Using UMIs with ThruPLEX Tag-Seq HV

ThruPLEX Tag-Seq HV adapters have been designed to include discrete unique molecular tags (UMIs) that can be analyzed using publicly available tools. Below is an example of a step-by-step pipeline for finding and using the molecular tags in your sequencing data using a command-line environment such as Linux.

Before you begin

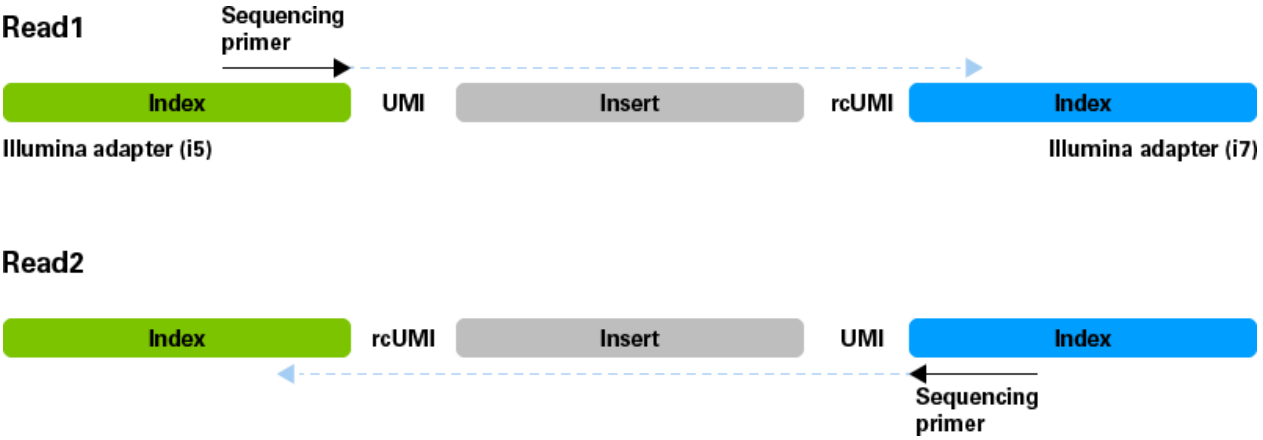
- Additional software dependencies*
  - Linux x86_64
  - Java 10.0.1 or later
  - Trimmomatic 0.36
  - Tools suite from fgbio 0.6.1
  - Picard 2.18.27
  - SAMtools 1.8
  - Bowtie 2 2.3.4.1

  *For information on system requirements and installation instructions, please consult the software documentation.
- Download the following input files from our site
  - FASTA file containing the sequences needed for adapter trimming (Step A)
  - TEXT file containing the sequences needed for read grouping (Step D)

Adapter trimming

Step A. Trim adapters and reverse complement UMIs

The UMIs are read during the first seven cycles of Read1 and Read2. However, if long reads are performed or if the inserts are short, it is possible to read the reverse complement of the UMI (rcUMI) after the insert and before the Illumina adapters (see figure below). To remove the artificial sequence before alignment to the genome assembly, the reverse complement of the UMI is added to the Illumina adapter during the trimming step. The FASTA files containing the sequences are available here and from Takara Bio technical support.



java -jar trimmomatic-0.36.jar PE **<read1.fastq.gz> <read2.fastq.gz> <paired_output1.fq.gz> <unpaired_output1.fq.gz> <paired_output2.fq.gz> <unpaired_output2.fq.gz>** ILLUMINACLIP:**TruSeq3-PE-2with_rcUMI.fa**:1:10:5:9:true MINLEN:20

where:

- <read1.fastq.gz> is the input Illumina sequencing file for Read 1
- <read2.fastq.gz> is the input Illumina sequencing file for Read 2
- <paired_output1.fq.gz> is the output file containing paired forward reads
- <unpaired_output1.fq.gz> is the output file containing unpaired forward reads
- <paired_output2.fq.gz> is the is the output file containing paired reverse reads
- <unpaired_output2.fq.gz> is the output file containing unpaired reverse reads
- TruSeq3-PE-2with_rcUMI.fa is the downloaded FASTA file containing the rcUMI sequences

