Huws Lab Illumina Short-Read

Metagenomic Protocol v1.0.0

Monday, 04 August 2025

This document will include filenames of scripts that can be modified and then submitted using the Kelvin2 HPC. They `should’ work with most modern HPC clusters using the SLURM submission system.

Below are the general commands used to run each tool or step, through a set of samples that are housed separately in individual directories.

**Some useful links:**

* **Kelvin Resources and descriptions:** [https://ni-hpc.github.io/nihpc-documentation/Modules%20%26%20Jobscripts/#job-scheduler-instructions](https://ni-hpc.github.io/nihpc-documentation/Modules%20%26%20Jobscripts/" \l "job-scheduler-instructions)
* **Illumina Info page on adapters:** <https://knowledge.illumina.com/software/general/software-general-reference_material-list/000002905>
* **Overviews of different Metagenome Assembly tools:** <https://doi.org/10.1371/journal.pone.0169662> | <https://doi.org/10.1186%2Fs12864-017-3918-9>

**GitHub links for the tools we will be using:** Github is a database of

* **FastQC:** <https://github.com/s-andrews/FastQC>
* **MultiQC:** https://github.com/ewels/MultiQC
* **Trimmomatic**: <https://github.com/usadellab/Trimmomatic> - Being replaced by FastP
* **FastP**: https://github.com/OpenGene/fastp
* **Bowtie2:**  <https://github.com/BenLangmead/bowtie2>
* **Meta/Spades:** <https://github.com/ablab/spades>
* **Kraken2:** <https://github.com/DerrickWood/kraken2>
* **MetaPhlan:** https://github.com/biobakery/MetaPhlAn
* **Prodigal/Pyrodigal:** https://github.com/althonos/pyrodigal - <https://github.com/hyattpd/Prodigal>
* **Eggnog-mapper:** <https://github.com/eggnogdb/eggnog-mapper>
* **Python3:** <https://www.python.org/>
* **MetaPont:** https://github.com/TheHuwsLab/MetaPont

**Anaconda environments available on Kelvin:** Several anaconda (conda) environments have been pre-installed on Kelvin and are available for everyone to use.

* Users no longer need request access to /mnt/scratch2/igfs-anaconda and should have read-only access. If this is not the case, request access via the Kelvin support ticket system (<https://www.qub.ac.uk/directorates/InformationServices/Services/ITServiceDesk/>)
* There are different anaconda versions available on Kelvin2 and I the current workflow seems to run fine ‘2024.06’ – To ensure that this is the version of anaconda you are using, please enure this line is in all submission scripts ‘**module load apps/anaconda3/2024.06/bin’**.

**Kelvin Job Submission:** Kelvin uses the ‘SLURM’ job submission system (<https://slurm.schedmd.com/sbatch.html>) to handle users and their jobs.

* We submit ‘Jobs’ to Kelvin using the SLURM system with the ‘sbatch’ command.
  + For example, **sbatch fastqc\_script.slurm**
* Submission scripts are written in Bash which is the same language you use to navigate the terminal so commands such as `cd’ and `mkdir’ work the same in a submission file and on the terminal.

General Notes:

* Where I have placed ‘…/’ (3 dots and a forward slash) in the code, I am referring to wherever YOUR data is. You must change this according to where your data is as copying this directly will not work.
* Where you see ‘$$’ (Double dollar signs), I am showing you where you need to add your own parameters. For example, ‘#SBATCH --mail-user=**$$**@qub.ac.uk’ is asking you to replace ‘$$’ with your QUB email address.

Databases:

* Kraken2 Database:
  + The Kraken2 database(s) used in this pipeline are available at : https://genome-idx.s3.amazonaws.com/kraken/$$
  + The database currently used in this workflow is ‘k2\_pluspfp\_20240904’.

