ONE ARM Metagenomics Pipeline

Prepared for and by: Project 4

Last updated September 21, 2024

Contents

Part I

Project overview

Metagenomic analysis of Antibiotic/Antimicrobial Resistance Genes (ARGs) in NCR (Metro Manila) hospitals, wastewaters, and surface waters.

- ✓ Done
- ☐ Pending
- ⟳ Needs refinement
- △ Unexpected issues
- ✎ Drafted
- ➔ Moved

- Essential
- Optional
- Robust
- ~~Deprecated~~

1 HPC Preparation

- ☐ Confirmation of Robustness of analysis
  - ☐ Agree upon the type of analyses
- ☐ `SLURM` request management
  - ☐ Send email for HPC usage requests
  - ☐ Setup Docker containers for `SLURM`
  - ☐ `SLURM` container for simulations
  - ☐ Automation of `SLURM` requests

File Management

- ☐ `BAM` file parsing
- ☐ Interconversion between `SAM` and `BAM`

Raw Read Processing

- ☐ Pipeline integration of `raw.bash`
- ☐ Data visualization
- ☐ Determination of optimal tool/s
- ☐ Determination of optimal parameters
- ☐ Tool combination randomization script

Assembly & Binning

- ☐ Testing binning pipeline

Creation

- ☐ metaSPAdes
- ☐ MASURCA
- ☐ PLASS
- ☐ AbySS

Testing

- ☐ metaSPAdes
- ☐ MASURCA
- ☐ PLASS
- ☐ AbySS
- ☐ KMA-iterative
- ☐ Benchmarking between all of them
- ✎ Iterative assembly
- ✎ Contig quality checking

Integration

- △ `MetaWrap` binning
  `Binning.smk`
- ☐ Testing on datasets of higher depth for MetaWrap tests
- ☐ Refinement of bins

Annotation

ARG-related scripts

- ⟳ RGI
  ~~prokka_ARG.bash,~~
  ~~shortbred_ARG.bash~~
- ⟳ ShortBRED
  shortbred_ARG.bash
- ⟳ AMRFinder

Testing of Annotation Scripts

- ☐ RGI
- ☐ ShortBRED
  shortbred_ARG.bash
- ☐ AMRFinder

Taxonomy releated scripts

- ☐ MetaPhlan4

Complete

General Taxo-Metagenomics Pipeline

Creation of modular scripts

- ✓ FasQC script for raw reads
  raw.bash
- ✓ Trimmomatic script
  ~~Trimmomatic.bash~~
  trimming-cleaning-checking.py
- ✓ ~~fastp~~
  ~~fastp.bash~~
  trimming-cleaning-checking.py
- ✓ Script to check if reads PASS or FAIL
  ~~FastQC.bash~~
  trimming-cleaning-checking.py
- ✓ FasQC script for trimmed reads
  ~~FastQC_check.py~~
  trimming-cleaning-checking.py
- ✓ Kraken2 script
  krakenpipeline.bash
- ✓ Bracken script
  krakenpipeline.bash
- ✓ In-house diversity calcs script
  calculate_diversity.py

Test Run of Modular Scripts

- ✓ FasQC script raw.bash
- ✓ Trimmomatic script
  ~~Trimmomatic.bash~~
  trimming-cleaning-checking.py
- ✓ ~~fastp~~
  ~~fastp.bash~~
  trimming-cleaning-checking.py
- ✓ Kraken2 script
  krakenpipeline.bash
- ✓ Bracken script
  krakenpipeline.bash
- ✓ In-house diversity calcs script
  calculate_diversity.py

Other Completed Pipeline Tasks

- ✓ Integration with config folder
- ✓ Testing with simulated datasets
- ✓ Pipeline integration
  metagenomics_general.smk
  - ✓ FastQC
    trimming-cleaning-checking.py
  - ✓ Trimmomatic
    trimming-cleaning-checking.py
  - ✓ Kraken2
    krakenpipeline.bash
  - ✓ Bracken
    krakenpipeline.bash
  - △ ~~QIIME2~~ krakenpipeline.bash
  - ✓ Change QIIME2 to in-house diversity script
    calculate_diversity.py

- ✓ Aggregate diversity metrics
  `calculate_diversity.py`
- ✓ Aggregate quality metrics
  `summary_stat.bash`
- ✓ Remove `fastp` from Shellscript

✓ Testing pipeline with simulated datasets
`metagenomics_general.smk`

File Management

✓ Compression and decompression scripts

✓ Demultiplexing script

✓ Testing on multiple datasets

Read Quality Control

Creation

✓ Parametric randomization script
`trim_randomizer.smk`

1. `Trimmomatic`
2. `CutAdapt`
3. `BBDuk`
4. `Sickle`
5. `fastp`

✓ Parsing QC metrics of all iterations
`parseFastQC.py`

✓ Bootstrapping script
`bootstrapping_rawreads.smk`

Test-run

✓ Parsing and aggregation of FastQC metrics from multi-site data (N=10)
`summary_stat.bash`

✓ Randomization of trimming parameters from different trimmers simultaneously (5 trimmers total)
`trim_randomizer.smk`

✓ Bootstrapping of randomization of trimming parameters. 5 Trimmers, 1000 bootstraps
`bootstrapping_rawreads.smk`

Pipeline Preparation for HPC

✓ Calculate using quotation from PGC

✓ Set up config files

✓ Taxo-metagenomics pipeline

Script Descriptions

Pipelines

This section covers all the scripts that were created during the Project.

Listen, I'm eternally curious and I don't just want to settle for any random journal. No offense, but I'm aiming for something high-impact!

These were created during off hours or during work hours, so some scripts might seem irrelevant at first, ***but trust me, there's a conscientiousness or meticulousness to this madness.***

I have separated them into folders/repositories (shameless plug here: https://github.com/GABallena) based on their relevance to the project.

- **Project4** - Essential scripts directly related to the core analyses of the project.
- **Side** - Scripts that can potentially be used to increase the robustness of the paper.
- **Main** - Scripts related to file organization and data management, crucial for handling large datasets.

Project 4 Scripts

1 ARG-MGE.smk

Stage: Draft General Purpose

This pipeline is designed for **comprehensive metagenomic analysis of ARGs**, it also includes placeholders for analyses of **mobile genetic elements (MGEs)**, and **plasmid detection**. It integrates tools for **read quality control, assembly, annotation, taxonomic profiling, and structural variant (SV) detection** to provide a high-resolution view of the genetic components in metagenomic samples.

Preprocessing

1.1 Pair merging

Technical Notes: `PEAR (Paired-End reAd mergeR)` compares paired-reads to correct infer the likely bases on its associated pair

Rationale: The main goal here is to ensure good read quality for downstream analyses and to maximize the amount of data by reducing gaps in the sequence and improving confidence of base calls within that region. `seqtk` then converts compressed `FASTQ` files to `FASTA`.

Rationale

Note: Another tool, `PandaSeq` which does the same thing is used later, this usage of `PEAR` here is because it is more optimized for larger datasets - which in this case are trimmed reads. **Note:** Conversion is necessary here as some tools cannot read FASTQ files, and instead rely on FASTA formats.

1.2 Translation and Reverse Translation

Technical Notes: `transeq` from `EMBOSS` translates nucleotide sequences to protein sequences based on **standard genetic code**. `backtranseq` reverses this to allow iterative alignments with nucleotide sequences.

Rationale: This leverages conserved protein-level information, which is lost at the nucleotide level due to synonymous mutations - while also increasing the sensitivity to potential ARG proteins from k-mer alignment. See Nature Methods, doi: doi.org/10.1038/s41592-019-0437-4 (2019) for more details.

Metagenomic Assembly

1.3 Iterative alignment of ARG-contigs

Technical Notes: `KMA` (K-mer Alignment) is used find the reverse-translated ARGs with the proximity filtering option to determine the surrounding regions of ARGs. This is done iteratively for each gene increasing the ARG-associated database.

Rationale: Firstly, `KMA` is used because, unlike `Bowtie2` and `BWA-MEM`, which were created specifically for Human metagenomics, KMA does not suffer (or suffers less) from multi-allelic databases. Secondly, iterative alignment using this process allows us to contextualize the region wherein ARGs reside - thereby narrowing our focus onto these local regions instead of looking at the global genomic context. **Notes:**

The script is designed to have a cap on the number of iterations KMA creates, increasing the database size.

