

Mar 01, 2019 Version 4

Stranded Transcript Count Table Generation from Long Reads V.4

 Version 1 is forked from [Transcript Coverage Analysis from Long Reads](#)

DOI

dx.doi.org/10.17504/protocols.io.yqefvte

David A Eccles¹

¹Malaghan Institute of Medical Research (NZ)



David A Eccles

Malaghan Institute of Medical Research (NZ)

OPEN  ACCESS



DOI: dx.doi.org/10.17504/protocols.io.yqefvte

Protocol Citation: David A Eccles 2019. Stranded Transcript Count Table Generation from Long Reads. **protocols.io**
<https://dx.doi.org/10.17504/protocols.io.yqefvte>

License: This is an open access protocol distributed under the terms of the **Creative Commons Attribution License**, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited

Protocol status: In development

We are still developing and optimizing this protocol

Created: March 01, 2019

Last Modified: March 01, 2019

Protocol Integer ID: 20966



Abstract

This protocol is for comparing different samples at the transcript level, using long reads that are mapped to transcripts.

Input(s): demultiplexed fastq files (see protocol [Demultiplexing Nanopore reads with LAST](#)), transcript reference fasta file, annotation file

Output(s): transcript table, sorted by differential coverage, annotated with gene name / description / location

Before start

Obtain a transcript fasta file, and an annotation file. For the mouse genome, I use the following files:

1. Transcript [CDS] sequences from [Ensembl](#); [this file](#) was the most current when I last checked.
2. Annotation file obtained from [Ensembl BioMart](#) (Ensembl Genes → Mouse Genes) as a compressed TSV file with the following attribute columns:

- Transcript stable ID
- Gene description
- Gene start (bp)
- Gene end (bp)
- Strand
- Gene name
- Chromosome/scaffold name

Barcode Demultiplexing

- 1 Demultiplex reads as per protocol [Demultiplexing Nanopore reads with LAST](#).

If this has been done, then the following command should produce output without errors:

```
for bc in $(awk '{print $2}' barcode_counts.txt); do ls
reads_${bc}.fastq.gz; done
```

Example output:

```
reads_BC03.fastq.gz
reads_BC04.fastq.gz
reads_BC05.fastq.gz
reads_BC06.fastq.gz
reads_BC07.fastq.gz
reads_BC08.fastq.gz
```

If the *barcode_counts.txt* file is missing, the output will look like this:


```
awk: fatal: cannot open file `barcode_counts.txt' for reading (No
such file or directory)
```

If one or more of the barcode-demultiplexed files are missing, the output will look something like this:

```
reads_BC03.fastq.gz
reads_BC04.fastq.gz
reads_BC05.fastq.gz
ls: cannot access 'reads_BC06.fastq.gz': No such file or directory
ls: cannot access 'reads_BC07.fastq.gz': No such file or directory
reads_BC08.fastq.gz
```

Adapter Mapping

- 2 Prepare a FASTA file containing adapter sequences (see attached FASTA file).

 adapter_seqs.fa

- 3 Prepare the LAST index for the adapter file. This will generate seven additional files of the form <index name>.XXX:

```
lastdb adapter_seqs.fa adapter_seqs.fa
```


Orienting Reads

- 4 Map the reads to the adapter sequences. In this case it's important that the direction of mapping is also recorded, so the *cut* command selects three fields (query name [7], target name [2], mapping direction [10]):

```
for bc in $(awk '{print $2}' barcode_counts.txt);  
do echo "*** ${bc} ***";  
  lastal -Q 1 -P10 adapter_seqs.fa <(pv reads_${bc}.fastq.gz) | \  
    maf-convert -n tab | cut -f 2,7,10 | uniq | \  
    gzip > adapter_assignments_${bc}.txt.gz  
done
```

- 5 Reads are filtered into two groups (and one group-by-omission) based on the mapped direction of the strand-switch primer, then reverse-complemented (if necessary) to match the orientation of the original RNA strand. I use my [fastx-fetch.pl](#) and [fastx-rc.pl](#) scripts for this.

