CHAPTER 2

SAW PIPELINES & ARGUMENTS

2.1 splitMask

splitMask is designed for splitting Stereo-seq Chip T mask file according to the SE FASTQ CID. The split count is directly related to the number of split FASTQs while writing FASTQ out from sequencer.

Running **splitMask** requires the following files:

- Stereo-seq Chip T mask file **(.h5)**

🕓 Expected running time for 1 cm x 1 cm chip: ~5 min, Memory: ~3G

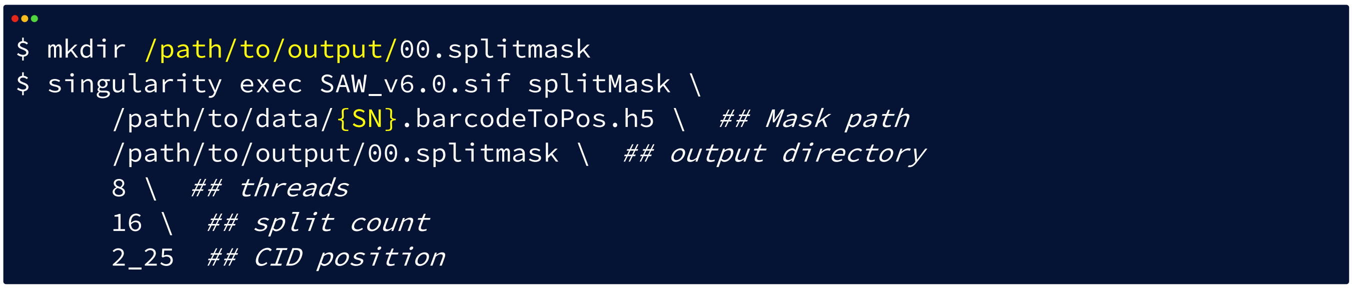
2.1.1 Arguments and Options

Table 2-1 **splitMask** Arguments and Options

Parameter index	Function
[1]	(Required) Stereo-seq Chip T mask file path (.h5).
[2]	(Required) Output directory stores split mask files.
[3]	(Required) Set the number of threads to be used.
[4]	(Optional) Split count. This count number needs to be set the same as SE FASTQ count. The options are the powers of 4, and the commonly used options are 16 and 64. If a library contains two barcodes sequenced in the same lane and the two barcodes were split when written out FASTQs, then the split count is 16; otherwise, if the barcodes were not split, then 64.
[5]	(Optional) CID position, commonly used option is "**2_25**". **splitMask** uses 24 bases (out of 25, 25 is the read length of CID) to split Stereo-seq Chip T mask file. This CID position parameter is a string that combines two index numbers with "_". The two numbers indicate the start and the end base index in the CID, and the bases in this range are used for splitting. For example, "**2_25**" means starting from the 2nd base to the 25th base used for splitting mask file.

2.1.2 Usage Example

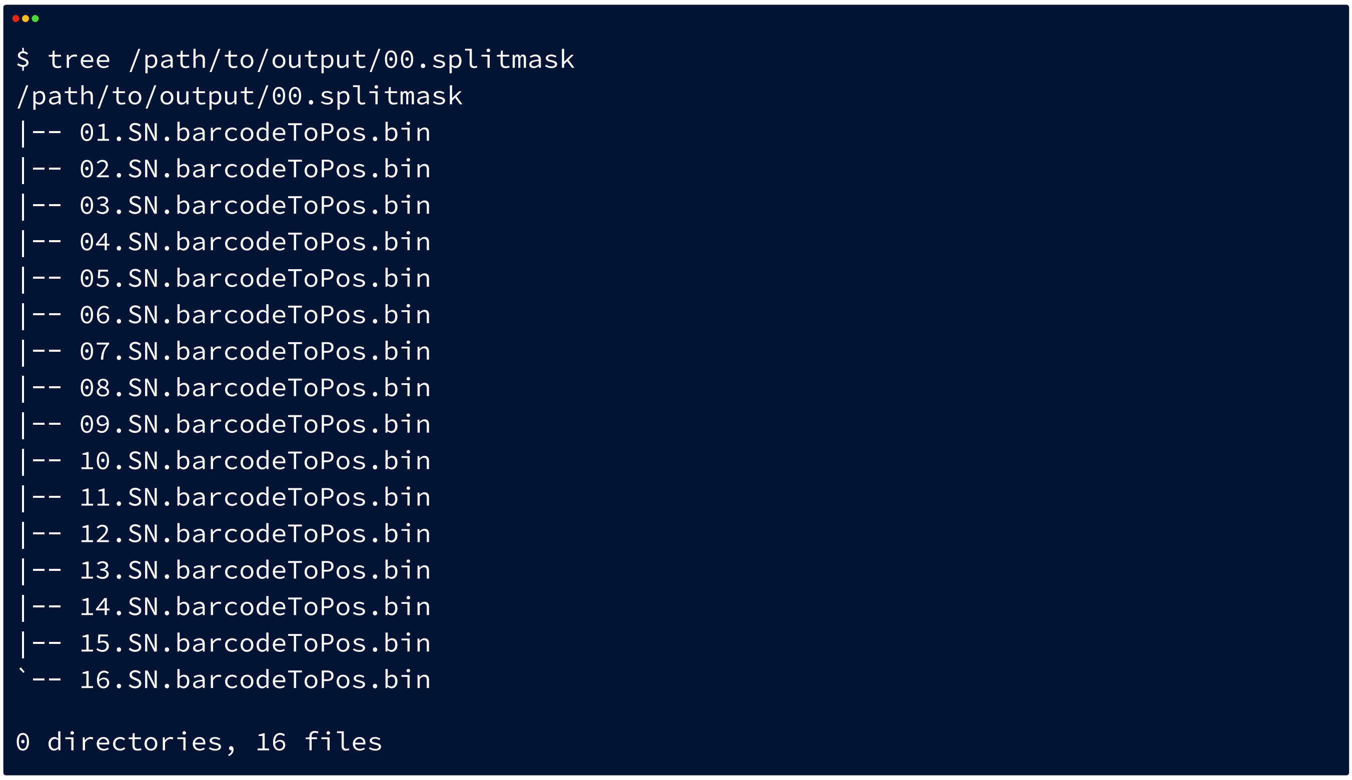
Please Note! Replace `{SN}` with your Stereo-seq Chip T serial number (SN, e.g. SS200000135TL_D1 or A00135D1)



```
$ mkdir /path/to/output/00.splitmask
$ singularity exec SAW_v6.0.sif splitMask \
      /path/to/data/{SN}.barcodeToPos.h5 \  ## Mask path
      /path/to/output/00.splitmask \  ## output directory
      8 \  ## threads
      16 \  ## split count
      2_25  ## CID position
```

2.1.3 Outputs

The output files of **`splitMask`** are organized as below:



```
$ tree /path/to/output/00.splitmask
/path/to/output/00.splitmask
|-- 01.SN.barcodeToPos.bin
|-- 02.SN.barcodeToPos.bin
|-- 03.SN.barcodeToPos.bin
|-- 04.SN.barcodeToPos.bin
|-- 05.SN.barcodeToPos.bin
|-- 06.SN.barcodeToPos.bin
|-- 07.SN.barcodeToPos.bin
|-- 08.SN.barcodeToPos.bin
|-- 09.SN.barcodeToPos.bin
|-- 10.SN.barcodeToPos.bin
|-- 11.SN.barcodeToPos.bin
|-- 12.SN.barcodeToPos.bin
|-- 13.SN.barcodeToPos.bin
|-- 14.SN.barcodeToPos.bin
|-- 15.SN.barcodeToPos.bin
`-- 16.SN.barcodeToPos.bin

0 directories, 16 files
```

2.2 CIDCount

`CIDCount` is a program for computing the number of CIDs in the Stereo-seq Chip T mask file and roughly estimating memory used in **`mapping`**.

Running **`CIDCount`** requires the following files:

- Stereo-seq Chip T mask file **(.h5)**

🕓 Expected running time for 1 cm x 1 cm chip: ~30 s, Memory: ~0.7G

2.2.1 Arguments and Options

Table 2-2 `CIDCount` Arguments and Options

Parameter	Function
`-i`	(Required) Stereo-seq Chip T mask file path (.h5 or .bin).
`-s`	(Required) A string of species name.
`-g`	(Required) Genome file size in GB.

2.2.2 Usage Example

For PE FASTQ input cases, run **`CIDCount`** as below:

```
$ singularity exec SAW_v6.0.sif CIDCount \
      -i /path/to/data/{SN}.barcodeToPos.h5 \  ## Stereo-seq Chip T mask file path
      -s {speciesName} \  ## species name
      -g {genomeSize}  ## genome file size in GB, can be acquired by “ls -l
--block-size=GB ${Genome file of the species after STAR indexing}”
```

For SE FASTQ input cases that require to run **`splitMask`** first, users may run **`CIDCount`** only once because the results for each individual small mask are close to each other. For chip size larger than 1 cm x 1 cm, we would recommend to run **`CIDCount`** for each CID class/small mask.

Please Note! {index} needs to be replaced by the real index number of the split mask file.

```
$ singularity exec SAW_v6.0.sif CIDCount \
      -i /path/to/output/00.splitmask/{index}.{SN}.barcodeToPos.bin \  ## Ste-
reo-seq Chip T mask file path
      -s {speciesName} \  ## species name
      -g {genomeSize}  ## genome file size in GB, can be acquired by “ls -l
--block-size=GB ${Genome file of the species after STAR indexing}”
```

2.2.3 Outputs

The output of **`CIDCount`** is shown as below,

```
$ singularity exec SAW_v6.0.sif CIDCount -i SN.barcodeToPos.h5 -s mouse -g 3
645784920  ## CID count
62  ## estimated memory for mapping
```

2.3 mapping

Each Stereo-seq sequenced read contains a CID sequence which is used as a key to spatially map the read back to its original location on the tissue section. `mapping` pipeline matches CID of the original sequencing reads stored in the FASTQ file with the records of CID-coordinates key-value pairs saved in the Stereo-seq Chip T mask file (allow 1 mismatch). Coordinate information for reads that CID could be paired with, based on the records of the mask file, will be added. Coordinate information for reads that CID could be paired with will be added based on the records of the mask file. Reads that get the coordinate annotations are Valid CID mRNA Reads (**Valid CID Reads**). After discarding reads with unqualified MID reads which do not satisfy further analysis, filtering reads with adapter, and filtering short reads (read length less than 30 after trimming consecutive A bases), the filtered **Valid CID Reads** are the **Clean Reads**. `mapping` pipeline maps **Clean Reads** to the reference genome, and output sorted BAM[7] format alignments and summary report.

Running `mapping` requires the following files:

- Stereo-seq sequenced reads FASTQ files **(.fq.gz)**
- Stereo-seq Chip T mask file **(.h5)**
- Indexed reference genome
- bcPara file **(.bcPara)**, please check the content of **Table 2-4**

Expected running time for ~1G reads: ~4 h, Memory: ~67G

If concerned about rRNA counting and filtering, necessary modification to the reference genome should be completed beforehand, mainly including 2 steps: 1) add rRNA information into the FASTA file, with the suffix '_rRNA' on the chromosome column, like '1_rRNA', to distinguish them from original reference; 2) build index using modified reference genome by STAR[8] function.

2.3.1 Arguments and Options

As `mapping` encapsulate STAR[8] function, it accepts additional options beyond those shown in the table below.

Table 2-3 `mapping` Arguments and Options

Parameter	Function
`--outSAMattributes spatial`	Set to turn on spatial BAM file format mode.
`--outSAMtype BAM SortedByCoordinate`	(STAR option) Set output BAM file sorted by coordinate.
`--genomeDir`	(STAR option) Path to the directory where the genome indices are stored.
`--runThreadN`	(STAR option; defaults to 1) Set the number of threads to be used. Usually set to 8 or higher.
`--outFileNamePrefix`	(STAR option) Custom output file prefix.
`--sysShell /bin/bash`	(STAR option) Path to the shell binary.
`--stParaFile`	(Required) Create a parameter file that defines CID mapping options. Options are specified in Table 2-4.
`--readNameSeparator \" \"`	(STAR option) Character(s) separating the part of the read names that will be trimmed in output.
`--limitBAMsortRAM`	(STAR option) Maximum available RAM (bytes) for sorting BAM.
`--limitOutSJcollapsed`	(STAR option) Max number of collapsed junctions.
`--limitIObufferSize`	(STAR option) Max available buffers size (bytes) for input/output, per thread.
`--outBAMsortingBinsN`	(STAR option; defaults to 50) Number of genome bins for sorting coordinates. If the file size of read2 FASTQ is greater than 200, it is better to set this value to 100.

Table 2-4 `mapping --stParaFile` Arguments and Options

Parameter	Function
`in`	(Required) Path to the Stereo-seq Chip T mask file (PE) or split mask file (SE).
`in1`	(Required) Path to the FASTQ file. For PE sequences specify the path to the FASTQ file of read1 here. For SE sequences, input FASTQ file with the same split index as the mask file. A more elegant way to input a long list of SE FASTQs is to organize them in a `FQ_{index}.list` file. Please check **2.3.2 Usage Example SE scenario 2** to get more information.
`in2`	(Optional) Path to the FASTQ file of read2. Only valid for PE sequencing.
`encodeRule`	(Required for PE) Encoding rule for the four bases. ACTG stands for A->0, C->1, T->2, G->3.
`out`	(Optional) Set output file prefix.
`action`	(Required; defaults to 1) Action number. Set to 4 in `mapping` Valid options: 1=CID stat, 2=CID overlap, 3=get CID position map, 4=map CID to slide, or 5=merge CID list.
`barcodeReadsCount`	(Required) Mapped CID list file with read counts for each CID. This is an output file in `mapping`
`platform`	(Optional) Sequencing platform. Valid options include but are not limited to: **SEQ500, G400, T1, T7, T10**.
`barcodeStart`	(Required; defaults to 0) CID start position. Set to 0.
`barcodeLen`	(Required; defaults to 25) CID length. Set to 25 for PE, 24 for SE.
`umiStart`	(Required; defaults to 25) MID start position.
`umiLen`	(Required; defaults to 10) MID length.
`umiRead`	(Required; defaults to 1) Declare the read contains MID.
`mismatch`	(Required; defaults to 0) Max mismatch tolerant. Usually set to 1 in `mapping`.
`bcNum`	(Required) CID count in mask. Please check **2.2 CIDCount** for more information.
`polyAnum`	(Optional) Number of consecutive A in the read that will be trimmed. Recommend to set this value to 15.
`mismatchInPolyA`	(Optional) Number of mismatch bases in searching poly A. Recommend to set this value to 2
`rRNAremove`	(Optional; default to disable) Remove rRNA reads. Note that the precondition of the switch is the reference genome combined with rRNA information. If not, this filtering function will not work anyway. Using the reference genome with the addition of rRNA, reads related to rRNA are not filtered by default, namely the switch is off. And the number and the ratio of rRNA reads in the statistical output file are both 0. When the switch is on, as long as the query reads are mapped to rRNA reference, they will be removed. rRNA count and ratio are recorded in the output file.

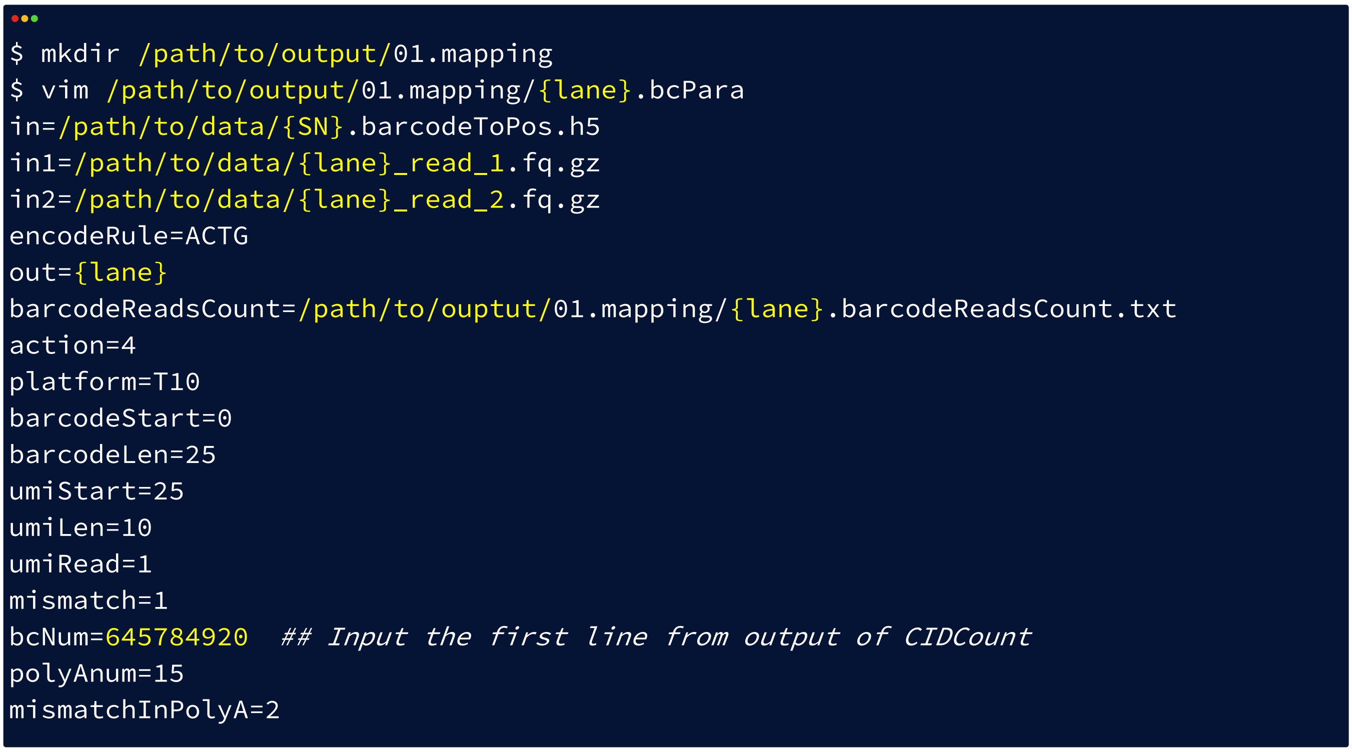
2.3.2 Usage Example

Two scenarios for preparing `mapping` **--stParaFile** input file `{lane}.bcPara` with PE FASTQ input:

Please Note! Replace `{SN}` with your Stereo-seq Chip T serial number (SN, e.g. SS200000135TL_D1 or A00135D1), and `{lane}` with the FASTQ lane name prefix (e.g. E100026571_L01)

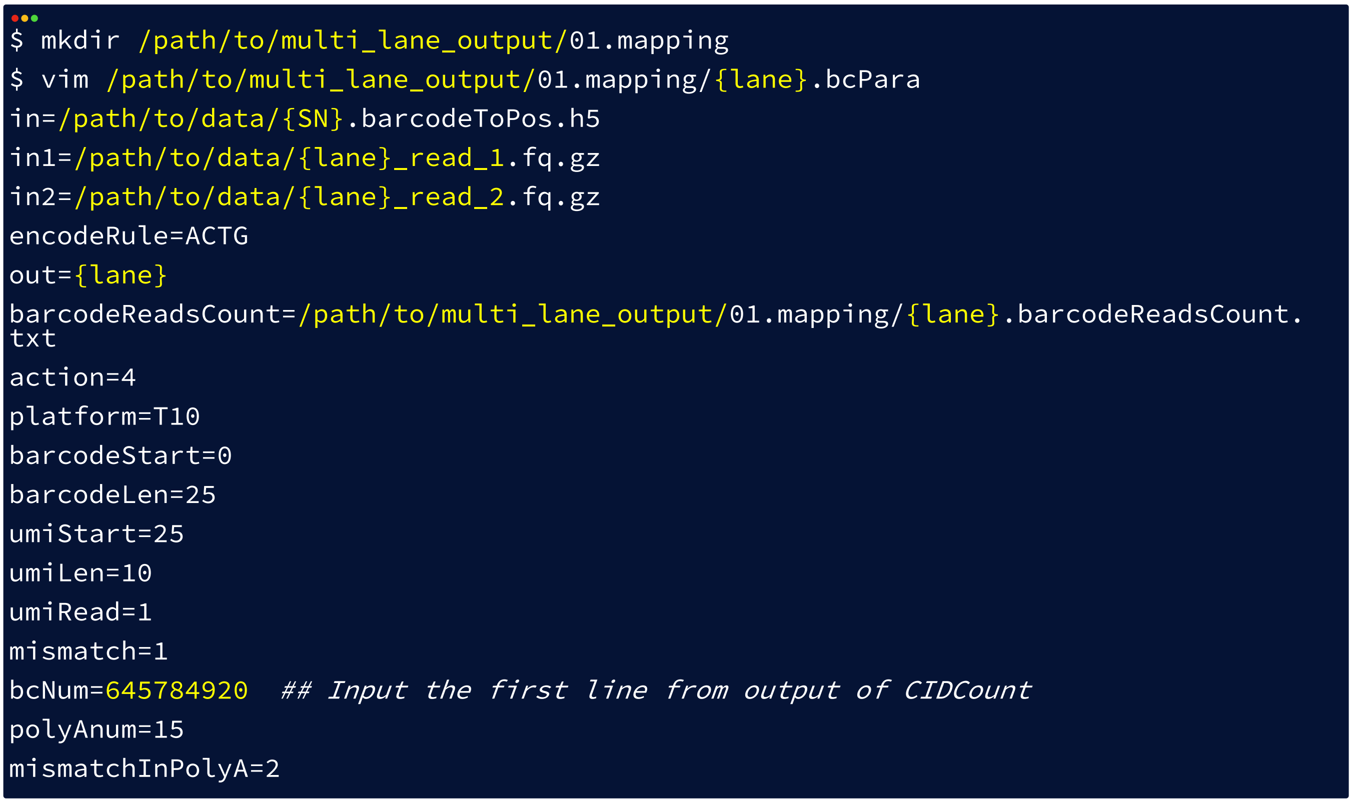
The same applies to all examples.

PE scenario 1: Prepare **`{lane}.bcPara`** file for PE one pair FASTQ as **`mapping`** input:



```
$ mkdir /path/to/output/01.mapping
$ vim /path/to/output/01.mapping/{lane}.bcPara
in=/path/to/data/{SN}.barcodeToPos.h5
in1=/path/to/data/{lane}_read_1.fq.gz
in2=/path/to/data/{lane}_read_2.fq.gz
encodeRule=ACTG
out={lane}
barcodeReadsCount=/path/to/ouptut/01.mapping/{lane}.barcodeReadsCount.txt
action=4
platform=T10
barcodeStart=0
barcodeLen=25
umiStart=25
umiLen=10
umiRead=1
mismatch=1
bcNum=645784920  ## Input the first line from output of CIDCount
polyAnum=15
mismatchInPolyA=2
```

PE scenario 2: Prepare **`{lane}.bcPara`** file for PE multiple pair of FASTQ as **`mapping`** input:



```
$ mkdir /path/to/multi_lane_output/01.mapping
$ vim /path/to/multi_lane_output/01.mapping/{lane}.bcPara
in=/path/to/data/{SN}.barcodeToPos.h5
in1=/path/to/data/{lane}_read_1.fq.gz
in2=/path/to/data/{lane}_read_2.fq.gz
encodeRule=ACTG
out={lane}
barcodeReadsCount=/path/to/multi_lane_output/01.mapping/{lane}.barcodeReadsCount.
txt
action=4
platform=T10
barcodeStart=0
barcodeLen=25
umiStart=25
umiLen=10
umiRead=1
mismatch=1
bcNum=645784920  ## Input the first line from output of CIDCount
polyAnum=15
mismatchInPolyA=2
```

Run **mapping** for PE input:

```
$ singularity exec SAW_v6.0.sif mapping \
                --outSAMattributes spatial \
                --outSAMtype BAM SortedByCoordinate \
                --genomeDir /path/to/genomeDir \
                --runThreadN 8 \
                --outFileNamePrefix /path/to/output/01.mapping/{lane}. \
                --sysShell /bin/bash \
                --stParaFile /path/to/output/01.mapping/{lane}.bcPara \
                --readNameSeparator \" \" \
                --limitBAMsortRAM 38582880124 \
                --limitOutSJcollapsed 10000000 \
                --limitIObufferSize=280000000 \
                --outBAMsortingBinsN 50 \
                > /path/to/output/01.mapping/{lane}_barcodeMap.stat
```

Two scenarios for preparing **mapping** --**stParaFile** input file **{index}.bcPara** with SE FASTQ input:

SE scenario 1: Prepare **{index}.bcPara** file for SE FASTQ that contains one barcode in library or did not split barcode when writing FASTQ:

Please Note! {index} need to be replaced by the index of the split mask file which is corresponding to the index of the SE FASTQ file.

```
$ mkdir /path/to/output/01.mapping
$ vim /path/to/output/01.mapping/{index}.bcPara
in=/path/to/output/00.splitmask/{index}.{SN}.barcodeToPos.bin  ## split mask file
in1=/path/to/data/{lane}_{index}.fq.gz  ## {index}th FASTQ file in {lane}
out={SN}.bc.out${index}
barcodeReadsCount=/path/to/ouptut/01.mapping/{index}.barcodeReadsCount.txt
action=4
platform=T10
barcodeStart=0
barcodeLen=24
umiStart=25
umiLen=10
umiRead=1
mismatch=1
bcNum=38284877  ## Input the first line from output of CIDCount
polyAnum=15
mismatchInPolyA=2
```

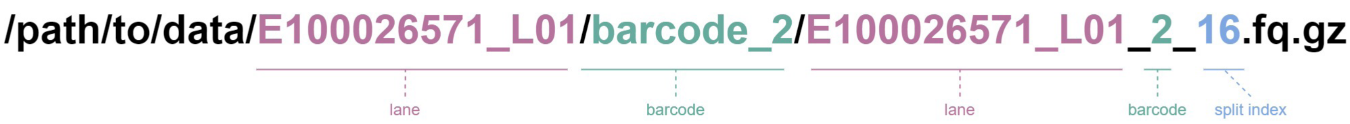
SE scenario 2: Prepare `{idx}.bcPara` file for SE FASTQ that has multiple barcodes in the library and split when writing FASTQ:

Please Note! `{index}` needs to be replaced by the real index number of the split mask file and `{idx}` needs to be replaced by the index number of the input FASTQ file. For example, if a library contains 2 barcodes that are sequenced in the same lane, the `{index}` and `{idx}` for the third FASTQ file of barcode 1 are 01 and 03, respectively.

```
$ mkdir /path/to/output/01.mapping
$ vim /path/to/output/01.mapping/{idx}.bcPara
in=/path/to/output/00.splitmask/{index}.{SN}.barcodeToPos.bin  ## split mask file
in1=/path/to/data/{lane}_{index}.fq.gz  ## {index}th FASTQ file in {lane}. FASTQ
files for different barcode usually stores in different directory, so /path/to/
data/ might change to /path/to/data/{barcode_n}
out={SN}.bc.out${idx}
barcodeReadsCount=/path/to/ouptut/01.mapping/{idx}.barcodeReadsCount.txt
action=4
platform=T10
barcodeStart=0
barcodeLen=24
umiStart=25
umiLen=10
umiRead=1
mismatch=1
bcNum=38284877 ## Input the first line from output of CIDCount
polyAnum=15
mismatchInPolyA=2
```

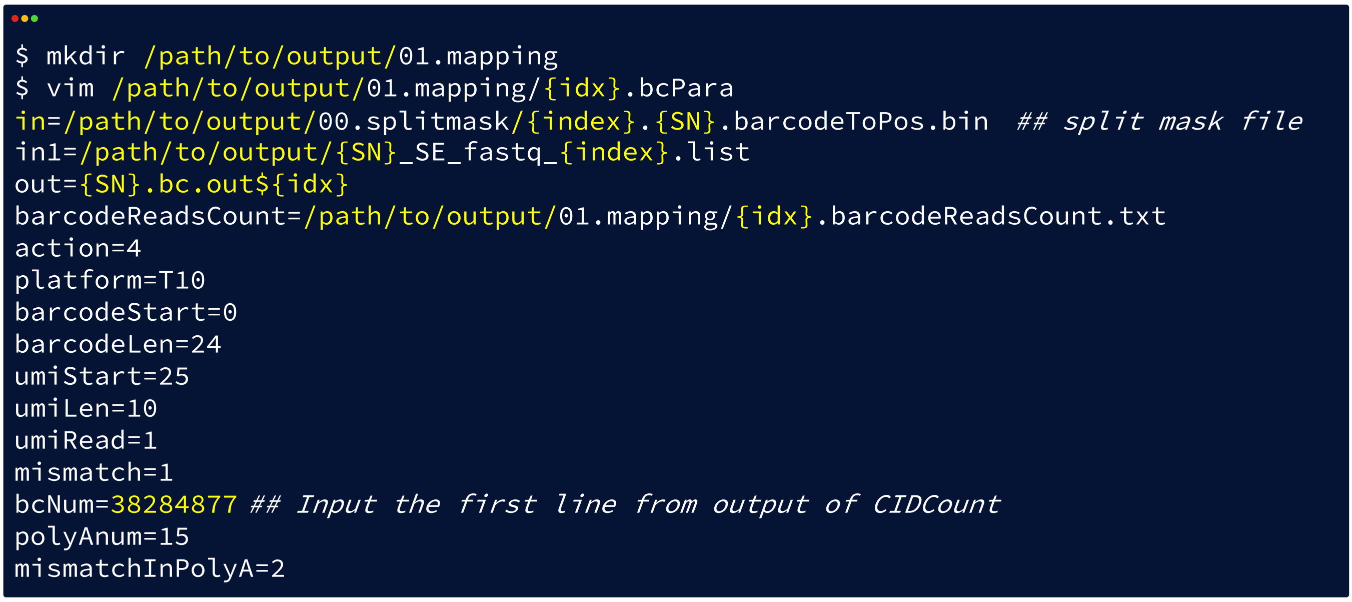
In case there are too many FASTQ files that need to be processed, an easier way is to organize them into a **`FQ_{index}.list`** file. The requirement for preparing a **`FQ_{index}.list`** file is to gather all the FASTQs with the same index as the split mask together, since these files are all split by the same logic.

SE FASTQ name


/path/to/data/E100026571_L01/barcode_2/E100026571_L01_2_16.fq.gz
lane
barcode
lane
barcode
split index

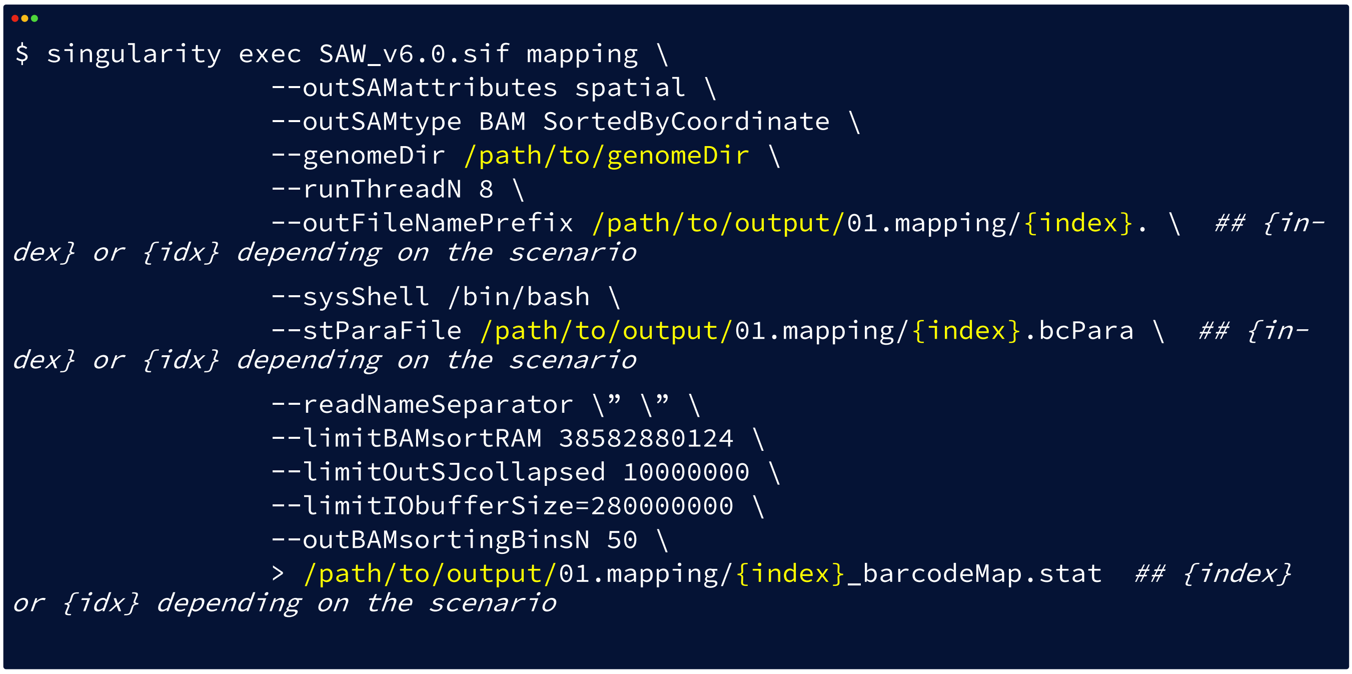
```
$ cat SN_SE_fastq_16.list  ## a FASTQ list file gathers all the FASTQs whose index
is 16
/path/to/data/lane_1_16.fq.gz
/path/to/data/lane_2_16.fq.gz
```

Then input the path of **`FQ_{index}.list`** file for **`in1`** parameter in the **`{idx}.bcPara`** file. The **`{index}`** of **`FQ_{index}.list`** and the **`{index}`** of split mask file have to be the same.



```
$ mkdir /path/to/output/01.mapping
$ vim /path/to/output/01.mapping/{idx}.bcPara
in=/path/to/output/00.splitmask/{index}.{SN}.barcodeToPos.bin  ## split mask file
in1=/path/to/output/{SN}_SE_fastq_{index}.list
out={SN}.bc.out${idx}
barcodeReadsCount=/path/to/output/01.mapping/{idx}.barcodeReadsCount.txt
action=4
platform=T10
barcodeStart=0
barcodeLen=24
umiStart=25
umiLen=10
umiRead=1
mismatch=1
bcNum=38284877 ## Input the first line from output of CIDCount
polyAnum=15
mismatchInPolyA=2
```

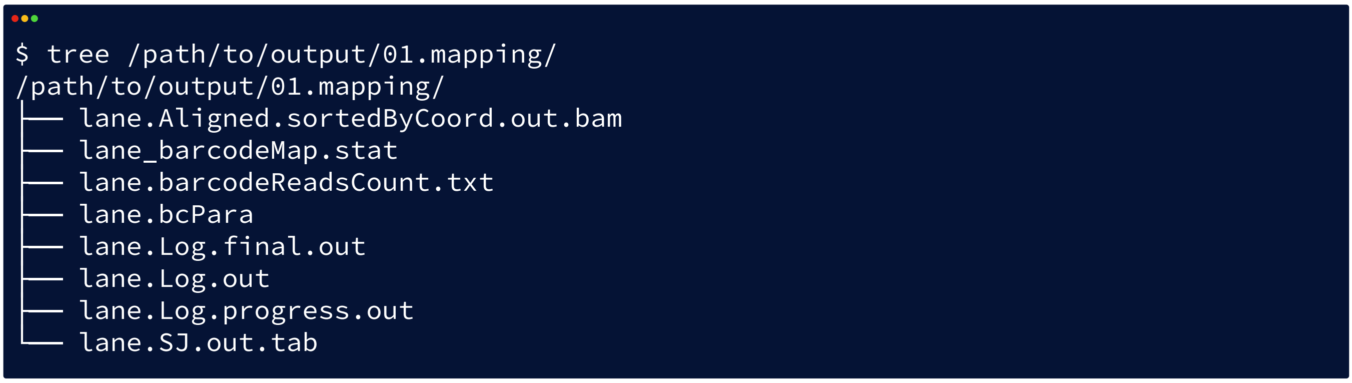
Run **`mapping`** pipeline



```
$ singularity exec SAW_v6.0.sif mapping \
                --outSAMattributes spatial \
                --outSAMtype BAM SortedByCoordinate \
                --genomeDir /path/to/genomeDir \
                --runThreadN 8 \
                --outFileNamePrefix /path/to/output/01.mapping/{index}. \  ## {index} or {idx} depending on the scenario

                --sysShell /bin/bash \
                --stParaFile /path/to/output/01.mapping/{index}.bcPara \  ## {index} or {idx} depending on the scenario

                --readNameSeparator \” \” \
                --limitBAMsortRAM 38582880124 \
                --limitOutSJcollapsed 10000000 \
                --limitIObufferSize=280000000 \
                --outBAMsortingBinsN 50 \
                > /path/to/output/01.mapping/{index}_barcodeMap.stat  ## {index} or {idx} depending on the scenario
```