1.4 Merging of paired reads

Technical Notes: `PANDASeq` is then used to further refine the paired-reads collated in the ARG-related genes database.
Rationale: `PANDASeq` was chosen as, while being slower than `PEAR`, it is more accurate. This mergins step is included to ensure that only high-confidence reads are assembled. **Note:** We leverage the fact that the ARG-related genes database is smaller compared to the raw metagenomic reads database.

1.5 Guided Metagenomic Assembly

Technical Notes: `metaSPAdes` is then used to create contiguous sequences from these local regions by extending them using reads from the whole metagenomic pool. Additionally, Contigs are filtered by length here to remove possible artefacts.
Rationale: `metaSPAdes` was chosen as, while being slower than `MEGAHIT`, it is optimized in handling highly diverse and mixed microbial populations. CARD is used here because it is manually curated and updated regularly - in order to be included in the database, there must be clinical data (e.g. ASTs(Antimicrobial susceptibility testing)) involved in the study.
Notably, this would decrease the sensitivity of our ARGs - and would mostly be biased towards those reported in the clinical setting. To counter this, we could also incorporate other tools such as `ResFinder`, the `NCBI AMR Database`, and `ARG-ANNOT`
Notes: Filtering of contig length is handled by a python script called `minimum_length_CARD.py`.
Notes: Contigs are further extended using contigextender to form scaffolds
Notes: Might add other contig extendending programs like `GapFiller` - which leverages mate-pair information

Contig Quality Checks

1.6 Confirmation of contigs with ARGs

Technical Notes: `RGI` scan the contigs and check whether which contigs created by `metaSPAdes` have ARGs in them.
Rationale: `metaSPAdes` may have created contigs that DO NOT contain ARGs, and have instead assembled them into a more matching contig (a false-positive misassembly) - this can happen because of the different databases being used; also parallelization of methods like this increases robustness because it has been confirmed independently from different starting points (bottom-up vs top-down approach). This allows us to filter ARG-containing contigs.
Notes: `RGI` is the official scanner of `CARD`.

1.7 Standard Contig Quality Metrics

Technical Notes:A custom `Python` script `calculate_contig_quality.py` is created to do another round of checking contig quality for downstream analysis, `R` scripts (TBA) are used to visualize the data.
Notes: The `Python` script will measure standard contig quality metrics: N50, L50, GC-content, and coverage, as well as more robust metrics: N90 and L90.

1.8 Read-mapping

Technical Notes: `Samtools` is used here to map the raw reads from the larger database back to the assembled contigs and then calculates the coverage over the entire contig.
Rationale: Read-mapping is a quality control protocol used in metagenomics, to determine the quality of the assembly. High-coverage means that many of the k-mers align well with that region of the contig, while low coverage is evidence of inconsistent mapping and that the contigs should be refined, split, or discarded.
Note If there are persistent (after further refinement and reassemblies) sudden differences in coverage across a contig, that contig could be chimeric, meaning, it could be from two different populations.
Extra note A `Python` script `plot_and_detect_intermediate_coverage.py` is included in the pipeline that is determine visualize and check how the coverage changes over contig regions. In general, they could be interpreted as the following:

1. **Smooth, Uniform Coverage**: Typically shown by well-assembled contigs.
2. **Sharp Coverage Drop**: May need to be split or flagged for reassembly. May also be a misassembly point (chimeric contig) or caused by a structural variant.
3. **Coverage Gaps**: Regions with little to no read support; a strong indicator of misassembly.
4. **Gradual Drops**: Overlapping reads, repetitive or duplicate regions, partial HGT, sequence heterogeneity, or coverage differences due to a mixed population. Repeats and duplications can be filtered out using tools like `RepeatMasker` or `BLAST` (TBA).
5. **Sharp Increase**: May be due to repetitive or duplicated regions, amplification bias from PCR, HGT, SV, chimeras.

Read mapping parameters

Technical Notes: Four (4) Tools will be used in parallel to do the read mapping process `BWA` `Bowtie` `KMA`, and `minimap2`, their parameters have been adjusted to map reads at 95 % identity to the contigs.
Rationale: 95 % identity is used to increase sensitivity - as is standard for determining homologous sequences. This adjustment was made because k-mers are either attach or don't. Parallelization is used to increase robusness.
Note K-mer extension is used to increase the accuracy of mapping. `BWA` k-mer lengths can be adjusted, while `KMA` does it by default. The others, cannot be adjusted.

Mapper	K-mer Length	Mismatch Penalty	Gap Opening Penalty	Gap Extension Penalty
BWA	21	5	7	2
BWA	31	4	6	2
BWA	51	3	5	1
Bowtie2	-	4,2	5,2	5,2
KMA	Default	95% identity	Automatic	Automatic
Minimap2	-	5	7,2	4,1

Table 1: Example table of mapper configurations without command example

Note

I chose to change the script to not allow gaps during this phase as we already used reverse translation earlier to correct for synonymous codons, and protein sequences are more important when it comes to ARG function, the new values are below. I also added protein-based read mapping.

Tool	K-mer Length	Mismatch Penalty	Gap Opening Penalty	Gap Extension Penalty
BWA	21, 31, 51	5, 4, 3	1000	1000
Bowtie2	-	4,2	1000,1000	1000,1000
KMA	-	95% identity	Automatic	Automatic
Minimap2	-	5	1000,1000	1000,1000

Table 2: Alignment parameters for ungapped alignments across BWA, Bowtie2, KMA, and Minimap2

Rationale: The main rationale for adding a protein-based read mapping protocol is because ARGs are primarily about their ***protein-protein interactions*** (biological relevance). This method also accounts for frameshifts with higher specificity to homologous regions.

On a personal note: This approach may also require further exploration into whether protein-protein interactions are altered—perhaps by investigating changes in binding sites. Which is a story for another day (***why do I do this to myself?***)

1.9 Taxonomic profiling of reads and contigs

Technical Notes: Kraken2 uses k-mer-based classification to assign taxonomy based on raw-reads. While SprayNPray complements this by assigning taxonomy at the contig level.

Rationale: By comparing their respective databases with our ARG-related databases, we will be able to connect our reads and/or contigs to their corresponding taxa, uncovering the microbial hosts responsible for carrying and potentially spreading ARGs in the environment.

1.10 Detect structural variants (SVs) in contigs

Technical Notes: Manta identifies large genomic rearrangements such as insertions, deletions, and duplications.

Rationale: Chimeric contigs may be due to systematic error or real biological signals. These chimeric contigs can be detected by Kraken2 and SprayNPray (i.e., when a portion of a contig is being assigned

Tool	Input Files	Key Parameters
tblastn	• Protein sequences: `Translated protein sequences contigs` • Nucleotide database: `Cleaned sequence database`	• `-outfmt 6` • `-evalue 1e-5` • `-gapopen 5` • `-gapextend 2` • `-matrix BLOSUM62`
blastp	• Protein sequences: `Translated protein sequences contigs` • Protein database: `Translated cleaned sequence database`	• `-outfmt 6` • `-evalue 1e-5` • `-gapopen 5` • `-gapextend 2` • `-matrix BLOSUM62`

Table 3: Protein read-mapping parameters for `tblastn` and `blastp`

to different taxa). The rationale behind this step is to investigate whether structural variations are present — which may be evidence of horizontal gene transfer (HGT) events.

1.11 Sketching contigs followed by calculating `Bray-Curtis` diversity

Technical Notes: `Mash Sketch` uses a MinHash approach to generate a presence/absence profile of ARGs across contigs. This gives us a quick snapshot of what the contigs "look like" in terms of ARG content. For `Bray-Curtis` diversity, we calculate a dissimilarity matrix from the abundance data of ARGs, followed by a PCoA plot to visualize similarities between contigs based on their ARG profiles.
Rationale: The `Mash Sketch` helps rapidly identify the genetic makeup of contigs in terms of ARGs, which provides a foundation for further investigation. By applying `Bray-Curtis` diversity and using PCoA, we can group contigs based on their ARG similarity. If contigs with the same sketch group together, we can trace them back to their taxonomic IDs to identify the microbial hosts. However, if contigs have similar ARG profiles but belong to different taxa, this could serve as **evidence for Horizontal Gene Transfer (HGT)**. This dual approach allows us to trace ARG spread and potential HGT events in a metagenomic context.
Note: `Bray-Curtis` (dis-)similarity is most often used as a presence or absence diversity metric.