E.g.,

java -jar trimmomatic-0.36.jar PE **read1_R1_001.fastq.gz read2_R2_001.fastq.gz trimmed_R1.fq.gz UnPaired_R1_001.fq.gz trimmed_R2.fq.gz UnPaired_R2_001.fq.gz** ILLUMINACLIP:**TruSeq3-PE-2with_rcUMI.fa**:1:10:5:9:true MINLEN:20

Unmapped BAM

Step B. Generate unmapped BAM with UMI in RX tag

This step involves converting your trimmed FASTQ files from Step A to an unmapped BAM format, moving the UMI information from the read itself to the RX tag of each read, then creating FASTQ format files with the UMI sequence removed that can be used for alignment. Both the FASTQ file without the UMI and the intermediate BAM file that has the UMI information will be used later in the pipeline.

1. Add your tag or UMI to an RX tag in the BAM file using fgbio's FastqToBam tool:

   java -jar fgbio-0.6.1.jar FastqToBam -i **<paired_output1.fq.gz> <paired_output2.fq.gz>** -r 7M1S+T 7M1S+T -o **<unmapped_output.bam>** -s true

   where:

   - <paired_output1.fq.gz> is the output file containing the paired forward reads from Step A
   - <paired_output2.fq.gz> is the is the output file containing paired reverse reads from Step A
   - <unmapped_output.bam> is the output file

   E.g.,

   java -jar fgbio-0.6.1.jar FastqToBam -i **trimmed_R1.fq.gz trimmed_R2.fq.gz** -r 7M1S+T 7M1S+T -o **unmapped.bam** -s true

Arguments	Explanation
-i	Input FASTQ files corresponding to each sequencing read
-r	Read structure: **T** identifies a template read (to be aligned later) **B** identifies a sample barcode read (not found in this part of the read, so not used) **M** identifies a unique molecular index read (seven bases for ThruPLEX Tag-Seq HV) **S** identifies a set of bases that should be skipped or ignored (skip the first base for a better alignment)
-o	Output file, in this example, named ummapped bam
-s	If true, queryname sort bam

2. Generate FASTQ output files from the unmapped BAM using Picard's SamToFastq tool:

```
java -jar picard.jar SamToFastq INPUT=<unmapped_output.bam> FASTQ=<unmapped_output1.fastq>
SECOND_END_FASTQ=<unmapped_output2.fastq>
```

where:

- <unmapped_output.bam> is the unmapped output file from Step B.1
- <unmapped_output1.fastq> is the FASTQ output file for first end (Read1) of the pair FASTQ
- <unmapped_output2.fastq> is the FASTQ output file for second end (Read2) of the pair FASTQ

E.g.,

```
java -jar picard.jar SamToFastq INPUT=unmapped.bam FASTQ=read1_minusUMI_R1.fastq SECOND_END_FASTQ=read2_minusUMI_R2.fastq
```

Now you have generated FASTQ files with the UMI sequences removed from the read, and a BAM file that contains the UMI information.

Read alignment

Step C. Align the new FASTQ files (with removed UMIs) with Bowtie 2

1. Align the processed FASTQ files from Step B.2 to the appropriate genome assembly with Bowtie 2:

```
bowtie2 -x /REFERENCES/HG19/bowtie2hg19 -1 <unmapped_output1.fastq> -2 <unmapped_output2.fastq> -p 4 -S <bowtie2_ouput.sam>
```

where:

- <unmapped_output1.fastq> is the FASTQ output file for first end (Read1) of the pair FASTQ from Step B.2
- <unmapped_output2.fastq> is the FASTQ output file for second end (Read2) of the pair FASTQ from Step B.2
- <bowtie2_output.sam> is the output file

E.g.,

```
bowtie2 -x /REFERENCES/HG19/bowtie2hg19 -1 read1_minusUMI_R1.fastq -2 read2_minusUMI_R2.fastq -p 4 -S bowtie2.sam
```

2. Sort by queryname with Picard's SortSAM tool:

```
java -jar picard.jar SortSam INPUT=<bowtie2_output.sam> OUTPUT=<sorted_bowtie2_output.sam> SORT_ORDER=queryname
```

where:

- <bowtie2_output.sam> is the Bowtie2 output file from Step C.1
- <sorted_bowtie2_output.sam> is the output file

E.g.,

```
java -jar picard.jar SortSam INPUT=bowtie2.sam OUTPUT=sorted.sam SORT_ORDER=queryname
```