2.3.3 Outputs

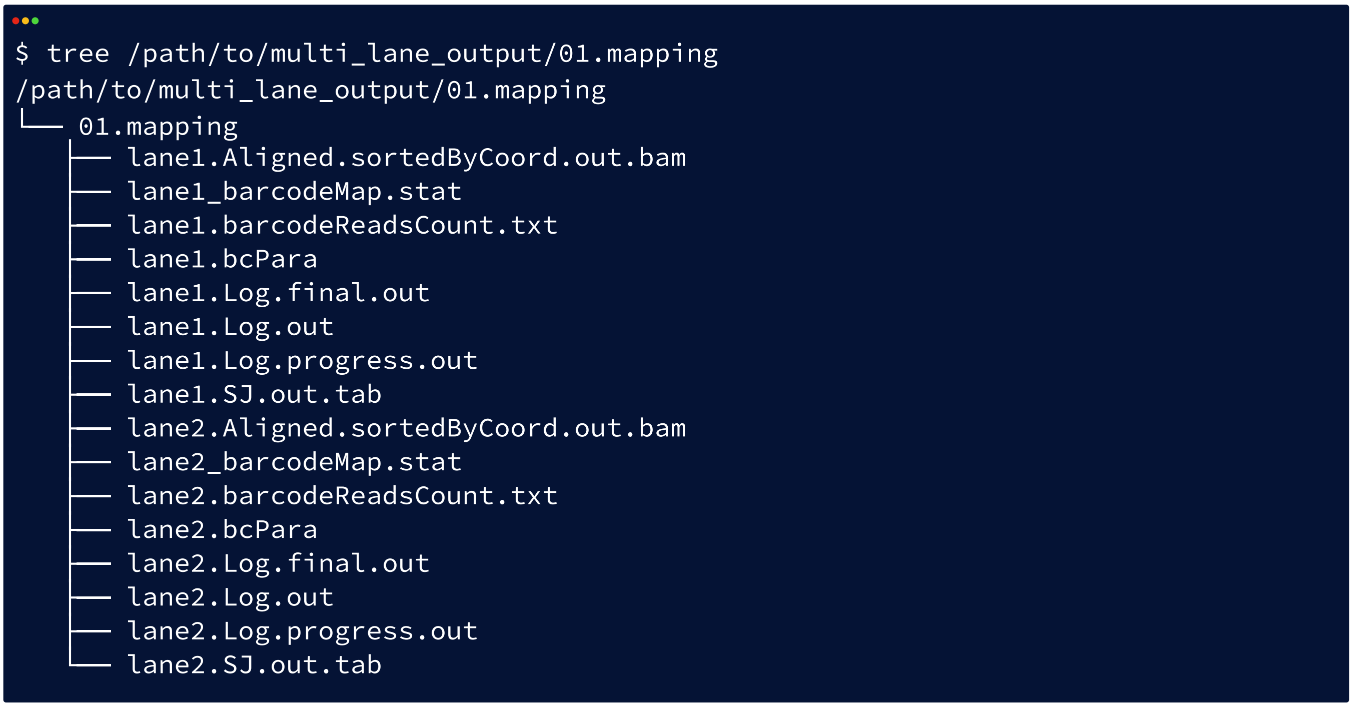
PE scenario 1 output files are organized as below:



```
$ tree /path/to/output/01.mapping/
/path/to/output/01.mapping/
├── lane.Aligned.sortedByCoord.out.bam
├── lane_barcodeMap.stat
├── lane.barcodeReadsCount.txt
├── lane.bcPara
├── lane.Log.final.out
├── lane.Log.out
├── lane.Log.progress.out
└── lane.SJ.out.tab
```

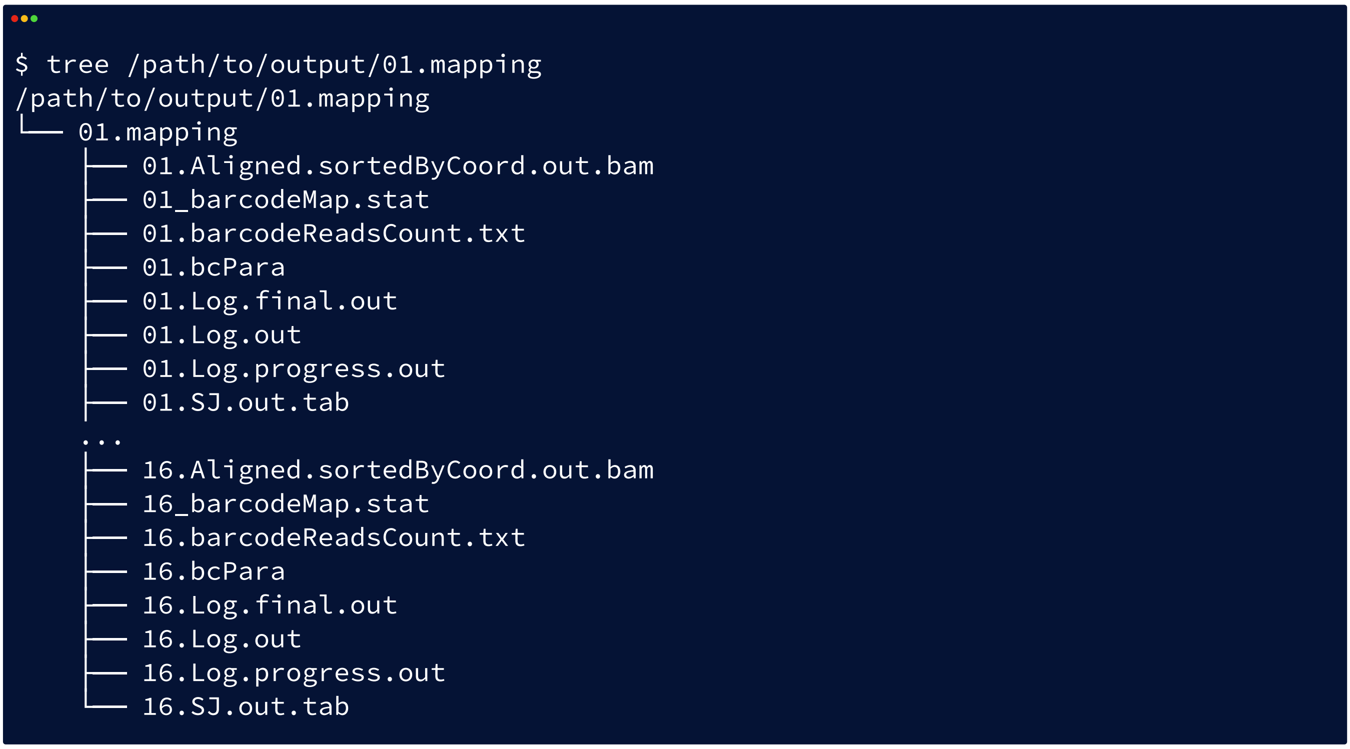
PE scenario 2 output files are organized as below (Here shows an example of 2 pairs of FASTQ):

If one sample has multiple FASTQ files, you need to run `mapping` for each FASTQ pair.



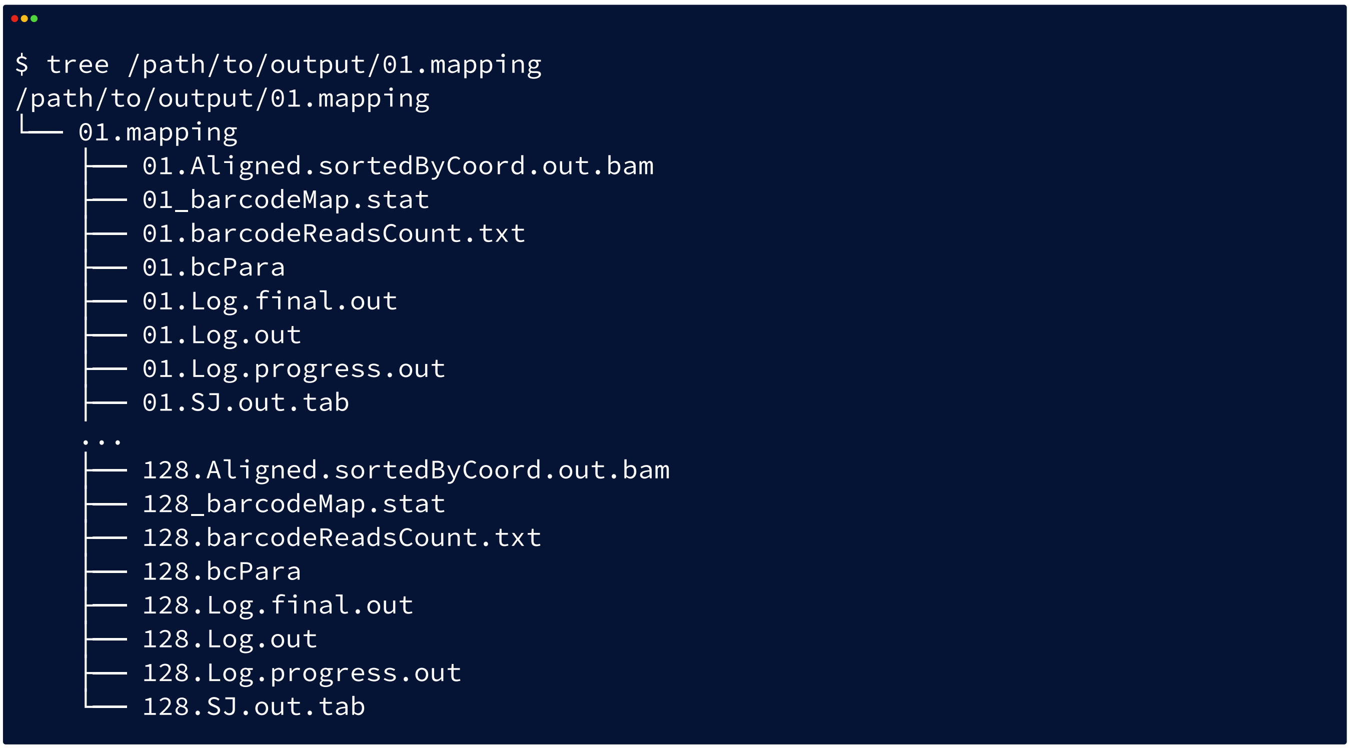
```
$ tree /path/to/multi_lane_output/01.mapping
/path/to/multi_lane_output/01.mapping
└── 01.mapping
    ├── lane1.Aligned.sortedByCoord.out.bam
    ├── lane1_barcodeMap.stat
    ├── lane1.barcodeReadsCount.txt
    ├── lane1.bcPara
    ├── lane1.Log.final.out
    ├── lane1.Log.out
    ├── lane1.Log.progress.out
    ├── lane1.SJ.out.tab
    ├── lane2.Aligned.sortedByCoord.out.bam
    ├── lane2_barcodeMap.stat
    ├── lane2.barcodeReadsCount.txt
    ├── lane2.bcPara
    ├── lane2.Log.final.out
    ├── lane2.Log.out
    ├── lane2.Log.progress.out
    └── lane2.SJ.out.tab
```

SE scenario 1 (one barcode in library or did not split barcode when writing FASTQ) output files are organized as below:



```
$ tree /path/to/output/01.mapping
/path/to/output/01.mapping
└── 01.mapping
    ├── 01.Aligned.sortedByCoord.out.bam
    ├── 01_barcodeMap.stat
    ├── 01.barcodeReadsCount.txt
    ├── 01.bcPara
    ├── 01.Log.final.out
    ├── 01.Log.out
    ├── 01.Log.progress.out
    ├── 01.SJ.out.tab
    ...
    ├── 16.Aligned.sortedByCoord.out.bam
    ├── 16_barcodeMap.stat
    ├── 16.barcodeReadsCount.txt
    ├── 16.bcPara
    ├── 16.Log.final.out
    ├── 16.Log.out
    ├── 16.Log.progress.out
    └── 16.SJ.out.tab
```

SE scenario 2 (multiple barcodes in the library and split when writing FASTQ) output files are organized as below (Here shows an example of 2 barcodes):



```
$ tree /path/to/output/01.mapping
/path/to/output/01.mapping
└── 01.mapping
    ├── 01.Aligned.sortedByCoord.out.bam
    ├── 01_barcodeMap.stat
    ├── 01.barcodeReadsCount.txt
    ├── 01.bcPara
    ├── 01.Log.final.out
    ├── 01.Log.out
    ├── 01.Log.progress.out
    ├── 01.SJ.out.tab
    ...
    ├── 128.Aligned.sortedByCoord.out.bam
    ├── 128_barcodeMap.stat
    ├── 128.barcodeReadsCount.txt
    ├── 128.bcPara
    ├── 128.Log.final.out
    ├── 128.Log.out
    ├── 128.Log.progress.out
    └── 128.SJ.out.tab
```

2.4 merge (optional)

SAW **`merge`** pipeline is used to combine the results of **`mapping`**.

Running **`merge`** requires the following file:

- **`mapping`** output mapped CID list files **(.txt)**

Expected running time for ~1G reads 2 lanes: ~5 min, Memory: ~5G

2.4.1 Arguments and Options

Table 2-5 `merge` Arguments and Options

Parameter	Function
`[1]`	(Required) Path to the Stereo-seq Chip T mask file.
`[2]`	(Required) Mapped CID list files with read counts for each CID.
`[3]`	(Required) Mapped CID list file which merges all input files.

2.4.2 Usage Example

```
$ mkdir /path/to/multi_lane_output/02.merge
$ singularity exec SAW_v6.0.sif merge \
      /path/to/data/{SN}.barcodeToPos.h5 \
      /path/to/multi_lane_output/01.mapping/{lane1}.barcodeReadsCount.txt,/path/to/
multi_lane_output/01.mapping/{lane2}.barcodeReadsCount.txt \  ## change {lane} to
{index} or {idx} for SE and change /path/to/multi_lane_output/ to /path/to/output/
for SE single lane scenario
      /path/to/multi_lane_output/02.merge/{SN}.barcodeReadsCount.txt
```

* Please be noted that we use the backward slash "\" to indicate the end of a line in a command that spans multiple lines.

2.4.3 Ouputs

The output file of **`merge`** has been organized as below:

```
$ tree /path/to/multi_lane_output/02.merge
/path/to/multi_lane_output/02.merge
└── {SN}.merge.barcodeReadsCount.txt
```

2.5 count

SAW **count** is an efficient general-purpose read annotation pipeline that label reads with their overlapped genomic features and outputs statistical information for the overall summarization result. Through quantification of annotated reads, **count** generates spatial gene expression data after de-duplicating reads according to CID, gene ID, and MID information. Usually, SAW **count** is run on the **Uniquely Mapped Reads** filtered from **mapping** output based on the reference genome annotation records. Starting from SAW v5.1.3, **count** allows the utilization of some Multi-Mapped Reads in quantification (**--multi_map**). The gene expression level in SAW pipeline is the summation of both intron and exon MID count. To support downstream analysis that might be required of different genomic features, **count** also outputs exon MID count separately.

Running **count** requires the following files:

- **mapping** output BAM file **(.bam)**
- Reference genome annotation GFF/GTF[9,10] file **(.gff / .gtf)**

🕒 Expected running time for ~1G reads: 0.5 h, Memory: ~45 G

2.5.1 Arguments and Options

Table 2-6 count Arguments and Options

Parameter	Function
-i	(Required) **mapping** output BAM file. Separate multiple files by comma.
-o	(Required) Set the **count** output BAM file name.
-a	(Required) Gene annotation GFF/GTF file.
-s	(Required) Set the **count** output statistical summary report file name.
-e	(Required) Set the **count** output gene expression file name.
--umi_len	(Required; defaults to 10) MID length.
--sn	(Required) Stereo-seq Chip T serial number (SN).
-c	(Optional; defaults to detected) CPU core number to use. Minimum value is 8, recommended value is 24.
--save_lq	(Optional; defaults to false) Save low-quality reads if set.
--save_dup	(Optional; defaults to false) Save duplicate reads if set.
--umi_on	(Optional; defaults to false) Correct MID if set.
--sat_file	(Optional; defaults to None) Set the saturation sampling file name which is prepared for sequencing saturation (requires **--umi_on**).
-m	(Optional; defaults to detected) Set available memory (GB).
--multi_map	(Optional; defaults to disable) Set to enable multi-mapped reads correction. This correction consists of two logics. 1) The first logic is the correction of multi-gene reads which is a read mapped uniquely to a genomic region where multiple genes overlap. In this scenario, the read has to overlap with a genomic locus greater than 50% of its read length. If the genomic feature (exon, intron, or intergenic) of all the mapped genes includes exon and intron, then select the alignment record mapped to exon in preference to intron. If more than one gene locus belongs to the same genomic feature type, then pick the one that has the longest overlap. Otherwise, label the read to "intergenic." 2) The second logic is to select and correct one of the multi-mapped reads and add the count to gene expression matrix. The first step is to group reads by QNAME. Select reads in the group mapped to exon in preference to those mapped to intron. Then correct the longest overlapped reads (at least overlapped greater than 50% of its read length) in the group to unique read and correct its MAPQ to 255. Set the MAPQ of the remaining reads to 0.