Transposition

1.12 Determination of Transposons (TBA)

Technical Notes: Tools such as the following can be used:

- `HMMER3` suite
- `Tnppred` - a transposon predictor tool

Rationale:

Plasmids

1.13 Determination of putative plasmids

Technical Notes: The tools listed below will be used in parallel. Plasmids are considered valid when all 4 tools predict plasmid signatures in the contig.

- `PlasPredict` pipeline
- `Recycler`
- `PlasmidFinder`
- `MOBSuite` plasmid marker annotator

If plasmid signatures are present, `plasmidSPAdes` along with `GapFiller` will be used to check if the contig can circularize. `oriTfinder` will then be applied to contigs with fewer than 4 fragments. A `Python` script (TBA) will calculate GC skews of the chimeric contig and compare it to its taxonomic counterparts. Another `Python` script (TBA) will normalize the data according to `16S rRNA` from trimmed reads. Lastly, plasmid percentage will be calculated based on reads mapped to putative plasmids over the total reads. A code snippet is present called `calculate_plasmid_percentage.py`.

Equation:

$$\text{Plasmid Percentage} = \left(\frac{\text{Plasmid Reads}}{\text{Total Reads}}\right) \times 100$$

Rationale: `oriTfinder` looks for `origin of transfer` sites (`oriT`), which are characteristics of conjugative plasmid . This whole sub-pipeline is to look for evidence of conjugative plasmid transfer as the cause of these chimeric contigs. Normalization and percentage counts are used here to further check whether these ”plasmids” align with our understanding of the average plasmid copy number.

Note: Will also be drafting a script (TBA) to do sliding window analysis of `GC-skews` - as different characteristics of this curve can be interpreted in different ways.

Phages

1.14 Phage influence signatures

Technical Notes: They will be determined using a variety of tools:

- `VirSorter`: Identifies viral signatures within microbial genomes and separates prophages from bacterial sequences.
- `PHASTER`: A web-based tool for phage search and annotation, identifying integrated prophages.
- `VIBRANT`: A tool that combines several approaches to identify and annotate phage elements in metagenomic sequences.

Rationale:This analysis aims to detect potential phage signatures in the chimeric contigs. Since phages are mobile genetic elements, their involvement in transferring ARGs through transduction is highly relevant.Phages, especially temperate phages, can integrate into bacterial genomes and excise themselves, sometimes carrying host genetic material, such as ARGs, with them.The integration and excision signatures detected in contigs will provide evidence of possible transduction events in our datasets, supporting the hypothesis of ARG dissemination via phages.

1.15 Phage Signature Extraction and Phylogenetic Analysis

Technical Notes: Phage-associated genes will be extracted from the chimeric contigs, followed by phylogenetic analysis to uncover evolutionary relationships. `FastTree` will be used to build a phylogenetic tree based on the extracted phage genes. For visualization, tools like `iTOL` or `FigTree` can be used to generate an interpretable phylogenetic tree.

Rationale: Phage genes embedded in chimeric contigs (their taxonomy) may serve as strong evidence of horizontal gene transfer (HGT) events. The aim here is to check the evolutionary origins of the phage genes found in our dataset and their potential involvement in the dissemination of ARGs.

1.16 Directory tree

Notes: The final directory tree should look like this to help you visualize. **Notice the multiple** `path/to/` **here as this is still (WIP)**:

```
/project_root/
├── *path/to/cleaned_reads/* (Prerequisite)
├── path/to/merged_reads/
├── *path/to/CARD_db/* (Prerequisite - Database)
├── path/to/output/
│   ├── final_annotated_plasmids.fasta
│   ├── fpkm_normalized_plasmids.txt
│   ├── categorized_MGEs.txt
│   └── final_filtered_contigs.fasta
└── path/to/logs/
    ├── predict_plasmids.log
    ├── run_mob_suite.log
    └── check_plasmidfinder.log
```

```
    │
    └── calculate_gc_skew.log
├── path/to/plaspredict_output/
├── *path/to/PFAM_db/* (Prerequisite - Database)
├── *path/to/TnpPred_db/* (Prerequisite - Database)
├── path/to/plasmid_prediction/
    ├── predicted_plasmids.fasta
    ├── plasmidfinder_report.txt
    ├── oritfinder_report.txt
    └── tnpred_report.txt
├── path/to/plasmidspades_output/
    └── plasmid_assembly.fasta
├── path/to/mob_suite_output/
    └── mob_suite_summary.txt
├── path/to/gc_skew_analysis/
    └── gc_skew_plot.png
└── path/to/read_mapping/
    ├── reads_mapped_to_plasmids.bam
    ├── plasmid_read_count.txt
    └── total_read_count.txt
```

Homework?

1.17 Future Considerations

Will continue improving this section (everything regarding HGT) by evaluating the results from these tools, aligning them with ARG presence, and refining the approach for identifying conjugation, transposon, and transduction events within chimeric contigs. This may also involve validating phage activity—*again, why do I do this to myself?*

2 metagenomics_general.smk

Stage: Done

General Purpose taxo-metagenomics

Specifics: This pipeline is designed for **The essentials in metagenomics**, which includes **quality checking, filtering, and trimming of raw reads to clean reads**. As well as the usual **taxo-metagenomic analysis**.

2.1 Quality control (Pre-processing)

Raw trimming of raw metagenomic data

Technical Notes: `FastQC` is used on **raw reads**

Rationale: This is mainly used as a point of comparison - determine whether the next step (trimming) was effective. This pre-processing step is the starting point in any and all metagenomics pipelines.

Notes: This whole quality control steps are interconnected with each other.

2.2 Trimming

Technical Notes: `Trimmomatic` is used on **raw reads**

Rationale: Trimming involves removing low quality bases (often depending on something called the Phred Score - which is just a measure of how ”confident” we are that the base on that site was accurate), adapters, and filtering reads that go below a specific length threshold.

Notes: Journals often report the parameters on `Trimmomatic` (or any trimmer they decide to use); this is often so that the study is reproducible, should one decide to actually reproduce the study starting from scratch (raw reads).

Notes: There are many different trimmers each with their own strengths and weaknesses, `Trimmomatic` is just the most popular trimmer and is thus used here, though studies differ in the parameters they used for trimming - which often dictates how strict they are with what they define as ”good enough”.

Perspective: Why different people choose different trimmers depend on the **strengths and weaknesses** of the trimmer e.g. trimmers like ”fastp” is used because it’s fast making it suitable for very large datasets like deep sequencing. While some trimmers like `Sickle` has automatic adjustment over the entire sequence - which makes it useful for very ancient datasets where DNA is often highly-degraded. Other times, it’s for **convenience** like `Trim-Galore` which combines `FastQC` and `Cutadapt` trimmer in a single command. Another good example is `BBDuk` which is part of a larger package called `BBtools`, `BBDuk` also has an built-in contamination detection - so it’s particularly good at filtering out usual contaminants like sequences known to be from the human genome. so you can simply just use all the modules in that package for all-in-one processing. Other times, it’s just **familiarity**.

Quality checks of post-trimmed data

Technical Notes: `FastQC` is used on **trimmed reads** to determine how effective the trimming process was.

Rationale: If the trimming process was effective, we should notice a better quality reads here, otherwise, we might have to adjust the trimming parameters.

Notes: Determination of whether the trimmed reads are ”clean enough” is more of an art rather than

actual science, though thresholds exist like Phred > 20 or Phred > 30 depending on how strict you are as a researcher.

Processing cleaned reads

2.3 Metagenomic Taxonomy

Taxonomy from Cleaned Reads

Technical Notes: `Kraken2` is used on **raw reads** to determine which species they came from.

Rationale: Taxonomy based on reads is standard on metagenomics instead of using assembled contigs because information is lost during the assembly process. By using cleaned reads we make sure that:

1. We are maximizing the amount of data
2. We are not being biased by low quality reads

Notes: There are many ways in which taxonomy is assigned to raw reads the `Kraken-based` packages use a curated database that links k-mers from your database to k-mers generated from their database (usually manually curated).