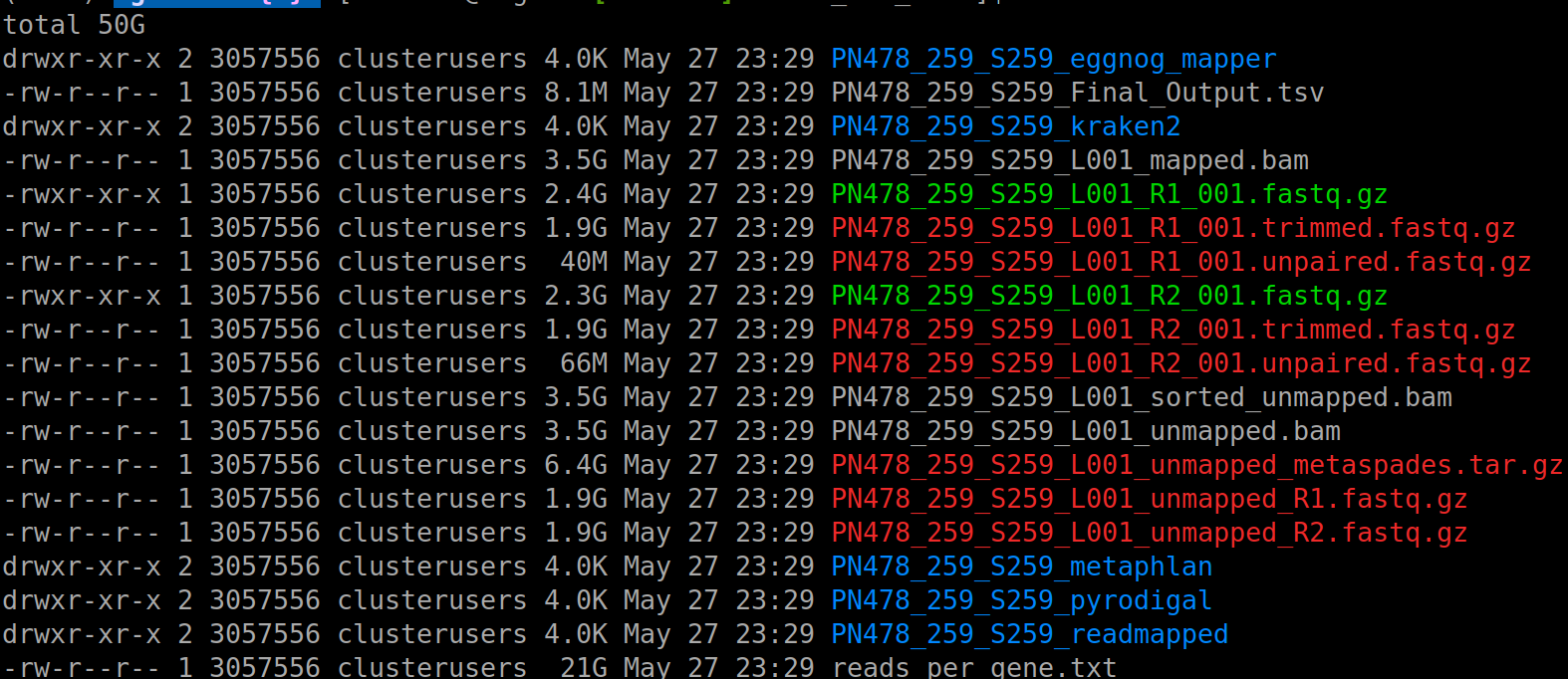
Conda Environments:

* We have a number of preinstalled anaconda environments in the igfs-anaconda directory with all the tools needed for this pipeline:
  + To view them you can use ‘conda env list’
* A screen shot of a computer

  Description automatically generated

**Metagenomic Workflow:**

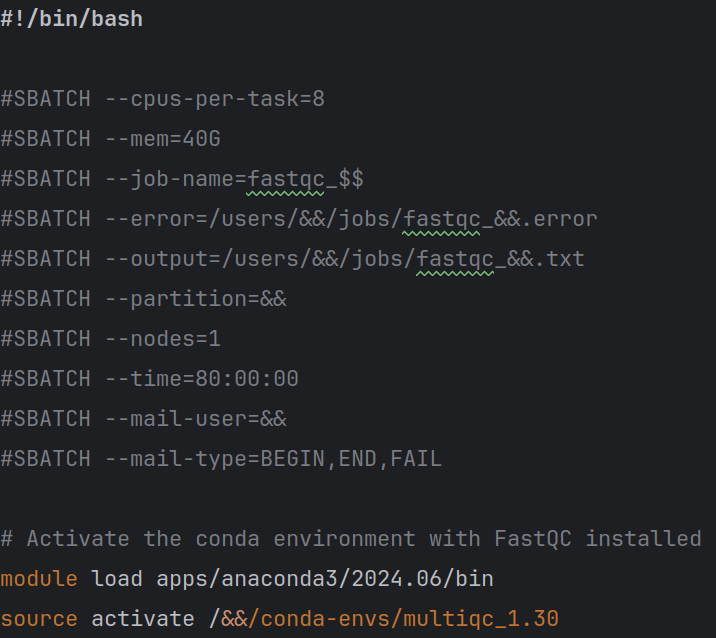
The workflow has been written to conform to a very specific format where each sample is separated and processed in its own directory. Outputs such as assemblies and annotations are given their own directory within the sample directory. The submission scripts rely on this directory structure and as such should not be changed. An example can be seen below:

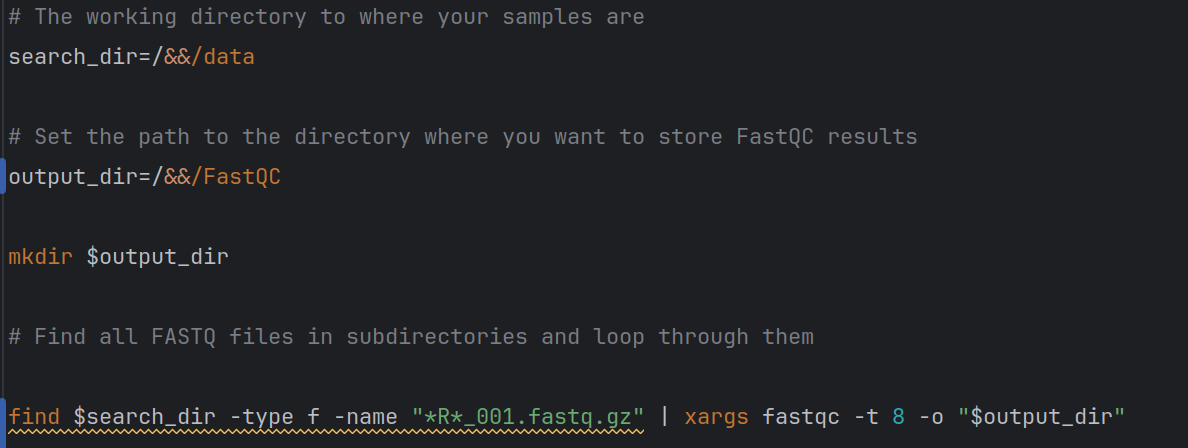


**Kelvin SLURM batch file handling:**

* The Kelvin HPC uses the SLURM job scheduling system to submit jobs into a queue that then run when ‘space’ becomes available.
* These few lines at the top of each SLURM submission script configures a job submission to a SLURM-managed compute cluster with specific resource requests, job parameters, partition preferences, and email notifications for job status updates. The actual computational tasks to be performed by the job would follow these SLURM directives in the script.A screen shot of a computer program

