most of the available WF platforms fall into two main categories: systems using a graphical user interface (GUI, e.g. Galaxy (7)) and command-line based systems (e.g. Nextflow (1), Snakemake (2) and SUSHI (6)). While intended for scientists with no programming experience, GUI-based systems usually have limited flexibility and transparency, and they often do require programming expertise in order to assimilate new tools or to perform complex WFs. On the other hand, command-line based systems are much more flexible and enable tailor made WF designs. However, command-line based systems require programming (e.g. in Groovy, Python or Ruby) even at the design stage of the WFs and are intended for dedicated, expert bioinformaticians.

We have developed NeatSeq-Flow, a lightweight, easy to use, yet powerful WF platform, which offers the advantages of the two worlds presented above. Similar to Galaxy, it is easy to use for non-programmers and programmers alike; however, it is executed through the command-line of computer clusters, thus gaining flexibility and power. NeatSeq-Flow is written in Python and is easily installed, optionally using CONDA package, dependency and environment manager (https://conda.io). It is modular, can use existing as well as newly devised modules, and can execute both publicly-available and in-house programs.

Main Advantages

NeatSeq-Flow WFs are conceptually based on three elements: sample information (files' physical location and type), modules and WF design. This setup enables advantages such as the use of the same WF design on different sets of samples as well as using different WFs on the same set of samples, all of this without changing the individual elements. Moreover, this independency of elements makes them easily shared between users and eventually forms a repository of WFs and modules ready to be used on new sample sets or to be re-edited to form new types of WFs and modules.

The three elements of a WF contain all the information required for its reproduction and are therefore stored by NeatSeq-Flow as a self-sufficient backup for this purpose. Execution of the WF on the cluster is fully under the user's control, using easily understood shell scripts generated by NeatSeq-Flow. These scripts also contain directives enabling parallelization and ensuring sequential execution. In addition, the user can also determine to which node a step will be sent according to the step requirements such as the amount of memory and number of CPUs or by specifying a specific node name.

Designing and running NeatSeq-Flow WFs using existing modules does not require any programming knowledge and with the use of the included "generic module" most Linux-based programs having command-line arguments are also covered, making NeatSeq-Flow accessible to a wide variety of users.

Finally, NeatSeq-Flow supports the use of CONDA for easy installation of NeatSeq-Flow with most of its dependent HTS analysis programs. The use of CONDA environments does not require "superuser" privileges (SUDO) and helps save time and effort in setting up a WF to work on the user's own computer system. Moreover, in complex WFs CONDA relieves the user from the need to deal with interdependency issues among the installed programs. Most importantly, CONDA facilitates sharing of WFs by enabling delivery of entire environments for HTS analyses.

Description of NeatSeq-Flow

NeatSeq-Flow can create and execute WFs on any set of samples and operations. A schematic diagram of NeatSeq-Flow and a detailed example are provided in Figs. 1 and S1.

