Enhanced Genomic Algorithms Pipeline - WSL Commands

This guide provides the complete command sequence to set up and run the enhanced genomic algorithms pipeline in WSL.

🚀 Quick Start (Automated)

Option 1: Automated Setup

bash

```bash
# Download and run the automated setup script
curl -sSL https://raw.githubusercontent.com/your-repo/genomic-pipeline/main/setup_genomic_pipel

# Or if you have the script locally:
chmod +x setup_genomic_pipeline.sh
./setup_genomic_pipeline.sh
```

📋 Manual Setup (Step by Step)

1. System Preparation

bash

```bash
# Update system packages
sudo apt update && sudo apt upgrade -y

# Install essential dependencies
sudo apt install -y \
    curl wget git build-essential pkg-config libssl-dev \
    python3 python3-pip python3-venv default-jre unzip \
    htop time jq

# Verify installations
gcc --version
python3 --version
java -version
```

2. Install Rust

bash

```bash
# Install Rust using rustup
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh -s -- -y

# Source Rust environment
source ~/.cargo/env

# Install additional components
rustup component add clippy rustfmt

# Verify installation
rustc --version
cargo --version
```

3. Install Nextflow

bash

```bash
# Download and install Nextflow
curl -s https://get.nextflow.io | bash

# Move to system path
sudo mv nextflow /usr/local/bin/
sudo chmod +x /usr/local/bin/nextflow

# Verify installation
nextflow -version
```

4. Set Up Python Environment

bash

```bash
# Create virtual environment
python3 -m venv genomic_env
source genomic_env/bin/activate

# Install scientific packages
pip install --upgrade pip
pip install numpy scipy pandas matplotlib seaborn scikit-learn jupyter
```

5. Create Project Structure

bash

```bash
# Create and enter project directory
mkdir enhanced_genomic_pipeline
cd enhanced_genomic_pipeline

# Create subdirectories
mkdir -p src data results scripts tests docs benchmarks

# Initialize Rust project
cargo init --name genomic_algorithms
```

6. Copy Project Files

bash

```bash
# Copy the artifact files to appropriate locations:

# 1. Copy Cargo.toml content to ./Cargo.toml
# 2. Copy enhanced main.rs content to ./src/main.rs
# 3. Copy main.nf content to ./main.nf
# 4. Copy generate_genomic_data.py content to ./scripts/generate_genomic_data.py
# 5. Copy setup script content to ./setup_genomic_pipeline.sh

# Make scripts executable
chmod +x scripts/generate_genomic_data.py
chmod +x setup_genomic_pipeline.sh
```

📊 Generate Test Datasets

Small Test Dataset

bash

```bash
# Activate Python environment
source genomic_env/bin/activate

# Generate small dataset (100 individuals, 1000 variants)
python3 scripts/generate_genomic_data.py \
    --small \
    --output data/small_cohort.csv \
    --metadata data/small_metadata.json

# View sample data
head -20 data/small_cohort.csv
```

Medium Dataset with Population Structure

bash

```bash
# Generate medium dataset (1000 individuals, 5000 variants, 2 populations)
python3 scripts/generate_genomic_data.py \
    --medium \
    --populations 2 \
    --output data/medium_cohort.csv \
    --metadata data/medium_metadata.json

# Check dataset info
cat data/medium_metadata.json | jq '.'
```

Large Performance Testing Dataset

bash

```bash
# Generate large dataset (5000 individuals, 20000 variants, 3 populations)
python3 scripts/generate_genomic_data.py \
    --large \
    --populations 3 \
    --missing-rate 0.03 \
    --output data/large_cohort.csv \
    --metadata data/large_metadata.json

# Check file size
ls -lh data/*.csv
```