3. Generate sorted BAM file with SAMtools:

```
samtools view -S -b <sorted_bowtie2_output.sam> > <sorted_bowtie2_output.bam>
```

where:

- <sorted_bowtie2_output.sam> is the sorted SAM file from Step C.2
- <sorted_bowtie2_output.bam> is the output file

E.g.,

```
samtools view -S -b sorted.sam > sorted.bam
```

4. Use the unmapped BAM generated in Step B.1 and the sorted BAM file from Step B.3 to generate a mapped BAM file that includes the UMI in the RX tag using Picard's MergeBamAlignment tool:

```
java -jar picard.jar MergeBamAlignment ALIGNED=<sorted_bowtie2_output.bam> UNMAPPED=<unmapped_output.bam>
OUTPUT=<aligned_output.bam> REFERENCE_SEQUENCE=hg19.fa SORT_ORDER=coordinate ALIGNER_PROPER_PAIR_FLAGS=true
ALIGNED_READS_ONLY=true CREATE_INDEX=true VALIDATION_STRINGENCY=SILENT EXPECTED_ORIENTATIONS=FR
MAX_INSERTIONS_OR_DELETIONS=-1
```

where:

- <sorted_bowtie2_output.bam> is the sorted BAM file from Step C.3
- <unmapped_output.bam> is the unmapped output file from Step B.1
- <aligned_output.bam> is the output file

E.g.,

```
java -jar picard.jar MergeBamAlignment ALIGNED=sorted.bam UNMAPPED=unmapped.bam OUTPUT=umi.bam
REFERENCE_SEQUENCE=hg19.fa SORT_ORDER=coordinate ALIGNER_PROPER_PAIR_FLAGS=true ALIGNED_READS_ONLY=true
CREATE_INDEX=true VALIDATION_STRINGENCY=SILENT EXPECTED_ORIENTATIONS=FR MAX_INSERTIONS_OR_DELETIONS=-1
```

Additional analysis

Now you have an aligned BAM file (<ouput.bam>) that contains the UMI information. This can be used by a variety of different downstream analysis programs. Below, we describe a method to group and filter the reads using the UMI.

Read grouping

Step D. Group reads per UMI and filter

This step provides instructions to group the UMIs for secondary analysis, such as identifying false positives from sequencing errors or collapsing PCR duplicates. At the end of this process, you will have an unmapped BAM format file with filtered consensus reads.

1. Correct the UMIs stored in BAM files when a set of fixed UMIs is in use, as is the case with ThruPLEX Tag-Seq HV, using fgbio's CorrectUmis. The UMI sequences are available here and via Takara Bio technical support.

```
java -jar fgbio-0.6.1.jar CorrectUmis -i <aligned_ouput.bam> -o <corrected_output.bam> -M <metrics_output.txt> -m 2 -d 2 -U
<expectedUMI.txt>
```

where:

- <aligned_output.bam> is the output file from Step C.4

- <corrected_output.bam> is the corrected output file
- <metrics_output.txt> is the output metrics file
- <expectedUMI.txt> is the downloaded text file containing the UMI sequence information

E.g.,

```
java -jar fgbio-0.6.1.jar CorrectUmis -i umi.bam -o corrected_umi.bam -M metrics.txt -m 2 -d 2 -U expectedUMI.txt
```

Argument	Explanation
-m	Maximum number of mismatches between a UMI and an expected UMI
-d	Minimum distance (in mismatches) to next best UMI

2. Group the reads together that appear to have come from the same original molecule using fgbio's GroupReadsByUmi. Reads are grouped by template, and then templates are sorted by the 5'-mapping positions of the reads from the template, from earliest mapping position to latest. Reads that have the same end positions are then subgrouped by UMI sequence.

```
java -jar fgbio-0.6.1.jar GroupReadsByUmi -i <corrected_output.bam> -o <grouped_ouput.bam> -s paired -m 20
```

where:

- <corrected_output.bam> is the output file from Step D.1
- <grouped_output.bam> is the output file

E.g.,

```
java -jar fgbio-0.6.1.jar GroupReadsByUmi -i corrected_umi.bam -o grouped.bam -s paired -m 20
```

Argument	Explanation
-s	Specifies the grouping strategy.
paired	Option for the grouping strategy arugment. It is similar to adjacency but for methods that produce a template with a pair of UMIs such that a read with A-B is related to but not identical to a read with B-A. Expects the pair of UMIs to be stored in a single tag, separated by a hyphen (e.g., ACGT-CCGG).
-m	Minimum mapping quality.

