**Bintu Lab CUT&RUN Pipeline**

**Written by Connor Ludwig**

**PART 1 – Preparation and Alignment**

Genomics assays involve the enrichment of a feature of interest and mapping the sequencing reads to a reference genome to understand the genome-wide distribution of the feature. When focusing on the distribution of these features across endogenous genes only, we can use pre-built reference genomes (e.g. hg19 or GRCh38). When interested in the distribution of these features across both endogenous and synthetic genes (e.g. reporter genes), we need to build a custom reference genome.

***Section 1.1 – Preparing a Reference Genome***

For the following section, you will be using the Linux command line. Open Ubuntu and navigate to the folder where your CUT&RUN scripts and data are/will be stored. You can change directories using ‘cd’ and list directory contents with ‘ls’ to help navigate. To go up a level you can use ‘../’, which can also precede a file or folder. The following section also uses Python 3, called from the command line.

To create a custom reference genome, we first need to create a FASTA file for your synthetic sequence in Sublime or in another text editor software. FASTA files have two lines per entry, where the first line starts with ‘>’ and contains the sequence identifier (e.g. gene name). Avoid spaces in these identifiers. The second line contains the DNA sequence, which you can copy from Benchling.

Create the custom genome with your synthetic sequences appended as extra chromosomes to the primary genome (e.g. hg19 or GRCh38). First, generate a FASTA file for this custom genome by using concat\_fastas.py:

python concat\_fastas.py [path\_to\_primary\_fasta] [path\_to\_synthetic\_sequence\_fasta] [output\_fasta\_save\_path]

Example:

python scripts/concat\_fastas.py ../genomes/hg19.fa ../genomes/no\_arms\_puro-pA-9xTO-pEF-H2B-cit-NS-H2B-mCherry.fasta ../genomes/hg19\_cit-NS-mCherry\_at\_end.fasta

Build the custom genome using bowtie2-build. You’ll need to specify the concatenated FASTA file from the previous step as well as a root name for your new genome in the directory where you want it saved. The easiest way to devise a root name for the genome is to use the concatenated FASTA name without the file extension. This script will produce six .bt2 files sharing the root name.

bowtie2-build [path\_to\_concat\_fasta] [desired\_save\_dir/root\_name\_for\_new\_genome]

Example:

bowtie2-build ../genomes/hg19\_cit-NS-mCherry\_at\_end.fasta ../genomes/hg19\_cit-NS-mCherry\_at\_end

***Section 1.2 – Demultiplexing Sequencing Data***

Download raw data from your NextSeq run. Name your sample sheet ‘SampleSheet.csv’ and place this in the run folder. Demultiplex your data to generate separate fastq.gz files for each sample and for each direction sequenced.

You can find an example sample sheet in the BINTU\_LAB\_CUT&RUN folder

bcl2fastq --runfolder-dir [dir\_in\_path] -p 12 --output-dir [dir\_out\_path] --no-lane-splitting

Example:

bcl2fastq --runfolder-dir 230110\_NB551514\_0003\_AHGY7HBGXN/ -p 12 --output-dir fastq\_files/ --no-lane-splitting

bcl2fastq --runfolder-dir 230122\_NB551514\_0010\_AHNYV5BGXN/ -p 12 --output-dir fastq\_files/ --no-lane-splitting

bcl2fastq --runfolder-dir 230123\_NB551514\_0011\_AHNYV7BGXN/ -p 12 --output-dir fastq\_files/ --no-lane-splitting

Create a sub-directory for each sample (named with the same root as the read files) and move the paired read fastq.gz files to the appropriate folder. Place these folders into a parent directory called ‘fastq’ or something similar (exact name does not matter). A script to automate or avoid this process would be helpful.

***Section 1.3 – Batch Processing Alignment and De-duplication***

Run the first part of the CUT&RUN pipeline, which batch processes the following:

* Alignment of reads to your specified genome using bowtie2
  + Input: pairs of fastq.gz files
  + Output: sam files
* (Optional) removal of reads that fall entirely within repeated elements (e.g. H2B, polyA)
  + Input: sam files
  + Output: edited sam files
* Conversion from sam to sorted bam
  + Input: sam files
  + Output: sorted bam files
* Marking and removing duplicate reads using Picard
  + Input: sorted bam files
  + Output: de-duplicated bam files

The script requires three inputs and can take an optional fourth:

1. Path to the directory containing sub-directories with pairs of fastq.gz files
2. Path to the reference genome (common root - exclude .bt2 extensions)
3. Path to the Picard tool for quantifying and marking duplicates
4. (Optional) Path to a script for editing sam files – these are specific to a particular synthetic sequence and require the user to define the coordinates of the elements they wish to exclude

This script keeps a TXT file record of all steps and compiles sample read counts at each stage in a CSV alignment summary file. Run the script as follows:

python scripts/cutrun\_pipeline\_v9\_2021-10-16.py [parent\_fastq\_directory] [genome\_directory/root\_name\_for\_new\_genome] [path\_to\_picard] [optional\_script\_for\_editing\_sam\_files]

Example:

python scripts/cutrun\_pipeline\_v9\_2021-10-16.py fastq/ ../genomes/hg19\_cit-NS-mCherry\_at\_end ../tools/picard.jar scripts/edit\_sam\_K562\_NS.py

python scripts/cutrun\_pipeline\_v9\_2021-10-16\_12-cores.py fastq/ reference\_genomes/hg19\_pjt039 tools/picard.jar scripts/edit\_sam\_pJT039.py

python scripts/cutrun\_pipeline\_v9\_2021-10-16\_12-cores.py fastq/reference\_genomes/ecoli tools/picard.jar scripts/edit\_sam\_pJT039.py

**PART 2 – Normalization**

Normalization methods are employed to correct for slight differences in how samples are treated during an assay and the depth to which they are sequenced. There are myriad normalization methods, each of which have pros/cons and are more appropriate in certain situations.