Custom Dataset Examples

bash

```bash
# High-density dataset with strong LD
python3 scripts/generate_genomic_data.py \
    --individuals 2000 \
    --variants 10000 \
    --populations 1 \
    --missing-rate 0.01 \
    --maf-distribution realistic \
    --output data/high_density.csv

# Multi-population study
python3 scripts/generate_genomic_data.py \
    --individuals 3000 \
    --variants 15000 \
    --populations 5 \
    --missing-rate 0.025 \
    --output data/multi_pop.csv

# Dataset without LD structure (for control analysis)
python3 scripts/generate_genomic_data.py \
    --medium \
    --no-ld \
    --output data/no_ld_control.csv
```

⚙️ Build the Rust Project

bash

```bash
# Build in debug mode (faster compilation)
cargo build

# Build in release mode (optimal performance - recommended)
cargo build --release

# Copy binary to project root (needed for Nextflow)
cp target/release/rust_core_algorithms .
chmod +x rust_core_algorithms

# Test the binary
./rust_core_algorithms --help
```

Troubleshooting Build Issues

bash

```bash
# Clean and rebuild if issues occur
cargo clean
cargo build --release

# Update Rust toolchain
rustup update

# Check for missing dependencies
sudo apt install pkg-config libssl-dev
```

🧬 Run Genomic Analysis

Method 1: Direct Rust Execution

Basic Allele Frequency Analysis

bash

```bash
# Simple allele frequency calculation
./rust_core_algorithms \
    --input data/small_cohort.csv \
    --output results/basic_analysis.txt \
    --format txt

# View results
cat results/basic_analysis.txt
```

Comprehensive Analysis with LD

bash

```bash
# Full analysis with linkage disequilibrium
./rust_core_algorithms \
    --input data/medium_cohort.csv \
    --output results/comprehensive.json \
    --format json \
    --compute-ld \
    --max-ld-pairs 1000 \
    --min-maf 0.05 \
    --verbose

# Pretty print JSON results
cat results/comprehensive.json | jq '.summary'
```

HMM Phasing Analysis

bash

```bash
# Haplotype phasing with HMM
./rust_core_algorithms \
    --input data/medium_cohort.csv \
    --output results/phasing_analysis.txt \
    --hmm-states 5 \
    --phase-all \
    --seed 123 \
    --verbose

# Check phasing results
grep -A 10 "Phasing Results" results/phasing_analysis.txt
```

Population Genetics Study

bash

```bash
# Comprehensive population analysis
./rust_core_algorithms \
    --input data/multi_pop.csv \
    --output results/population_study.csv \
    --format csv \
    --compute-ld \
    --max-ld-pairs 2000 \
    --phase-all \
    --min-maf 0.01 \
    --hmm-states 4 \
    --verbose

# Analyze CSV output
head -20 results/population_study.csv
```

Method 2: Nextflow Pipeline Execution

Basic Pipeline Run

bash

```bash
# Simple Nextflow execution
nextflow run main.nf \
    --input data/small_cohort.csv

# Check results
ls -la genomic_results/
```

Advanced Pipeline with All Features

bash

```bash
# Comprehensive analysis pipeline
nextflow run main.nf \
    --input data/medium_cohort.csv \
    --output_dir results/nextflow_comprehensive \
    --format json \
    --compute_ld true \
    --max_ld_pairs 1500 \
    --phase_all true \
    --hmm_states 4 \
    --min_maf 0.02

# View HTML report
# Open results/nextflow_comprehensive/analysis_report.html in browser
```

Production Pipeline with Monitoring

bash

```bash
# Production run with comprehensive reporting
nextflow run main.nf \
    --input data/large_cohort.csv \
    --output_dir results/production_analysis \
    --format json \
    --compute_ld true \
    --max_ld_pairs 5000 \
    --phase_all true \
    --min_maf 0.05 \
    -with-report results/execution_report.html \
    -with-timeline results/timeline.html \
    -with-trace results/trace.txt \
    -with-dag results/flowchart.html

# Resume failed pipeline
nextflow run main.nf -resume
```

