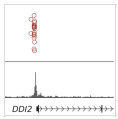


Oct 25, 2019

# Processing Single Cell Calling Card Sequencing Data

Arnav Moudgil[1], Rob Mitra[1]

[1]Washington University, Saint Louis

1 Works for me dx.doi.org/10.17504/protocols.io.4phgvj6

Transposon Calling Cards

Arnav Moudgil
Washington University, Saint Louis

ABSTRACT

Here we present a computational pipeline for processing single cell calling card (scCC) data. These data will have been generated from single cell RNA-seq libraries following transfection/-duction of either undirected *piggyBac* transposase or your favorite transcription factor (YFTF) fused to *piggyBac*. This workflow demonstrates how to process scCC sequencing data derived from a 10x Chromium-based scCC library; the workflow can be parallelized on distributed computing architectures (e.g. slurm).

MATERIALS TEXT

The following external programs are required:

- cutadapt (≥ 1.16)
- samtools (≥ 1.9)
- cellranger (≥ 2.1.0)

You will need a .2bit version of the genome sequence you are aligning to; these are readily available from the UCSC Genome Browser. (They can also be generated from a FASTA file using the faToTwoBit utility)

The following programs are optional, but highly recommended:

- bedtools (≥ 2.27)
- bedops (≥ 2.4)

In addition, this workflow calls some calling card-specific scripts, which use Python 3. It is recommended that your Python installation be relatively up-to-date (i.e. ≥ 3.4). To check your python version, type

```
python -V
```

You will need to install the following Python modules:

- numpy
- pandas
- pysam
- twobitreader

All of these packages are available on PyPI and can be installed via pip:

```
pip install numpy pandas pysam twobitreader
```

(If Python3 is not the default on your system, replace pip with pip3)

Finally, these are the calling card-specific scripts you will need, all of which are available on GitHub:

- TagBam.py
- AnnotateInsertionSites.py
- BamToCallingCard.py
- UMIFilter.py
- FilterUniqueBarcodes.py
- FilterBAMByBarcodes.py

- FilterCCFByBarcodes.py

BEFORE STARTING

Please make sure you have installed the required software and packages (see Materials section).

This protocol describes how to analyze a single cell calling cards* (scCC) experiment. These libraries are derived from single cell RNA-seq libraries (scRNA-seq). Currently, we only support 10x Chromium 3'-based libaries. Unlike bulk calling card libraries, in which we require multiple replicates (e.g. 10-12), each cell in a scCC library is considered a replicate. Sensitivity is driven by the total number of insertions recovered, which is directly proportional to the number of transformed cells in the scRNA-seq library. Therefore, we advise processing as many cells as feasible to maximize discovery of transcription factor binding sites.

*If you are unfamiliar with calling card libraries, we recommend reading our quick start guide and our scCC library preparation protocol.

## Preamble

1 The objective of this protocol is to take sequencing reads from single cell calling cards (scCC) library and process them into a CCF (calling card format; .ccf) file. A CCF file is a modified BED file (BED3+3) that concisely enumerates every transposition event in the sequenced library.