***Section 2.1A – Normalization via CPM***

The most basic method normalizes sample counts to the total number of aligned reads to account for differences in sequencing depth. This is the default method for the CUT&RUN pipeline and only requires calling the script (no need to specify inputs). From the command line, run the followingscripts/cutrun\_pipeline\_part2\_20220425.py

The script produces bedgraph files that can be viewed in a genome browser, such as IGV, and used for downstream analyses.

***Section 2.1B – Normalization via E. coli Spike-in***

More recent CUT&RUN workflows involve spiking *E. coli* DNA into the STOP buffer when quenching pAG-MNase activity. Because the same amount of *E. coli* DNA is added to each sample, this normalization method accounts for differences in sample processing from genomic DNA extraction onward. This method requires that the user run PART 1 of the CUT&RUN pipeline again using the *E. coli* genome as a reference. Make sure not to overwrite the data generated from alignment to the primary genome! It may help to make a new folder for this analysis.

Next, run the second part of the CUT&RUN pipeline (same script in Section 2.1A) with an additional argument – the CSV alignment summary file produced from alignment to the *E. coli* genome. This information is used in combination with the analogous CSV alignment summary file produced from alignment to the primary genome to calculate a scaling factor for each sample based on the amount of detected *E. coli* DNA.

python scripts/cutrun\_pipeline\_part2\_20220425.py ../ecoli/analysis\_summary.csv

The script produces bedgraph files that can be viewed in a genome browser, such as IGV, and used for downstream analyses.

***Section 2.1C – Normalization via KmetStat Panel***

More recent CUT&RUN workflows (Epicypher specifically) involve spiking in something called the KmetStat panel during antibody incubation. This panel includes bead-tethered recombinant nucleosomes harboring different histone modifications and modification-specific DNA barcodes. While primarily used for quality control (i.e. how specific is an antibody for its target histone modification?), the KmetStat panel can also be used for normalization for samples within an antibody condition. The antibody incubation step is rather early in the experimental workflow, meaning that the KmetStat normalization method accounts for differences in sample processing from this incubation onward. Next, run the second part of the CUT&RUN pipeline (same script in Section 2.1A) with an additional argument – the CSV alignment summary file produced from alignment to Kmet barcodes This information is used in combination with the analogous CSV alignment summary file produced from alignment to the primary genome to calculate a scaling factor for each sample based on the amount of detected Kmet reads.

First generate KMET read counts for normalization by counting barcodes corresponding for your antibody target.

Run the following to generate metstat read counts for normalization and antibody QC:

kmetstat\_analysis\_summary.py

python scripts/kmetstat\_analysis\_summary.py fastq/ analysis\_summary.csv 3

Can plot the QC matrix (antibody specificity) with:

plot\_kmetstat\_qcmatrix.py

python scripts/plot\_kmetstat\_qcmatrix.py kmetstat\_counts\_matrix.csv analysis\_summary\_kmet.csv 3

After obtaining KMET counts run part2 of the cut&run pipeline with kmet as an argument and the kmet counts analysis summary as an additional input.

python scripts/cutrun\_pipeline\_part2\_20220425.py ../kmet/analysis\_summary.csv

**PART 3 – Peak Calling**

Beyond simply viewing sample tracks in a genome browser and qualitatively assessing distribution differences between samples, we can use quantitative peak-calling methods to identify regions of significant signal. While there are several peak-calling tools available, SEACR is one that is better suited to the lower background associated with CUT&RUN samples. SEACR provides options to call peaks relative to an IgG control sample (preferred) or without this control, with options such as stringent versus relaxed peak-calling and performing normalization or not if this was already performed. The called peaks can then be used for a number of downstream analyses, some of which are explored in later parts of this document.

***Section 3.1 – SEACR***

First, copy the SEACR folder containing the bash script (or just the script) to your scripts directory. Next, run the following on the command line with bash:

bash SEACR\_1.3.sh [experimental\_bedgraph] [control\_bedgraph OR value\_from\_0\_to\_1\_for\_top\_fraction\_peaks] [norm OR non] [stringent OR relaxed] [output\_prefix]

Example:

bash scripts/SEACR/SEACR\_1.3.sh dedup\_scaled/K562\_CL048\_dox\_BR1\_FLAG\_S1\_pic.bam\_dedup.scaled.bedgraph dedup\_scaled/K562\_CL048\_dox\_BR1\_IgG\_S9\_pic.bam\_dedup.scaled.bedgraph norm stringent SEACR\_peaks/K562\_CL048\_dox\_BR1

For parallel processing of peaks can use Run\_Seacr.py python script. Place it in your scripts folder and modify the paths within the script.