Metagenomic Analysis Pipeline

A comprehensive SLURM-based pipeline for metagenomic sequencing data analysis, from raw reads to high-quality metagenome-assembled genomes (MAGs).

Table of Contents

Overview

This pipeline processes paired-end metagenomic sequencing data through a complete workflow:

1. **Quality Filtering** - Remove adapters and low-quality reads using Trimmomatic
2. **Assembly** - Assemble contigs using MetaSPAdes with singleton support
3. **Plasmid Detection** - Identify plasmid sequences using PlasClass and MOB-suite
4. **Binning** - Generate genome bins using MetaBAT2, MaxBin2, and CONCOCT
5. **Bin Refinement** - Refine bins using MetaWRAP
6. **Bin Reassembly** - Reassemble bins for improved quality
7. **MAGpurify** - Remove contaminating contigs from bins
8. **CheckM2** - Assess bin quality and completeness
9. **CoverM** - Calculate bin abundance and coverage
10. **Bin Collection** - Collect and categorize high-quality bins
11. **Treatment Analysis** - Combine results across treatments
12. **Final Report** - Generate comprehensive reports

Pipeline Stages

Stage	Name	Description	Tools Used
0	Quality Filtering	Remove adapters, filter low-quality reads	Trimmomatic
1	Assembly	Assemble contigs from filtered reads	MetaSPAdes
2	Plasmid Detection	Identify plasmid sequences	PlasClass, MOB-suite
3	Binning	Generate genome bins	MetaBAT2, MaxBin2, CONCOCT
4	Bin Refinement	Refine and improve bins	MetaWRAP
5	Bin Reassembly	Reassemble bins for better quality	SPAdes
6	MAGpurify	Remove contaminating contigs	MAGpurify
7	CheckM2	Assess bin quality	CheckM2
8	CoverM	Calculate abundance and coverage	CoverM
9	Bin Collection	Collect high-quality bins	Custom scripts
10	Treatment Analysis	Combine results by treatment	Custom scripts
11	Final Report	Generate comprehensive reports	Custom scripts

Prerequisites

System Requirements

- Linux-based HPC system with SLURM job scheduler
- Conda/Miniconda package manager
- Python 3.6+ with pandas and openpyxl
- At least 256GB RAM per node (recommended)
- 32+ CPU cores per node (recommended)

Software Dependencies

All software is managed through conda environments:

- **Trimmomatic** - Quality filtering
- **SPAdes** - Assembly and reassembly
- **PlasClass** - Plasmid classification
- **MOB-suite** - Plasmid detection
- **MetaBAT2** - Genome binning
- **MaxBin2** - Genome binning
- **CONCOCT** - Genome binning
- **MetaWRAP** - Bin refinement
- **MAGpurify** - Contamination removal
- **CheckM2** - Quality assessment

- **CoverM** - Coverage calculation
- **Bowtie2** - Read mapping
- **Samtools** - BAM processing

Installation

1. Clone the Repository

bash

```bash
git clone <repository-url>
cd metagenomic-pipeline
```

2. Run Setup Script

bash

```bash
./setup_pipeline.sh -i /path/to/input/fastq -o /path/to/output -d /path/to/databases
```

Setup Options:

- `-i, --input-dir`: Directory containing FASTQ files
- `-o, --output-dir`: Output directory for results
- `-d, --databases`: Directory for databases
- `-c, --conda-base`: Path to conda installation [default: ~/miniconda3]
- `-e, --envs-only`: Only create conda environments
- `-t, --test`: Run installation test

3. Manual Environment Setup (if needed)

bash

```bash
# Create individual environments
conda create -n trimmomatic -c bioconda trimmomatic
conda create -n spades -c bioconda spades
conda create -n metawrap-env -c bioconda metawrap-mg
conda create -n checkm -c bioconda checkm2 coverm
# ... (see setup script for complete list)
```

4. Database Setup

bash

```
# CheckM2 database
conda activate checkm
checkm2 database --download --path /path/to/checkm2/db

# Trimmomatic adapters (usually included with conda installation)
# MAGpurify database (follow MAGpurify documentation)
```

Input Data Preparation

Directory Structure

Organize your FASTQ files in a single directory:

```
input_directory/
├── sample1_R1.fastq.gz
├── sample1_R2.fastq.gz
├── sample2_R1.fastq.gz
├── sample2_R2.fastq.gz
└── sample_sheet.xlsx
```

Sample Sheet Format

Create an Excel file (.xlsx) or TSV file with the following columns:

Sample_Name	Treatment	R1_File	R2_File
sample1	treatment1	sample1_R1.fastq.gz	sample1_R2.fastq.gz
sample2	treatment1	sample2_R1.fastq.gz	sample2_R2.fastq.gz
sample3	treatment2	sample3_R1.fastq.gz	sample3_R2.fastq.gz

Column Descriptions:

- **Sample_Name**: Unique identifier for each sample
- **Treatment**: Treatment/condition group
- **R1_File**: Forward read filename (relative to input directory)
- **R2_File**: Reverse read filename (relative to input directory)

Create Sample Sheet Template

```bash
./run_pipeline.sh -c -i /path/to/input/directory
```

Configuration

Edit Configuration File

Edit `00_config_utilities.sh` to set default paths:

```bash
# Base directories
export INPUT_DIR="/path/to/input/fastq/files"
export OUTPUT_DIR="/path/to/output/directory"

# Sample sheet
export SAMPLE_SHEET="${INPUT_DIR}/sample_sheet.xlsx"

# Database paths
export TRIMMOMATIC_DB="/path/to/trimmomatic/adapters"
export MAGPURIFYDB="/path/to/magpurify/database"

# Conda installation
export CONDA_BASE="/path/to/miniconda3"
```

Trimmomatic Parameters

Adjust quality filtering parameters:

```bash
export TRIMMOMATIC_ADAPTERS="TruSeq3-PE-2.fa"
export TRIMMOMATIC_LEADING="3"
export TRIMMOMATIC_TRAILING="3"
export TRIMMOMATIC_SLIDINGWINDOW="4:15"
export TRIMMOMATIC_MINLEN="36"
```

Quality Thresholds

Set bin quality thresholds:

```bash
export MIN_COMPLETENESS="90"
export MAX_CONTAMINATION="5"
```

Running the Pipeline

Complete Pipeline

```bash
```

```
./run_pipeline.sh -i /path/to/input -o /path/to/output -S sample_sheet.xlsx
```

Basic Usage Examples

bash

```bash
# Run with auto-discovery (no sample sheet)
./run_pipeline.sh -i /path/to/input -o /path/to/output

# Run specific stages only
./run_pipeline.sh --start-stage 0 --end-stage 3 -i /path/to/input -o /path/to/output

# Run for specific treatment
./run_pipeline.sh --treatment treatment1 -i /path/to/input -o /path/to/output

# Run for specific sample
./run_pipeline.sh --sample sample1 -i /path/to/input -o /path/to/output

# Dry run to preview
./run_pipeline.sh --dry-run -i /path/to/input -o /path/to/output
```

Command Line Options

bash

```bash
./run_pipeline.sh [OPTIONS]

OPTIONS:
    -i, --input-dir PATH      Input directory containing FASTQ files
    -o, --output-dir PATH     Output directory for results
    -S, --sample-sheet PATH   Sample sheet (Excel or TSV)
    -s, --start-stage NUM     Start from stage NUM (0-11) [default: 0]
    -e, --end-stage NUM       End at stage NUM (0-11) [default: 11]
    -t, --treatment NAME      Run only for specific treatment
    -m, --sample NAME         Run only for specific sample
    -c, --create-template     Create sample sheet template
    -d, --dry-run             Show what would be run without executing
    -h, --help                Show help message
```

Individual Stage Execution

bash

```
# Run individual stages manually
sbatch --array=0-9 00_quality_filtering.sh
sbatch --array=0-9 01_assembly.sh
# ... etc
```

Output Structure

The pipeline creates a comprehensive output directory structure:

```
output_directory/
├── logs/                    # Log files organized by treatment
├── checkpoints/             # Progress tracking files
├── quality_filtering/       # Trimmomatic output
├── assembly/                # MetaSPAdes assemblies
├── plasmids/                # Plasmid detection results
├── binning/                 # Genome bins from all binners
├── bin_refinement/          # Refined bins from MetaWRAP
├── reassembly/              # Reassembled bins
├── magpurify/               # Purified bins
├── checkm2/                 # Quality assessment
├── coverm/                  # Abundance calculations
├── bin_collection/          # High-quality bins by sample
├── treatment_analysis/      # Combined results by treatment
├── final_report/            # Final comprehensive reports
└── high_quality_bins/       # Final high-quality MAGs
```

Key Output Files

Quality Filtering

- `filtered_1.fastq.gz` / `filtered_2.fastq.gz` - Filtered paired reads
- `singletons.fastq.gz` - Unpaired reads (used in assembly)