3. Call consensus sequences from reads with the same unique molecular tag using fgbio's CallMolecularConsensusReads tool. This step generates unmapped consensus reads from the output of GroupReadsByUmi.

```
java -jar fgbio-0.6.1.jar CallMolecularConsensusReads -I <grouped_output.bam> -o <consensus_output.bam> --error-rate-post-umi=25 --min-read=2
```

where:

- <grouped_output.bam> is the grouped output file from Step D.2
- <consensus_output.bam> is the output file

E.g.,

```
java -jar fgbio-0.6.1.jar CallMolecularConsensusReads -I grouped.bam -o consensus_unmapped.bam --error-rate-post-umi=25 --min-read=2
```

Argument	Explanation
--error-rate-post-umi	The Phred-scaled error rate for an error post when the UMIs have been integrated.
--min-read	Particular attention should be paid to setting the --min-reads parameter, as this can have a dramatic effect on both results and runtime. For libraries with low duplication rates (e.g., 100–300X exome libraries) in which it is desirable to retain singleton reads while making consensus reads from sets of duplicates, --min-reads=1 is appropriate. For libraries with high duplication rates where it is desirable to only produce consensus reads supported by 2+ reads to allow error correction, --min-reads=2 or higher is appropriate.

4. Filter consensus reads generated by CallMolecularConsensusReads using fgbio's FilterConsensusReads tool.

```
java -jar fgbio-0.6.1.jar FilterConsensusReads -i <consensus_output.bam> -o <filtered_output.bam> -r hg19.fa -M 1 -E 0.05 -e 0.1 -N 30 -n 0.1
```

where:

- <consensus_output.bam> is the consensus reads output file from Step D.3
- <filtered_output.bam> is the output file

E.g.,

```
java -jar fgbio-0.6.1.jar FilterConsensusReads -i consensus_unmapped.bam -o filtered_unmapped.bam -r hg19.fa -M 1 -E 0.05 -e 0.1 -N 30 -n 0.1
```

Argument	Explanation
-r	Reference FASTA file
-M	The minimum number of reads supporting a consensus base/read
-E	The maximum raw-read error rate across the entire consensus read
-e	The maximum error rate for a single consensus base
-N	Mask (make N) consensus bases with quality less than this threshold
-n	Maximum fraction of no-calls in the read after filtering
-o	Output file

These filters depend on the quality of the run and the number of cycles performed for Read1 and Read2. For best results, stringency can be increased when using short reads (e.g., PE 75 x 75 cycles) and relaxed when using long reads (e.g., PE 150 x 150 cycles).

Final alignment

Step E. Align grouped and filtered reads with Bowtie 2

In this part, the final alignment is performed with the filtered consensus read files (duplicates removed) from Step D.4. This process results in an aligned BAM file that can be visualized and used for downstream variant calling.

1. Sort the reads per queryname with Picard's SortSAM tool:

```
java -jar picard.jar SortSam INPUT=<filtered_output.bam> OUTPUT=<sorted_filtered_output.bam> SORT_ORDER=queryname
```

where:

- <filtered_output.bam> is the filtered consensus reads output file from Step D.4
- <sorted_filtered_output.bam> is the output file

E.g.,

```
java -jar picard.jar SortSam INPUT=filtered_unmapped.bam OUTPUT=filtered_unmapped_sorted.bam SORT_ORDER=queryname
```

2. Create new FASTQ files from the unmapped reads using Picards' SamToFastq tool:

```
java -jar picard.jar SamToFastq INPUT=<sorted_filtered_output.bam> FASTQ=<sorted_filtered_output1.fastq>
SECOND_END_FASTQ=<sorted_filtered_output2.fastq>
```

where:

- <sorted_filtered_output.bam> is the sorted output file from Step E.1
- <sorted_filtered_output1.fastq> is the filtered FASTQ output file for first end (Read1) of the pair FASTQ
- <sorted_filtered_output2.fastq> is the filtered FASTQ output file for second end (Read2) of the pair FASTQ