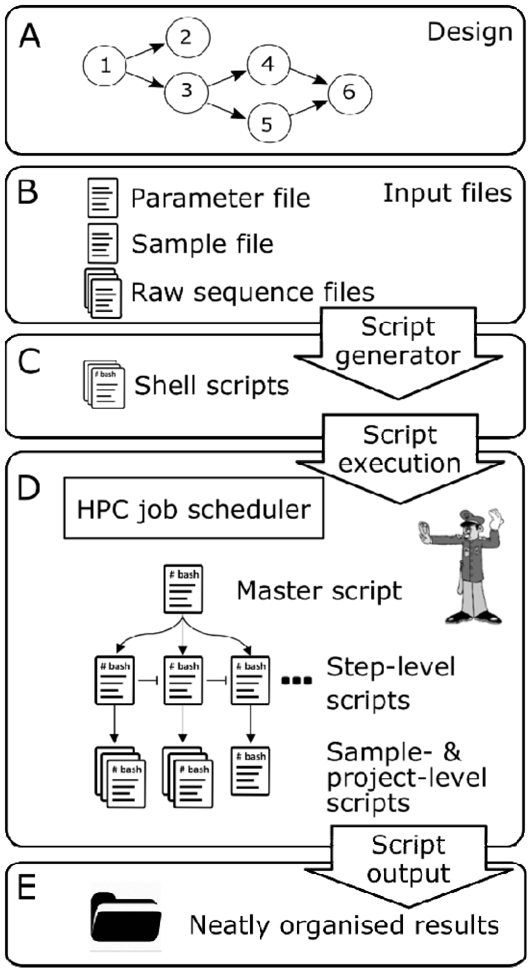


Figure 1. Outline of workflow execution with NeatSeq-Flow. **A.** A conceptual design of the WF takes on the form of a directed acyclic graph where nodes represent steps, arrows represent interdependencies between steps, and convergence (e.g. step 6) represents a step which is dependent on several previous steps. **B.** Based on the WF design, the user creates sample- and parameter-definition files, and provides the input files. **C.** NeatSeq-Flow Script generator is executed, creating a set of structured shell scripts. **D.** These are executed on a computer cluster using the cluster job scheduler, which manages step dependencies and parallelization. **E.** Script outputs, WF log and other accompanying files are neatly organized in a directory structure.

NeatSeq-Flow operations are implemented as modules, where each module is a wrapper for a program. A program could be anything executable from the Linux command-line, from a simple script to a complex software tool, either publicly available (e.g. Trinity (8), BWA (9), Bowtie (10)) or an in-house program. A list of pre-built modules is available at NeatSeq-Flow module and workflow repository, http://neatseq-flow.readthedocs.io/projects/neatseq-flow-modules/en/latest/. Creation of new modules requires basic Python programming knowledge and is easily achieved using a provided template (Fig. S2). In addition, NeatSeq-Flow includes a generic module which can execute any Linux-based program having command-line arguments (in conventional formats). Usage of the generic module does not require any programming knowledge and may thus enable non-programmers to make full use of NeatSeq-Flow for running their own set of programs even if they are not available in NeatSeq-Flow module repository.

The order of the operations, i.e. the WF, is specified in a user-provided "parameter file" (see an example in Fig S1C). The WF is composed of steps, where each step calls a module. A certain module may be called by several distinct steps, e.g. each time with different parameters. For each step, the user defines which previous step(s) need to be completed before the current step executes, thus imposing "step dependencies". The flow of steps may be perceived as a directed acyclic graph (Fig. 1A and see example in Fig. S1A), meaning that a step may be preceded either by a single step (e.g. Bowtie2 (11) precedes Samtools (12) in Fig S1A), or by a convergence of several steps (e.g. MultiQC (13) in Fig S1A). Typically, steps are implemented at one of two levels: per sample (e.g. alignment to a reference) or per project (e.g. *de novo* assembly of the reads from all samples). Finally, for each step the user may define the module's parameters (e.g. the use of the mem algorithm at the BWA mapper module in Fig. S1C), the program's parameters (e.g. –B for mismatch penalty in BWA mem in Fig. S1C) and step-specific cluster parameters (e.g. use nodes with certain memory/CPU requirements or run on specific node name(s)).

The set of raw input files (e.g. FASTQ and FASTA files) can be placed in a directory or directories of the user's choice, and their location(s) and sample attributions should be defined in a "sample file" (see example in Fig. S1B). From this point onwards, the user is relieved from the need to know or manage the locations of intermediate or final files, or to transfer files between WF steps. WF output file locations are determined by NeatSeq-Flow such that they are neatly organized in an intuitive directory structure (Fig. S1E).

Once the user provides NeatSeq-Flow with sample and parameter files, NeatSeq-Flow creates a hierarchy of shell scripts (Fig 1C,D and Fig S1E): a "master script" that calls all step-level scripts; step level scripts that call all sample- or project-level scripts; and sample- and/or project-level scripts that call the relevant programs. The latter shell scripts contain the code for executing the programs, including input and output file locations, user-defined parameters and dependency directives. All scripts are stored in a neat directory structure (see example in Fig. S1E).

Execution of the WF takes place on the computer cluster upon submission of the master shell script to the cluster job scheduler (e.g. to qsub) (Fig. 1D). By definition, WF execution is parallelized by the job scheduler, while step dependencies encoded in the shell scripts ensure the correct order of step execution. Parallelization is both sample-wise as well as step-wise for steps that are on independent branches of the WF (e.g. running several mapping programs as in Fig. S1A or running the same program with different parameter sets). The user may choose to execute only part of the WF; only a certain step; or even only a certain step on a certain sample, by executing the relevant shell script(s) from the script hierarchy. During WF execution, NeatSeq-Flow "Terminal Monitor" may be used to follow the WF progress in real time and to alert for execution errors (Fig S1D).