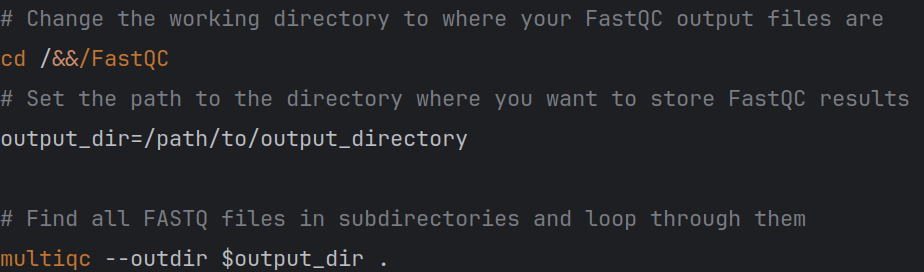
  Description automatically generated
* The SLURM directives are as follows:
  + #SBATCH --cpus-per-task=8: Requests 8 CPU cores for the task, specifying the number of cores allocated per task.
  + #SBATCH --mem=20G: Requests 20 GB of memory for the job, setting the maximum amount of memory the job can use.
  + #SBATCH --job-name=trimm: Sets the name of the job to "trimm", which is useful for identifying the job in the job queue and logs.
  + #SBATCH --error=Trim\_$$.error: Directs the standard error output (errors) to a file named Trim\_$$.error, where $$ is replaced with the job ID, ensuring a unique error file for each job instance.
  + #SBATCH --output=Trim\_$$.txt: Directs the standard output (regular output) to a file named Trim\_$$.txt. The $$ is replaced with the job ID, ensuring a unique output file.
  + #SBATCH --partition=bio-compute,k2-medpri: Specifies that the job should be run on either the "bio-compute" or "k2-medpri" partition, which are groups of compute resources in the cluster. The job scheduler will place the job on any available node in these partitions.
  + #SBATCH --nodes=1: Requests one compute node for the job, where a node is a single machine in the cluster.
  + #SBATCH --time=24:00:00: Sets a time limit of 24 hours for the job, ensuring the job will be terminated if it exceeds this time.
  + #SBATCH --mail-user=$$@qub.ac.uk: Specifies the email address to send notifications to. The $$ will be replaced with the job ID. This placeholder should be replaced with an actual email address before running the script.
  + #SBATCH --mail-type=BEGIN,END,FAIL: Configures SLURM to send email notifications at the beginning, end, and failure of the job, which is useful for monitoring job status without needing to manually check the job queue.

1. **Read quality checking with FastQC:**
   1. **FastQC** is a tool that reads the quality data provided by the Illumina sequencing machine (embedded in the FastQ files) and provides a visual overview of the overall ‘predicted’ quality of the sample (The sequencing machine is not 100% certain of its ‘own’ accuracy).
   2. To run **FastQC**:
      1. Activate the multiqc environment (FastQC and MultiQC are installed together in a single environment ‘multiqc’ on the Kelvin cluster).
      2. Set working directory and output directory and find all ‘.fastq’ files and run the fastqc command on them individually.