2.5.2 Usage Example

```
$ mkdir -p /path/to/output/03.count
$ geneExp=/path/to/output/03.count/{SN}.raw.gef
$ saturationSamplingFile=/path/to/output/03.count/{SN}_raw_barcode_gene_exp.txt
$ singularity exec SAW_v6.0.sif count \
        -i /path/to/output/01.mapping/{lane}.Aligned.sortedByCoord.out.bam \
        -o /path/to/output/03.count/{SN}.Aligned.sortedByCoord.out.merge.q10.dedup.
target.bam \
        -a /path/to/reference/genes.gtf \
        -s /path/to/output/03.count/{SN}.Aligned.sortedByCoord.out.merge.q10.dedup.
target.bam.summary.stat \
        -e ${geneExp} \
        --umi_len 10 \
        --sat_file ${saturationSamplingFile} \
        --sn {SN} \
        --umi_on \
        --save_lq \
        --save_dup \
        -c 24 \
        -m 128
```

* Please be noted that we use the backward slash "\" to indicate the end of a line in a command that spans multiple lines.

For more than one pair of FASTQ files (Here shows an example of 2 pairs of FASTQ):

```
$ mkdir -p /path/to/multi_lane_output/03.count  ## change /path/to/multi_lane_out-
put/ to /path/to/output/ for SE single lane scenario
$ geneExp=/path/to/multi_lane_output/03.count/{SN}.raw.gef
$ saturationSamplingFile=/path/to/multi_lane_output/03.count/{SN}_raw_barcode_gene_
exp.txt
$ singularity exec SAW_v6.0.sif count \
        -i /path/to/multi_lane_output/01.mapping/{lane1}.Aligned.sortedByCoord.out.
bam,/path/to/ multi_lane_output/01.mapping/{lane2}.Aligned.sortedByCoord.out.bam \
## change {lane} to {index} or {idx} for SE
        -o /path/to/multi_lane_output/03.count/{SN}.Aligned.sortedByCoord.out.
merge.q10.dedup.target.bam \
        -a /path/to/reference/genes.gtf \
        -s /path/to/multi_lane_output/03.count/{SN}.Aligned.sortedByCoord.out.
merge.q10.dedup.target.bam.summary.stat \
        -e ${geneExp} \
        --umi_len 10 \
        --sat_file ${saturationSamplingFile} \
        --sn {SN} \
        --umi_on \
        --save_lq \
        --save_dup \
        -c 24 \
        -m 128
```

* Please be noted that we use the backward slash "\" to indicate the end of a line in a command that spans multiple lines.

For dealing with multi-mapped reads:

```
$ singularity exec SAW_v6.0.sif count \
        -i /path/to/output/01.mapping/{lane}.Aligned.sortedByCoord.out.bam \
        -o /path/to/output/03.count/{SN}.Aligned.sortedByCoord.out.merge.q10.dedup.
target.bam \
        -a /path/to/reference/genes.gtf \
        -s /path/to/output/03.count/{SN}.Aligned.sortedByCoord.out.merge.q10.dedup.
target.
bam.summary.stat \
        -e ${geneExp} \
        --umi_len 10 \
        --sat_file ${saturationSamplingFile} \
        --sn {SN} \
        --umi_on \
        --save_lq \
        --save_dup \
        -c 24 \
        -m 128 \
        --multi_map
```

2.5.3 Ouputs

The **`count`** output files are organized as below:

```
$ tree /path/to/output/03.count
/path/to/output/03.count
├── SN.Aligned.sortedByCoord.out.merge.q10.dedup.target.bam
├── SN.Aligned.sortedByCoord.out.merge.q10.dedup.target.bam.csi
├── SN.Aligned.sortedByCoord.out.merge.q10.dedup.target.bam.summary.stat
├── SN_raw_barcode_gene_exp.txt
└── SN.raw.gef
```

When you run **`mapping`** using reference genome with rRNA information and the switch of **`rRNAremove`** is on, **`PASS FILTER`** item in**`*.bam.summary.stat`** file will not contain the number of the reads mapped to rRNA reference.

2.6 register

SAW `register` pipeline aligns the microscopic tissue staining image (multiple staining images, including DAPI image for ssDNA and IF image, immunofluorescence image, for protein) with the plot of the gene expression matrix generated by `count` based on the track lines on the chip while establishing the mapping relationship between images and spatial gene expression distribution. SAW `register` includes four main modules, stitching, tissue segmentation, cell segmentation, and registration. Stitching combines microscopic images with overlapping sections to create a panoramic image (skip stitching if the input image is already a panoramic image). Tissue and cell segmentation modules detect and mask out tissue and cell coverage region, respectively. Registration module aligns stitched image and the expression matrix, and at the same time registered masks from segmentation modules with the gene expression matrix using the same parameters. When it comes to the scenario of DAPI and mIF (multiple immunofluorescence images), this module outputs the registered DAPI and mIF image files. Tissue segmentation of IF image adopts the method of threshold segmentation, while cell segmentation of it is the intersection result of IF tissue segmentation, DAPI tissue segmentation and DAPI cell segmentation.

Image stitching and segmentation modules can be run separately with registration.

Running `register` requires the following files:

- `count` output gene expression matrix file **(.raw.gef)**
- ImageQC processed microscopic tissue staining image file **(.tar.gz)**
- ImageQC Image process record file **(.ipr)**

Expected running time for 1 cm x 1 cm Stereo-seq Chip T image: ~2 h, Memory: ~20 G

The change of output file versions as below:

- **Image process record file (.ipr) updated from v0.0.1 to v0.1.0**
- **Registered pyramid image file (.rpi) updated from v0.0.1 to v0.0.2**

Supporting the connection of manually processed images back from each module of ImageStudio;

Cell segmentation is the most time-consuming part. You may run `rapidRegister` to skip cell segmentation, but still perform stitching, tissue segmentation and registration. Please check 2.14.2 rapidRegister to get more information about `rapidRegister`.

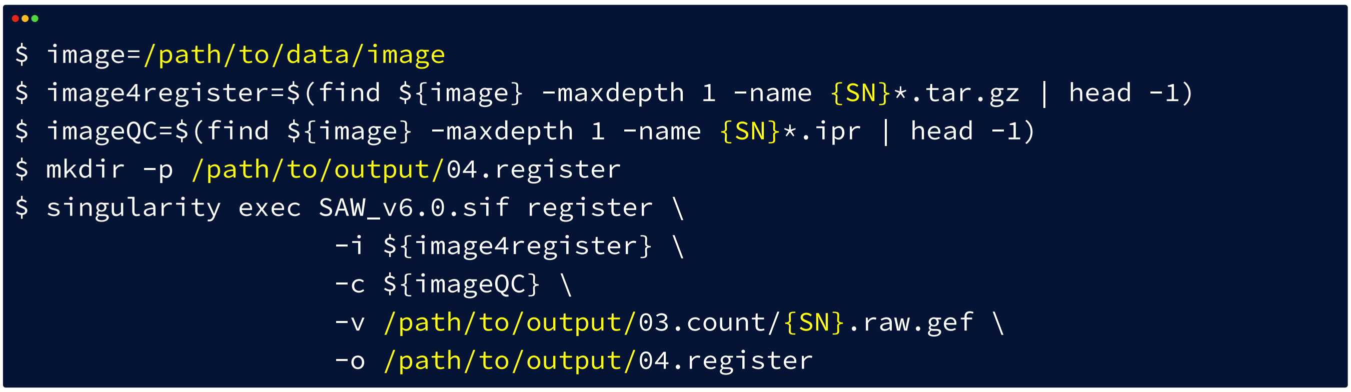
2.6.1 Arguments and Options

Table 2-7 `register` Arguments and Options

Parameter	Function
`-i`	(Required) ImageQC/ImageStudio processed staining image TAR.GZ file or image pre-processed output directory.
`-c`	(Required) ImageQC/ImageStudio IPR (image process record) file. IPR is designed to efficiently hold images and process records generated from each image processing step along with metadata in various data types. Please check **Stereo-seq File Format Manual** to get more information on the IPR file.
`-v`	(Optional. Depends on `-i`) `count` output gene expression matrix GEF file.
`-o`	(Required) Path to the directory where to store the `register` outputs.

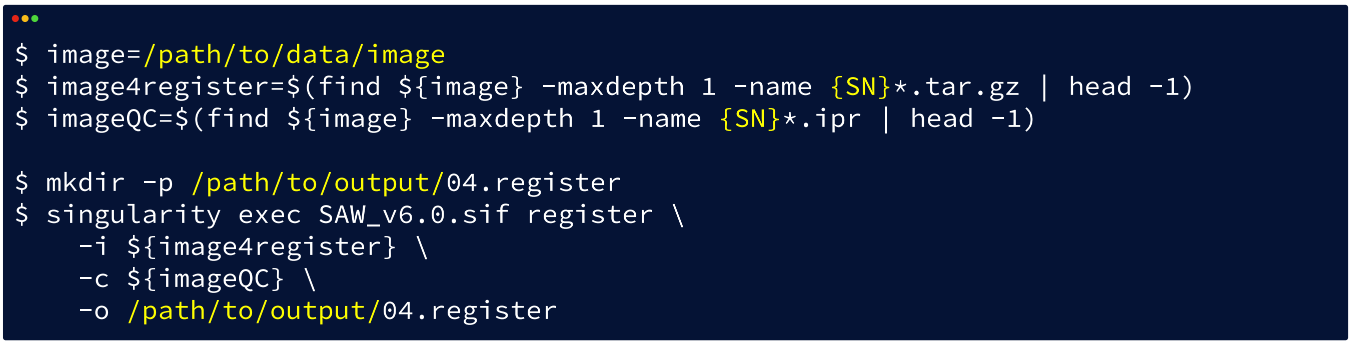
2.6.2 Usage Example

Scenario 1: Process images from ImageQC/ImageStudio output data and gene expression matrix. Perform stitching, tissue segmentation, cell segmentation, and registration with gene expression matrix.



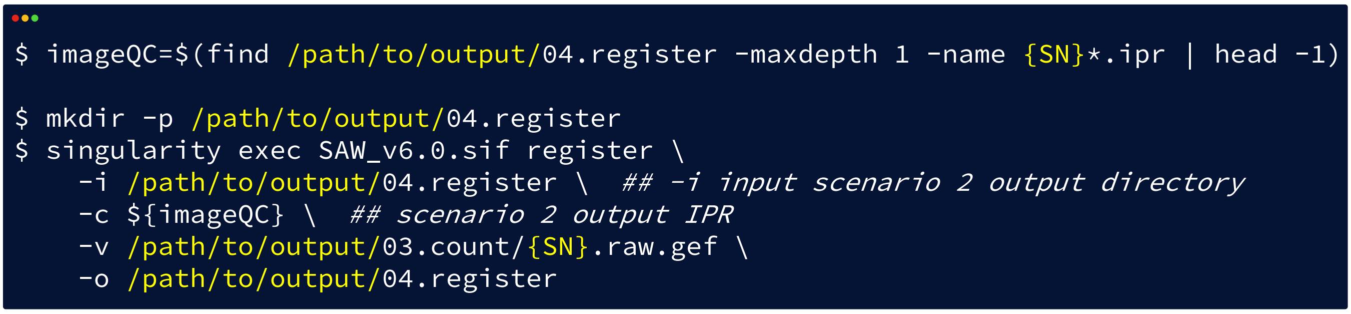
```
$ image=/path/to/data/image
$ image4register=$(find ${image} -maxdepth 1 -name {SN}*.tar.gz | head -1)
$ imageQC=$(find ${image} -maxdepth 1 -name {SN}*.ipr | head -1)
$ mkdir -p /path/to/output/04.register
$ singularity exec SAW_v6.0.sif register \
                    -i ${image4register} \
                    -c ${imageQC} \
                    -v /path/to/output/03.count/{SN}.raw.gef \
                    -o /path/to/output/04.register
```

Scenario 2: Process images from ImageQC/ImageStudio output data. Perform stitching, tissue segmentation, and cell segmentation without doing registration with the gene expression matrix.



```
$ image=/path/to/data/image
$ image4register=$(find ${image} -maxdepth 1 -name {SN}*.tar.gz | head -1)
$ imageQC=$(find ${image} -maxdepth 1 -name {SN}*.ipr | head -1)

$ mkdir -p /path/to/output/04.register
$ singularity exec SAW_v6.0.sif register \
    -i ${image4register} \
    -c ${imageQC} \
    -o /path/to/output/04.register
```

Scenario 3: Process images from scenario 2. `register` processed images with gene expression matrix.



```
$ imageQC=$(find /path/to/output/04.register -maxdepth 1 -name {SN}*.ipr | head -1)

$ mkdir -p /path/to/output/04.register
$ singularity exec SAW_v6.0.sif register \
    -i /path/to/output/04.register \  ## -i input scenario 2 output directory
    -c ${imageQC} \  ## scenario 2 output IPR
    -v /path/to/output/03.count/{SN}.raw.gef \
    -o /path/to/output/04.register
```

2.6.3 Outputs

`register`output files of scenario 1 and scenario 3 (if `-i` and `-o` are identical) are organized as below:

```
$ tree /path/to/output/04.register
/path/to/output/04.register
...   ## skip setting files, logs and image folder
├── <stainType>_fov_stitched_transformed.tif
├── SN_<chipType>_date_time_version.ipr
└── SN_tissue_bbox.csv
```

`register` output files of scenario 2:

```
$ tree /path/to/output/04.register
/path/to/output/04.register
...   ## skip setting files, logs and image folder
├── <stainType>_fov_stitched_transformed.tif
├── SN_<chipType>_date_time_version.ipr
```

`register` output files of scenario 3 if `-i` and `-o` are two different paths:

```
$ tree /path/to/output/04.register
/path/to/output/04.register
...  ## skip logs
├── SN_<chipType>_date_time_version.ipr
└── SN_tissue_bbox.csv
```

2.7 imageTools

SAW `imageTools` is a handy toolkit designed to play with image data in SAW. `imageTools ipr2img` (or `ipr2img`, available from SAW ST >= v5.0.0) is required in SAW core pipeline "decode" images from the processed IPR file. SAW `imageTools ipr2img` outputs TIFF images from IPR such as pre-registered ssDNA stitched image, tissue segmentation and cell segmentation mask TIFFs, as well as registered three types of images.

Running `imageTools ipr2img` requires the following files:

- ImageQC/ImageStudio processed microscopic tissue staining image file **(.tar.gz)**
- `register` processed Image process record file **(.ipr, compatible with v0.0.1 and v0.1.0)**

🕓 Expected running time for 1 cm x 1 cm Stereo-seq Chip T image: ~5 min, Memory: ~10 G

The change of output file versions as below:

- **Image process record file (.ipr) updated from v0.0.1 to v0.1.0**
- **Registered pyramid image file (.rpi) updated from v0.0.1 to v0.0.2**

2.7.1 Arguments and Options

Table 2-8 imageTools ipr2img Arguments and Options

Command	Parameter	Function
ipr2img	-i	(Required) ImageQC/ImageStudio processed staining image TAR.GZ file.
	-c	(Required) IPR (image process record) file. IPR is designed to efficiently hold image information generated from each processing step and datasets in various compound types along with metadata. Please check **Stereo-seq File Format Manual** to get more information on IPR file.
	-d	(Required) Segmentation module names. Convert segmentation mask TIFF from IPR for the selected modules. Valid options: `tissue` and `cell`. Separate modules by space.
	-r	(Required; default to True) Whether output registered images or pre-registered images. "Pre-registered" state stands for the images that have been stitched to a single panoramic image and transformed to have the same scale with the gene expression matrix, but still need to be flipped, 90°rotated, or translated. The options are `True` for output registered images and `False` for output pre-registered images.
	-o	(Required) Path to the directory where to store the `ipr2img` outputs.

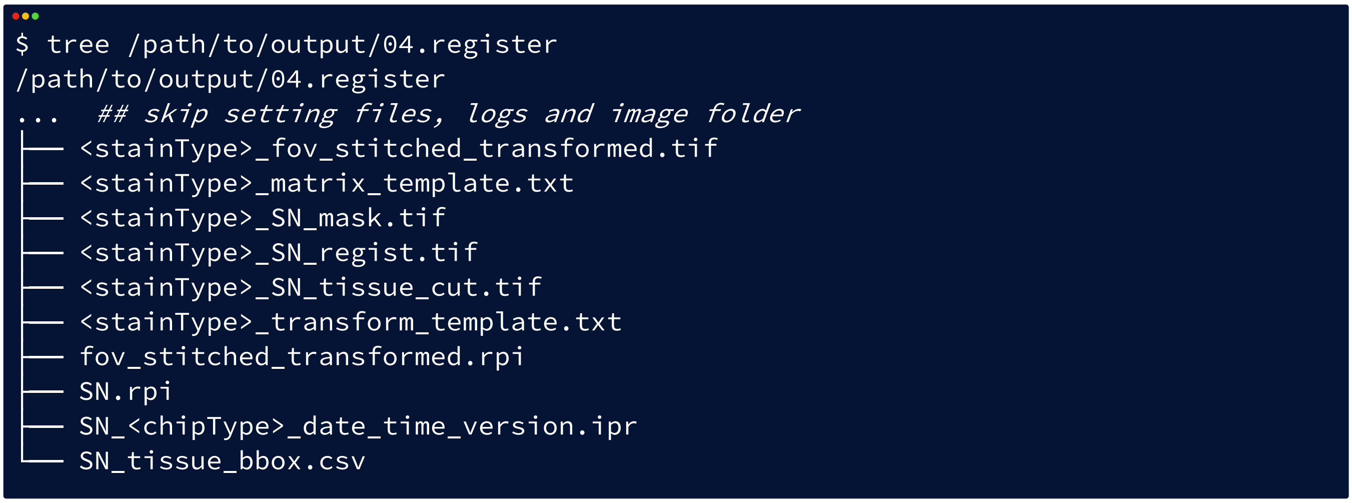
Please check **2.14.4 Other applications of imageTools** to learn more about **imageTools**.