Batch Processing Multiple Cohorts

bash

```bash
# Process multiple datasets
for cohort in data/*_cohort.csv; do
    cohort_name=$(basename "$cohort" .csv)
    echo "Processing $cohort_name..."

    nextflow run main.nf \
        --input "$cohort" \
        --output_dir "results/${cohort_name}_analysis" \
        --compute_ld true \
        --max_ld_pairs 1000
done
```

📊 Performance Testing and Benchmarking

Timing Analysis

bash

```bash
# Measure execution time
time ./rust_core_algorithms \
    --input data/large_cohort.csv \
    --output results/performance_test.txt \
    --compute-ld \
    --max-ld-pairs 3000 \
    --verbose

# Detailed timing with GNU time
/usr/bin/time -v ./rust_core_algorithms \
    --input data/large_cohort.csv \
    --output results/detailed_perf.txt \
    --compute-ld \
    --phase-all \
    --verbose 2> results/timing_report.txt
```

Memory Profiling

bash

```bash
# Monitor memory usage during execution
./rust_core_algorithms \
    --input data/large_cohort.csv \
    --output results/memory_test.txt \
    --compute-ld \
    --max-ld-pairs 2000 \
    --verbose &

# Monitor in another terminal
PID=$!
while kill -0 $PID 2>/dev/null; do
    ps -p $PID -o pid,vsz,rss,pcpu,pmem,time,comm --no-headers >> results/memory_usage.log
    sleep 5
done
```

Parallel Performance Testing

bash

```bash
# Test different thread counts (if using parallel features)
for threads in 2 4 8 16; do
    echo "Testing with $threads threads..."
    RAYON_NUM_THREADS=$threads time ./rust_core_algorithms \
        --input data/medium_cohort.csv \
        --output "results/perf_${threads}threads.txt" \
        --compute-ld \
        --max-ld-pairs 1000 \
        2>> results/parallel_benchmark.log
done
```

🔍 Analyzing Results

Text Format Results

bash

```bash
# View summary statistics
grep -E "Total|Mean|Variants" results/comprehensive.txt

# Check LD results
awk '/Linkage Disequilibrium/{flag=1; next} /Phasing/{flag=0} flag' results/comprehensive.txt |

# Extract high LD pairs
awk '/Linkage Disequilibrium/{flag=1; next} /Phasing/{flag=0} flag && $3 > 0.8' results/compreh
```

JSON Format Analysis

bash

```bash
# Extract key metrics using jq
cat results/comprehensive.json | jq '.summary'

# Get high MAF variants
cat results/comprehensive.json | jq '.variants[] | select(.maf > 0.3) | {id, maf, allele_freque

# Analyze LD results
cat results/comprehensive.json | jq '.ld_results[] | select(.r_squared > 0.8) | {variant1, vari

# Phasing quality metrics
cat results/comprehensive.json | jq '.phasing_results[] | {individual_id, mean_confidence: (.cc
```

CSV Format Processing

bash

```bash
# Sort variants by MAF
head -1 results/population_study.csv > results/sorted_variants.csv
tail -n +2 results/population_study.csv | sort -t',' -k5 -nr >> results/sorted_variants.csv

# Filter high-quality variants
awk -F',' 'NR==1 || ($5 > 0.05 && $6 > 0.95)' results/population_study.csv > results/high_quali
```