Notes: Others like `MetaPhlan` first create "markers" that are based off of known sequences, and then scan your raw reads for these markers, which it then checks for under what taxon/taxa that marker fell under.

2.4 Diversity analysis

Diversity analysis per site or sample

Technical Notes: Here `Bracken` - an extension of the `Kraken` packages or `Qiime2` are often used to calculate diversity. Instead I opted to create my own `Python` script `calculate_diversity.py` because:

1. I find that the diversity indices in either tools are limited to only the most often used i.e. the most popular indices - so it is not comprehensive
2. I've had problems integrating them into the pipeline because some dependency limitations, un-updated scripts, and a pre-processing step that requires converting all of `Kraken2, Bracken` files, then importing them to `Qimme2` which is too time-consuming and inefficient - my script just automatically calulates from `Bracken` outputs and just puts out all the possible (non-phylogenetic-based-which you would need a phylogenetic tree to build first) indices out there.

Rationale: Taxonomy based on reads is standard on metagenomics instead of using assembled contigs because information is lost during the assembly process. By using cleaned reads we make sure that:

1. We are maximizing the amount of data
2. We are not being biased by low quality reads

Notes: There are many ways in which taxonomy is assigned to raw reads the `Kraken-based` packages use a curated database that links k-mers from your database to k-mers generated from their database (usually manually curated).

Notes: Others like `MetaPhlan4` first create "markers" that are based off of known sequences, and then scan your raw reads for these markers, which it then checks for under what taxon/taxa that marker fell under.

2.5 Directory tree

```
/project_root/
├── configs/
│   └── config.yaml
├── **path/to/raw_reads_dir/** (Prerequisite - Raw Data)
│   ├── sample1_R1.fastq.gz
│   ├── sample1_R2.fastq.gz
│   ├── sample2_R1.fastq.gz
│   └── sample2_R2.fastq.gz
├── **path/to/trimmed_reads_dir/** (Prerequisite - Trimmed Data)
│   ├── sample1_R1_paired.fastq.gz
│   ├── sample1_R2_paired.fastq.gz
│   ├── sample2_R1_paired.fastq.gz
│   └── sample2_R2_paired.fastq.gz
├── path/to/fastqc_output_dir/
├── path/to/kraken_output_dir/
│   ├── sample1.k2report
│   ├── sample1.kraken2
│   ├── sample2.k2report
│   └── sample2.kraken2
├── path/to/bracken_output_dir/
│   ├── sample1.bracken
│   ├── sample1.breport
│   ├── sample2.bracken
│   └── sample2.breport
├── path/to/cleaning_results_dir/
│   └── summary_report.txt
├── logs/
│   └── calculate_diversity.log
└── path/to/output/
    └── diversity_matrices.tsv
```

3 trim_randomizer.smk

Stage: Done

Purpose: Randomization of trimming parameters

This is a module that will be part of a bigger pipeline. The idea here to is randomize parameters in a variety of trimmers in this case `Trimmomatic`, `fastp`, `CutAdapt`, `BBDuk`, and `Sickle` - famous bioinformatics trimming tools. `Trimmomatic` and `Sickle` in particular are widely used in `Illumina`-based data.

3.1 Random Parameter Generation

Technical Notes: Random parameters are generated for each trimming tool (Trimmomatic, Fastp, Cutadapt, BBDuk, Sickle). This is done using the `random module`, which creates random values for parameters such as quality scores, read length, adapter sequences, and error rates. These parameters are stored in the `generated_parameters` dictionary to ensure consistency across iterations for each sample.
Rationale:By randomizing parameters, the workflow allows for testing different parameter sets across multiple iterations to find optimal settings for trimming and quality control.
Notes: This approach helps with parameter exploration, particularly when you are unsure which trimming settings will give the best results. The randomness provides variability, which can highlight which parameters consistently lead to good results.

3.2 Log Parameters to a `TSV` File

Technical Notes: Each set of generated parameters is logged into a separate TSV file (e.g., `trimmomatic_params.tsv`, `fastp_params.tsv`). The file headers are written only once, and parameters are appended as the trimming steps proceed. This is done in a structured way so that you can track the exact parameters used for each sample and iteration.
Rationale:Logging ensures reproducibility and transparency in bioinformatics workflows. Having a record of all the parameter values used in each iteration is crucial for comparing results and for future reference
Notes: This practice is a standard in scientific workflows where random parameter generation is involved. It helps maintain a clear audit trail of the steps performed and aids in troubleshooting or refining workflows later.

3.3 Define `Rules` for Each Tool

Technical Notes: The script uses `Snakemake rules` to define separate rules for each tool which include

1. Input folder (as the script is designed to go through all the `FASTQ` samples within the folder)
2. Output folder, where the processed files will be saved (e.g., trimmed paired and unpaired reads). The script is designed to keep all the `trimmed reads` in separate files.
3. Params: Fetches the parameters to be randomized.

Rationale:Using Snakemake here allows for parallel execution of the workflow. This parallelization if very important as the generation of random paramaters is created using a random `seed`. Using a random `seed` like this allows us to replicate what the parameters that had the optimal results were by tracking down what seed was assigned as dictated in the `TSV` file.
Note keep in mind that this will create a large number of folders if you decide to iterate many times - as each iteration, per tool, will have its own folder full of trimmed reads, per sample/site.

3.4 Interpretation and Analysis of Results (TBA)

Technical Notes: Once all iterations have completed, the trimmed files can be analyzed to determine which parameters led to the best results in terms of quality and length distribution of reads. There are many different metrics that can be used to interpret the results, including (but not limited to)

1. Quality Scores - significant differences in quality among sites and across entire sequences (Phred score, Contamination, Adapter Removal, N Content, Length Distribution, etc.). Can be done using tools like `FastQC`

2. Visualization can be done using `Rstudio` or `Python's matplotlib` to visually look for differences in abnormalities.

Rationale:Evaluating read quality and assessing key metrics post-trimming helps to ensure that the data is suitable for downstream analyses. Optimal trimming should maximize the number of high-quality, usable reads while eliminating low-quality bases and adapter contamination.
Notes See Further Investigations `Box on Biological Information` for more possible details on this.

3.5 Annotated Directory Tree with File Temporary files and Prerequisites

```
/project_root/
├── raw_reads/ (permanent) **(prerequisite)**
│   ├── sample1_R1.fastq.gz (permanent) **(prerequisite)**
│   ├── sample1_R2.fastq.gz (permanent) **(prerequisite)**
│   ├── sample2_R1.fastq.gz (permanent) **(prerequisite)**
│   └── sample2_R2.fastq.gz (permanent) **(prerequisite)**
└── output_dir/
    ├── fastp_output/ (temporary) (to be deleted)
    │   ├── iteration_1/
    │   │   ├── sample1_R1_fastp_trimmed.fastq.gz (temporary)
    │   │   ├── sample1_R2_fastp_trimmed.fastq.gz (temporary)
    │   │   ├── sample2_R1_fastp_trimmed.fastq.gz (temporary)
    │   │   └── sample2_R2_fastp_trimmed.fastq.gz (temporary)
    │   ├── iteration_2/
    │   │   └── (same as iteration_1 but with iteration_2 files) (temporary)
    │   └── iteration_3/
    │       └── (same as iteration_1 but with iteration_3 files) (temporary)
    └── trimmomatic_output/ (temporary) (to be deleted)
```

```
    ├── iteration_1/
    │   ├── sample1_R1_paired.fastq.gz (temporary)
    │   ├── sample1_R1_unpaired.fastq.gz (temporary)
    │   ├── sample1_R2_paired.fastq.gz (temporary)
    │   ├── sample1_R2_unpaired.fastq.gz (temporary)
    │   └── (same for sample2) (temporary)
    ├── iteration_2/
    │   └── (same structure as iteration_1) (temporary)
    └── iteration_3/
        └── (same structure as iteration_1) (temporary)
├── cutadapt_output/ (temporary) (to be deleted)
    ├── iteration_1/
    │   ├── sample1_R1_cutadapt_trimmed.fastq.gz (temporary)
    │   └── sample1_R2_cutadapt_trimmed.fastq.gz (temporary)
    ├── iteration_2/
    │   └── (same structure as iteration_1) (temporary)
    └── iteration_3/
        └── (same structure as iteration_1) (temporary)
├── bbduk_output/ (temporary) (to be deleted)
    ├── iteration_1/
    │   ├── sample1_R1_bbduk_trimmed.fastq.gz (temporary)
    │   └── sample1_R2_bbduk_trimmed.fastq.gz (temporary)
    ├── iteration_2/
    │   └── (same structure as iteration_1) (temporary)
    └── iteration_3/
        └── (same structure as iteration_1) (temporary)
├── sickle_output/ (temporary) (to be deleted)
    ├── iteration_1/
    │   ├── sample1_R1_sickle_trimmed.fastq.gz (temporary)
    │   ├── sample1_R2_sickle_trimmed.fastq.gz (temporary)
    │   └── sample1_singles_sickle_trimmed.fastq.gz (temporary)
    ├── iteration_2/
    │   └── (same structure as iteration_1) (temporary)
    └── iteration_3/
        └── (same structure as iteration_1) (temporary)
└── logs/ (permanent) **(prerequisite)**
    ├── fastp_params.tsv (permanent) **(prerequisite)**
    ├── trimmomatic_params.tsv (permanent) **(prerequisite)**
    ├── cutadapt_params.tsv (permanent) **(prerequisite)**
    ├── bbduk_params.tsv (permanent) **(prerequisite)**
    └── sickle_params.tsv (permanent) **(prerequisite)**
```

