“Don’t let a desperate situation make you do desperate things.” ☺

**CAUTION**

The code for the automated pipeline as well as for the additional features is contained in a separate code-folder. Please do not mess with this folder, unless you know what you are doing or you are troubleshooting as described below. Feel free to open the folder to copy the visualization or conversion files onto the local computer, but do not open or format the automation files.

**Note**

When copying/using code: <text> does indicate that something has to be substituted. The brackets <> should not be in the final code!

**Working Instructions [code shown in small caps]**

Steps 7 – 9 are optional to get a better understanding and to double-check correct execution

1. Navigate to working folder: cd Scratch/spades/
2. Create new project folder for each new project or new round of analysis: (do not use whitespace in folder name!)

mkdir <project\_folder\_name>

1. Import sample .fastq files into this folder using Globus Connect Personal (access from local computer)

🡪 importing via command line is also possible, but takes considerably longer than using the Global Personal Connect GUI

1. Execute run-command from project folder:

**./../code\_do\_not\_mess/iterative\_bas.sh –i <integer> -t <meta|isolate> -o <string> -f <string>**

* The variables stand for –i [ID], -t [sample\_type], -o [project\_foldername], -f [forward\_suffix]
* Variables can be passed in any order
* Use example: for a sample that is an isolate sample and is identified by the first 5 characters of the file name, a project folder called “project1”, the suffix for the forward reads is “R1”
  + ./../code\_folder/spades –i 5 –t isolate –o project1 –f R1

1. Check console output or info.log file from project folder: nano info.log
2. If ERROR is being thrown for any of the samples or the program execution, please :

* check the error message
* adjust files/code
* cancel sample with error: scancel <jobID> or cancel all runs: scancel –u pauluga
* re-run program
* if ERROR is being repeatedly thrown: check slurm-files (within code-folder) with correct jobID. Potential issues might be:

runtime error: selected server-time (6h) not enough for sample file size/complexity 🡪(see below)

🡪 go to the code-folder, open file *spades\_script\_40threads\_180G\_META\_input\_from\_iteration.sh (or spades\_script\_40threads\_180G\_Isolate.sh, depending on what type of samples you are using):* nano <script\_file\_name>

🡪 increase time in #SBATCH --time=6:00:00 (eg. to 8 or 12)

permission error: check automation-file permissions (ls –l) if the ouput print “-rw-rw-rw-” instead of “-rwxrw-rw” do the following from code-folder:

chmod 744 automation\_file\_name

1. If no ERROR is being thrown: check execution: squeue –u pauluga
2. Further information on errors/details can be found in the resulting slurm-files or (in more detail) in the sample-output folder in the files params.txt and spades.log
3. Come back in a few hours or the next day to see contigs.fasta outputs in individual sample main folders within the project folder 🡪 done ☺
   * + NOTE: Ignore the contigs.fasta files within the K21, K33 and K55 folders, these are “partial” results

**Detailed Instructions and Explanations**

**What this pipeline does**

The pipeline automates the process of raw NGS read assembly and antibiotic resistance gene (ARG) identification. *SPAdes* is used for raw read assembly on the server and subsequent analysis on the local Linux computer (e.g. d*eepARG* analysis for ARG identification).

The analysis is done automatically and iteratively, so that a large number of files can be dumped into the designated project folder and be run at the same time (using a loop in the script), so that the analyzer does not have to wait for one sample to be analyzed to then manually run the analysis on the next sample.

**What this pipeline does NOT do**

Any quality control (QC)and (de-)compression steps must completed beforehand. Please be aware that raw read QC is crucial as they will determine the results accuracy and quality. Visualization will be needed to be completed separately.