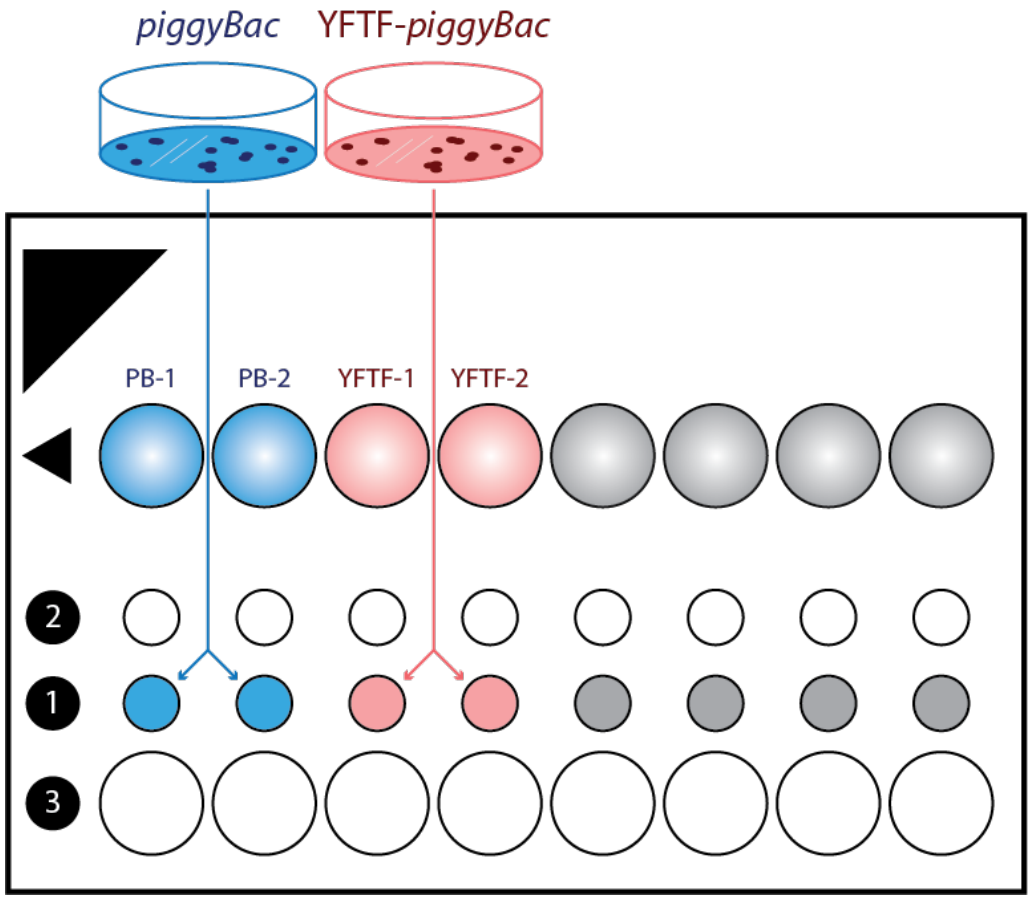
CCF files typically have six columns:

1. chrom: chromosome
2. start: beginning coordinate of the insertion site
3. end: ending coordinate of the insertion site; since *piggyBac* inserts into TTAA's, this typically spans the motif itself.
4. count: the number of reads supporting this insertion
5. strand: + or -, indicating which strand was targetted (optional but highly recommended)
6. barcode: a string identifying the library or cell from which this insertion originated. For scCC, this is the sequence of the cell barcode.

Each line of the final .ccf file represents an independent calling card insertion, and the value in the cell barcode column specifies in which cell this insertion was observed. If your sample is heterogeneous, you may find that your cells can be grouped into biologically meaningful clusters (e.g. different cell types) based on their scRNA-seq expression profiles. In this case, you will have assigned cell barcodes to each cluster, and can use this information to split the .ccf file to generate insertion profiles for each cluster (see Step 19). These split .ccf files can then be used to identify differentially bound loci, or for visualization of TF binding in different clusters.

This workflow walks through how to perform quality control, alignment, filtering, and processing of single cell calling card sequencing libraries to generate a .ccf file. This file can then be used in downstream applications, such as visualization on the (legacy) WashU Epigenome Browser (instructions here), and as input for peak calling.

2 To illustrate the workflow, let's say that we have performed scCC on our favorite transcription factor (YFTF) in a human cell line. We have prepared scRNA-seq and scCC libraries from 10,000 cells transfected with wild-type *piggyBac* transposase and 10,000 cells transfected with YFTF-*piggyBac*. Both the wild-type and YFTF transfectants were loaded across two wells each of a Chromium chip.




Example of a 10x Chromium chip loaded for a single cell calling cards experiment

This image illustrates our example experiment. Two wells have been loaded with *piggyBac*-transfected cells in Row 1 (blue) and two wells with YFTF-*piggyBac* transfectants (pink). The remaining wells, in grey, were empty (loaded with 50% glycerol). After emulsion generation, the resulting libraries–in the row marked by the triangle–were kept separate and processed according to the scCC molecular protocol.

3 scCC analysis requires two separate sequencing runs: one for the scRNA-seq library (which will be used for dimensionality reduction and cell type identification) and one for the scCC library (to assign insertions to specific cells). The scRNA-seq library can be sequenced using 10x's standard recommendations and processed using cellranger. **We will assume this step has already been completed.** This step establishes a set of high-quality cell barcodes from each library; we will cross-reference these with the scCC library to assign insertions to specific cells.