The WF output files are neatly organized in the "data" directory by module, step and sample (see example in Fig. S1E), making it easy to locate required information. Additionally, execution start and end times as well as maximum memory requirements are written to a log file. Debugging is facilitated by storing STDERR and STDOUT of the shell scripts in dedicated directories. All WF elements necessary for its execution, i.e. its parameter file, sample file and used modules, are copied into a dedicated backup directory. This enables reproducing the WF at any time in the future. Needless to say, the shell scripts themselves, together with the sample and parameter files, constitute the ultimate documentation for the WF performed. Sharing WFs is facilitated by a shared

repository of modules and parameter files (http://neatseq-flow.readthedocs.io/projects/neatseq-flow-modules/en/latest/).

Implementation

NeatSeq-Flow script generator, as well as the modules, are written in Python. The parameter file uses the intuitive YAML format. Program paths (e.g. physical location of Bowtie2 executable) are specified by the user at the top of the parameter file and are easily edited, thus ensuring portability of NeatSeq-Flow and its modules across different computer clusters (Fig. S1C). Input and output file paths of WF steps are determined "on the fly" by the script generator (see below), and are not hard coded in NeatSeq-Flow nor in the parameter file. This concept enables the modules and parameter files to be independent of actual file locations and therefore shareable.

Step dependencies are implemented as follows: *In the parameter file*, the user specifies for each step, which other step(s), called "base step(s)", must precede it (Fig. S1C); *During execution of the script generator*, for each step, the script generator writes a directive into the relevant shell script to hold the execution of the current step's program until the base step's program terminates.

Information sharing between WF steps is implemented as follows: each module contains a definition of the required input file types and the expected output file types, e.g. for BLAST, the module defines FASTA and BLAST-database as inputs and BLAST result file as output. The output file locations are determined by the script generator for each step and stored in an internal data structure which is then passed on to the next step in the WF. In turn, each step can search the data structure for its required input file types (for more details see Fig S3). This design enables great flexibility for the user to thread together steps, with the only requirement being that for each step its required input file type(s) were created by at least one of its predecessor step(s).

The generic module does not contain a definition of input and output file types, therefore in steps that use a generic module, the user has to specify the input and output file types in the parameter file. An example of calling the generic module is provided in Fig. S4, and a full specification is available in NeatSeq-Flow documentation (http://neatseq-flow.readthedocs.io/projects/neatseq-flow-modules/en/latest/Module_docs/GenericModule.html)

To summarize, the script generator generates the shell scripts as follows: for each step it (1) writes a directive into the relevant shell script, to hold the execution of the current step's program until the base step's program terminates; (2) checks that the input file type(s) of the current step's module are compatible with the output file types of its predecessors step(s); (3) constructs unique file paths for the outputs of the current step and stores them in the internal data structure as appropriate types; (4) retrieves from the data structure the file paths of its required input files, generated by previous steps; (5) constructs the shell command for calling this step's program with the input and output file paths, and writes the command in the relevant shell script.

Conclusions and Future Perspective

NeatSeq-Flow allows the user to execute diverse and extensive HTS analyses on computer clusters, while avoiding the tedious task of composing numerous error free shell scripts. Execution of the actual WF is controlled by the cluster job scheduler, while the user has control over which steps and which samples to execute. A WF in NeatSeq-Flow is defined by sample and parameter files and together with the modules used they ensure clear documentation and reproducibility. Furthermore, once the shell scripts are produced by

NeatSeq-Flow, they plainly reveal all the operations applied to the data, with nothing "hidden behind the scenes". NeatSeq-Flow is written in plain Python, such that adding new modules to the software is a straightforward process. A generic module is also provided, enabling calling programs directly, without pre-built modules. Accordingly, NeatSeq-Flow may easily be extended to include new protocols and software packages. It is our hope that the community of users will contribute additional modules as well as dedicated WF designs to the public. NeatSeq-Flow is in constant use by our group for a multitude of analysis procedures, and has proven to be priceless in time saving and error reduction. NeatSeq-Flow is under continuous agile development and improvement. NeatSeq-Flow can be generalized to work on many types of biological data other than HTS data.