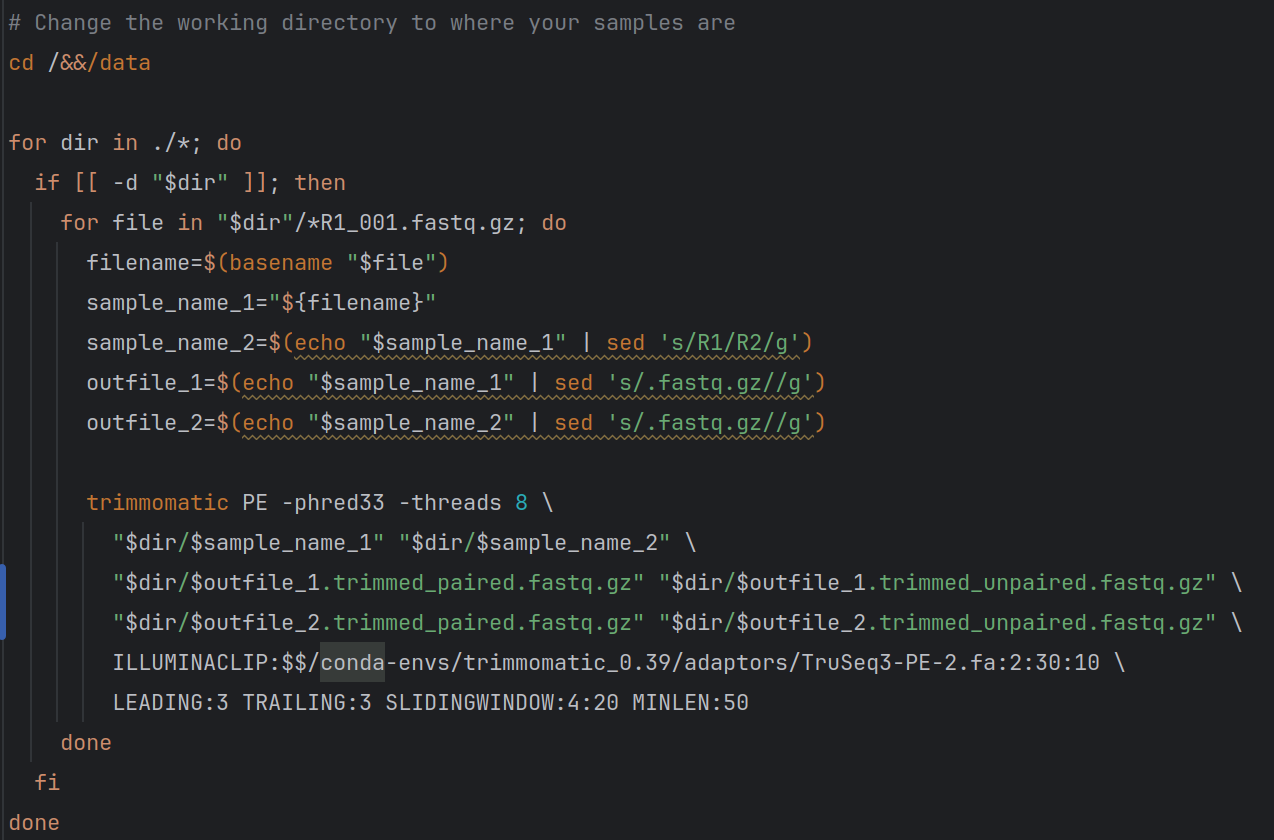
* 1. See Submission file **fastqc\_v1.slurm** to run **FastQC** on a collection of samples.

1. **Overview FastQC quality check with MultiQC:**
   1. **MultiQC** takes the individual output files created by **FastQC** and collates them into a single output that allows for easier comparison across a study.
   2. To run **MultiQC:**

****

* + 1. Activate the multiqc environment.
    2. Set the output directory.
    3. multiqc --outdir /path/to/output\_directory.

1. **(To be replaced with fastp) Trimming and pairing reads with Trimmomatic:**
   1. We are using the Trimmomatic provided adapter sequences (TruSeq3-PE-2) available at: <https://github.com/timflutre/trimmomatic/blob/master/adapters/TruSeq3-PE-2.fa>
   2. To run Trimmomatic:

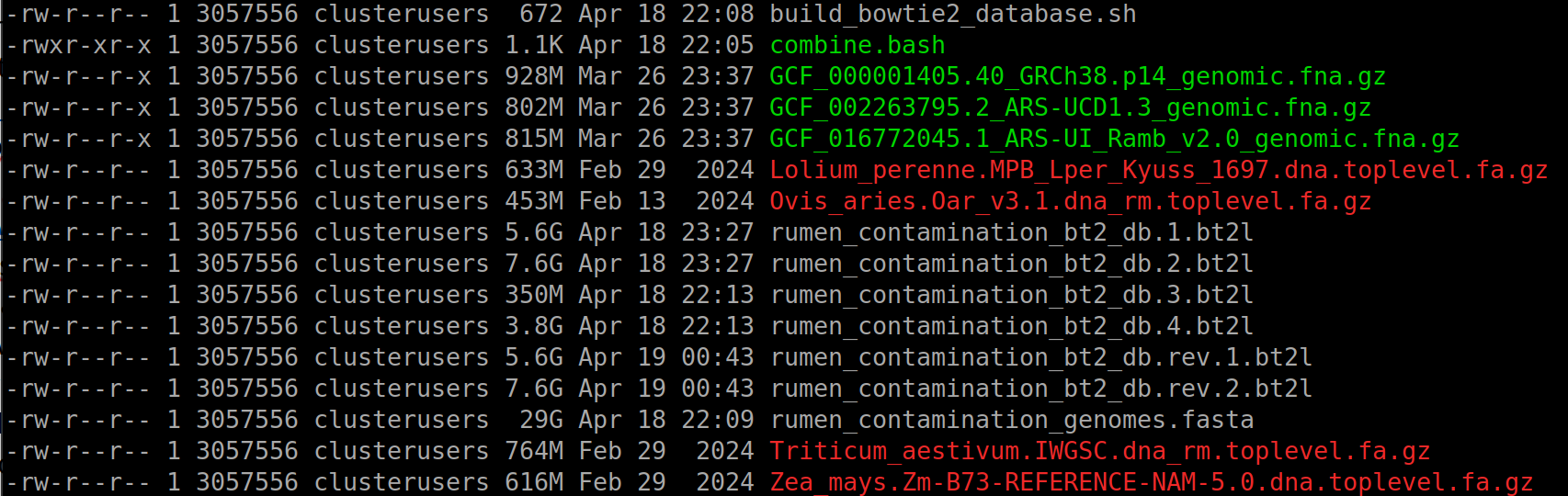
****

* + 1. This command aims to improve the quality of the reads, remove adapter sequences, and ensure that the reads are of a minimum length and have a mate. It's commonly used as a preprocessing step before downstream analysis of sequencing data – Check the manual for further details.
  1. See Submission file **trimmomatic\_v1.slurm** to run **Trimmomatic** on a collection of samples.