4 The scCC library should have been sequenced as recommended in our scCC molecular workflow. Specifically, on a dual indexed-compatible Illumina sequencer; we prefer to sequence these libraries on an Illumina NextSeq 500 with 50% phiX, allocating 26 bases to read 1, 50 bases to read 2, and 8 bases each to index 1 and index 2. Although scCC libraries should be demultiplexable with unique index sequences, this does not always work and the index 1 read can fail, reporting all N's. If this happens, reads from all libraries will be mixed together. We can identify scCC reads from phiX and other artifacts by demultiplexing with the index 2 read (should be GCGTCAAT). To further identify reads from the constituent libraries, we will demultiplex using the cell barcode.

5 In this example, we will be analyzing scCC sequencing run demultiplexed using index 2 only. The read 1 file will contain the cell barcode and UMI, while read 2 will contain the junction between the transposon and the genome. The read 1 and read 2 files are, respectively:

- PB_YFTF-PB_combined_R1_001.fastq.gz
- PB_YFTF-PB_combined_R2_001.fastq.gz

Read 1 is 26 bases long: the first 16 bases comprise the cell barcode, while the final 10 bases are the UMI.
Read 2 is 50 bases long: the first 2 bp will contain transposon sequence (GG) followed by the 4bp TTAA insertion site; the rest of the read is genomic sequence (and maybe some P7 adapter).

## Adapter Trimming

6 We first ensure that read 2 begins with GGTTAA. If it does, those bases are trimmed (hard clipped) to facilitate genomic alignment. Only reads with perfect matches are carried forward.

```
cutadapt \
    -g ^GGTTAA \
    -o PB_YFTF-PB_combined-trim1_R2_001.fastq.gz \
    -p PB_YFTF-PB_combined-trim1_R1_001.fastq.gz \
    --minimum-length 1 \
    --discard-untrimmed \
    -e 0 \
    --no-indels \
    PB_YFTF-PB_combined_R2_001.fastq.gz \
    PB_YFTF-PB_combined_R1_001.fastq.gz
```

Typically 70-90% of reads will pass this filter, although there may be sample-dependent variation.

7 Next, we re-examine the passing reads and trim any reads that end in the P7 adapter that was added during scCC library preparation. This step reduces the amount of non-genomic bases, which should accelerate alignment. Only a small fraction (5-10%) typically have any adapter sequence at all, so the majority of reads pass this filter.

```
cutadapt \
    -a AGAGACTGGCAAGTACACGTCGCACTCACCATGANNNNNNNNNATCTCGTATGCCGTCTTCTGCTTG \
    -o PB_YFTF-PB_S1_L001_R2_001.fastq.gz \
    -p PB_YFTF-PB_S1_L001_R1_001.fastq.gz \
    --minimum-length 1 \
    PB_YFTF-PB_combined-trim1_R2_001.fastq.gz \
    PB_YFTF-PB_combined-trim1_R1_001.fastq.gz
```

Here, NNNNNNNNN indicates where the index 1 sequence would be. The N's do not have to be replaced as cutadapt can tolerate ambiguous bases.

For running cellranger in next step, the input FASTQ filenames MUST conform to the following pattern:

[Sample Name]_S1_L00[Lane Number]_[Read Type]_001.fastq.gz

[Read Type] is R1 or R2. [Lane Number] can be varied; for simplicity, we use L001 here.

## Alignment

8 Now that our reads are trimmed, we are ready to align them to the genome. At the same time, we need to perform error-correction on our cell barcode and UMI sequences. We will use cellranger, as it can perform both whole-genome alignment and barcode curation at once. We need to specify the directory where the trimmed FASTQ files can be found. Note that this directory should be "flat", i.e have no subdirectories.

```
cellranger count \
    --id=PB_YFTF-PB_map_scCC \
    --fastqs=fastq_dir/ \
    --transcriptome=/opt/refdata-cellranger-GRCh38-3.0.0 \
    --sample=PB_YFTF-PB \
    --expect-cells=5000 \
    --nosecondary \
    --chemistry=SC3Pv2 \
    --localcores=16 \
    --localmem=30
```

A few notes on this command:

- --id= will specify the output directory that will be created. This will be familiar to anyone who was worked with cellranger before.
- --expect-cells= is an estimate for how many cells are present in the libray. This is much more important for scRNA-seq libraries than scCC libraries, and so can probably be safely excluded.
- --nosecondary skips the dimensionality reduction step of the cellranger pipeline. We are only concerned with mapping insertions to the genome.
- --chemistry= specifies the chemistry of the 10x kit. We prefer to explicitly specify this. Here, we used version 2 of the single cell 3' kit. The scCC workflow should also be immediately compatible with v3 chemistry.
- --localcores= and --localmem=30 specify machine settings. Here, we used 16 cores and 30 GB of memory. These can be adjusted to fit your setup.

cellranger automatically performs barcode whitelisting and error-correction of UMIs, which are encoded in the program's output .bam file. The **CB** tag contains the read's verified cell barcode, and the **UB** tag denotes the corrected UMI. A full description of cellranger BAM tags can be found here.

