BAM File Preparation - Comprehensive Guide

Overview

BAM (Binary Alignment Map) files from aligners often require preparation steps before downstream analysis. This includes indexing for random access, calculating metadata, and creating subsamples for efficient quality control.

Applicable to: Any workflow producing BAM files (RNA-seq, WGS, WES, ChIP-seq, ATAC-seq)

Why Prepare BAM Files?

Common Requirements

Most downstream tools require:

1. **Indexed BAM** - Fast random access to specific regions
2. **Standardized naming** - Consistent filenames across samples
3. **Metadata** - Read length, library type, etc.
4. **Subsamples** - Smaller BAMs for fast QC

Without Preparation

Problems:

- Tools fail without index (.bai file)
- Inconsistent filenames cause tracking issues
- Full BAM QC takes hours
- Unknown read lengths break scripts

With Preparation

Benefits:

- All tools work seamlessly
- Fast QC analysis
- Reproducible pipelines
- Clear file organization

BAM Indexing

What is a BAM Index?

BAM index (.bai file):

- Allows random access to BAM regions

- Required by most downstream tools
- Maps genomic coordinates to file positions
- Small file (~1-5MB for human genome)

When Indexing is Required

Tools that need indexed BAM:

- IGV (genome browser)
- samtools view -r (region queries)
- GATK (variant calling)
- RSeQC (quality control)
- Qualimap (QC statistics)
- Any region-based analysis

Indexing Requirements

BAM must be:

1. **Sorted by coordinate** - Not by name or unsorted
2. **Valid format** - Proper SAM/BAM structure
3. **Complete** - Not truncated or corrupted

Indexing Tools Comparison

samtools index

Command:

```
samtools index input.bam
```

Characteristics:

- Single-threaded
- Standard tool (most compatible)
- Slower for large files

Speed: ~5-10 minutes for human WGS BAM

---

sambamba index

Command:

```
sambamba index -t 8 input.bam
```

Characteristics:

- Multi-threaded (parallel)
- Faster than samtools

- Compatible with samtools format

Speed: ~1-2 minutes for human WGS BAM (8 cores)

Command breakdown:

```
sambamba index \          # Sambamba indexing command
    -t 8 \                # Use 8 threads (parallel processing)
    input.bam             # Input BAM file
```

Output: `input.bam.bai` (same format as samtools)

Performance Comparison

Tool	Threads	Time (Human WGS)	Memory
samtools	1	8-10 min	100MB
sambamba (4 cores)	4	2-3 min	500MB
sambamba (8 cores)	8	1-2 min	800MB

Recommendation: Use sambamba with 4-8 threads for speed

Read Length Detection

Why Detect Read Length?

Many tools need read length as parameter:

- RSeQC scripts (deletion_profile, mismatch_profile)
- Custom QC scripts
- Coverage calculations
- Fragment size analysis

Detection Methods

Method 1: From BAM (Recommended)

```
# Extract sequence length from first read
samtools view input.bam | head -n1 | awk '{print length($10)}'
```

Command breakdown:

```
samtools view input.bam     # Convert BAM to SAM text
    | head -n1               # Take only first read
    | awk '{print length($10)}'  # Column 10 = sequence, get length
```

Output: Single number (e.g., `150`)

Method 2: Robust Detection (Multiple Reads)

```
# Sample multiple reads, find most common length
samtools view input.bam | \
    head -n 10000 | \
    awk '{print length($10)}' | \
    sort | uniq -c | sort -rn | head -n1 | \
    awk '{print $2}'
```

Command breakdown:

```
samtools view input.bam          # Convert to SAM
    | head -n 10000              # Sample 10K reads
    | awk '{print length($10)}'  # Extract all lengths
    | sort                       # Sort numerically
    | uniq -c                    # Count occurrences of each length
    | sort -rn                   # Sort by count (descending)
    | head -n1                   # Take most common
    | awk '{print $2}'           # Extract length value
```

Why sample multiple reads?:

- Handles trimmed reads (variable length)
- Identifies dominant length
- More robust than single read

Example output:

```
   9847 150    # 9847 reads are 150bp (most common)
    103 149    # 103 reads are 149bp (trimmed)
     50 148    # 50 reads are 148bp (trimmed)
```

Result: `150`

Method 3: From FASTQ (Before Alignment)

```
# Get length from original FASTQ
zcat reads.fastq.gz | head -n 4 | sed -n '2p' | awk '{print length}'
```

Why from BAM is better:

- No need to keep FASTQ files
- Reflects actual aligned read length
- Handles soft-clipping automatically

BAM Subsampling

Why Subsample?