1. **(Replacing Trimmomatic) Trimming and pairing reads with Fastp:**
   1. As a replacement for trimmomatic, we can use Fastp to trim and pair our reads.
      1. As we may be unsure of which adapters were used, we use the –detect\_adapter\_for\_pe option. Users can provide their own adapters in fasta format.
   2. See Submission file **fastp\_v1.slurm.**



1. **Removing contamination and preparing reads for assembly:**
   1. We need to identify the host and environmental DNA we might expect to be `contaminating’ our samples. Most of our animal studies will likely be with cows or sheep along with human operators. Therefore, the standard dataset we will use to remove contamination is a combination of the bovine, ovine and human genomes. We have also included some typical agricultural plant genomes. The full list can be found in the ‘rumen\_genomes.txt’ file inside the Contamination\_Control dir.
   2. ***\*\*This has already been done\*\**:** To prepare these genomes for contamination removal we must first download them from NCBI (See links below) and next create a bowtie2 database from them.
      1. These genomes and the bowtie2 database are already available in **‘/mnt/$$/reference\_genomes/rumen\_contamination\_genomes/rumen\_contamination\_bt2\_db** and the bowtie2 database has already been created. See build\_bowtie2\_database.slurm in the Contamination\_Control dir for more details.
         1. Bovine: https://www.ncbi.nlm.nih.gov/datasets/taxonomy/9913/
         2. Ovine: https://www.ncbi.nlm.nih.gov/datasets/taxonomy/9940/
         3. Human: https://www.ncbi.nlm.nih.gov/datasets/taxonomy/9606/
         4. [Lolium perenne: https://www.ncbi.nlm.nih.gov/datasets/taxonomy/4522/](https://www.ncbi.nlm.nih.gov/datasets/genome/GCF_019359855.1/)
         5. Triticum aestivum: https://www.ncbi.nlm.nih.gov/datasets/taxonomy/4565/
         6. Zea mays: https://www.ncbi.nlm.nih.gov/datasets/taxonomy/4577/



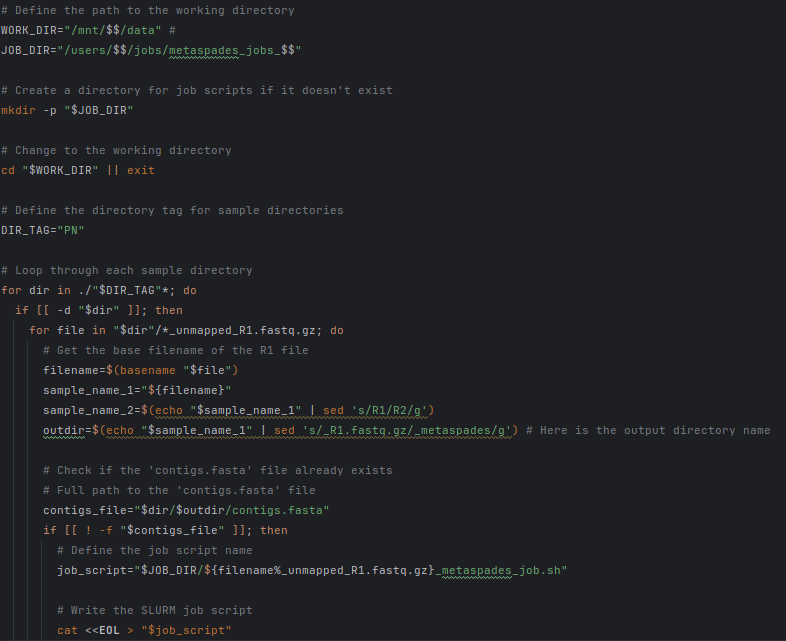
* 1. Due to the time each sample will take to process this script is written differently where each sample has its own SLURM submission. This allows concurrent processing of each sample to significantly speed up processing of very large numbers of samples.
  2. The script is described below:
     1. Its main purpose is to automatically create and submit separate jobs to check for contamination in sequencing data.
     2. Below are details for the script ***contamination\_removal\_array\_v1.slurm***:
        1. WORK\_DIR: where the raw sequencing data is stored.
        2. JOB\_DIR: where the new job scripts and their logs will be saved.
        3. Makes sure the job directory exists (mkdir -p).
        4. Moves into the working directory to look for data.
        5. Looks for folders starting with “PN” (e.g., PN12345) – Change accordingly.
        6. Inside each folder, looks for files ending in R1\_001.trimmed\_paired.fastq.gz – Change accordingly.
        7. For each such file:
           1. Figures out the name of the matching “R2” file.
           2. Uses the folder name as the sample name (PNxxx).
        8. For each sample, creates placeholder variable names for:
           1. A sorted BAM file of reads that did not match contaminants.
           2. Two new FASTQ files for those unmapped paired reads.
        9. For each sample, it writes a separate SLURM submission script that:
           1. Requests computing resources (30 CPUs, 120 GB RAM, 3-hour runtime).
           2. Loads the Anaconda3 module and activates a Conda environment with Bowtie2 installed.
           3. Runs Bowtie2 to align reads against known contamination genomes.
           4. Keeps only the read pairs where both reads didn’t match (using samtools view -f 12).
           5. Sorts the resulting BAM file by read name (with samtools sort -n).
           6. Converts this sorted BAM file back to paired FASTQ files (with bedtools bamtofastq).