2.7.2 Usage Example

```
$ image=/path/to/data/image
$ image4register=$(find ${image} -maxdepth 1 -name {SN}*.tar.gz | head -1)
$ imageIPR=$(find /path/to/output/04.register -maxdepth 1 -name {SN}*.ipr | head -1)
## has to be a processed IPR

$ singularity exec SAW_v6.0.sif imageTools ipr2img \
      -i ${image4register} \
      -c ${imageIPR} \
      -d tissue cell \  ## output both tissue and cell segmentation mask TIFF
      -r True \
      -o /path/to/output/04.register
```

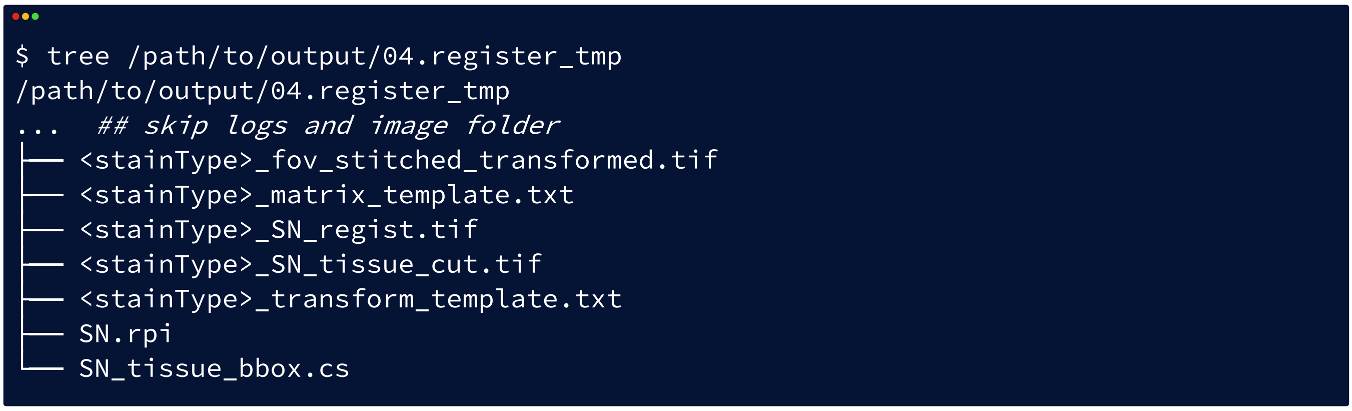
2.7.3 Outputs

`imageTools ipr2img` output files (if the parent directory path of `-c` IPR and `-o` are identical) are organized as below:



```
$ tree /path/to/output/04.register
/path/to/output/04.register
...  ## skip setting files, logs and image folder
├── <stainType>_fov_stitched_transformed.tif
├── <stainType>_matrix_template.txt
├── <stainType>_SN_mask.tif
├── <stainType>_SN_regist.tif
├── <stainType>_SN_tissue_cut.tif
├── <stainType>_transform_template.txt
├── fov_stitched_transformed.rpi
├── SN.rpi
├── SN_<chipType>_date_time_version.ipr
└── SN_tissue_bbox.csv
```

`imageTools ipr2img` output files (if the parent directory path of `-c` IPR and `-o` are two different paths) are organized as below:



```
$ tree /path/to/output/04.register_tmp
/path/to/output/04.register_tmp
...  ## skip logs and image folder
├── <stainType>_fov_stitched_transformed.tif
├── <stainType>_matrix_template.txt
├── <stainType>_SN_regist.tif
├── <stainType>_SN_tissue_cut.tif
├── <stainType>_transform_template.txt
├── SN.rpi
└── SN_tissue_bbox.cs
```

2.8 tissueCut

SAW **`tissueCut`** pipeline can delineate and extract the tissue coverage area based on the aligned image generated from **`register`** and **`imageTools`** or from the plot of gene expression matrix (if microscopic tissue staining images are not available). **`tissueCut`** outputs expression data in GEF format.

If the output of `tissueCut` doesn't match the morphology of the tissue, users could handle this issue with the help of ImageStudio, StereoMap or Stereopy[11]. If the `tissueCut` was run with an image, users could manually redo tissue segmentation for the image via ImageStudio and run `register`, `imageTools` and `tissueCut` again. Otherwise, manually delineate tissue coverage region from the gene expression distribution plot via StereoMap and then put the result into the SAW-lasso pipeline (2.14.6 lasso), or do lasso selection and expression matrix extraction via Stereopy (Stereopy->Examples->Interactive).

Running **`tissueCut`** requires the following files:

- Mapped CID list file **(.txt)**
- **`count`** output gene expression matrix file **(.raw.gef)**
- `imageTools ipr2img` output tissue segmentation mask TIFF file **(.tif optional)**

Expected running time for ~1G reads: ~6 min, Memory: ~6 G

2.8.1 Arguments and Options

Table 2-9 `tissueCut` Arguments and Options

Parameter	Function
`--dnbfile`	(Optional) Mapped CID list file with reads counts for each CID. Input merged mapped CID list file if more than one list file was generated in `mapping`.
`-i`	(Required) `count` output gene expression matrix file.
`-o`	(Required) Path to the directory where to store the `tissueCut` outputs.
`-s`	(Optional) Path to the `imageTools` `ipr2img` output tissue segmentation mask file. Only valid when `register` has been performed.
`--sn`	(Required) Stereo-seq Chip T serial number (SN).
`--omics,-O`	(Required; default to Transcriptomics) String that specifies the omics.
`-d`	(Required) Set to generate required plots for `report`.
`-t`	(Optional) Number of threads used in numpy (recommended to fill 8 here).

2.8.2 Usage Example

Run `tissueCut` if `register` aligned microscopic staining image is provided:

```
$ nucleusLayer=$(find /path/to/output/04.register -maxdepth 1 -name *fov_stitched_
transformed.tif -exec sh -c 'for f do basename -- "$f" _fov_stitched_transformed.
tif;done' sh {} + | grep -v IF)
$ tissueMaskFile=$(find /path/to/output/04.register -maxdepth 1 -name ${nucle-
usLayer}*_tissue_cut.tif

$ mkdir -p /path/to/output/05.tissuecut
$ singularity exec SAW_v6.0.sif tissueCut \
    --dnbfile /path/to/output/02.merge/{SN}.merge.barcodeReadsCount.txt \
    -i /path/to/output/03.count/{SN}.raw.gef \
    -o /path/to/output/05.tissuecut \
    -s ${tissueMaskFile} \
    --sn {SN} -O Transcriptomics -d
```

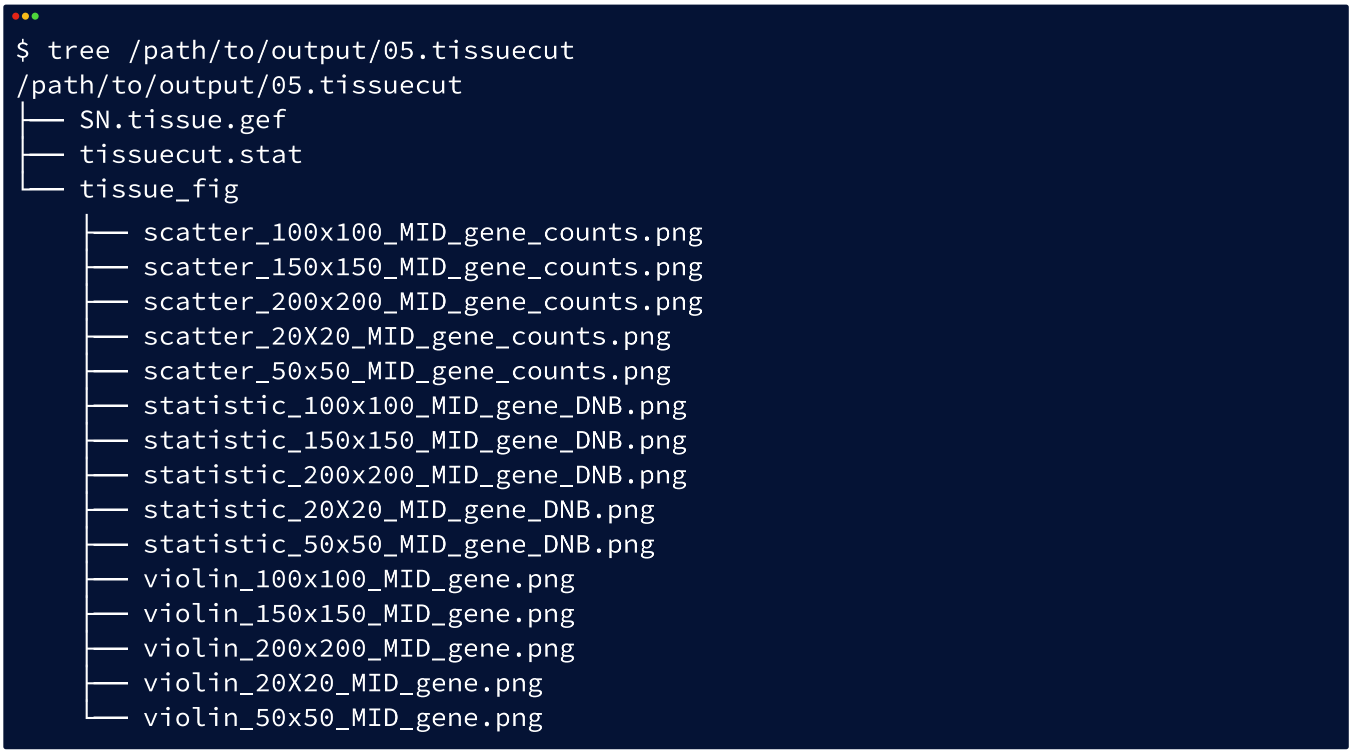
Run `tissueCut` if image is not available:

```
$ mkdir -p /path/to/output/05.tissuecut
$ singularity exec SAW_v6.0.sif tissueCut \
    --dnbfile /path/to/output/02.merge/{SN}.merge.barcodeReadsCount.txt \
    -i /path/to/output/03.count/{SN}.raw.gef \
    -o /path/to/output/05.tissuecut \
    --sn {SN} -O Transcriptomics -d
```

2.8.3 Outputs

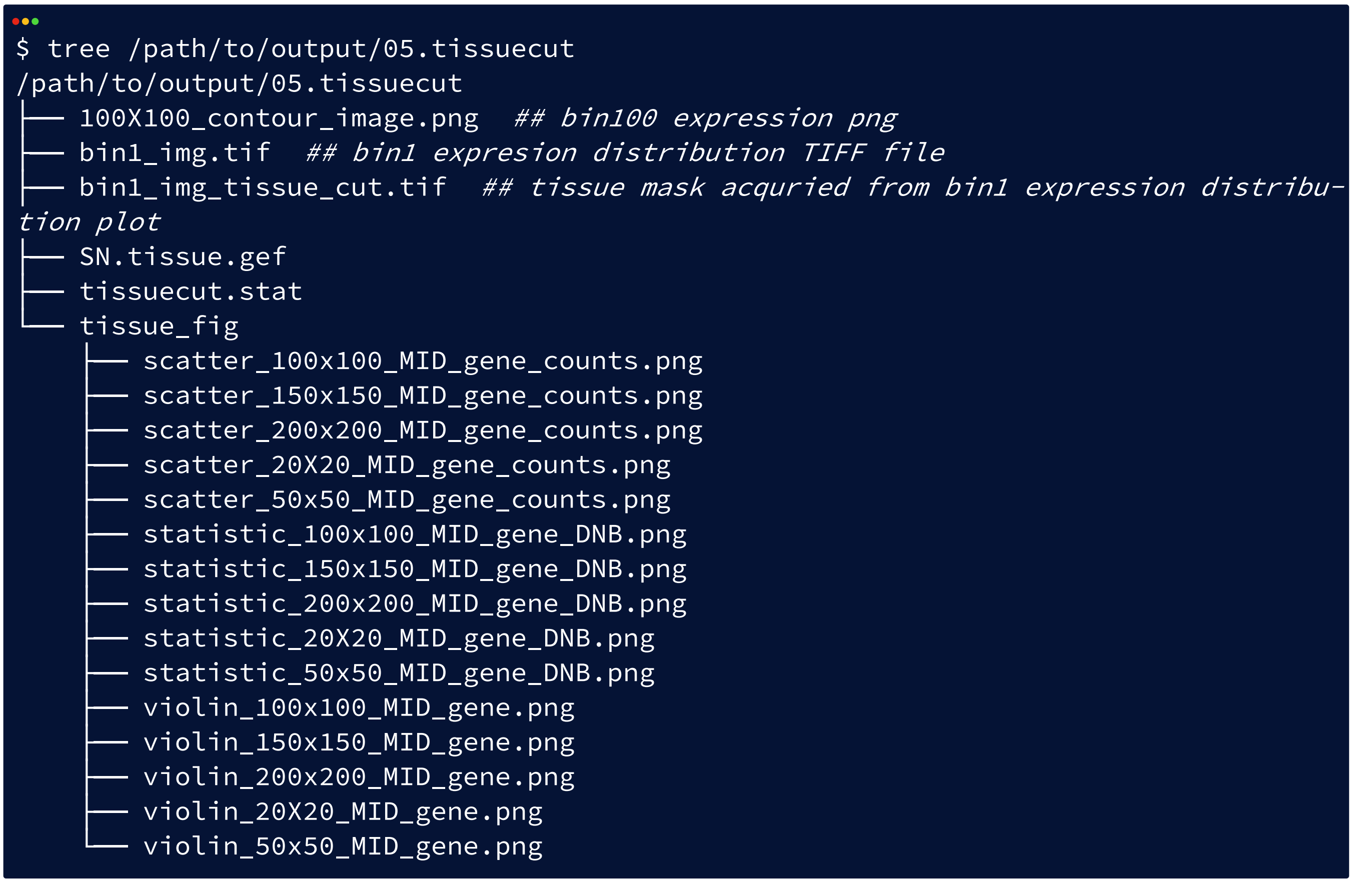
`tissueCut` output files:

Image is provided:



```
$ tree /path/to/output/05.tissuecut
/path/to/output/05.tissuecut
├── SN.tissue.gef
├── tissuecut.stat
└── tissue_fig
    ├── scatter_100x100_MID_gene_counts.png
    ├── scatter_150x150_MID_gene_counts.png
    ├── scatter_200x200_MID_gene_counts.png
    ├── scatter_20X20_MID_gene_counts.png
    ├── scatter_50x50_MID_gene_counts.png
    ├── statistic_100x100_MID_gene_DNB.png
    ├── statistic_150x150_MID_gene_DNB.png
    ├── statistic_200x200_MID_gene_DNB.png
    ├── statistic_20X20_MID_gene_DNB.png
    ├── statistic_50x50_MID_gene_DNB.png
    ├── violin_100x100_MID_gene.png
    ├── violin_150x150_MID_gene.png
    ├── violin_200x200_MID_gene.png
    ├── violin_20X20_MID_gene.png
    └── violin_50x50_MID_gene.png
```

Image is not provided:



```
$ tree /path/to/output/05.tissuecut
/path/to/output/05.tissuecut
├── 100X100_contour_image.png  ## bin100 expression png
├── bin1_img.tif  ## bin1 expresion distribution TIFF file
├── bin1_img_tissue_cut.tif  ## tissue mask acquried from bin1 expression distribu-
tion plot
├── SN.tissue.gef
├── tissuecut.stat
└── tissue_fig
    ├── scatter_100x100_MID_gene_counts.png
    ├── scatter_150x150_MID_gene_counts.png
    ├── scatter_200x200_MID_gene_counts.png
    ├── scatter_20X20_MID_gene_counts.png
    ├── scatter_50x50_MID_gene_counts.png
    ├── statistic_100x100_MID_gene_DNB.png
    ├── statistic_150x150_MID_gene_DNB.png
    ├── statistic_200x200_MID_gene_DNB.png
    ├── statistic_20X20_MID_gene_DNB.png
    ├── statistic_50x50_MID_gene_DNB.png
    ├── violin_100x100_MID_gene.png
    ├── violin_150x150_MID_gene.png
    ├── violin_200x200_MID_gene.png
    ├── violin_20X20_MID_gene.png
    └── violin_50x50_MID_gene.png
```

2.9 spatialCluster

SAW **`spatialCluster`** pipeline performs clustering analysis for spots using Stereopy. The clustering procedure includes 4 main steps: 1) preprocess gene expression data from the tissue-coverage region (normalize, logarithmize, identify highly-variable genes, and scale each gene), 2) reduce the dimensionality of the data by running PCA, 3) compute the neighborhood graph and embed neighborhood graph using UMAP, 4) clustering by Leiden algorithm.

Running **`spatialCluster`** requires the following files:

- **`tissueCut`** output GEF file for the tissue-covered region **(.tissue.gef)**

🕓 Expected running time for ~1G reads: ~1 min, Memory: ~5 G

2.9.1 Arguments and Options

Table 2-10 `spatialCluster` Arguments and Options

Parameter	Function
`-i`	(Required) **`tissueCut`** output GEF file for the tissue coverage area.
`-o`	(Required) Output path for the clustering result in H5AD format.
`-s`	(Required; default to 50) Bin size.

2.9.2 Usage Example

```
$ mkdir -p /path/to/output/06.spatialcluster
$ singularity exec SAW_v6.0.sif spatialCluster \
    -i /path/to/output/05.tissuecut/{SN}.tissue.gef \
    -o /path/to/output/06.spatialcluster/{SN}.spatial.cluster.h5ad \
    -s 200
```

2.9.3 Outputs

`spatialCluster` output files are:

```
$ tree /path/to/output/06.spatialcluster
/path/to/output/06.spatialcluster
└── SN.spatial.cluster.h5ad
```

2.10 cellCut

SAW `cellCut` pipeline runs to extract expression matrix of cell nucleus based on the aligned image generated from `register` and `imageTools`. `cellCut` outputs expression data in cell bin GEF format. Run `cellCut` if cell bin results are desired.