Full BAM QC is slow:

- Gene body coverage: 20-30 minutes per sample
- Read distribution: 10-15 minutes
- 100 samples = 50+ hours total

Subsampled BAM QC is fast:

- Gene body coverage: 2-3 minutes per sample
- Read distribution: 1-2 minutes
- 100 samples = 5-6 hours total

Trade-off: ~10x speedup with <5% accuracy loss

Subsampling Strategies

Strategy 1: First N Reads (Not Recommended)

```
# Take first 1M reads
samtools view -h input.bam | head -n 1000000 | samtools view -b > subset.bam
```

Problems:

- Not random (biased by chromosome order)
- Misses later chromosomes
- Not representative of whole genome

Don't use for QC

Strategy 2: Random Sampling (Recommended)

```
# Random 1M reads with reproducible seed
sambamba view -s 0.1 --subsampling-seed=42 -f bam input.bam -o subset.bam
```

Command breakdown:

```
sambamba view \                    # Sambamba view/filter command
    -s 0.1 \                       # Subsample to 10% of reads
    --subsampling-seed=42 \        # Reproducible random seed
    -f bam \                       # Output format: BAM
    input.bam \                    # Input BAM file
    -o subset.bam                  # Output filename
```

Benefits:

- Random sampling (unbiased)

- Reproducible (same seed = same subset)
- Representative of whole BAM
- Even chromosome distribution

Recommended for QC

Calculating Sample Fraction

Goal: Get exactly 1M reads

```
# Count total reads
TOTAL=$(samtools view -c input.bam)

# Calculate fraction
FRACTION=$(awk -v total=$TOTAL 'BEGIN {print 1000000/total}')

# If BAM has <1M reads, use all (fraction > 1.0)
FRACTION=$(awk -v total=$TOTAL 'BEGIN {
    if (total <= 1000000)
        print 1.0
    else
        print 1000000/total
}')

# Subsample
sambamba view -s $FRACTION --subsampling-seed=42 -f bam input.bam -o subset.bam
```

Example:

```
Total reads: 50,000,000
Target: 1,000,000
Fraction: 1,000,000 / 50,000,000 = 0.02 (2%)
```

Subsampling Parameters

-s (Fraction) **Format**: Decimal between 0.0 and 1.0

Examples:

```
-s 0.1    # Keep 10% of reads
-s 0.05   # Keep 5% of reads
-s 0.02   # Keep 2% of reads
-s 1.0    # Keep all reads (no subsampling)
```

--subsampling-seed **Purpose**: Reproducible random sampling

Why set a seed?:

- Same seed = same subset each time
- Different seed = different subset
- Useful for reproducibility

Examples:

```
--subsampling-seed=42      # Common choice
--subsampling-seed=123     # Alternative
--subsampling-seed=0       # Another option
```