4 bootstrapping_rawreads.smk

Stage: Further refinement

Purpose: Bootstrapping the taxo_metagenomic pipeline itself

Bootstrapping is the process of randomly selecting from a pool of samples (with replacement) and using that in a specific process you want to bootstrap. This is a standard method used in molecular phylogenetics to determine the robustness of trees where a > 70 support from bootstrapped data is considered robust enough.

The principle of bootstrapping in phylogenetics

Phylogenetics uses this statistical technique because (in principle) it effectively means that removing parts of the entire sequence does not alter the topology of the tree.
Here I'm adapting this method with `Kraken2`'s taxonomic profiling to see whether the taxonomic support of its k-mer assignment is also consistent even with changes in the sampling sites.
This is still WIP because I plan to go step further and start bootstrapping the raw reads themselves to see if changes in reads changes the topology of taxonomic assignment.

4.1 Directory setup of temporary files

Technical Notes:Snakemake starts by ensuring the existence of necessary directories. Most notably, the temporary bootstrap directories.
Rationale: Bootstrapping is sometimes done iteratively thousands of times, so making this a temporary directory helps manage space.
Notes:

4.2 Sample Identification

Technical Notes:The workflow identifies sample names by parsing filenames in the raw reads directory.
Rationale: Inclusion of these in the script allows the user to flexibly configure the naming convention and the directory in which they want to bootstrap.
Notes: Presently it looks for _R1.fastq.gz and it's associated pair, _R2.fastq.gz in the raw_reads directory.

4.3 Bootstrapping

Technical Notes:This executes `bootstrap_reads.py` in the scripts directory to start bootstrapping the paired-end reads and outputs them in the temporary folders I mentioned earlier.
Rationale: Moving bootstrapping logic into a Python script leverages its ability to create randomizations from its libraries. Also it allows us to define a `seed` so it is reproducible (if you want to reproduce) the results anyway.
Note: The number of bootstraps and the fraction of samples you want to retain can be controlled in the config.yaml file in the configuration folder.
On a personal note: Before writing this part of the document (Sep 19, 2024, 3:00 AM), I decided to change the bootstrapping rule. It used to rely on a simplified approximation. The probability of not being selected after N independent draws from a sample of size N is given by:

$$P = \left(1 - \frac{1}{N}\right)^N$$

This is a well-known mathematical equation describing the probability of not selecting a sample at least once in N draws. As $N \to \infty$, this probability approaches:

$$\frac{1}{e}$$

Previously, I used the probability of a sample *being selected*, which is:

$$1 - \frac{1}{e}$$

This was used as an approximation for bootstrapping by shuffling and adjusting the sample size. However, the updated script now performs **actual bootstrapping**, sampling with replacement, which is a more accurate statistical method for resampling.

4.4 Annotated Directory Tree with File Movement and Prerequisites

```
/project_root/
├── configs/
│   └── config.yaml **(prerequisite)**
├── path/to/raw_reads_dir/ (permanent) **(prerequisite)**
│   ├── sample1_R1.fastq.gz (permanent) **(prerequisite)**
│   ├── sample1_R2.fastq.gz (permanent) **(prerequisite)**
│   ├── sample2_R1.fastq.gz (permanent) **(prerequisite)**
│   └── sample2_R2.fastq.gz (permanent) **(prerequisite)**
└── path/to/temp_bootstrap_dir/ (temporary) (to be deleted after pipeline resolves)
    ├── sample1/ (temporary)
    │   ├── rep_1_R1.fastq.gz (temporary)
    │   ├── rep_1_R2.fastq.gz (temporary)
    │   ├── rep_2_R1.fastq.gz (temporary)
    │   ├── rep_2_R2.fastq.gz (temporary)
    │   └── total_reads.txt (temporary)
    ├── sample2/ (temporary)
    │   ├── rep_1_R1.fastq.gz (temporary)
    │   ├── rep_1_R2.fastq.gz (temporary)
    │   ├── rep_2_R1.fastq.gz (temporary)
    │   ├── rep_2_R2.fastq.gz (temporary)
    │   └── total_reads.txt (temporary)
    └── rep_1/ (temporary)
        ├── diversity_matrices_sample1_rep_1.tsv (temporary) (moved to diversity_results/)
        └── diversity_matrices_sample2_rep_1.tsv (temporary) (moved to diversity_results/)
```

```
    ├── rep_2/ (temporary)
    │   ├── diversity_matrices_sample1_rep_2.tsv (temporary) (moved to diversity_results/)
    │   └── diversity_matrices_sample2_rep_2.tsv (temporary) (moved to diversity_results/)
    ├── logs/ (permanent) **(prerequisite)**
    │   ├── metagenomics_pipeline_sample1_rep_1.log (permanent) **(prerequisite)**
    │   ├── metagenomics_pipeline_sample1_rep_2.log (permanent) **(prerequisite)**
    │   ├── metagenomics_pipeline_sample2_rep_1.log (permanent) **(prerequisite)**
    │   └── metagenomics_pipeline_sample2_rep_2.log (permanent) **(prerequisite)**
    ├── diversity_results/ (permanent) (contains moved files) **(prerequisite)**
    │   ├── diversity_rep_1.tsv (moved from temp_bootstrap_dir)
    │   └── diversity_rep_2.tsv (moved from temp_bootstrap_dir)
    └── aggregated_results/ (permanent) **(prerequisite)**
        └── diversity_aggregate.tsv (permanent)
```

4.5 Run `kraken_pipeline.bash`

Technical Notes: This Shellscript (or bash file) is used to automate the processing of all files from the bootstrapping. It runs them under `Kraken2` then `Bracken` to generate taxonomy profiles for all of them. It also creates a log file for each replicate to provide traceability and error checking, helping diagnose any issues with specific replicates or samples.

Rationale: Read why I'm bootstrapping from the textbox above. The reason why I also included `Bracken` and measurements diversity metrics in the analysis per sampling replicate is so we can analyze how the topology of diversity also changes - similiar to how we look at topology of phylogenetic trees. The final process consolidates all the diversity metrics (alpha diversity) and matrices (beta diversity) into a TSV file.

Note: The decision to use Snakemake and Shellscripts here is so that the bootstrapping comes first before the pipeline is introduced. Otherwise Snakemake will run the entire thing in parallel, taking up so much memory because it runs `Kraken2` for every single sample instead of doing it in batches - which takes up so much unnecessary time.

5 Binning.smk

Stage: To test on higher coverage data
Binning pipeline to create high quality (Metagenome Assembled Genomes) MAGs
Specifics: This pipeline passes through multiple quality checks during binning of using a variety of tools (both wrappers and modules) including `MetaWrap`, `DasTool`, `CheckM2`, `MagPurify` etc.

5.1 Universal Configurations

Technical Notes: The pipeline allows the user to configure settings they want for the binning process. By default, the settings are Minimum Contig Length (2500bp), Completeness (50%), Contamination (10%).
Rationale: The default settings were curated by me and the reason I chose them is because of the following

1. Longer contigs tend to represent more complete genomic fragments. Setting a threshold of 2500bp ensures better assembly quality. Others prefer a lower threshold for more sensitivity like 2000 bp.