9 We then filter mapped reads for primary alignments, to eliminate multi-mapped reads:

```
samtools view \
    -b -h -F 260 \
    -o PB_YFTF-PB_map_scCC_uniq.bam \
    PB_YFTF-PB_map_scCC/outs/possorted_genome_bam.bam
```

## Annotation

10 As when processing bulk RNA calling card data, we annotate the filtered BAM file with various tags to connect reads to insertion sites. A usefule side effect of this is that the tagged BAM serves as an archive of a scCC experiment. We use the following custom tags for scCC libraries:

- XI: insertion site annotation
- XZ: adjacent sequence (to verify transposase motif)

(The XP, XJ, and XK tags are only used in bulk RNA calling cards.)

11 Now we will annotate reads with respect to the insertion site. This script checks each read to make sure that it maps next to the *piggyBac* insertion site motif TTAA. Remember, this part of read 1 was trimmed in step 5. By double checking that the read maps next to a genomic TTAA, we add an extra layer of specificity to the alignment. The sequence of the adjacent bases will also be annotated with the XZ tag. Reads that pass will be annotated with the insertion site coordinates in the XI tag and written to the output file.

```
python AnnotateInsertionSites.py \
    --transposase PB \
    -f \
    PB_YFTF-PB_map_scCC_uniq.bam \
    hg38.2bit \
    PB_YFTF-PB_map_scCC_tagged.bam
```

You can provide a path to the .2bit file if your genome references are in another directory.

12 We perform one last quality control check on the processed scCC reads. Since scCC libraries involve intramolecular circularization, there is a small chance that concatamerization can occur. These would appear as singleton events where a cell barcode and UMI are linked to an insertion in a different cell. To guard against this, we require all insertions in a given cell (i.e. sharing the same cell barcode) to have at least two different UMIs each. This yields libraries with excellent specificity (see e.g. Figure 3B in our preprint)

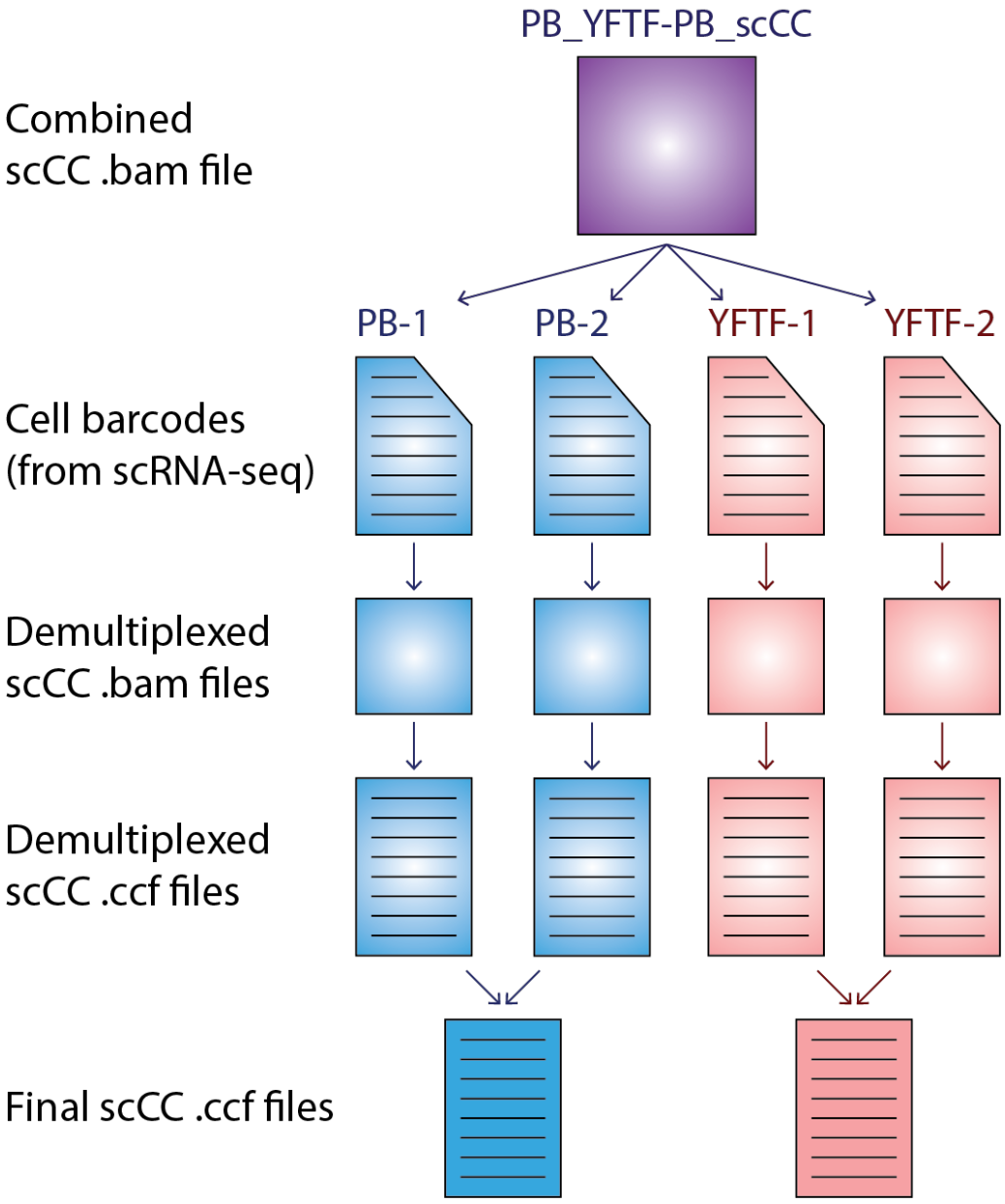
```
python UMIFilter.py \
    -p 10x \
    -i PB_YFTF-PB_map_scCC_tagged.bam \
    --verbose \
    -o PB_YFTF-PB_map_scCC_final.bam
```