Without seed: Different result each run

Complete Preparation Workflow

Step-by-Step Example

```
#!/bin/bash
SAMPLE="Sample1"
INPUT="${SAMPLE}.Aligned.sortedByCoord.out.bam"
CORES=8

# Step 1: Rename to standard format
mv "$INPUT" "${SAMPLE}.bam"

# Step 2: Index full BAM
sambamba index -t $CORES "${SAMPLE}.bam"
echo "Full BAM indexed"

# Step 3: Calculate read length
READ_LENGTH=$(sambamba view "${SAMPLE}.bam" | \
    head -n 10000 | \
    awk '{print length($10)}' | \
    sort | uniq -c | sort -rn | head -n1 | \
    awk '{print $2}')
echo "$READ_LENGTH" > "${SAMPLE}.read_length.txt"
echo "Read length: $READ_LENGTH"

# Step 4: Count total reads
TOTAL=$(sambamba view -c -t $CORES "${SAMPLE}.bam")
echo "Total reads: $TOTAL"

# Step 5: Calculate subsample fraction
FRACTION=$(awk -v total=$TOTAL 'BEGIN {
    if (total <= 1000000)
        print 1.0
    else
```

```
        print 1000000/total
}')
echo "Subsample fraction: $FRACTION"

# Step 6: Create subsample
sambamba view \
    -t $CORES \
    -f bam \
    -s $FRACTION \
    --subsampling-seed=42 \
    "${SAMPLE}.bam" \
    -o "${SAMPLE}.1M.bam"
echo "Subsample created"

# Step 7: Index subsample
sambamba index -t $CORES "${SAMPLE}.1M.bam"
echo "Subsample indexed"

# Verify outputs
ls -lh ${SAMPLE}*
```

Output files:

```
Sample1.bam                 # Full BAM (5-10GB)
Sample1.bam.bai             # Full BAM index (1-5MB)
Sample1.1M.bam              # Subsampled BAM (100-500MB)
Sample1.1M.bam.bai          # Subsample index (50-200KB)
Sample1.read_length.txt     # Read length (1 line)
```

Use Cases for Each Output

Full BAM + Index

Uses:

- Visualization (IGV, UCSC Browser)
- Variant calling (GATK, bcftools)
- Detailed analysis (full resolution)
- Publication figures
- Long-term archival

Keep: Indefinitely

Subsampled BAM + Index

Uses:

- Fast QC (RSeQC, Qualimap)
- Gene body coverage
- Read distribution
- Quick validation
- Preliminary analysis

Keep: Until QC complete (can regenerate)

Read Length File

Uses:

- RSeQC script parameters
- Coverage normalization
- Custom QC scripts
- Documentation

Keep: With analysis results

Quality Checks

Verify Indexing Success

```
# Check index exists
ls -lh sample.bam.bai

# Verify index is valid
samtools idxstats sample.bam | head

# Quick region query (tests index)
samtools view sample.bam chr1:1000-2000 | head
```

Expected: Fast query response (1-2 seconds)

If slow or fails: Index is corrupted or missing

Verify Subsample Quality

```
# Count reads in subsample
samtools view -c subsample.bam

# Should be ~1M (within 10%)
# Example: 950,000 - 1,050,000

# Check chromosome distribution
```

```
samtools idxstats subsample.bam | awk '{print $1, $3}'

# Should have reads from all chromosomes
```

Verify Read Length

```
# Check read length file
cat sample.read_length.txt

# Expected: Single number (e.g., 150)

# Verify against BAM
samtools view sample.bam | head -n1 | awk '{print length($10)}'

# Should match read_length.txt
```

Performance Optimization

Multi-threading

Sambamba scales well:

```
# Benchmarks for indexing 10GB BAM
1 thread:  8 minutes
2 threads: 4.5 minutes
4 threads: 2.5 minutes
8 threads: 1.5 minutes
16 threads: 1.2 minutes (diminishing returns)
```

Recommendation: 4-8 threads optimal

Storage Considerations

Disk space needed:

```
Original BAM:      10GB
Index (.bai):      2MB
Subsample (1M):    200MB
Subsample index:   50KB
Read length:       1KB
Total:             ~10.2GB (2% overhead)
```

Temporary space: ~5GB during processing

I/O Optimization

Fast operations:

- Store BAMs on local SSD
- Avoid network storage during processing
- Can move to slower storage after indexing

Slow operations:

- Network-mounted BAMs
- HDD storage
- Compressed filesystems

Troubleshooting

Problem: Indexing Fails

Error: "bam file must be sorted by coordinate"

Solution:

```
# Check if sorted
samtools view -H sample.bam | grep "@HD"

# Expected: SO:coordinate
# If says SO:queryname or nothing, need to sort

# Sort by coordinate
sambamba sort -t 8 -o sample.sorted.bam sample.bam
mv sample.sorted.bam sample.bam
```