2. Completeness 50% and Contamination 10% are actually based from a standard called the `MIMAG` standards.

Notes: Making configurations universal this way creates consistency across the script, i.e. when specifically asked by a specific tool, this returns a universal parameter.
Notes: Additionally, the user can specify the memory usage and number of threads they want to allocate per tool as well as other tool-specific parameters at the top of the script for ease of use. *I plan to add this to the configuration file soon once I have tested the file to be working at higher coverage - since you can't make high quality bins with low read counts - and my PC can't practically handle that sorry.*

5.2 `FastUniq` (Deduplication)

Technical Notes: `FastUniq` used to remove duplicate reads.
Rationale: Since we are focused on creating MAGs or genomes based on populations of genomes, removing duplicates is less risky during binning and is thus included. It also allows us to completely remove amplification bias from `PCR` reactions. Moreoever, deduplication reduces redundancy and thereby memory usage downstream.

5.3 `Seqtk` (`FASTA` Conversion)

Technical Notes: Seqtk converts FASTQ to FASTA.
Rationale: This conversion prepares sequences for tools that require FASTA inputs, such as `CD-HIT-EST`.
Notes: I used to include a decompress-then-compress mechanism in the script to minimize memory storage but according to my calculations from the sequencing facility quotations, re-compressing may actually be more costly when done throughout the pipeline. Hence, it should be more cost-efficient to start compressing files ones the bins are done.

5.4 `CD-HIT-EST` (Clustering) at Identity: 90%

Technical Notes: `CD-HIT-EST` clusters similar sequences at 90% identity.
Rationale: Clustering reduces redundancy in the contig data while maintaining closely related sequences.
Notes:I chose 90% ID because that's what I often see in published journals that is all Perhaps, the 90% identity threshold balances removing duplicates while preserving diversity and that higher thresholds would result in fewer clusters but might oversimplify the data. *Fine, I'll make it my homework assignment why this specific threshold is used (I HAVE TO KNOW).*

Bin Refinement

5.5 `MetaWRAP` Binning and Reassembly

Technical Notes: `MetaWRAP` is what is known as a wrapper program - meaning it makes use of other tools as its modules. For the binning process in particular it uses 3:`MetaBAT2`, `MaxBin2`, and `CONCOCT`. Each binning process goes through internal quality control checks and the one with the best bin-qualities are selected. It also has a reassembly feature wherein it reassembles the contigs again to try and further refine the bins.
Rationale: Using 3 binners in parallel and choosing the best bins, quality checking, and then reassembling (not-so-good) bins make the binning process very robust, creating very refined bins *not sponsored by the way, talking as a fellow researcher*.
Notes: No moving forward, the process of further refinement of bins seems redundant. But do note that I have checked and validated that the process used in refining and checking by the tools used here cover different metrics - and therefore can be seen as parallel processes.

5.6 DAS Tool (Bin Refinement)

Technical Notes: `DAS_Tool` is very similar to `MetaWrap` in that they both choose the best bins from a pool of bins from different binners (in this pipeline DAS Tool is designed to **ALSO** use information from `MetaBAT2`, `MaxBin2`, and `CONCOCT` outputs to improve binning)
Rationale: However, as rationale for including it, is that it focuses more on single-copy gene (SCG) analysis. In contrast, in `MetaWrap`, bins are evaluated using completeness and contamination thresholds.
Notes: Notably, in the checking phase of this pipeline we will not be using DAS_Tool for SCG analysis. It is optimized for refining bins not quality checking bins. Instead we will use a more updated and optimized software for the latter called `BUSCO`.

5.7 MAGpurify (Contamination Removal)

Technical Notes: MAGpurify is also a bin refiner, in a sense that it uses several modules to detect and **prune** contamination in genome bins. It also uses other metrics to define bin quality specifically it looks for differences in

1. **Phylo-markers**,

2. **Clade-markers**,
3. **Tetranucleotide-frequencies**,
4. **GC-content**,
5. and then removes **known-contaminants** from it's manually curated database (created back in 2013)

Rationale: This step improves genome quality by removing low-confidence contigs or contamination from other taxa. Each module targets different contamination types (phylogenetic, clade, etc.).
Notes: Similar to `DAS_tool`, `MAGPurify` is relatively old (in the bioinformatics world where new tools are being published every day). So in checking our bins we will be using more recently updated tools.

Quality checking

5.8 `MetaQUAST` (Assembly Quality Assessment)

Technical Notes: `MetaQUAST` assesses the quality of genome assemblies via the following:

1. N50 and L50 to determine contiguity
2. Number of contigs to determine fragmentation
3. GC content - since a single MAG should have a constant GC content across its entirety (usually)
4. Alignment to a reference sequence

   Additionally, it also detects other metrics using modular tools
5. structural variations (requires `GRIDSS`)
6. presence or rRNA (requires `SILVA`)
7. Conserved gene sets (requires `BUSCO`)

Rationale: This step quantifies the completeness and accuracy of the assembled genomes, and is updated frequently.
Notes: Using reference genomes improves the accuracy of the assessment, but it's optional if references are unavailable.

A Personal Note

Personal Note: As of this writing, `BUSCO` has updated beyond the version required by `QUAST` (`BUSCO` od9). Unfortunately, this version is not available in the archives (you'll encounter a 404 error). Likewise, `SILVA` and `GRIDSS` frequently update. I recommend downloading each separately and manually linking their databases to avoid potential issues with `QUAST`'s download management. *Good luck and have fun!*

5.9 `dRep` (Dereplication)

Technical Notes: dRep dereplicates genomes by clustering them based on similarity.
Rationale: Dereplication reduces redundancy in the assembled genome data, ensuring unique genome representations. Basically `CD-HIT` but for whole genomes.
Notes: `FastANI` is utilized for quick, precise clustering, and is part of the `dRep` package. It defaults to a 95% ANI (Average Nucleotide Identity) threshold, a common yet somewhat subjective metric used to determine microbial species boundaries. This is often sufficient for microbial genomes due to their high gene density, frequently organized in operons. However, it may not be as suitable for eukaryotic genomes, which are laden with repetitive elements.

A Personal Note

Personal Note: I cannot find a newer version of either dRep or FastANI (both dating back to 2013, basically ancient in bioinformatics terms). There's a newer version called `pyani`, which is available in `Bioconda`, that might be a good replacement. However, I still need to reverse-engineer its source code to fully understand how it operates. Might be worth trying!

5.10 `CheckM2`

Technical Notes: `CheckM2` is used to predict genome completeness and contamination using `low-memory mode` (essential for resource-limited systems like mine; remember to adjust this on HPC systems).
Rationale: `CheckM2` is an updated version of `CheckM`, but many tools in this pipeline haven't been updated to recognize it. I haven't tested whether aliasing `CheckM2` as `CheckM` would work, so it's added here as a final step to ensure the results meet `MIMAG` standards for completeness and contamination.