13 Finally, we convert this BAM file to a (sorted) CCF file. The sorting step relies on bedops; see here for alternatives.

```
python BamToCallingCard.py \
    -b CB \
    -i PB_YFTF-PB_map_scCC_final.bam \
    -o PB_YFTF-PB_map_scCC_unsorted.ccf
sort-bed PB_YFTF-PB_map_scCC_unsorted.ccf > PB_YFTF-PB_map_scCC_final.ccf
```

## Demultiplexing

14 At this point, PB_YFTF-PB_map_scCC_final.bam contains all insertions from all cells in our single cell library, both from the wild-type *piggyBac* transfectants and the YFTF-*piggyBac* transfectants. How can we determine which insertions came from which library? We will use the cell barcodes to further demultiplex the CCF file.




This figure summarizes Steps 15-18, wherein we use the cell barcodes (obtained from the respective scRNA-seq libraries) to demultiplex the combined scCC .bam file, generate .ccf files from each library, and finally create master .ccf files for each condition (i.e. *piggyBac* and YFTF-*piggyBac* treatments).

15 In step 2, we prepared our 10x libraries by loading four wells of the Chromium chip: two for wild-type PB, and two for YFTF-PB. Let us call these libraries PB-1, PB-2, YFTF-1, and YFTF-2; further, assume that we have completed the scRNA-seq portions of the scCC workflow, including analysis with cellranger. For each of these four libraries, we can get a list of high-quality barcodes.

After cellranger has finished, each library's cell barcodes can be found in the following locations, respectively:

```
PB-1/outs/filtered_gene_bc_matrices/hg38/barcodes.tsv
PB-2/outs/filtered_gene_bc_matrices/hg38/barcodes.tsv
YFTF-1/outs/filtered_gene_bc_matrices/hg38/barcodes.tsv
YFTF-2/outs/filtered_gene_bc_matrices/hg38/barcodes.tsv
```

Starting with cellranger v3, the barcodes.tsv file may be gzipped (barcodes.tsv.gz). If that is the case, you will need to unzip before proceeding (gunzip barcodes.tsv.gz)