Problem: Subsampling Produces Wrong Count

Expected: 1M reads
Got: 50K reads

Diagnosis:

```
# Check total reads
sambamba view -c sample.bam

# If total < 1M, subsample will have fewer
# This is correct behavior
```

Solution: Adjust expectations or use full BAM

Problem: Read Length Detection Fails

Error: Empty read_length.txt

Diagnosis:

```
# Check BAM has reads
sambamba view sample.bam | head

# If empty, BAM is corrupted or truncated
```

Solution: Re-align or use known read length

Alternative Tools

Tool Comparison

Feature	sambamba	samtools	biobambam2
Indexing	Fast (parallel)	Slow (single)	Fast
Subsampling	Built-in	Via pipe	Built-in
Speed	Excellent	Good	Excellent
Memory	Moderate	Low	Moderate
Stability	Stable	Very stable	Stable

samtools Equivalent Commands

```
# Indexing
samtools index input.bam

# Subsampling (requires awk calculation)
FRAC=0.02  # 2%
samtools view -s $FRAC -b input.bam > subset.bam

# Read counting
samtools view -c input.bam

# Read length
samtools view input.bam | head -n1 | awk '{print length($10)}'
```

Note: samtools subsampling uses different algorithm

Best Practices

For Reproducibility

Set subsampling seed:

```
--subsampling-seed=42  # Always same result
```

Document parameters:

```
echo "Subsample: 1M reads, seed=42" > sample.subsample.log
```

Verify outputs:

```
# Check all expected files exist
for f in *.bam *.bai *read_length.txt; do
    [ -f "$f" ] && echo "OK: $f" || echo "MISSING: $f"
done
```

For Performance

Use local storage during processing

Parallel indexing:

```
sambamba index -t 8  # Not just -t 1
```

Batch processing:

```
# Process multiple samples in parallel
parallel -j 4 'sambamba index -t 2 {}' ::: *.bam
```

For Storage

Delete intermediate files after QC:

```
# After RSeQC completes
rm *.1M.bam *.1M.bam.bai
```

Keep full BAM + index for long-term

Compress old BAMs if needed:

```
# CRAM is smaller than BAM
samtools view -T reference.fa -C -o sample.cram sample.bam
```

Integration with Pipelines

Nextflow Example

```
process BAM_PREP {
    input:
    tuple val(sample_id), path(bam)

    output:
    tuple val(sample_id),
          path("${sample_id}.bam"),
          path("${sample_id}.bam.bai"),
          path("${sample_id}.1M.bam"),
          path("${sample_id}.1M.bam.bai")

    script:
    """
    sambamba index -t ${task.cpus} ${bam}
    sambamba view -s 0.02 --subsampling-seed=42 \
        -f bam ${bam} -o ${sample_id}.1M.bam
    sambamba index -t ${task.cpus} ${sample_id}.1M.bam
    """
}
```

Snakemake Example

```
rule bam_prep:
    input:
        bam="aligned/{sample}.bam"
    output:
        bam="prepared/{sample}.bam",
        bai="prepared/{sample}.bam.bai",
        sub="prepared/{sample}.1M.bam",
        sub_bai="prepared/{sample}.1M.bam.bai"
    threads: 8
    shell:
        """
        cp {input.bam} {output.bam}
        sambamba index -t {threads} {output.bam}
        sambamba view -s 0.02 --subsampling-seed=42 \
            -f bam {output.bam} -o {output.sub}
        sambamba index -t {threads} {output.sub}
        """
```

Related Documentation

- **sambamba Documentation**: https://lomereiter.github.io/sambamba/
- **samtools Documentation**: http://www.htslib.org/doc/samtools.html
- **SAM/BAM Format**: https://samtools.github.io/hts-specs/SAMv1.pdf
- **RSeQC Tools**: `docs/rseqc.md`

Document Version: 2.0
Last Updated: January 2026
Applicable to: Any workflow producing BAM files requiring indexing and QC