🧪 Testing and Validation

Run Test Suite

bash

```bash
# Create and run comprehensive test script
cat > test_genomic_pipeline.sh << 'EOF'
#!/bin/bash
set -e

echo "=== Genomic Pipeline Test Suite ==="

# Test 1: Binary functionality
echo "Testing binary functionality..."
./rust_core_algorithms --help > /dev/null && echo "✓ Help command works"

# Test 2: Small dataset analysis
echo "Testing small dataset analysis..."
./rust_core_algorithms \
    --input data/small_cohort.csv \
    --output results/test_small.txt \
    --compute-ld \
    --max-ld-pairs 50 && echo "✓ Small dataset analysis works"

# Test 3: JSON output
echo "Testing JSON output..."
./rust_core_algorithms \
    --input data/small_cohort.csv \
    --output results/test_json.json \
    --format json && echo "✓ JSON output works"

# Test 4: Nextflow pipeline
if command -v nextflow &> /dev/null && [ -f main.nf ]; then
    echo "Testing Nextflow pipeline..."
    nextflow run main.nf \
        --input data/small_cohort.csv \
        --output_dir results/test_nf && echo "✓ Nextflow pipeline works"
fi

echo "All tests passed!"
EOF

chmod +x test_genomic_pipeline.sh
./test_genomic_pipeline.sh
```

Validation Tests

bash

```bash
# Test with different data sizes
for size in small medium; do
    echo "Validating $size dataset..."
    ./rust_core_algorithms \
        --input "data/${size}_cohort.csv" \
        --output "results/validate_${size}.txt" \
        --compute-ld \
        --verbose
done

# Compare results consistency
diff <(grep "Total variants" results/validate_small.txt) \
     <(grep "Total variants" results/validate_medium.txt) || true
```

🔧 Troubleshooting

Common Issues and Solutions

Memory Issues

bash

```bash
# Reduce memory usage
./rust_core_algorithms \
    --input data/large_cohort.csv \
    --output results/memory_efficient.txt \
    --max-ld-pairs 500  # Reduce LD computation

# Monitor memory usage
free -h
htop  # In another terminal
```

Build Errors

bash

```bash
# Clean rebuild
cargo clean
rm -rf target/
cargo build --release

# Update dependencies
cargo update
rustup update
```

Nextflow Issues

bash

```bash
# Clean Nextflow cache
rm -rf work/ .nextflow/
nextflow clean -f

# Debug Nextflow execution
nextflow run main.nf -with-trace -with-report --input data/small_cohort.csv
```

File Format Issues

bash

```bash
# Validate CSV format
head -5 data/small_cohort.csv
python3 -c "
import csv
with open('data/small_cohort.csv', 'r') as f:
    reader = csv.reader(f)
    for i, row in enumerate(reader):
        if i > 5: break
        print(f'Row {i}: {len(row)} columns')
"
```

🚀 Performance Optimization

System Tuning

bash

```bash
# Increase file descriptor limits
ulimit -n 65536

# Set CPU governor to performance (if available)
sudo cpupower frequency-set -g performance 2>/dev/null || echo "cpupower not available"

# Monitor I/O performance
iostat -x 2 &  # In background
```

Application Tuning

bash

```bash
# Set environment variables for optimization
export RUST_LOG=info
export RAYON_NUM_THREADS=$(nproc)

# Use optimized builds
export RUSTFLAGS="-C target-cpu=native"
cargo build --release
```

Monitoring Resources

bash

```bash
# Real-time monitoring during analysis
watch -n 2 'free -h; echo; ps aux | grep rust_core_algorithms | head -5'

# Continuous monitoring script
cat > monitor_analysis.sh << 'EOF'
#!/bin/bash
LOG_FILE="analysis_monitor.log"
echo "Time,CPU%,Memory_MB,Disk_IO" > $LOG_FILE

while pgrep rust_core_algorithms > /dev/null; do
    TIMESTAMP=$(date '+%Y-%m-%d %H:%M:%S')
    CPU=$(ps -p $(pgrep rust_core_algorithms) -o %cpu --no-headers | awk '{sum+=$1} END {print
    MEM=$(ps -p $(pgrep rust_core_algorithms) -o rss --no-headers | awk '{sum+=$1} END {print s
    echo "$TIMESTAMP,$CPU,$MEM" >> $LOG_FILE
    sleep 5
done
EOF

chmod +x monitor_analysis.sh
```

This comprehensive guide provides all the commands needed to successfully set up, run, and optimize the enhanced genomic algorithms pipeline in WSL. Adjust parameters based on your specific dataset sizes and system capabilities.