E.g.,

```
java -jar picard.jar SamToFastq INPUT=filtered_unmapped_sorted.bam FASTQ=filtered_R1.fastq SECOND_END_FASTQ=filtered_R2.fastq
```

3. Align the new FASTQ files with Bowtie 2:

```
bowtie2 -x /REFERENCES/HG19/bowtie2hg19 -1 <sorted_filtered_output1.fastq> -2 <sorted_filtered_output2.fastq> -p 4 -S
<aligned_filtered_output.sam>
```

where:

- <sorted_filtered_output1.fastq> is the filtered FASTQ output file for first end (Read1) of the pair FASTQ from Step E.2
- <sorted_filtered_output2.fastq> is the filtered FASTQ output file for second end (Read2) of the pair FASTQ from Step E.2
- <aligned_filtered_ouput.sam> is the output file

E.g.,

```
bowtie2 -x /REFERENCES/HG19/bowtie2hg19 -1 filtered_R1.fastq -2 filtered_R2.fastq -p 4 -S filtered.sam
```

4. Sort by queryname with Picard's SortSAM tool:

```
java -jar picard.jar SortSam INPUT=<aligned_filtered_ouput.sam> OUTPUT=<aligned_filtered_sorted_output.sam> SORT_ORDER=queryname
```

where:

- <aligned_filtered_ouput.sam > is the aligned filtered output file from Step E.3
- <aligned_filtered_sorted_output.sam> is the output file

E.g.,

```
java -jar picard.jar SortSam INPUT=filtered.sam OUTPUT=sorted_filtered.sam SORT_ORDER=queryname
```

5. Create BAM file with SAMtools:

```
samtools view -S -b <aligned_filtered_ sorted_output.sam> > <aligned_filtered_ sorted_output.bam>
```

where:

- <aligned_filtered_sorted_output.sam> is the sorted output file from Step E.4
- <aligned_filtered_sorted_output.bam> is output file

E.g.,

```
samtools view -S -b sorted_filtered.sam > sorted_filtered.bam
```

6. Use the unmapped BAM generated in Step E.1 and the aligned bam from Step E.5 to generate a mapped BAM that includes the UMI in the RX tag using Picard's MergeBamAlignment tool:

```
java -jar picard.jar MergeBamAlignment ALIGNED=<aligned_filtered_sorted_output.bam> UNMAPPED=<sorted_filtered_output.bam>
OUTPUT=<consensus_filtered_output.bam> REFERENCE_SEQUENCE=hg19.fa SORT_ORDER=coordinate
ALIGNER_PROPER_PAIR_FLAGS=true ALIGNED_READS_ONLY=true CREATE_INDEX=true VALIDATION_STRINGENCY=SILENT
EXPECTED_ORIENTATIONS=FR MAX_INSERTIONS_OR_DELETIONS=-1
```

where:

- <aligned_filtered_sorted_output.bam> is the sorted BAM file from Step E.5
- <sorted_filtered_output.bam> is the unmapped output file from Step E.1
- <consensus_filtered_output.bam> is the output file

E.g.,

```
java -jar picard.jar MergeBamAlignment ALIGNED=sorted_filtered.bam UNMAPPED=filtered_unmapped_sorted.bam OUTPUT=consensus.bam
REFERENCE_SEQUENCE=hg19.fa SORT_ORDER=coordinate ALIGNER_PROPER_PAIR_FLAGS=true ALIGNED_READS_ONLY=true
CREATE_INDEX=true VALIDATION_STRINGENCY=SILENT EXPECTED_ORIENTATIONS=FR MAX_INSERTIONS_OR_DELETIONS=-1
```

Related Products

Cat. #	Product	Size	License	Quantity	Details
R400742	ThruPLEX® Tag-Seq HV	24 Rxns		*	

ThruPLEX Tag-Seq HV uses a simple, three-step workflow to generate high-complexity DNA libraries with unique molecular tags from standard samples and challenging sample sources such as FFPE and cell-free plasma DNA. This product contains reagents for 24 reactions and includes unique dual indexing (UDI) primers.

Documents	Components	Image Data

R400743	ThruPLEX® Tag-Seq HV	96 Rxns		*	

Add to Cart

Takara Bio USA, Inc.
United States/Canada: +1.800.662.2566 • Asia Pacific: +1.650.919.7300 • Europe: +33.(0)1.3904.6880 • Japan: +81.(0)77.565.6999