Running `cellCut` requires the following files:

- `count` output gene expression matrix file **(.raw.gef)**
- `register` and `imageTools` output cell segmentation mask TIFF file **(.tif)**

🕓 Expected running time for ~1G reads: ~2 min, Memory: ~10 G

2.10.1 Arguments and Options

Table 2-11 `cellCut` Arguments and Options

Commands	Parameters	Function
cgef	-i, --input-file	(Required) Input GEF file.
	-m, --mask-file	(Required) Input cell segmentation mask file.
	-o, --output-file	(Required) Output cell bin GEF file.
	-b, --block	(Optional; default to 256,256) Block size.
	-r, --rand-celltype	(Optional; default to 0) Number of random cell type.
	-t, --threads	(Optional; default to 1) Number of threads.
	-v, --verbose	(Optional) Verbose output.
	-n, --cnum	(Optional; default to 5000) Top level cell number.
	-R, --ratio	(Optional; default to 20) Other level cell number ratio.
	-a, --allocat	(Optional; default to 2) Allocation strategy.
	-g, --raw-gem	(Optional) Raw GEM file.
	-c, --canvas	(Optional; default to 0,0,90000,90000) Set canvas size, minX,minY,-maxX maxY.
	-l, --limit	(Optional; default to 16,16) Set block limit.
	-S, --split	(Optional; default to 0) Split cellID to layers and blocks.
	-w, --errorCode-file	(Optional; default to false) Set to turn on error code mode.

Please check 2.14.1 Other applications of cellCut to learn more about `cellCut`.

2.10.2 Usage Example

Run **`cellCut`** only if **`register`** aligned microscopic staining image is provided:

```
$ mkdir -p /path/to/output/051.cellcut
$ nucleusLayer=$(find /path/to/output/04.register -maxdepth 1 -name *fov_stitched_
transformed.tif -exec sh -c 'for f do basename -- "$f" _fov_stitched_transformed.
tif;done' sh {} + | grep -v IF)
$ nucleusMask=$(find /path/to/output/04.register -maxdepth 1 -name ${nucleusLayer}*_
mask.tif)

$ singularity exec SAW_v6.0.sif cellCut cgef \
      -i /path/to/output/03.count/{SN}.raw.gef \
      -m ${nucleusMask} \
      -o /path/to/output/051.cellcut/{SN}.cellbin.gef
```

2.10.3 Outputs

`cellCut` output files:

```
$ tree /path/to/output/051.cellcut
/path/to/output/051.cellcut
└── SN.cellbin.gef
```

2.11 cellCluster

SAW **`cellCluster`** pipeline runs by clustering cells using the Leiden algorithm similarly to the procedures of **`spatialCluster`**. Run **`cellCluster`** if cell bin results are desired.

Running **`cellCluster`** requires the following files:

- **`cellCut`** output cell bin gene expression matrix file **(.cellbin.gef)**

🕓 Expected running time for ~1G reads: ~5 min, Memory: ~5 G

2.11.1 Arguments and Options

Table 2-12 `cellCluster` Arguments and Options

Parameter	Function
-i	(Required) **`cellCut`** output cell bin gene expression matrix file.
-o	(Required) Output path to the clustering result in H5AD format.

2.11.2 Usage Example

```
$ mkdir -p /path/to/output/061.cellcluster
$ singularity exec SAW_v6.0.sif cellCluster \
        -i /path/to/output/051.cellcut/{SN}.cellbin.gef \
        -o /path/to/output/061.cellcluster/{SN}.cell.cluster.h5ad
```

2.11.3 Outputs

`cellCluster` output files:

```
$ tree /path/to/output/061.cellcluster
/path/to/output/061.cellcluster
└── SN.cell.cluster.h5ad
```

2.12 saturation

SAW **saturation** pipeline is performed to compute the sequencing saturation for the tissue coverage area.

Running **saturation** requires the following files:

- **mapping** output statistical report of CID mapping **(.stat)**
- **count** output saturation sampling file **(.txt)**
- **count** output statistical report of annotation **(.stat)**
- **tissueCut** output GEF file for the tissue coverage area **(.tissue.gef)**

🕓 Expected running time for ~1G reads: ~5 min, Memory: ~5 G

2.12.1 Arguments and Options

Table 2-13 `saturation` Arguments and Options

Parameter	Function
`-i`	(Required) **count** output saturation sampling file.
`--tissue`	(Required) **tissueCut** output GEF file for the tissue coverage area.
`-o`	(Required) Path to the directory where to store the **saturation** outputs. There has to be an existing path.
`--bcstat`	(Required) **mapping** output statistical report of CID mapping. Separate multiple files by comma.
`--summary`	(Required) **count** output statistical report of annotation.

2.12.2 Usage Example

```
$ mkdir -p /path/to/output/07.saturation
$ singularity exec SAW_v6.0.sif saturation \
      -i /path/to/output/03.count/{SN}_raw_barcode_gene_exp.txt \
      --tissue /path/to/output/05.tissuecut/{SN}.tissue.gef \
      -o /path/to/output/07.saturation \
      --bcstat /path/to/output/01.mapping/{lane}_barcodeMap.stat \
      --summary  /path/to/output/03.count/{SN}.Aligned.sortedByCoord.out.merge.q10.
dedup.target.bam.summary.stat
```

* Please be noted that these two lines belong to the same line of command.

For more than one pair of FASTQ files (Here showing an example of 2 pairs of FASTQ),

```
$ mkdir -p /path/to/multi_lane_output/07.saturation
$ singularity exec SAW_v6.0.sif saturation \
      -i /path/to/multi_lane_output/03.count/{SN}_raw_barcode_gene_exp.txt \
      --tissue /path/to/multi_lane_output/05.tissuecut/{SN}.tissue.gef \
      -o /path/to/multi_lane_output/07.saturation \
      --bcstat /path/to/multi_lane_output/01.mapping/{lane1}_barcodeMap.stat,/path/
to/multi_lane_output/01.mapping/{lane2}_barcodeMap.stat \

      --summary  /path/to/multi_lane_output/03.count/{SN}.Aligned.sortedByCoord.out.
merge.q10.dedup.target.bam.summary.stat
```

* Please be noted that we use the backward slash “\” to indicate the end of a line in a command that spans multiple lines.

2.12.3 Outputs

`saturation` output files:

```
$ tree /path/to/output/07.saturation
├── plot_1x1_saturation.png
├── plot_200x200_saturation.png
└── sequence_saturation.tsv
```

2.13 report

SAW `report` pipeline is performed to integrate analysis results from each step and generate the report in JSON format as well as a web report in HTML format. HTML analytical report integrate genes' spatial expression distribution, key statistical metrices, sequencing saturation plots, clustering analysis results, and image processing information. When it comes to the scenario of mIF, pseudo color (up to 7) is used on the bottom layer of clustering results.

Running `report` requires the following files and information:

- `mapping` output statistical report of CID mapping and STAR alignment **(.stat, .out)**
- `count` output statistical report of annotation **(.stat)**
- `register` processed image process record file and image pyramid RPI file **(.ipr, .rpi)**
- `tissueCut` and `cellCut` (if available) output GEF file, statistical report of tissue-covered region, plots and image pyramid RPI file **(.gef, .stat, .png)**
- `spatialCluster` and `cellCluster` (if available) output clustering H5AD file **(.h5ad)**
- `saturation` output bin200 sequence saturation plot **(.png)**
- species, tissue, and reference information

Expected running time for ~1G reads: ~1 min, Memory: 1G

2.13.1 Arguments and Options

Table 2-14 `report` Arguments and Options

Parameter	Function
-m	(Required) Statistical report of CID mapping. Separate multiple files by comma.
-a	(Required) Statistical report of STAR alignment. Separate multiple files by comma.
-g	(Required) Statistical report of annotation.
-l	(Required) Statistical report of tissue-covered region.
-n	(Required) GEF file that has wholeExp/bin200. Please check **2.14.1 Other applications of cellCut** to get more information of GEF completion.
-b	(Required) `tissueCut` output bin 200 scatter plot.
-v	(Required) `tissueCut` output bin 200 violin plot.
-c	(Required) `tissueCut` output bin 200 statistics plot.
--bin20Saturation	(Required) `tissueCut` output bin 20 scatter plot.
--bin20violin	(Required) `tissueCut` output bin 20 violin plot.
--bin20MIDGeneDNB	(Required) `tissueCut` output bin 20 statistics plot.
--bin50Saturation	(Required) `tissueCut` output bin 50 scatter plot.
--bin50violin	(Required) `tissueCut` output bin 50 violin plot.
--bin50MIDGeneDNB	(Required) `tissueCut` output bin 50 statistics plot.
--bin100Saturation	(Required) `tissueCut` output bin 100 scatter plot.
--bin100violin	(Required) `tissueCut` output bin 100 violin plot.
--bin100MIDGeneDNB	(Required) `tissueCut` output bin 100 statistics plot.
--bin150Saturation	(Required) `tissueCut` output bin 150 scatter plot.

Parameter	Function
`--bin150violin`	(Required) `tissueCut` output bin 150 violin plot.
`--bin150MIDGeneDNB`	(Required) `tissueCut` output bin 150 statistics plot.
`-d`	(Required) `spatialCluster` output H5AD file.
`-t`	(Required) `saturation` output bin 200 sequence saturation plot.
`-r standard_version`	(Required) Set to specifying report version.
`-s`	(Required) The Stereo-seq Chip T serial number.
`--pipelineVersion`	(Required) Set to specifying analysis pipeline version.
`-o`	(Required) The directory to store outputs.
`--species`	(Required) A string of species name.
`--tissue`	(Required) A string of tissue type.
`-reference`	(Required) A string of reference used for `mapping`
`--rpi_resolution`	(Optional; default to 100) The resolution of RPI. Valid options: **2, 10, 50, 100, 150**.
`-i`	(Optional) The image pyramid RPI file.
`--iprFile`	(Optional) A processed IPR (image process record) file.
`--cellBinGef`	(Optional) The cell bin GEF file.
`--cellCluster`	(Optional) `cellCluster` output H5AD file.

2.13.2 Usage Example

Run `report` if `register` aligned microscopic staining image is provided and has cell bin output:

Please Note! Replace `{SN}`, `{lane}`,`{species_name}`, `{tissue_type}`, `{reference_index}` with the real information.

```
$ imageIPR=$(find /path/to/output/04.register -maxdepth 1 -name {SN}*.ipr | head
-1)  ## has to be a processed IPR
$ singularity exec SAW_v6.0.sif cellCut bgef \
    -i /path/to/output/03.count/{SN}.raw.gef \
    -o /path/to/output/05.tissuecut/{SN}.gef \
    -b 1,10,20,50,100,200,500 \
    -O Transcriptomics

$ mkdir -p /path/to/output/08.report
$ singularity exec SAW_v6.0.sif report \
      -m /path/to/output/01.mapping/{lane}_barcodeMap.stat \
       -a /path/to/output/01.mapping/{lane}.Log.final.out \
      -g /path/to/output/03.count/{SN}.Aligned.sortedByCoord.out.merge.q10.dedup.
target.bam.summary.stat \
      -l /path/to/output/05.tissuecut/tissuecut.stat \
      -n /path/to/output/05.tissuecut/{SN}.gef \
      -d /path/to/output/06.spatialcluster/{SN}.spatial.cluster.h5ad \
      -t /path/to/output/07.saturation/plot_200x200_saturation.png \
      -b /path/to/output/05.tissuecut/tissue_fig/scatter_200x200_MID_gene_counts.
png \
```

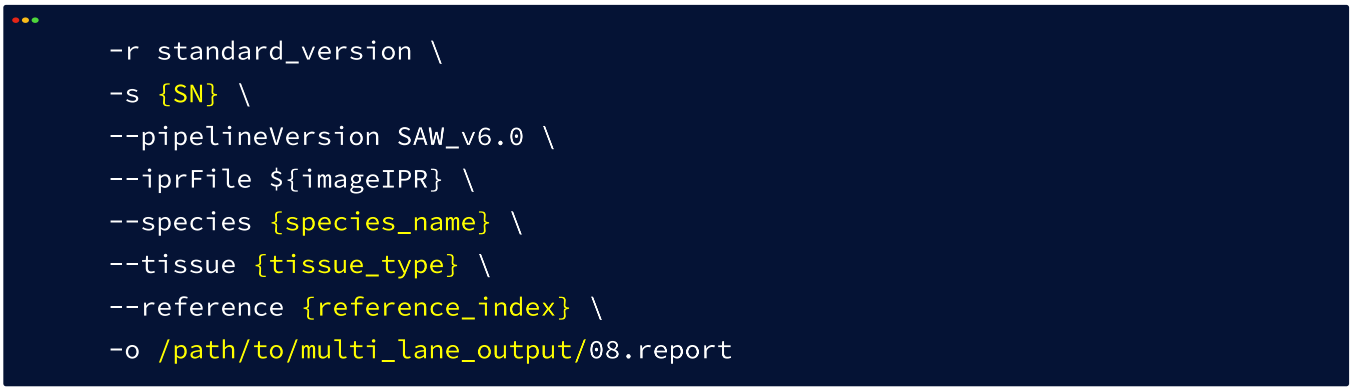
* Please be noted that we use the backward slash “\” to indicate the end of a line in a command that spans multiple lines.

```
      -v /path/to/output/05.tissuecut/tissue_fig/violin_200x200_MID_gene.png \
      -c /path/to/output/05.tissuecut/tissue_fig/statistic_200x200_MID_gene_DNB.png \
      --bin20Saturation,--bin20violin,--bin20MIDGeneDNB
      --bin50Saturation /path/to/output/05.tissuecut/tissue_fig/scatter_50x50_MID_gene_counts.png \
      --bin50violin /path/to/output/05.tissuecut/tissue_fig/violin_50x50_MID_gene.png \
      --bin50MIDGeneDNB /path/to/output/05.tissuecut/tissue_fig/statistic_50x50_MID_gene_DNB.png \
      --bin100Saturation /path/to/output/05.tissuecut/tissue_fig/scatter_100x100_MID_gene_counts.png \
      --bin100violin /path/to/output/05.tissuecut/tissue_fig/violin_100x100_MID_gene.png \
      --bin100MIDGeneDNB /path/to/output/05.tissuecut/tissue_fig/statistic_100x100_MID_gene_DNB.png \
      --bin150Saturation /path/to/output/05.tissuecut/tissue_fig/scatter_150x150_MID_gene_counts.png \
      --bin150violin /path/to/output/05.tissuecut/tissue_fig/violin_150x150_MID_gene.png \
      --bin150MIDGeneDNB /path/to/output/05.tissuecut/tissue_fig/statistic_150x150_MID_gene_DNB.png \
      --cellBinGef /path/to/output/051.cellcut/{SN}.cellbin.gef \
      --cellCluster /path/to/output/061.cellcluster/{SN}.cell.cluster.h5ad \
      -i /path/to/output/04.register/{SN}.rpi \
      -r standard_version \
      -s {SN} \
      --pipelineVersion SAW_v6.0 \
      --iprFile ${imageIPR} \
      --species {species_name} \
      --tissue {tissue_type} \
      --reference {reference_index} \
      -o /path/to/output/08.report
```

* Please be noted that we use the backward slash “\” to indicate the end of a line in a command that spans multiple lines.

For more than one pair of FASTQ files (Here showing an example of 2 pairs of FASTQ),