**Additional features (within the code folder)**

* A visualization Python script will be provided for use in *Jupyter* notebooks if desired (please give credit if the visualization script is used: Gabriela Karina Paulus), otherwise the final output from the *deepARG* pipeline (ARGs) can be visualized using any kind of statistical programming language (e.g. R, Python etc.)
* A short script for converting .fastq to .fasta files will be provided

(NOTE: using this script on raw NGS read .fastq files will **not** perform assembly; please use the automates *SPAdes* pipeline for this)

**Requirements for and output of this pipeline**

*SPAdes*:

* Unzipped raw NGS reads in .fastq format will be needed as input files
* forward and reverse reads must be provided in separate files (merging step does therefore NOT have to be done for this pipeline which saves time)
* Output: assembled sequences in.fasta file format
* The assembled .fasta output can be used for a multitude of other programs and analyses. If no ARG identification is required, only perform the first part of the process on the server (*SPAdes*) and feel free to export and used the generated assembled sequences in .fasta format for any other desired application

*deepARG*:

* Assembled .fasta files obtained from *SPAdes* will be needed as input for *deepARG*
* Unassembled raw reads (.fastq or .fasta format) could also be used for analysis, but take considerably longer to process and analyze using *deepARG* 🡪 this script therefore only uses assembled sequences
* Output: a number of different files will be created when using *deepARG,* including files with the suffix:
  + .daa.tsv
  + .mapping.ARG
  + .mapping.potential.ARG
* Files with the suffix .mapping.ARG will contain the desired ARG information and can directly be imported into the provided visualization script (or into any other script that may be used)
  + Example for importing the file into the pandas library (Python): pd.read\_table(path\_to\_file)

**Pipeline Overview**

This pipeline automates the use of *SPAdes* for raw read assembly on a server and the use of d*eepARG* for ARG identification on the local Linux computer. The output .fasta files from the automated assembly (on server) will have to be downloaded to the local computer to then run in the installed *Docker*. As *Docker* is not supported by the server, the download of the .fasta files will have to be completed manually.

A number of variables can be inputted into the run-command to take into account differences in the provided raw NGS reads, as well as in sample types for analysis.

Variables that can be individualized are:

* Suffix for the forward read files: [ -f ]

In most cases sequencing facilities provide/return two files per sample a forward and a reverse read file.

While these files have the same identifier, their suffixes differ. Frequently used suffixes are “R1” and “R2” or “FW” and “RV” for forward and reverse reads, respectively.

The suffix for the forward read will need to be passed to the run-command, so that the script can loop over all files provided in the project folder and run the analysis once per sample (even though 2 files are provided per sample). It is further needed to determine which of the two files is the forward and which the reverse read, to further pass this information to the *SPAdes* program.

* Identifier for each sample: [ -i ]

It is important to clearly be able to identify and distinguish different samples in research. IN most projects, each sample are therefore given unique identifiers. Sequencing facilities often use the researchers identifiers, so that the raw forward and reverse read files contain the identifying information used by the researcher. In other cases, sequencing facilities use their own identifiers which can then be linked to the original samples. To make sure that identifying information does not get lost/confused during the analysis process, the identifying information is taken into account in this pipeline.

The number of characters from the file/sample name that is used for sample identification can be passed to the run-command to use in processing and to save the output correctly in individual folders for each individual sample.

*e.g. forward file name = “CWW\_38860\_CGGAGCCTCT\_L001\_R1\_001\_HTCKBCX2.filt.fastq”*

*a) relevant identifying information = “CWW”* 🡪 *the number of characters would be 3*

*b) relevant identifying information = “CWW\_38860”* 🡪 *the number of characters would be 9*

* Project folder: [ -o ]

For each new project, a new individual project folder should be created. The name of this project folder needs to be passed to the run-command, to make sure the right data/samples are being processed and analyzed.

* Sample type: [ -t ]

Metagenomes and isolate samples need to be processed/analyzed in a completely different way. Therefore, the type of sample (metagenome or isolate) needs to be passed to the run-command to ensure correct analysis.

**Graphic Overview**

1. **Part 1 on server**
2. **Part 2 locally**