5.11 Annotated Directory Tree with File Movements and Temporary Files

```
/project_root/
├── trimmed_reads/ (prerequisite)
│   ├── site1_R1_paired.fastq.gz (permanent)
│   ├── site1_R2_paired.fastq.gz (permanent)
│   ├── site2_R1_paired.fastq.gz (permanent)
│   └── site2_R2_paired.fastq.gz (permanent)
├── fastuniq/ (temporary) (to be deleted)
│   ├── site1_R1_uniq.fastq (temporary)
│   ├── site1_R2_uniq.fastq (temporary)
│   ├── site2_R1_uniq.fastq (temporary)
│   └── site2_R2_uniq.fastq (temporary)
├── fasta/ (permanent)
│   ├── site1_R1_clean.fasta (permanent)
│   ├── site1_R2_clean.fasta (permanent)
│   ├── site2_R1_clean.fasta (permanent)
│   └── site2_R2_clean.fasta (permanent)
└── megahit_output/ (permanent)
```

```
    ├── site1_assembly/
    │   └── site1_filtered_contigs.fa (permanent)
    └── site2_assembly/
        └── site2_filtered_contigs.fa (permanent)
├── cdhit_contigs/ (permanent)
    ├── site1_cdhit_contigs.fasta (permanent)
    └── site2_cdhit_contigs.fasta (permanent)
├── CLEAN_READS/ (permanent) (moved)
    ├── site1/
    │   ├── site1_1.fastq (moved from fastuniq)
    │   └── site1_2.fastq (moved from fastuniq)
    └── site2/
        ├── site2_1.fastq (moved from fastuniq)
        └── site2_2.fastq (moved from fastuniq)
├── binning_output/ (permanent)
    ├── site1_binning_output/
    │   ├── concoct_bins.scaffolds2bin.tsv (permanent)
    │   ├── metabat2_bins.scaffolds2bin.tsv (permanent)
    │   └── maxbin2_bins.scaffolds2bin.tsv (permanent)
    └── site2_binning_output/
        └── (same as site1) (permanent)
├── dastool_output/ (permanent)
    ├── DAS_Tool_bins_site1 (permanent)
    └── DAS_Tool_bins_site2 (permanent)
├── reassembly_output/ (permanent)
    ├── site1_reassembly_output/
    │   └── assembly.fasta (permanent)
    └── site2_reassembly_output/
        └── assembly.fasta (permanent)
├── magpurify_output/ (permanent)
    ├── site1_cleaned.fasta (permanent)
    └── site2_cleaned.fasta (permanent)
├── metaquast_output/ (permanent)
    ├── site1_metaquast_output/
    │   └── (MetaQUAST results) (permanent)
    └── site2_metaquast_output/
        └── (MetaQUAST results) (permanent)
├── dereplication_output/ (permanent)
    └── dereplicated_genomes/
        ├── dereplicated_genomes.fasta (permanent)
        └── checkm2_output/ (permanent)
            └── quality_report.tsv (permanent)
```

```
└── logs/ (permanent)
    ├── run_megahit_site1.log (permanent)
    ├── run_cdhit_site1.log (permanent)
    └── (logs for each step) (permanent)
```

6 krakenpipeline.bash

Stage: Done

The entire basic taxo-metagenomic pipeline with the usual tools

Specifics: This pipeline passes through a loop between `FastQC` and `Trimmomatic` `trimming_cleaning_checking.py` before progressing to `Kraken2` then `Bracken` then `calculate_diversity.py`.

Notes: This does not include `FastQC`-ing of raw reads (yet), you have to run `raw.bash` for that. Additionally, it also currently does not automate the production of aggregated summaries of the FastQC files for you (yet), you have to run `summary_stat.bash` for that as well.

6.1 Directory tree

```
/project_root/
├── trimmed_reads/ **(Prerequisite - Trimmed reads directory)**
│   ├── sample1_R1_paired.fastq.gz
│   ├── sample1_R2_paired.fastq.gz
│   ├── sample2_R1_paired.fastq.gz
│   └── sample2_R2_paired.fastq.gz
├── kraken_db/ **(Prerequisite - Kraken2 database directory)**
│   └── database files (.k2d, etc.)
├── kraken_output/ (Output directory for Kraken2 reports)
│   ├── sample1.k2report
│   ├── sample1.kraken2
│   ├── sample2.k2report
│   └── sample2.kraken2
└── bracken_output/ (Output directory for Bracken reports)
    ├── sample1.bracken
    ├── sample1.breport
    ├── sample2.bracken
    └── sample2.breport
```

7 raw.bash; completely optional

Stage: Done

`FastQC` **Automation script to run** `FastQC` **on raw reads**

Specifics: This script takes raw reads and creates a summary report for each of them in FasQC_raw/

Notes: `summary_stat.bash` can already do that for you, so it's a bit redundant. **But if you're only interested in seeing QC from the raw reads, then have fun!**

Personal notes: *The only reason I am keeping this alive is because I am not quite sure yet if some pipelines rely on this and it's corresponding output files;* ***I have to check.***

8 summary_stat.bash

Stage: Done

`FastQC` **Automation script**

Specifics: This script takes reads from the `raw reads` directory and the `trimmed reads` directory, then creates a summary directory (if not already existent) (you can manually set this as user). The summary directory will contain summary statistics extracted from FastQC sub-directories, it also creates. It has two notable functions:

1. Counts the `number of reads` in raw and trimmed directories.
2. Extract quality metrics from `FastQC` subdirectories specifically: `per base sequence quality`, and `per sequence quality scores`.

It then saves them into a file the summary subfolder.

Notes: The way it counts reads from `FASTQ` file is by reading the number of lines and dividing it by 4 (since each read in a FASTQ file consists of 4 lines: sequence identifier, sequence, plus sign, and quality score).

Notes: This does not include config file integration yet (TBA).

Notes: Not integrated into any pipeline yet (TBA).

8.1 Directory tree

```
/project_root/
├── **raw_reads/** (Prerequisite - Raw Data)
│   ├── sample1.fastq.gz
│   ├── sample2.fastq.gz
│   └── ...
├── **trimmed_reads/** (Prerequisite - Trimmed Data)
│   ├── sample1_trimmed.fastq.gz
│   ├── sample2_trimmed.fastq.gz
│   └── ...
└── summary_statistics/
    ├── fastqc_raw/
    │   ├── sample1_fastqc.zip
    │   ├── sample2_fastqc.zip
    │   └── ...
    ├── fastqc_trimmed/
    │   ├── sample1_trimmed_fastqc.zip
    │   ├── sample2_trimmed_fastqc.zip
    │   └── ...
    └── extracted_metrics/
```

```
    ├── sample1_base_quality.txt
    ├── sample1_sequence_quality.txt
    └── ...
├── raw_read_counts.txt
└── trimmed_read_counts.txt
```

9 plot_and_detect_intermediate_coverage.py

Stage: Done

Processing of coverage data

Specifics: This script takes the coverage data from a coverage file, which is expected to have 3 columns: `chrom`(chromosome/contig name), `pos`(position along the contig), `cov`(coverage value at that specific position). It then takes the latter two columns and uses `matplotlib` to create a line plot of the coverage values - and saves it as an image.

Notes: Presently, it determines, intermediate coverage by looking first looking for the minimum and maximum value of coverage data.

Sept 20 2024 Update

I updated this one here to be a bit more robust in detection of chimeric contigs. Instead of visual inspection (which is a bit too subjective for my taste), I added 4 new features to this

1. Sliding window approach to detect changes coverage (rather than visualization alone) other than that we apply sliding window to detect changes in:
   (a) Codon usage bias
   (b) GC content
   (c) Tetranucleotide frequencies
2. Each of these ”windows” are then subjected to ANOVA, to determine statistical significance and
3. PCA, that combines all the (3) features to generate a 2D plot which to see whether different sections of the contig cluster differently - which would indicate likely chimerism.

10 Shortbred_ARG.bash

Stage: Back to drawing board and to test

Specifics: Marks trimmed reads with ARGs **Technical Details** This script takes cleaned reads (dicated by the user), unzips them (to FASTQ), then starts finding ARG-markers on them. Markers are determined by the function `shortbred_identify.py` (instrinsic to the tool), taking first the `CARD` database and aligning it with the `RefSeq`. After that, it clusters the `RefSeq` database at 95% similarity with those on the CARD database, to create markers. After that, it uses another function `shortbred_quantify.py`

Notes: I included an additional step here that filters out specific keywords from the `FASTQ` files from a reference presently. It called `filter_keywords.txt` which you can make yourself should you want to filter out specific ARGs that you consider "low-confidence", or just create the file and leave it blank (up to you).

Sept 20 2024 Update

I am revisiting this process because:

- It turns out that ShortBRED is typically used on **contigs**, not raw reads. This makes sense because marker-based analyses are more accurate on contigs, as they reduce false positives and provide a clearer view of the genetic background.
  Note: ShortBRED is used to quantify **genes**, and raw reads are fragmented sequences it is the **CONTIGS THAT CONTAIN GENES.**
- The current method of filtering low-confidence ARGs is inefficient: it creates a temporary file full of ARGs with the keywords, and then filters the marker database using those. There must be a more efficient way to skip this unnecessary step of generating temporary FASTQ files.

I have also streamlined the script to:

- Unzip FASTQ files in parallel.
- Remove the temporary file afterwards.
- Focus only on high-confidence ARGs. Previously, it also produced markers on the RefSeq database, but checking marker percentages turned out to be inefficient and pointless.
- Add checks when directory creation or unzipping fails.
- Use a consistent naming scheme.
- and finally, it now **includes MEGAHIT assembly** and a filtering step that removes contigs below 200 bp before ShortBRED processing.