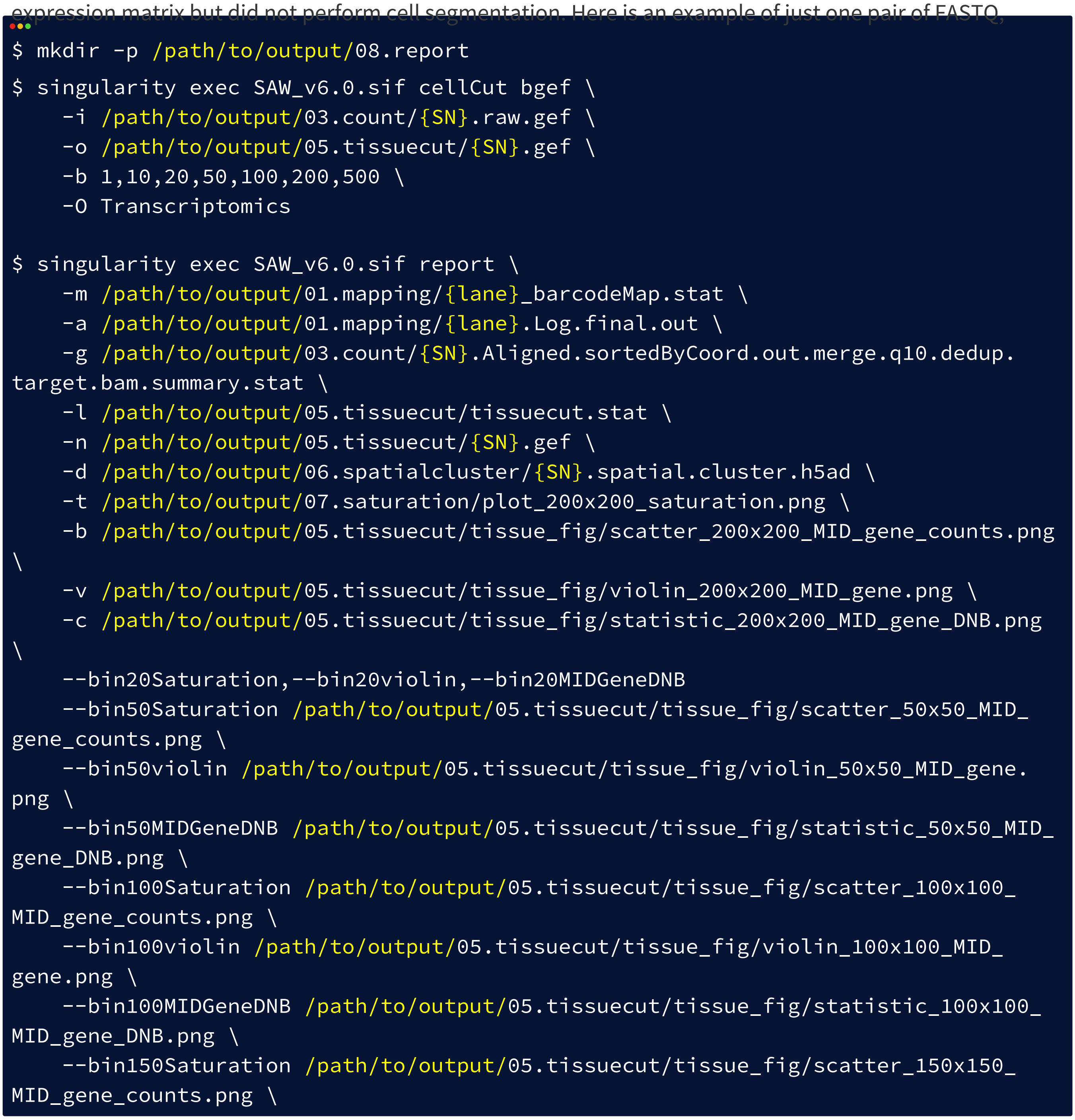
```
$ mkdir -p /path/to/multi_lane_output/08.report$
singularity exec SAW_v6.0.sif cellCut bgef \
    -i /path/to/output/03.count/{SN}.raw.gef \
    -o /path/to/output/05.tissuecut/{SN}.gef \
    -b 1,10,20,50,100,200,500 \
    -O Transcriptomics
$ singularity exec SAW_v6.0.sif report \
      -m /path/to/multi_lane_output/01.mapping/{lane1}_barcodeMap.stat,/path/to/
multi_lane_output/01.mapping/{lane2}_barcodeMap.statt \
      -a /path/to/multi_lane_output/01.mapping/{lane1}.Log.final.out,/path/to/
multi_lane_output/01.mapping/{lane2}.Log.final.out \
      -g /path/to/multi_lane_output/03.count/{SN}.Aligned.sortedByCoord.out.merge.
q10.dedup.target.bam.summary.stat \
      -l /path/to/multi_lane_output/05.tissuecut/tissuecut.stat \
      -n /path/to/multi_lane_output/05.tissuecut/{SN}.gef \
      -d /path/to/multi_lane_output/06.spatialcluster/{SN}.spatial.cluster.h5ad \
      -t /path/to/multi_lane_output/07.saturation/plot_200x200_saturation.png \
      -b /path/to/multi_lane_output/05.tissuecut/tissue_fig/scatter_200x200_MID_
gene_counts.png \
      -v /path/to/multi_lane_output/05.tissuecut/tissue_fig/violin_200x200_MID_
gene.png \
      -c /path/to/multi_lane_output/05.tissuecut/tissue_fig/statistic_200x200_MID_
gene_DNB.png \
      --bin20Saturation,--bin20violin,--bin20MIDGeneDNB
      --bin50Saturation /path/to/mult_lane_output/05.tissuecut/tissue_fig/scat-
ter_50x50_MID_gene_counts.png \
      --bin50violin /path/to/multi_lane_output/05.tissuecut/tissue_fig/vio-
lin_50x50_MID_gene.png \
      --bin50MIDGeneDNB /path/to/multi_lane_output/05.tissuecut/tissue_
fig/statistic_50x50_MID_gene_DNB.png \
      --bin100Saturation /path/to/multi_lane_output/05.tissuecut/tissue_
fig/scatter_100x100_MID_gene_counts.png \
      --bin100violin /path/to/multi_lane_output/05.tissuecut/tissue_
fig/violin_100x100_MID_gene.png \
      --bin100MIDGeneDNB /path/to/multi_lane_output/05.tissuecut/tissue_
fig/statistic_100x100_MID_gene_DNB.png \
      --bin150Saturation /path/to/multi_lane_output/05.tissuecut/tissue_
fig/scatter_150x150_MID_gene_counts.png \
      --bin150violin /path/to/multi_lane_output/05.tissuecut/tissue_
fig/violin_150x150_MID_gene.png \
      --bin150MIDGeneDNB /path/to/multi_lane_output/05.tissuecut/tissue_
fig/statistic_150x150_MID_gene_DNB.png \
      --cellBinGef /path/to/output/051.cellcut/{SN}.cellbin.gef \
      --cellCluster /path/to/output/061.cellcluster/{SN}.cell.cluster.h5ad \
      -i /path/to/multi_lane_output/04.register/{SN}.rpi \
```



```
      -r standard_version \
      -s {SN} \
      --pipelineVersion SAW_v6.0 \
      --iprFile ${imageIPR} \
      --species {species_name} \
      --tissue {tissue_type} \
      --reference {reference_index} \
      -o /path/to/multi_lane_output/08.report
```

* Please be noted that we use the backward slash "\" to indicate the end of a line in a command that spans multiple lines.

Run **`report`** of **`rapidRegister`** outputs in which the microscopic image has been registered with gene expression matrix but did not perform cell segmentation. Here is an example of just one pair of FASTQ,



```
$ mkdir -p /path/to/output/08.report
$ singularity exec SAW_v6.0.sif cellCut bgef \
    -i /path/to/output/03.count/{SN}.raw.gef \
    -o /path/to/output/05.tissuecut/{SN}.gef \
    -b 1,10,20,50,100,200,500 \
    -O Transcriptomics

$ singularity exec SAW_v6.0.sif report \
    -m /path/to/output/01.mapping/{lane}_barcodeMap.stat \
    -a /path/to/output/01.mapping/{lane}.Log.final.out \
    -g /path/to/output/03.count/{SN}.Aligned.sortedByCoord.out.merge.q10.dedup.
target.bam.summary.stat \
    -l /path/to/output/05.tissuecut/tissuecut.stat \
    -n /path/to/output/05.tissuecut/{SN}.gef \
    -d /path/to/output/06.spatialcluster/{SN}.spatial.cluster.h5ad \
    -t /path/to/output/07.saturation/plot_200x200_saturation.png \
    -b /path/to/output/05.tissuecut/tissue_fig/scatter_200x200_MID_gene_counts.png
\
    -v /path/to/output/05.tissuecut/tissue_fig/violin_200x200_MID_gene.png \
    -c /path/to/output/05.tissuecut/tissue_fig/statistic_200x200_MID_gene_DNB.png
\
    --bin20Saturation,--bin20violin,--bin20MIDGeneDNB
    --bin50Saturation /path/to/output/05.tissuecut/tissue_fig/scatter_50x50_MID_
gene_counts.png \
    --bin50violin /path/to/output/05.tissuecut/tissue_fig/violin_50x50_MID_gene.
png \
    --bin50MIDGeneDNB /path/to/output/05.tissuecut/tissue_fig/statistic_50x50_MID_
gene_DNB.png \
    --bin100Saturation /path/to/output/05.tissuecut/tissue_fig/scatter_100x100_
MID_gene_counts.png \
    --bin100violin /path/to/output/05.tissuecut/tissue_fig/violin_100x100_MID_
gene.png \
    --bin100MIDGeneDNB /path/to/output/05.tissuecut/tissue_fig/statistic_100x100_
MID_gene_DNB.png \
    --bin150Saturation /path/to/output/05.tissuecut/tissue_fig/scatter_150x150_
MID_gene_counts.png \
```

* Please be noted that we use the backward slash "\" to indicate the end of a line in a command that spans multiple lines.

```
    --bin150violin /path/to/output/05.tissuecut/tissue_fig/violin_150x150_MID_gene.
png \
    --bin150MIDGeneDNB /path/to/output/05.tissuecut/tissue_fig/statistic_150x150_
MID_gene_DNB.png \
    -i /path/to/output/04.register/{SN}.rpi \
    -r standard_version \
    -s {SN} \
    --pipelineVersion SAW_v6.0 \
    --iprFile ${imageIPR} \
    --species {species_name} \
    --tissue {tissue_type} \
    --reference {reference_index} \
    -o /path/to/output/08.report
```

Run **`report`** for **`register`** aligned microscopic staining image is absent (Here shows an example of just one pair of FASTQ, similar to multiple pairs),

```
$ mkdir -p /path/to/output/08.report
$ singularity exec SAW_v6.0.sif cellCut bgef \
    -i /path/to/output/03.count/{SN}.raw.gef \
    -o /path/to/output/05.tissuecut/{SN}.gef \
    -b 1,10,20,50,100,200,500 \
    -O Transcriptomics
$ singularity exec SAW_v6.0.sif report \
      -m /path/to/output/01.mapping/{lane}_barcodeMap.stat \
      -a /path/to/output/01.mapping/{lane}.Log.final.out \
      -g /path/to/output/03.count/{SN}.Aligned.sortedByCoord.out.merge.q10.dedup.
target.bam.summary.stat \
      -l /path/to/output/05.tissuecut/tissuecut.stat \
      -n /path/to/output/05.tissuecut/{SN}.gef \
      -d /path/to/output/06.spatialcluster/{SN}.spatial.cluster.h5ad \
      -t /path/to/output/07.saturation/plot_200x200_saturation.png \
      -b /path/to/output/05.tissuecut/tissue_fig/scatter_200x200_MID_gene_counts.
png \
      -v /path/to/output/05.tissuecut/tissue_fig/violin_200x200_MID_gene.png \
      -c /path/to/output/05.tissuecut/tissue_fig/statistic_200x200_MID_gene_DNB.
png \
      --bin20Saturation,--bin20violin,--bin20MIDGeneDNB
      --bin50Saturation /path/to/output/05.tissuecut/tissue_fig/scatter_50x50_MID_
gene_counts.png \
      --bin50violin /path/to/output/05.tissuecut/tissue_fig/violin_50x50_MID_gene.
png \
      --bin50MIDGeneDNB /path/to/output/05.tissuecut/tissue_fig/statistic_50x50_
MID_gene_DNB.png \
      --bin100Saturation /path/to/output/05.tissuecut/tissue_fig/scatter_100x100_
MID_gene_counts.png \
      --bin100violin /path/to/output/05.tissuecut/tissue_fig/violin_100x100_MID_
gene.png \
      --bin100MIDGeneDNB /path/to/output/05.tissuecut/tissue_fig/statis-
```

* Please be noted that we use the backward slash “\” to indicate the end of a line in a command that spans multiple lines.

```
tic_100x100_MID_gene_DNB.png \
      --bin150Saturation /path/to/output/05.tissuecut/tissue_fig/scatter_150x150_
MID_gene_counts.png \
      --bin150violin /path/to/output/05.tissuecut/tissue_fig/violin_150x150_MID_
gene.png \
      --bin150MIDGeneDNB /path/to/output/05.tissuecut/tissue_fig/statis-
tic_150x150_MID_gene_DNB.png \
      -r standard_version \
      -s {SN} \
      --pipelineVersion SAW_v6.0 \ --
      species {species_name} \
      --tissue {tissue_type} \
      --reference {reference_index} \
      -o /path/to/output/08.report
```

* Please be noted that we use the backward slash "\" to indicate the end of a line in a command that spans multiple lines.

2.13.3 Outputs

`report` output files that have cell bin data input are organized as below:

```
$ tree /path/to/output/08.report
/path/to/output/08.report
├── scatter_1x1_MID_gene_counts.png
├── SN.report.html
├── SN.statistics.json
├── statistic_1x1_MID_gene_DNB.png
└── violin_1x1_MID_gene.png
```

`report` output files that the cell bin data inputs are absent:

```
$ tree /path/to/output/08.report
/path/to/output/08.report
├── SN.report.html
└── SN.statistics.json
```

2.14 Other Applications

2.14.1 Other applications of cellCut

SAW **cellCut** is also an application for manipulating GEF file. SAW contains this tool to convert GEF format gene expression matrix to plain table or complete a GEF. Users may also manipulate the GEF files using the separate individual C++ compiled program geftools[12] or its python encapsulated package gefpy[13].

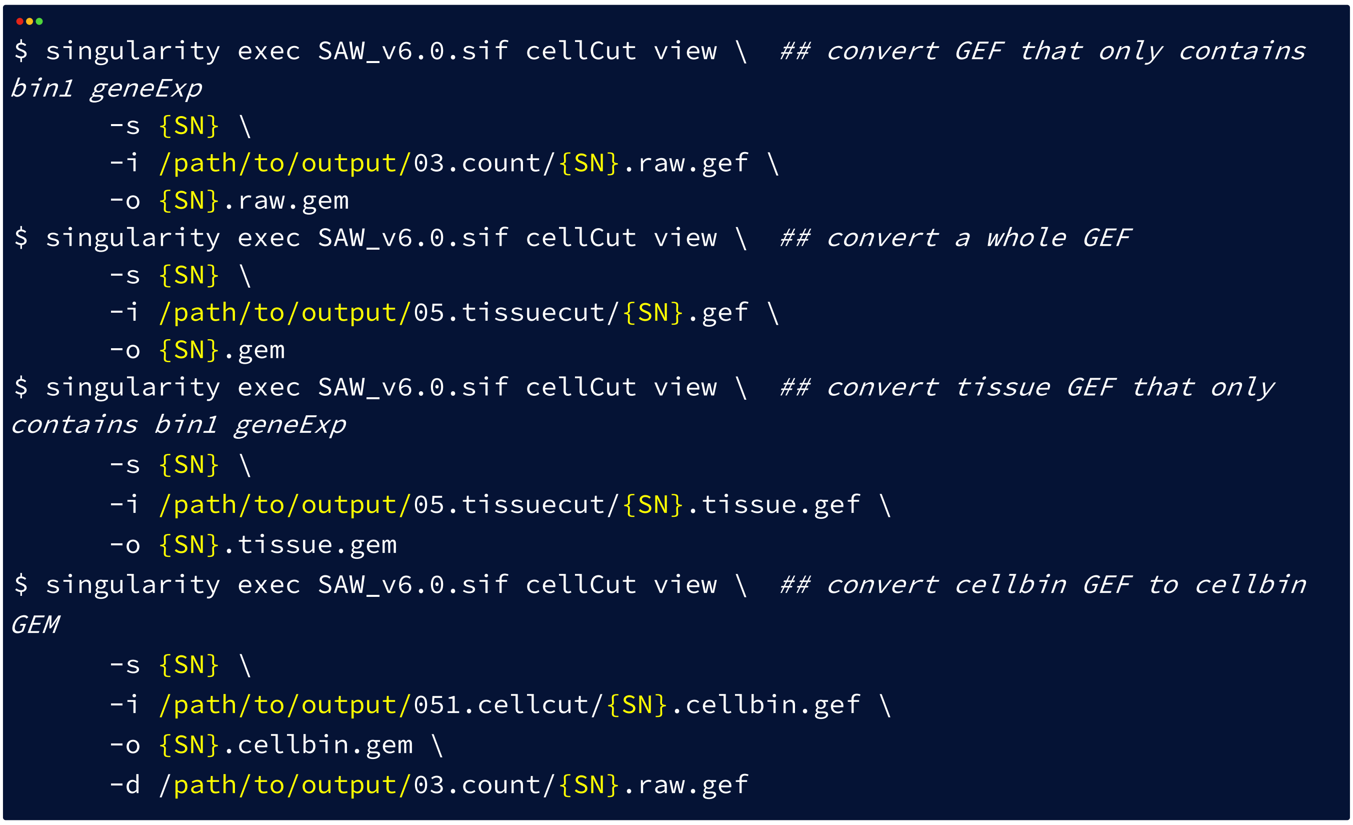
2.14.1.1 Arguments and Options

Table 2-15 Other applications of cellCut Arguments and Options

Commands	Parameters	Function
view	**-i, --input-file**	(Required) Input bin GEF file or cell bin GEF file.
	-o, --output-file	(Optional; default to stdout) Output GEM file.
	-d, --exp_data	(Optional; default to “”) Input bin1 GEF to get cell bin GEM.
	-b, --bin-size	(Optional; default to1) Set bin size for bin GEF file. Only valid for bin GEF.
	-s, --serial-number	(Required) Stereo-seq Chip T serial number.
	-e, --exon	(Optional; default to 1) Whether or not output exon group.
	-w, --errorCode-file	(Optional; default to false) Determine output error code to file.
bgef	**-i, --input-file**	(Required) Input gene expression matrix file (.gem/.gem.gz) or bin1 GEF file.
	-o, --output-file	(Required) Output bin GEF file.
	-b, --bin-size	(Required; default to 1,10,20,50,100,200,500) Comma-separate bin size list.
	-r, --region	(Optional; default to “”) A rectangular region represented by the comma-separated list of vertex coordinates. For example, minx,maxX-,minY,maxY.
	-t, --threads	(Optional; default to 8) Number of threads.
	-s, --stat	(Optional; default to true) Whether create stat group. Stat group includes a gene dataset which contains total MIDcount and E10 score for each gene.
	-O, --omics	(Required; default to Transcriptomics) Specify the omics.
	-v, --verbose	(Optional) Verbose output.
	-w, --errorCode-file	(Optional; default to false) Determine output error code to file.

