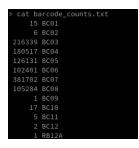


Apr 01, 2019 Version 2

© Demultiplexing Nanopore reads with LAST V.2

DOI

dx.doi.org/10.17504/protocols.io.znhf5b6



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.znhf5b6

External link: https://doi.org/10.5281/zenodo.2535894

Protocol Citation: David A Eccles 2019. Demultiplexing Nanopore reads with LAST. protocols.io

https://dx.doi.org/10.17504/protocols.io.znhf5b6

Manuscript citation:

License: This is an open access protocol distributed under the terms of the <u>Creative Commons Attribution License</u>, 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: April 01, 2019

Last Modified: April 01, 2019

Protocol Integer ID: 21929

Keywords: demultiplexing, nanopore, high-throughput sequencing



Abstract

This protocol is for a semi-manual method for read demultiplexing, as used after my presentation Sequencing DNA with Linux Cores and Nanopores to work out the number of reads captured by different barcodes.

Input: reads as a FASTQ file, barcode sequences as a FASTA file Output: reads split into single FASTQ files per target [barcode]

Note: barcode / adapter sequences are not trimmed by this protocol



Generating Barcode Index

Prepare a FASTA file containing barcode sequences (see attached FASTA file). To reduce the chance of mismatched adapters, this should *only* contain the barcode sequences. That restriction means this approach will not work for short reads, where the barcode sequences are very likely to occur within sequences.

```
barcode_base.fa
```

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

```
lastdb barcode_base.fa barcode_base.fa
```

Mapping Reads to Barcodes

3 Combine all input reads into a single file

```
pv ../called_all/*.fastq | gzip > reads_all.fastq.gz
```

Note: I'm using the pipe viewer command pv to produce a progress indicator while the command is running. If this command is not available, it can be replaced with cat with no change in function (apart from not showing progess).

Use LAST in FASTQ alignment mode (-Q 1) to map the reads. In this example, it is distributed over 10 processing threads (-P 10). Here *maf-convert* is used to convert to a single line per match, *cut* retains only the barcode and read IDs, and *uniq* is used to make sure that multiple same barcodes per read (e.g. for reverse / complement barcodes at each end) will not produce duplicates:

```
lastal -Q 1 -P10 barcode_base.fa <(pv reads_all.fastq.gz) | \
  maf-convert -n tab | cut -f 2,7 | uniq | \
  gzip > barcode_assignments.txt.gz
```

For an extremely stringent search, the output of lastal can be piped through last-mapprobs, which will reduce the likelihood of a partial barcode match to other DNA



sequences. The downside is that this is more likely to drop reads due to slight mismatches in the barcode portion of the read:

```
lastal -0 1 -P10 barcode_base.fa <(pv reads_all.fastq.qz) | last-
map-probs | \
  maf-convert -n tab | cut -f 2,7 | uniq | \setminus
  gzip > barcode_assignments.txt.gz
```

The output of this command will be a gripped tab-separated 2-column file with barcode names in the first column, and read IDs in the second column.

Optional [but recommended]: identifying chimeric reads

5 Identify reads with unique barcode classes attached to each read. The sort command sorts by the second field (read ID), then uniq identifies duplicated lines when ignoring the first field (barcode). Two uniq commands are used to allow for the possibility that a sequence could be tailed by matching barcodes at both ends. The downside of this is that it will also collapse chimeric reads with the same barcode. This uses 'uniq -u' to only print the unique lines.

```
pv barcode_assignments.txt.gz | zcat | sort -k 2,2 | uniq | \
  uniq -f 1 -u | gzip > unique_assignments.txt.gz
```

6 Identify reads with multiple barcodes (i.e. potentially chimeric reads). This is identical to the last step, except for using 'uniq -D' to only print duplicated reads. This step is only strictly needed when chimeric reads need to be inspected.

```
pv barcode_assignments.txt.gz | zcat | sort -k 2,2 | uniq | \
  uniq -f 1 -D | gzip > duplicate_assignments.txt.gz
```

Splitting Read File Per Barcode

7 Create a file containing barcode read counts for the appropriate read category. For example, for non-chimeric reads:

```
pv unique_assignments.txt.gz | zcat | awk '{print $1}' | \
  sort | uniq -c > barcode_counts.txt
```



Or for all reads:

```
pv barcode_assignments.txt.gz | zcat | awk '{print $1}' | \
  sort | uniq -c > barcode_counts.txt
```

8 For each discovered barcode, using the appropriate read category assignment file, find the corresponding read IDs, then extract those IDs out of the read FASTQ file. This uses one of my own scripts, fastx-fetch.pl, to do this directly from a FASTQ file:

```
for bc in $(awk '{print $2}' barcode_counts.txt);
 do echo "** ${bc} **";
 fastx-fetch.pl -i <(zgrep ${bc} unique_assignments.txt.gz | awk</pre>
'{print $2}') \
    <(pv reads_all.fastq.gz) | \
    gzip > reads_${bc}.fastq.gz;
done
```

Note: this step processes through the read file once per barcode, which could take a while depending on how many barcodes are detected.

Or, alternatively, for all reads:

```
for bc in $(awk '{print $2}' barcode_counts.txt);
 do echo "** ${bc} **";
 fastx-fetch.pl -i <(zgrep ${bc} barcode_assignments.txt.gz | awk</pre>
'{print $2}') \
    <(pv reads_all.fastq.gz) | \
    gzip > reads_${bc}.fastq.gz;
done
```