Uses pigz to compress the two fastq files to .gz files.

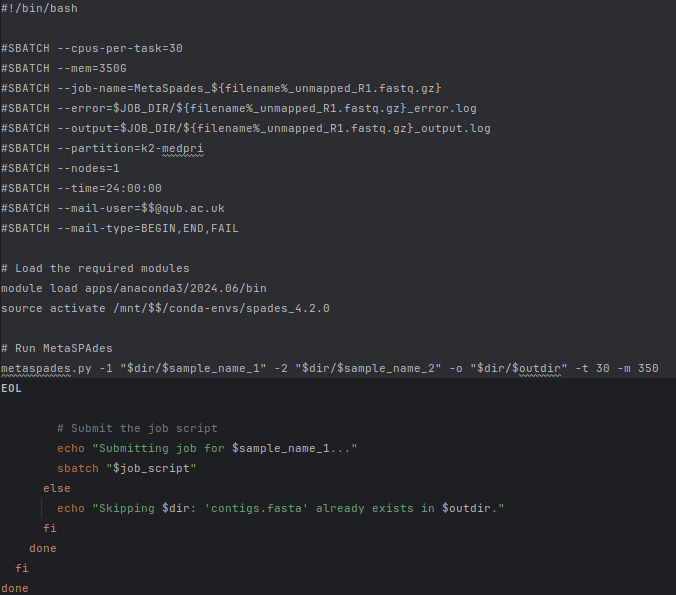
* + - 1. After creating each sample’s SLURM script, the main script uses sbatch to submit it to the cluster.
      2. Prints info about each job so the user knows what’s happening.
      3. At the end, you have:
         1. A cleaned BAM file of reads that didn’t match known contaminants.
         2. New paired FASTQ files of these clean reads.
         3. Log files recording what happened during the job.
         4. The SLURM job script itself (for reproducibility).
    1. **In short:**
       1. This script automates the process of removing known contaminant sequences from many sequencing samples by:
          1. Creating one job script per sample.
          2. Submitting each to the cluster.
          3. Keeping only clean, unmapped reads for further analysis.

See submission the **contamination\_removal\_array\_v1.slurm** scriptin the Contamination\_Control directory for more details.

1. **Assemble reads into Metagenomic contigs: metaspades\_array\_v1.slurm**
   1. Here is where we will assemble our cleaned reads into metagenomic contigs. As with the decontamination step, this SLURM script will automatically create individual submission scripts for each sample.
   2. The script is described below:
      1. Its main purpose is to automatically create and submit separate jobs to assemble each samples set of cleaned reads.
      2. Below are details for the script ***metaspades\_array\_v1.slurm***:
         1. WORK\_DIR: where the raw sequencing data is stored.
         2. JOB\_DIR: where the new job scripts and their logs will be saved.
         3. Makes sure the job directory exists (mkdir -p).
         4. Moves into the working directory to look for data.
         5. Looks for folders starting with “PN” (e.g., PN12345) – Change accordingly.
         6. Inside each folder, looks for files ending in \_unmapped\_R1.fastq.gz (the first read in paired-end sequencing) – Change accordingly (This is the output format provided in the decontamination script).
         7. For each such file:
            1. Figures out the name of the matching “R2” file.
            2. Uses the folder name as the sample name (PNxxx).
         8. For each sample, creates placeholder variable name for:
            1. Output directory name for assembly results (replacing \_unmapped\_R1.fastq.gz with \_metaspades)
         9. Checks if contigs.fasta already exists in the output folder:
            1. **If it does →** skip this sample (don’t submit a new job).
            2. **If it doesn’t →** prepare a new SLURM job script.



* + - 1. For each sample, it writes a separate SLURM submission script that:
         1. Requests computing resources (30 CPUs, 350 GB RAM, 24-hour runtime).
         2. Loads the Anaconda3 module and activates a Conda environment with spades\_4.2.0 installed.
         3. Runs metaspades to metagenomically assemble the cleaned and decontamination reads.



* + - 1. After creating each sample’s SLURM script, the main script uses sbatch to submit it to the cluster.
      2. Prints info about each job so the user knows what’s happening.
      3. At the end, you have:
         1. A new directory containing all the intermediatary assembly files and a final `contigs.fasta’ file containing the assembled contigs.
         2. Log files recording what happened during the job.
         3. The SLURM job script itself (for reproducibility).

See submission the ***metaspades\_array\_v1.slurm*** criptin the Contamination\_Control directory for more details.