Acknowledgements

This research used the High Performance Computing Facility at Ben-Gurion University.

Conflict of Interest: none declared.

References

1. Di Tommaso,P., Chatzou,M., Floden,E.W., Barja,P.P., Palumbo,E. and Notredame,C. (2017) Nextflow enables reproducible computational workflows. *Nat. Biotechnol.,* **35,** 316-319.

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

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

4. Stocker,G., Rieder,D. and Trajanoski,Z. (2004) ClusterControl: A web interface for distributing and monitoring bioinformatics applications on a linux cluster. *Bioinformatics,* **20,** 805-807.

5. Linke,B., Giegerich,R. and Goesmann,A. (2011) Conveyor: A workflow engine for bioinformatic analyses. *Bioinformatics,* **27,** 903-911.

6. Hatakeyama,M., Opitz,L., Russo,G., Qi,W., Schlapbach,R. and Rehrauer,H. (2016) SUSHI: An exquisite recipe for fully documented, reproducible and reusable NGS data analysis. *BMC Bioinformatics,* **17,** 228.

7. Goecks,J., Nekrutenko,A. and Taylor,J. (2010) Galaxy: A comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biol.,* **11,** R86.

8. Grabherr,M.G., Haas,B.J., Yassour,M., Levin,J.Z., Thompson,D.A., Amit,I., Adiconis,X., Fan,L., Raychowdhury,R. and Zeng,Q. (2011) Full-length transcriptome assembly from RNA-seq data without a reference genome. *Nat. Biotechnol.,* **29,** 644.

9. Li,H. and Durbin,R. (2009) Fast and accurate short read alignment with burrows-wheeler transform. *Bioinformatics,* **25,** 1754-1760.

10. Langmead,B., Trapnell,C., Pop,M. and Salzberg,S.L. (2009) Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol.,* **10,** R25.

11. Langmead,B. and Salzberg,S.L. (2012) Fast gapped-read alignment with bowtie 2. *Nat. Methods,* **9,** 357-359.

12. Li,H., Handsaker,B., Wysoker,A., Fennell,T., Ruan,J., Homer,N., Marth,G., Abecasis,G., Durbin,R. and 1000 Genome Project Data Processing Subgroup. (2009) The sequence Alignment/Map format and SAMtools. *Bioinformatics,* **25,** 2078-2079.

13. Ewels,P., Magnusson,M., Lundin,S. and Käller,M. (2016) MultiQC: Summarize analysis results for multiple tools and samples in a single report. *Bioinformatics,* **32,** 3047-3048.

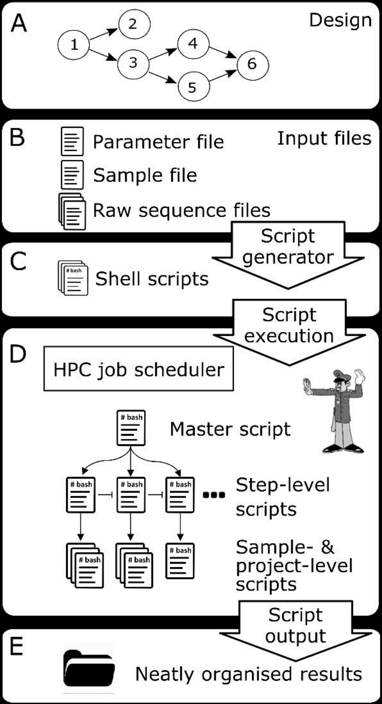
Supplementary Figure legends

Figure S1. Example of NeatSeq-Flow workflow: A. Graphical presentation **B.** Sample file **C**. Parameter file **D**. Terminal Monitor **E**. Output directory structure

Figure S2. Template for a new Module: A. Sample-level module template **B.** Project-level module template

Figure S3. Implementation of managing file transfer between steps

Figure S4. Example of usage and implementation of the generic module


A
Design
1
2
3
4
5
6
B
Parameter file
Sample file
Raw sequence files
Input files
Script generator
C
Shell scripts
# bash
Script execution
D
HPC job scheduler
# bash
Master script
# bash
Step-level scripts
Sample- & project-level scripts
Script output
E
Neatly organised results