The most important addition is the conversion from `FASTQ` to `FASTA` using `seqtk`. I had overlooked this step, as some tools cannot process `FASTQ` format.

Personal Note: Dear Reader, please be aware that ShortBRED is quite particular about FASTA headers. It requires the headers to be numeric identifiers, which adds an extra step of renaming all sequences to numbers. You'll also need to create an index to trace the original sequence headers back to their numeric counterparts.
While troubleshooting, I encountered persistent naming errors that were confusing at first. After diving into the source code, I eventually discovered that ShortBRED requires headers to be strictly numeric. This unexpected requirement added to the complexity of the process and was not immediately obvious.

11 refseq_bacteria.bash

Stage: Done

`FastQC` **Automation script to run download from the bacteria** `RefSeq`

Specifics: This only downloads the ones that end in `.faa` as opposed to `.fna` - the latter are genomes, the former are just all the annotated proteins themselves, with a genome or not.

Notes: This script is kept here because it was difficult to find the link to the `FTP site`, because `NCBI` seemed to have had a recent restructuring.

Sept 20 2024 Update

Removed the hard-cap of 1-511.faa files, to add flexibility; now downloads without the need for the user to check how many to download. Also now uses aria2c, which has and intrinsic `redownload` unfinished files instead of `wget` function.

Project Side Scripts

```
1 # 1
```

Project Main Scripts

```
1 # 1
```

Further investigations

This section comprises of concepts or ideas that require further investigation. They are written in boxes to help categorize them cleanly. I will refer to this section often as no script is perfect - and can be further improved.

Biological Information

I have long pondered upon the idea of creating a graph with `biological information` on the y-axis and `read length` on the x-axis. I hypothesize that we will find a **"sweet spot"** wherein we can optimize the amount of `information/read` using such trimmers. Unfortunately, it is extremely difficult to define what `a biological sequence` really is, because technically you can generate any random sequence - whether protein or nucleotide. That's the main reason I have been stuck on this problem for quite a while, I've been trying to find the answer first.

The core difficulty here lies in defining what constitutes biological information in a meaningful, quantifiable way. Since any random sequence of nucleotides or proteins could be technically "valid" (false-positives).

When people are asked this question they often give **DESCRIPTIONS** of what a `biological sequence` is but not what **DEFINES** a `biological sequence`. I understand that there are many characteristics that can help in determining the signal from the noise, but I am just not satisfied with whether this "thing" checks most (if not all) the boxes - I need a non-subjective answer to this problem.

Daily entries

1 September 23, 2024

To-do List
Note:

- ☐ Call for a group meeting regarding logistics (or when they will be available via Zoom call)
  - ☐ Discuss the issue about VMMC review fee
  - ☐ Raise concerns about data storage and management
  - ☐ Inquire if there are scripts I can develop to streamline the bureaucratic process
- ☐ Contact engineering/sanitation departments via landline
  - Talk about possible sampling sites and that we will present out authorization upon arrival
  - ☐ VMMC
    - Tell them that we are have a go-signal from the admin, but we have yet to pay for the fee - which can be held off for later
  - ☐ Mary Johnston
    -
  - ☐ ManilaMed
    - Tell them we already have a go signal from Ma'am Eula
  - ☐ St. Lukes
    - Inquire where Engr. Valenzuela is there
    - Inquire the status of their renovation and if they are available for sampling this October already as talked about last year

Completed Tasks

✓

✓

✓

Scripts Worked On

✓

Issues Encountered

-

Need to Troubleshoot

-

2 September 20, 2024

To-do List

To do list

☐ Project4 Modules

- ✓ krakenpipeline.bash
- ✓ raw.bash (optional)
- ✓ summary_stat.bash
- ✓ plot_and_detect...py
- ✓ refseq_bacteria.bash

✓ Send the signed AR, PCF, and COS documents as pdf

Completed Tasks

✓ Created directory trees template for each script (if relevant, which includes prerequisites and temporary folders)

✓ Restructuring of this Documentation to include Further Investigations, Supplementary Data, and Information section for more streamlined reading of the script portions.

Scripts Worked On

✓ Improved plot_and_detect...py for more robust analysis (Details on its own section)

✓ Drafted a fix to the Shortbred_ARG.bash shell (Details on its own section)

Issues Encountered

- Landline-based to-do list moved to next week due to landline down and it's weekend tomorrow so offices will be closed.

Need to Troubleshoot

-

3 September 19, 2024

To-do List

- ☐ Try to complete LaTeX documentation of scripts
  - ✓ Project 4 pipelines
    - ✓ ARG-MGE.smk
    - ✓ metagenomics_pipeline.smk
    - ✓ Binning_.smk
  - ➔ Project Main pipelines
  - ➔ Project Side pipelines
- ✓ Counter-check Project4 scripts with documentation to see redundancies

Completed Tasks

- ✓ Deleted redundancies in Project 4 scripts
  - FastQC_check.py
  - fastp.bash
  - README.md (since this document exists and is more comprehensive)
  - fastp metrics from summary_stat.bash
- ✓ Decided to move to Project Side K-mer_diversity.smk as it is not essential (yet).
- ✓ Finally resolved an issue with wakatime not integrating properly with TeXStudio
- ✓ I realize now that the entry dates were one-day behind.
- ✓ Also added the names of the python scripts I refer for proper documentation.

Scripts Worked On

- calculate_contig_quality.py now updated to calculate coverage statistics.

Issues Encountered

- Landline-based to-do list moved due to landline down

Need to Troubleshoot

-

4 September 18, 2024

To-do List

- 🡒 Call for a group meeting regarding logistics
    - ✓ Offer to use landline to minimize on-site visits and reduce transportation costs; as point-persons have not been replying via email as quickly lately
    - ✓ Discuss PGC calculations and talk about allocatable budget
    - ✓ On-site orientation with PGC engineering department (OETS) regarding sampling sites **taken care of by James and Roch**
    - 🡒 Raise concerns about data storage and management
    - 🡒 Inquire if there are scripts I can develop to streamline the bureaucratic process

Completed Tasks

- Setup wakatime in VS Code and Ubuntu to track my coding productivity
- Helped in receiving and moving the autoclave and fridge needed by the Program

Scripts Worked On

- ~~rawAPIformatter.sh~~

✓ checkingAPI.sh

- ~~wakatime.sh~~

Issues Encountered

✓ Connection blockage from wakatime API

🡒 raw API results is unreadable

Need to Troubleshoot

- conky display of wakatime API for desktop

5 September 17, 2024

To-do List

- Call for a group meeting regarding logistics
  - Offer to use landline to minimize on-site visits and reduce transportation costs; as point-persons have not been replying via email as quickly lately
  - Discuss PGC calculations and talk about allocatable budget
  - Raise concerns about data storage and management
  - Inquire if there are scripts I can develop to streamline the bureaucratic process

Completed Tasks

✓ Updated GitHub repositories; created new repositories: `Documentation` and `Confidential`

✓ Created a new bash script to track progress across all repositories

✓ Set up this research diary to document daily progress and tasks

✓ Python script that calculates contig quality metrics L50, N50, but also L90, N90, GC skew etc.

✓ Python script that detects and calculates changes in coverage over contigs for read mapping

✓ Typesetting: explaining the process and rationale inside the ARG-MGE.smk pipeline

✓ Integration of the two scripts to the ARG-MGE pipeline

Scripts Worked On

- `Project4/ARG_MGE.smk` - updated

✓ `Project4/calculate_contig_quality.py`

✓ `Project4/plot_and_detect_intermediate_coverage.py`

Issues Encountered

✓ Could not access GitHub, needed to reset the SSH keys and update my Git histories for version control

Need to Troubleshoot

- None identified at this time

Supplementary data

This will contain data on my investigations on standardized or simulated datasets. This is mainly a testing ground for the scripts, and their accompanying data visualizations.

Information dump

This is part is mainly created as an information dump outside of just bioinformatics, that perhaps we can one day apply to bioinformatic data analysis (mostly). This part could also serve as an index of papers that I've read regarding various topics that seem interesting to me and that I think can be applied in the wider field of computational or mathematical biology.