Assembly

- `contigs.fasta` - Assembled contigs
- `assembly_statistics.txt` - Assembly metrics

Binning

- `metabat2_bins/` - MetaBAT2 bins
- `maxbin2_bins/` - MaxBin2 bins
- `concoct_bins/` - CONCOCT bins

Final Results

- `high_quality_bins/` - High-quality MAGs (≥90% complete, ≤5% contamination)
- `treatment_analysis/` - Combined analysis by treatment
- `final_report.html` - Interactive HTML report

Monitoring and Troubleshooting

Monitor Job Progress

```bash
# Check job queue
squeue -u $USER

# Check job details
sacct -j <job_id>

# Check logs
tail -f output_directory/logs/treatment1/sample1_pipeline.log
```

Check Pipeline Progress

```bash
# Check checkpoints
ls output_directory/checkpoints/*/

# Check specific stage completion
ls output_directory/checkpoints/treatment1/sample1_*_complete
```

Common Issues and Solutions

1. Out of Memory Errors

- Increase memory allocation in SLURM headers
- Reduce number of concurrent jobs
- Use smaller datasets for testing

2. Conda Environment Issues

bash

```bash
# Recreate environments
./setup_pipeline.sh --envs-only

# Test environment
conda activate trimmomatic
trimmomatic -version
```

3. Missing Dependencies

```bash
# Install missing Python packages
pip install pandas openpyxl

# Check CheckM2 database
conda activate checkm
checkm2 testrun
```

4. File Permission Issues

```bash
# Fix permissions
chmod +x *.sh
chmod -R 755 output_directory/
```

Restart from Specific Stage

```bash
# Remove checkpoints to restart
rm output_directory/checkpoints/treatment1/sample1_assembly_complete

# Run from specific stage
./run_pipeline.sh --start-stage 1 -i /path/to/input -o /path/to/output
```

Advanced Usage

Custom Quality Filtering Parameters

Edit the configuration file to adjust Trimmomatic parameters:

```bash
```

```bash
export TRIMMOMATIC_LEADING="5"       # Higher quality threshold
export TRIMMOMATIC_TRAILING="5"      # Higher quality threshold
export TRIMMOMATIC_SLIDINGWINDOW="4:20" # Stricter sliding window
export TRIMMOMATIC_MINLEN="50"       # Longer minimum length
```

Custom Bin Quality Thresholds

bash

```bash
export MIN_COMPLETENESS="80"  # Lower completeness threshold
export MAX_CONTAMINATION="10" # Higher contamination threshold
```

Running on Different Schedulers

For non-SLURM systems, modify the SBATCH headers in each script:

bash

```bash
# For PBS/Torque
#PBS -l nodes=1:ppn=32
#PBS -l walltime=24:00:00

# For LSF
#BSUB -n 32
#BSUB -W 24:00
```

Customizing Resource Requirements

Edit SLURM parameters in individual scripts:

bash

```bash
#SBATCH --cpus-per-task=16  # Reduce CPU usage
#SBATCH --mem=128G          # Reduce memory usage
#SBATCH --time=12:00:00     # Reduce time limit
```

FAQ

Q: Can I run this pipeline without SLURM?

A: The pipeline is designed for SLURM, but you can run individual stages manually by removing SLURM headers and running scripts directly.

Q: What file formats are supported?

A: The pipeline supports compressed FASTQ files (.fastq.gz). It expects paired-end reads with standard naming conventions (_R1/_R2 or _1/_2).

Q: How do I handle single-end reads?

A: The pipeline is designed for paired-end reads. For single-end reads, you would need to modify the assembly and binning scripts.

Q: Can I skip certain stages?

A: Yes, use the `--start-stage` and `--end-stage` options to run specific portions of the pipeline.

Q: How long does the pipeline take?

A: Runtime depends on data size and system resources. For typical metagenomic samples (5-10GB), expect 12-48 hours for the complete pipeline.

Q: What if I don't have a sample sheet?

A: The pipeline can auto-discover samples in the input directory. It will use the directory structure to infer sample names and treatments.

Q: How do I cite this pipeline?

A: Please cite the individual tools used in the pipeline. A list of citations is provided in the final report.

Q: Can I modify the pipeline for my specific needs?

A: Yes, the pipeline is modular. You can modify individual scripts or add new stages as needed.

Support

For issues and questions:

1. Check the log files in `output_directory/logs/`
2. Review the troubleshooting section above
3. Check individual tool documentation
4. Open an issue in the repository

License

This pipeline is released under the MIT License. Individual tools may have their own licenses.