1. **Taxonomically annotate contigs with Kraken2:**
   1. Here is where we will taxonomically classify our assembled contigs with Kraken2. As with the two previous steps, this SLURM script will automatically create individual submission scripts for each sample.
   2. The script is described below:
      1. Its main purpose is to automatically create and submit separate jobs to classify each set of contigs one sample at a time.
      2. Below are details for the script ***kraken\_contigs\_array\_v1.slurm:***
         1. WORK\_DIR: where the raw sequencing data is stored.
         2. JOB\_DIR: where the new job scripts and their logs will be saved.
         3. Makes sure the job directory exists (mkdir -p).
         4. Moves into the working directory to look for data.
         5. Looks for folders starting with “PN” (e.g., PN12345) – Change accordingly.
         6. Inside sample directory, looks for the contigs.fasta file produced by the previous assembly step *\*\_unmapped\_metaspades/contigs.fasta* – Change accordingly (This is the output format provided specifically by meta/spades).
         7. For each sample, creates placeholder variable names for ($dir is the name of the sample):
            1. sample\_name=$(basename "$dir")
            2. outdir="${dir}/${sample\_name}\_kraken2"
            3. kraken\_output="${outdir}/${sample\_name}\_kraken2.txt"
            4. report\_mpa="${outdir}/${sample\_name}\_kraken2\_report\_mpa.txt"
            5. report="${outdir}/${sample\_name}\_kraken2\_report.txt"
         8. Checks if $kraken\_output, $report\_mpa and $report already exists in the output folder:
            1. **If they do →** skip this sample (don’t submit a new job).
            2. **If they don’t →** prepare a new SLURM job script.

# Define Kraken2 database  
DBNAME=/$$/conda-dbs/kraken2/k2\_pluspfp\_20240904  
  
for dir in "${DIR\_TAG}"\*; do  
 if [[ -d "$dir" ]]; then  
 for file in "$dir"/\*\_unmapped\_metaspades/contigs.fasta; do  
 sample\_name=$(basename "$dir")  
 outdir="${dir}/${sample\_name}\_kraken2"  
 kraken\_output="${outdir}/${sample\_name}\_kraken2.txt"  
 report\_mpa="${outdir}/${sample\_name}\_kraken2\_report\_mpa.txt"  
 report="${outdir}/${sample\_name}\_kraken2\_report.txt"  
  
 # Check if output files exist and are non-empty  
 if [[ -s "$kraken\_output" && -s "$report\_mpa" && -s "$report" ]]; then  
 echo "Skipping $sample\_name - Kraken2 results already exist."  
 continue  
 fi  
  
 # Create job script  
 job\_script="$JOB\_DIR/${sample\_name}\_kraken\_job.sh"  
  
 cat <<**EOL** > "$job\_script"

* + - 1. For each sample, it writes a separate SLURM submission script that:
         1. Requests computing resources (10 CPUs, 200 GB RAM, 24-hour runtime).
         2. Loads the Anaconda3 module and activates a Conda environment with kraken2\_2.1.3 installed.
         3. Runs kraken2 to on the provided contigs.fasta file.

#!/bin/bash  
  
#SBATCH --cpus-per-task=10  
#SBATCH --mem=200G  
#SBATCH --job-name=Kraken2\_${sample\_name}  
#SBATCH --error=$JOB\_DIR/${sample\_name}\_error.log  
#SBATCH --output=$JOB\_DIR/${sample\_name}\_output.log  
#SBATCH --partition=k2-medpri  
#SBATCH --nodes=1  
#SBATCH --time=24:00:00  
#SBATCH --mail-user=$$@qub.ac.uk  
#SBATCH --mail-type=BEGIN,END,FAIL  
  
module load apps/anaconda3/2024.06/bin  
  
source activate /mnt/$$/conda-envs/kraken2\_2.1.3  
  
mkdir -p "$outdir"  
  
# Run Kraken2 only if output files are missing or empty  
if [[ ! -s "$kraken\_output" || ! -s "$report\_mpa" || ! -s "$report" ]]; then  
 kraken2 --threads 10 --output "$kraken\_output" --report "$report\_mpa" \\  
 --db "$DBNAME" --use-names --use-mpa-style "$file"  
  
 kraken2 --threads 10 --output "$kraken\_output" --report "$report" \\  
 --db "$DBNAME" --use-names "$file"  
else  
 echo "Skipping $sample\_name - Kraken2 results already exist."  
fi  
**EOL** # Submit the job  
 echo "Submitting job for $sample\_name..."  
 sbatch "$job\_script"  
 done  
 fi  
done

* + - 1. After creating each sample’s SLURM script, the main script uses sbatch to submit it to the cluster.
      2. Prints info about each job so the user knows what’s happening.
      3. At the end, you have:
         1. A new directory containing all the kraken2 output files.
         2. Log files recording what happened during the job.
         3. The SLURM job script itself (for reproducibility).
    1. See submission the ***kraken\_contigs\_array\_v1.slurm*** scriptin the Contamination\_Control directory for more details.

1. **Predict CoDing Sequences (CDS) with P(y)rodigal:**
   1. Pyrodigal is a Python implementation of Prodigal (<https://github.com/hyattpd/Prodigal>) which has a few bug fixes and user interface improvements. We use it in this workflow to efficiently identify the CoDing Sequences in our metagenomically assembled contigs.
   2. The SLURM script ***pyrodigal\_v1.slurm*** (available in the Gene\_Function\_Prediction directory) is very basic and does not involve anything we haven’t seen before.
      1. As Pyrodigal is considered a fast tool, the script below processes the contig.fasta assembly files for each of our samples linearly.
      2. We run Pyrodigal under the ‘-p meta’ parameter and request 3 output files:
         1. A GFF file containing the coordinates for each predicted gene.
         2. A FASTA file with the amino acid sequences for each predicted gene.
         3. A FASTA file with the nucleotide sequences for each predicted gene.