16 Since GEM generation was performed independently for each sample, there is a small chance (see note below) that the same cell barcode was captured more than once across libraries. This could, in theory, confound interpretation of TF binding, as a shared cell barcode may belong to cells of different types or states. While the effect of these is likely small, we recommend discarding shared barcodes between libraries. The following command takes in a set of barcode files; for each, an output file is created containing the subset of unique barcodes found only in the respective input file.

```
python UMIFilter.py \
    -i PB-1/outs/filtered_gene_bc_matrices/hg38/barcodes.tsv \
       PB-2/outs/filtered_gene_bc_matrices/hg38/barcodes.tsv \
       YFTF-1/outs/filtered_gene_bc_matrices/hg38/barcodes.tsv \
       YFTF-2/outs/filtered_gene_bc_matrices/hg38/barcodes.tsv \
    -o PB-1_unique_barcodes.txt \
       PB-2_unique_barcodes.txt \
       YFTF-1_unique_barcodes.txt \
       YFTF-2_unique_barcodes.txt
```

Note that this can command can take in any number of input files (2, 3, 4, 5, etc.). The only requirements are: (1) a matched list of output files is provided; and (2) the input barcode files contain one barcode per line.

For the curious: the probability of a shared barcode (i.e. barcode collision) between two 10x scRNA-seq libraries is quite small, but is dependent on library size. For two libraries of 5,000 cells each, the probability is < 1%. As the number of libraries increases, the probability of collision increases approximately. We have filtered unique cell barcodes across as many as six libraries and have discarded no more than 5% of total cell barcodes.

17 Now that we have a list of cell barcodes unique to each library, we can demultiplex our calling card BAM file. The example show is for PB_1 but can be generalized to all samples.

```
python FilterBAMByBarcodes.py \
    -i PB_YFTF-PB_map_scCC_final.bam
    -b PB-1_unique_barcodes.txt \
    -o PB-1_scCC_final.bam
```

We can now convert this BAM file to CCF output.

```
python BamToCallingCard.py \
    -b CB \
    -i PB-1_scCC_final.bam \
    -o PB-1_scCC_unsorted.ccf
```

We can also combine the two wild-type *piggyBac* libraries into a single, sorted CCF file. (The second step requires bedops; see here for alternative sorting commands).

```
cat PB-1_scCC_unsorted.ccf PB-2_scCC_unsorted.ccf | sort-bed - > PB_scCC_final.ccf
cat YFTF-PB-1_scCC_unsorted.ccf YFTF-PB-2_scCC_unsorted.ccf | sort-bed > YFTF-PB_scCC_final.ccf
```

18 At last, we have a CCF file containing all insertions across all (unique) cells in a scCC experiment. This file can be further visualized on the WashU Epigenome Browser and used as input for peak calling. Here is an example of scCC CCF output.

```
chr1 29884 29888 3 + GCATGATCAGACGTAG-1
chr1 30355 30359 4 - CAGCTGGTCGCAAACT-1
chr1 32116 32120 11 - GTGTGCGAGCTTCGCG-1
chr1 32303 32307 674 + GTCGTAAAGGTAGCTG-1
chr1 33031 33035 2 - TTAGTTCTCAACACTG-1
chr1 33031 33035 21 + GCAATCAGTGGTTTCA-1
chr1 33031 33035 25 + GCACTCTAGTAGCCGA-1
chr1 33031 33035 98 + GTTTCTACAGACGCAA-1
chr1 33169 33173 26 - CAAGAAAGTACAGCAG-1
chr1 34572 34576 4 - CGTTCTGCAAATTGCC-1
```

Notes

19 In the course of analzying your scRNA-seq data, you may find biologically meaningful clusters and may wish to identify differentially bound loci. Let us suppose that in the your analysis of the YFTF-PB transfectants, you find two clusters of cells (Alfa and Bravo) and wish to stratify insertions specific to each cluster. If the cell barcodes in each cluster are in Barcodes_Alfa.txt and Barcodes_Bravo.txt, we can directly filter insertions from the YFTF-PB CCF file, instead of going back to the BAM file.

```
python FilterCCFByBarcodes.py \
    -i YFTF-PB_scCC_final.ccf \
    -b Barcode_Alfa.txt \
    -o YFTF-PB_scCC_Alfa.ccf

python FilterCCFByBarcodes.py \
    -i YFTF-PB_scCC_final.ccf \
    -b Barcode_Bravo.txt \
    -o YFTF-PB_scCC_Bravo.ccf
```

Note that if the input CCF file is sorted, the output file should automatically be sorted as well.

20 Currently, the scCC pipeline **does not** support 10x scRNA-seq libraries merged using *cellranger aggr*. Guidance on this will be provided in the future.