 [fastx-fetch.pl](#)

 [fastx-rc.pl](#)

```
mkdir -p oriented
for bc in $(awk '{print $2}' barcode_counts.txt);
do echo "*** ${bc} ***";
fastx-fetch.pl -i <(zgrep 'SSP' adapter_assignments_${bc}.txt.gz
| awk '{if($3 == "+"){print $2}}') <(pv reads_${bc}.fastq.gz) | \
gzip > oriented/${bc}_reads_fwd.fastq.gz
fastx-fetch.pl -i <(zgrep 'SSP' adapter_assignments_${bc}.txt.gz
| awk '{if($3 == "-"){print $2}}') <(pv reads_${bc}.fastq.gz) | \
fastx-rc.pl | gzip > oriented/${bc}_reads_rev.fastq.gz
done
```

- 6 Forward and reverse-oriented sequences are combined together to form a single group of RNA-oriented reads.

```
for bc in $(awk '{print $2}' barcode_counts.txt);
do echo "*** ${bc} ***";
pv oriented/${bc}_reads_fwd.fastq.gz
oriented/${bc}_reads_rev.fastq.gz | \
zcat | gzip > oriented/${bc}_reads_dirAdjusted.fastq.gz
done
```

Transcriptome mapping

- 7 Reads are mapped to the transcriptome with LAST.

The results of that mapping are piped through *last-map-probs* to exclude unlikely hits, then through *maf-convert* to convert to a one-line-per-mapping tab-separated format using the same fields as with the adapter mapping (query name [7], target name [2], mapping direction [10]).

This format is further processed to make sure that there is only one mapping per transcript-read pair, and then aggregated to sum up counts per transcript.

```
mkdir -p mapped
for bc in $(awk '{print $2}' barcode_counts.txt);
do echo "*** ${bc} ***";
lastal -Q 1 -P 10 Mus_musculus.GRCm38.cds.all.fa <(pv
oriented/${bc}_reads_dirAdjusted.fastq.gz | zcat) | \
last-map-probs | maf-convert -n tab | cut -f 2,7,10 | \
sort | uniq | awk -F'\t' -v "bc=${bc}" '{print bc,$1,$3}' | \
sort | uniq -c | \
gzip >
mapped/trnCounts_LAST_${bc}_vs_Mmus_transcriptome.txt.gz;
done
```

Annotation and Result Generation

- 8 Transcript counts are merged with ensembl gene annotation, then converted into wide format (one line per transcript) using an R script.

The transcript annotation in this case is from ensembl BioMart (see Guidelines for more details).



count_analysis.r

```
#!/usr/bin/env Rscript

library(dplyr);
library(tidyr);

## load used barcode identifiers
bcNames <- read.table("barcode_counts.txt",
stringsAsFactors=FALSE)[,2];

## load count data into "narrow" array (one line per count)
trn.counts <- tibble();
for(bc in bcNames){
  trn.counts <-
    bind_rows(trn.counts,
              as.tbl(read.table(

sprintf("mapped/trnCounts_LAST_%s_vs_Mmus_transcriptome.txt.gz",
bc),

col.names=c("count", "barcode", "transcript", "dir"),
            stringsAsFactors=FALSE)));
}

## remove revision number from transcript names (if present)
trn.counts$transcript <- sub("\\.[0-
9]+$", "", trn.counts$transcript);

## convert to wide format (one line per transcript)
trn.counts.wide <- spread(trn.counts, barcode, count) %>%
  mutate(dir = c("+="fwd", "-="rev")[dir]);
for(bd in colnames(trn.counts.wide[, -1])){
  trn.counts.wide[[bd]] <- replace_na(trn.counts.wide[[bd]], 0);
}

## load ensemble transcript metadata (including gene name)
ensembl.df <-
as.tbl(read.delim("ensembl_mm10_geneFeatureLocations.txt.gz",

col.names=c("transcript", "Description", "Start", "End",
            "Strand", "Gene", "Chr"),
            stringsAsFactors=FALSE));
ensembl.df$Description <- sub(" \\
[.*$", "", ensembl.df$Description);
```

```
ensembl.df$Description <- sub("^({50}).+${", "\\1...", ensembl.df$Description);
ensembl.df[,1:7] <- ensembl.df[,c(1,7,5,3,4,2,6)];
colnames(ensembl.df)[1:7] <- colnames(ensembl.df)
[c(1,7,5,3,4,2,6)];
options(scipen=15); ## don't show scientific notation for large
positions

## merge ensembl metadata with transcript counts
gene.counts.wide <- inner_join(ensembl.df, trn.counts.wide,
by="transcript");
gene.counts.wide <- gene.counts.wide[order(-
rowSums(gene.counts.wide[, -(1:8)])), ];
bcNames <- colnames(gene.counts.wide[, -(1:8)]);
## ignore columns with extremely low read counts
bcNames <- bcNames[colSums(gene.counts.wide[, bcNames]) > 10];
## write result out to a file
write.csv(gene.counts.wide,
file="wide_transcript_counts_LAST.csv",
row.names=FALSE);
```