# Loop through all directories in the current folder  
for dir in "${DIR\_TAG}"\*; do  
 # Check if the element is a directory  
 if [[ -d "$dir" ]]; then  
 # Loop through all files matching the pattern in the subdirectories  
 for file in "$dir"/\*\_unmapped\_metaspades/contigs.fasta; do  
 # Extract the sample name from the file path  
 sample\_name=$(echo $file | sed 's/\.\///;s/\/.\*//')  
 # Define the output directory and output file paths  
 outdir="${sample\_name}/${sample\_name}\_pyrodigal"  
 pyrodigal\_out\_gff="${outdir}/${sample\_name}\_pyrodigal.gff"  
 pyrodigal\_out\_nt="${outdir}/${sample\_name}\_pyrodigal\_nt.fa"  
 pyrodigal\_out\_aa="${outdir}/${sample\_name}\_pyrodigal\_aa.fa"  
  
 echo "$outdir"  
 # Create the output directory  
 mkdir "$outdir"  
 # Run Pyrodigal with the specified input and output files  
 pyrodigal -i "$file" \  
 -o "$pyrodigal\_out\_gff" \  
 -a "$pyrodigal\_out\_aa" \  
 -d "$pyrodigal\_out\_nt" \  
 -p meta -j 12  
 done  
 fi  
done

1. **Predict Functions for the P(y)rodigal Reported CoDing Sequences:**
   1. Here is where we will functionally classify the predicted CoDing Sequences identified by P(y)rodigal with eggnogMapper. As with some previous steps, this SLURM script will automatically create individual submission scripts for each sample as eggnogMapper can be a slow tool to use.
   2. The script is described below:
      1. Its main purpose is to automatically create and submit separate jobs to classify each set of predicted CoDing Sequences in amino acid form, one sample at a time.
      2. Below are details for the script ***eggnog\_mapper\_contigs\_v1.slurm:***
         1. WORK\_DIR: where the raw sequencing data is stored.
         2. JOB\_DIR: where the new job scripts and their logs will be saved.
         3. Makes sure the job directory exists (mkdir -p).
         4. Moves into the working directory to look for data.
         5. Looks for folders starting with “PN” (e.g., PN12345) – Change accordingly.
         6. Inside sample directory, looks for the***sample*\_pyrodigal/*sample*\_pyrodigal\_aa.fa** file produced by the P(y)rodigal CoDing Sequence prediction step– Change accordingly
         7. For each sample, creates placeholder variable names for ($dir is the name of the sample):
            1. sample\_name=$(basename "$dir")
            2. outdir="${dir}/${sample\_name}\_eggnog\_mapper"
            3. eggnog\_output="${sample\_name}\_pyrodigal\_eggnog\_mapped"
         8. Checks if $eggnog\_output already exists in the output folder:
            1. **If they do →** skip this sample (don’t submit a new job).
            2. **If they don’t →** prepare a new SLURM job script.

for dir in "${DIR\_TAG}"\*; do  
 if [[ -d "$dir" ]]; then  
 for file in "$dir"/\*\_pyrodigal/\*\_pyrodigal\_aa.fa; do  
 sample\_name=$(basename "$dir")  
 outdir="${dir}/${sample\_name}\_eggnog\_mapper"  
 eggnog\_output="${sample\_name}\_pyrodigal\_eggnog\_mapped"  
  
  
 # Check if output files exist and are non-empty  
 if [[ -s "${outdir}/${sample\_name}\_pyrodigal\_eggnog\_mapped.emapper.annotations.xlsx" ]]; then  
 echo "Skipping $sample\_name - eggnogMapper results already exist."  
 continue  
 fi  
  
 # Create job script  
 job\_script="$JOB\_DIR/${sample\_name}\_eggnogMapper\_job.sh"  
  
 cat <<**EOL** > "$job\_script"

#!/bin/bash  
  
#SBATCH --cpus-per-task=20  
#SBATCH --mem=200G  
#SBATCH --job-name=EggnogMapper\_${sample\_name}  
#SBATCH --error=$JOB\_DIR/${sample\_name}\_error.log  
#SBATCH --output=$JOB\_DIR/${sample\_name}\_output.log  
#SBATCH --partition=k2-medpri  
#SBATCH --nodes=1  
#SBATCH --time=24:00:00  
#SBATCH --mail-user=$$@qub.ac.uk  
#SBATCH --mail-type=BEGIN,END,FAIL  
  
module load apps/anaconda3/2024.06/bin  
  
source activate /mnt/$$/conda-envs/eggnog-mapper-2.1.12  
  
# Set location of eggnog-mapper database to use:  
export EGGNOG\_DATA\_DIR=/&&/conda-envs/eggnog-mapper-2.1.12/db  
  
mkdir -p "$outdir"  
  
# Run EggnogMapper only if output files are missing or empty  
if [[ -fs "${outdir}/${sample\_name}\_pyrodigal\_eggnog\_mapped.emapper.annotations.xlsx" ]]; then  
 emapper.py -i "$file" --output\_dir "$outdir" --output "$eggnog\_output" --score 60 \\  
 --subject\_cover 60 --sensmode sensitive --dbmem --decorate\_gff yes --excel --cpu 20 --override  
  
else  
 echo "Skipping $sample\_name - EggnogMapper results already exist."  
fi  
**EOL** # Submit the job  
 echo "Submitting job for $sample\_name..."  
 sbatch "$job\_script"  
 done  
 fi  
done