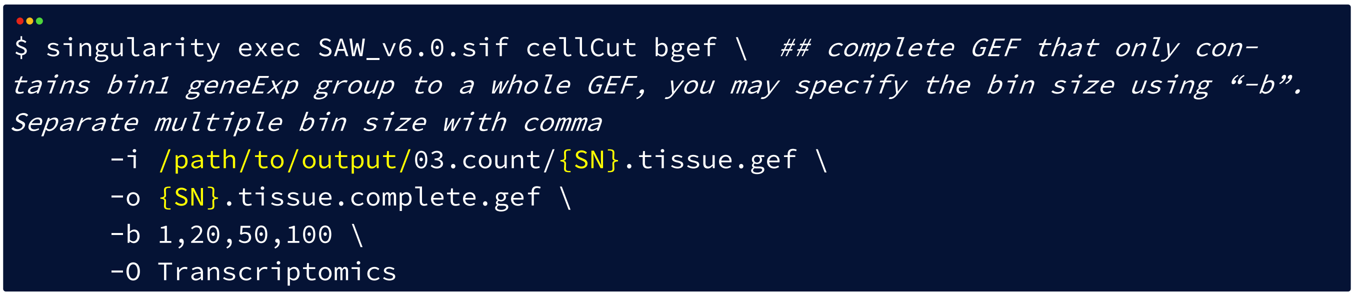
2.14.1.2 Usage Examples

Function 1: GEF to plain table GEM format



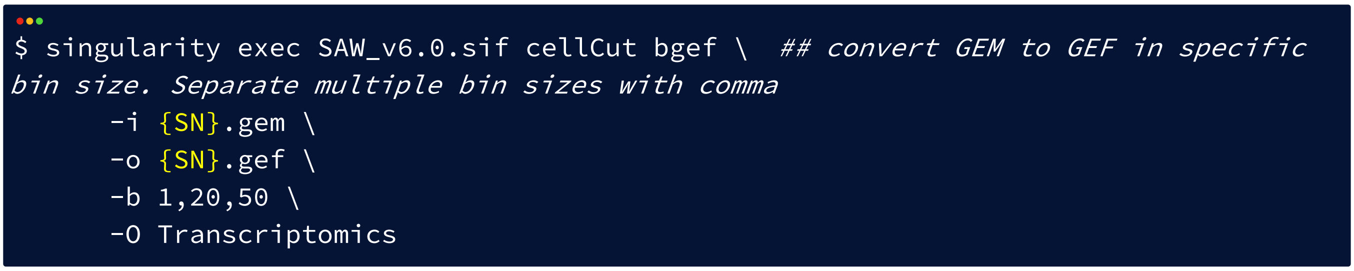
```
$ singularity exec SAW_v6.0.sif cellCut view \  ## convert GEF that only contains
bin1 geneExp
      -s {SN} \
      -i /path/to/output/03.count/{SN}.raw.gef \
      -o {SN}.raw.gem
$ singularity exec SAW_v6.0.sif cellCut view \  ## convert a whole GEF
      -s {SN} \
      -i /path/to/output/05.tissuecut/{SN}.gef \
      -o {SN}.gem
$ singularity exec SAW_v6.0.sif cellCut view \  ## convert tissue GEF that only
contains bin1 geneExp
      -s {SN} \
      -i /path/to/output/05.tissuecut/{SN}.tissue.gef \
      -o {SN}.tissue.gem
$ singularity exec SAW_v6.0.sif cellCut view \  ## convert cellbin GEF to cellbin
GEM
      -s {SN} \
      -i /path/to/output/051.cellcut/{SN}.cellbin.gef \
      -o {SN}.cellbin.gem \
      -d /path/to/output/03.count/{SN}.raw.gef
```

Function 2: completion of a GEF



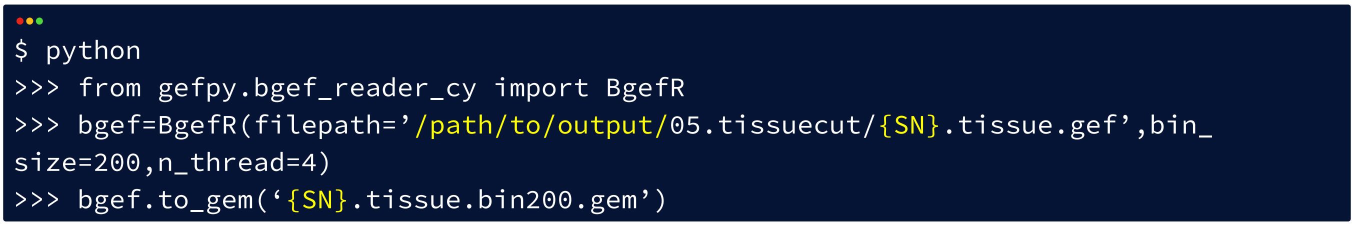
```
$ singularity exec SAW_v6.0.sif cellCut bgef \  ## complete GEF that only con-
tains bin1 geneExp group to a whole GEF, you may specify the bin size using “-b”.
Separate multiple bin size with comma
      -i /path/to/output/03.count/{SN}.tissue.gef \
      -o {SN}.tissue.complete.gef \
      -b 1,20,50,100 \
      -O Transcriptomics
```

Function 3: converting GEM to GEF



```
$ singularity exec SAW_v6.0.sif cellCut bgef \  ## convert GEM to GEF in specific
bin size. Separate multiple bin sizes with comma
      -i {SN}.gem \
      -o {SN}.gef \
      -b 1,20,50 \
      -O Transcriptomics
```

Example of GEF to GEM conversion using `gefpy`, users may specify the bin size.



```
$ python
>>> from gefpy.bgef_reader_cy import BgefR
>>> bgef=BgefR(filepath=’/path/to/output/05.tissuecut/{SN}.tissue.gef’,bin_
size=200,n_thread=4)
>>> bgef.to_gem(‘{SN}.tissue.bin200.gem’)
```

2.14.2 rapidRegister

SAW `rapidRegister` is a lite `register` pipeline that performs all modules in the `register` except cell segmentation. The usage is the same as `register`.

Running `rapidRegister` requires the following files:

- `count` output gene expression matrix file **(.raw.gef)**
- ImageQC processed microscopic tissue staining image file **(.tar.gz),compatible with the QC outputs from ImageStudio v1 and v2**
- ImageQC Image process record file **(.ipr)**

Expected running time for 1 cm x 1 cm Stereo-seq Chip T image: ~20 min, Memory: ~20 G

The change of output file versions as below:

- **Image process record file (.ipr) updated from v0.0.1 to v0.1.0**
- **Registered pyramid image file (.rpi) updated from v0.0.1 to v0.0.2**

2.14.2.1 Arguments and Options

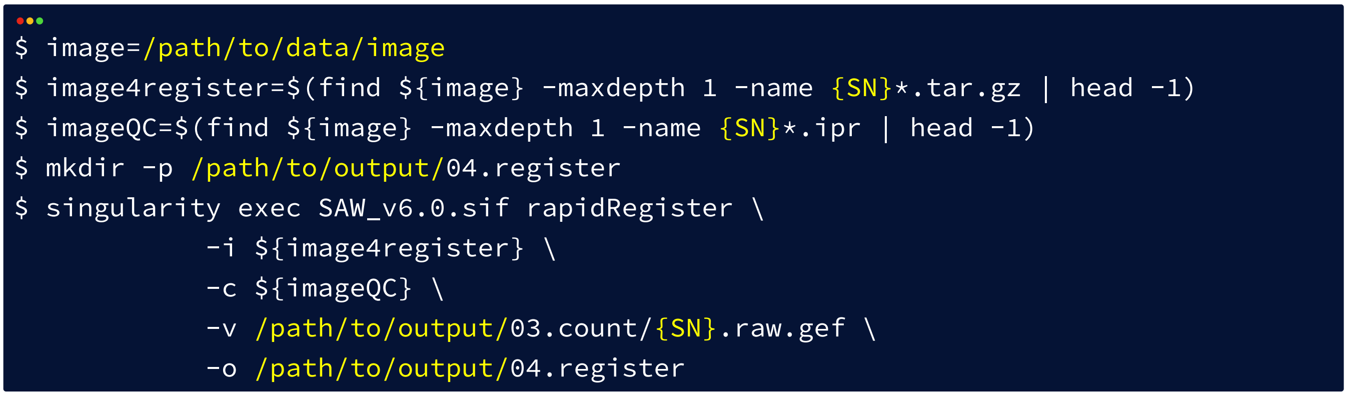
Table 2-16 `rapidRegister` Arguments and Options

Parameter	Function
`-i`	(Required) ImageQC/ImageStudio processed staining image TAR.GZ file or image pre-processed output directory.
`-c`	(Required) ImageQC/ImageStudio IPR (image process record) file. IPR is designed to efficiently hold images and process records generated from each image processing step along with metadata in various data types. Please check **Stereo-seq File Format Manual** to get more information on the IPR file.
`-v`	(Optional; depends on `-i`) `count` output gene expression matrix GEF file.
`-o`	(Required) Path to the directory where to store the `rapidRegister` outputs.

2.14.2.2 Usage Examples

The usage of `rapidRegister` is similar to `register`. Here we only take scenario 1 as an example.

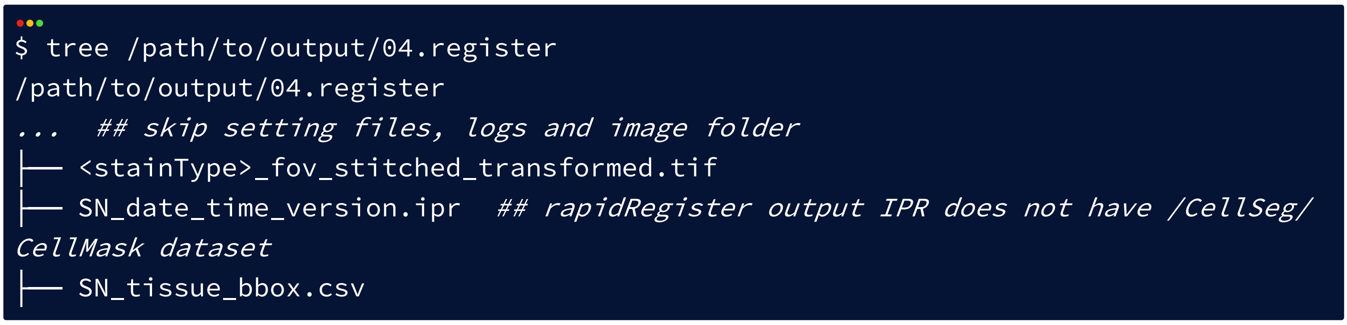
Scenario 1: Process images from ImageQC/ImageStudio output data and gene expression matrix. Perform stitching, tissue segmentation, and registration with gene expression matrix.



```
$ image=/path/to/data/image
$ image4register=$(find ${image} -maxdepth 1 -name {SN}*.tar.gz | head -1)
$ imageQC=$(find ${image} -maxdepth 1 -name {SN}*.ipr | head -1)
$ mkdir -p /path/to/output/04.register
$ singularity exec SAW_v6.0.sif rapidRegister \
          -i ${image4register} \
          -c ${imageQC} \
          -v /path/to/output/03.count/{SN}.raw.gef \
          -o /path/to/output/04.register
```

2.14.2.3 Outputs

`rapidRegister` output files are organized as below:



```
$ tree /path/to/output/04.register
/path/to/output/04.register
...  ## skip setting files, logs and image folder
├── <stainType>_fov_stitched_transformed.tif
├── SN_date_time_version.ipr   ## rapidRegister output IPR does not have /CellSeg/
CellMask dataset
├── SN_tissue_bbox.csv
```

2.14.3 checkGTF

SAW **`checkGTF`** is an application for checking whether the GTF/GFF file is in the correct format as the input for **`count`**.

Running **`checkGTF`** requires the following files:

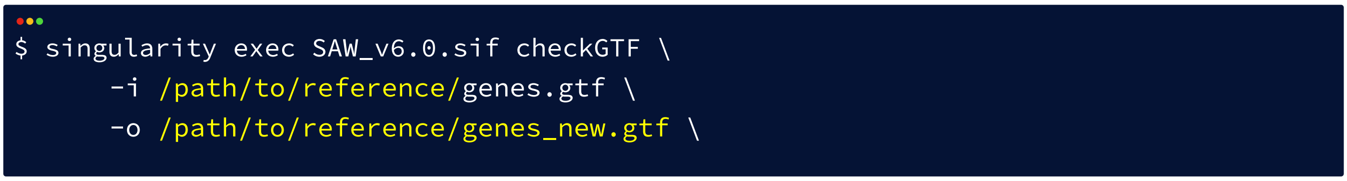
- Reference genome annotation GFF/GTF[9,10] file **(.gff / .gtf)**

2.14.3.1 Arguments and Options

Table 2-17 `checkGTF` Arguments and Options

Parameter	Function
-i	(Required) Gene annotation GFF/GTF file.
-o	(Required) Output a re-format GFF/GTF file.

2.14.3.2 Usage Examples



```
$ singularity exec SAW_v6.0.sif checkGTF \
      -i /path/to/reference/genes.gtf \
      -o /path/to/reference/genes_new.gtf \
```

2.14.4 Other applications of imageTools

Besides `ipr2img`, `imageTools` has three more functions: `img2rpi`(writes images in RPI format), `merge` (merges stain image with tissue segmentation and cell segmentation images to check the segmentation result), and `overlay` (stacks inferred track line template with stitched panoramic image to check stitching result or stacks track line template from matrix with registered panoramic image to check registration result).

2.14.4.1 Arguments and Options

Table 2-18 Other applications of `imageTools` Arguments and Options

Commands	Parameters	Function
`img2rpi`	`-i`	(Required) Input TIFF file. Separate multiple files by comma.
	`-g`	(Required) Set the '<stainType>/<imageType>' of the input TIFF stores in the RPI. Separate multiple files by comma, the order of groups needs to be exactly same with the input TIFF files.
	`-b`	(Required) Bin sizes. Separate multiple bin sizes with space, for example 1 10 50 100
	`-o`	(Required) Output path for the RPI file. Has to use absolute path or relative path.
`merge`	`-i`	(Required) Input TIFF files. Separate multiple files by comma. The maximum number of TIFF files to be composited is three, and the channel order is R-G-B.
	`-o`	(Required) Output path for the multichannel image.
`overlay`	`-i`	(Required) Input TIFF file. Usually input a pre-registered panoramic image if overlaid with the stitching template derived from track cross, and input a registered microscopic image if overlaid with the track line template acquired from gene expression matrix.
	`-c`	(Required) Image configuration file (.ipr) or template points file (.txt).
	`-o`	(Required) Output path of the TIFF file which has the template overlaid on the selected image.
	`-d`	(Required) Module name. Convert template from IPR for the selected modules. Valid options: `stitch` or `register`.
	`-l`	(Optional, default to 30) Cross limbs length in pixel.
	`-w`	(Optional, default to 1) Cross line thickness in pixel.

2.14.4.2 Usage Examples

Function 1: `img2rpi`

```
$ singularity exec SAW_v6.0.sif imageTools img2rpi \
    -i /path/to/output/04.register/<stainType1>_fov_stitched_transformed.tif,/
path/to/output/04.register/<stainType2>_fov_stitched_transformed.tif \
    -g <stainType1>/<ImageType>,<stainType2>/<ImageType>\
    -b 1 10 50 100 \
    -o /path/to/output/04.register/fov_stitched_transformed.rpi   ## this RPI is
used for manual registration in StereoMap
```

```
# An easy example of transforming multiple <stainType>_fov_stitched_transformed.
tif
# and storing as '<stainType>/Image' into fov_stitched_transformed.rpi

registerTif=$(find /path/to/output/04.register -maxdepth 1 -name *fov_stitched_
transformed.tif)
regTifStr=$(echo $registerTif | tr ' ' ',')
regGroup=$(find /path/to/output/04.register -maxdepth 1 -name '*fov_stitched_
transformed.tif' -exec sh -c 'for f do basename -- "$f" _fov_stitched_transformed.
tif;done' sh {} +)
regGroupStr=$(echo $regGroup | sed 's/ \|$/\/Image,/g' | sed 's/.$//')

singularity exec SAW_v6.0.sif imageTools img2rpi \
    -i ${regTifStr}  \
    -g ${regGroupStr} \ #<stainType>/Image, <stainType>/Image
    -b 1 10 50 100 \
    -o /path/to/output/04.register/fov_stitched_transformed.rpi
```

The change of output file versions as below:

- **Registered pyramid image file (.rpi) updated from v0.0.1 to v0.0.2**

Function 2: `merge` (`-i` **Arbitrary up to three grayscale images (.tif)**)

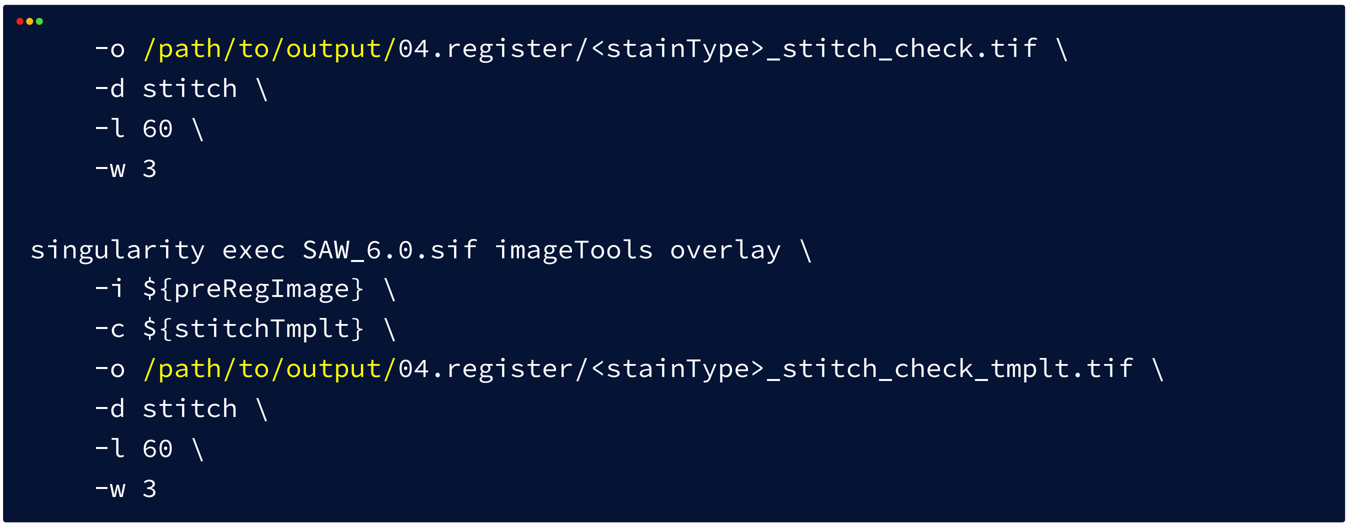
```
 $ singularity exec SAW_v6.0.sif imageTools merge \
    -i /path/to/output/04.register/<stainType>_{SN}_regist.tif,/path/to/output/04.
register/<stainType>_{SN}_tissue_cut.tif,/path/to/output/04.register/<stainType>_
{SN}_mask.tif \
    -o /path/to/output/04.register/merge.tif
```

* Please be noted that we use the backward slash "\" to indicate the end of a line in a command that spans multiple lines.

Function 3: `overlay`

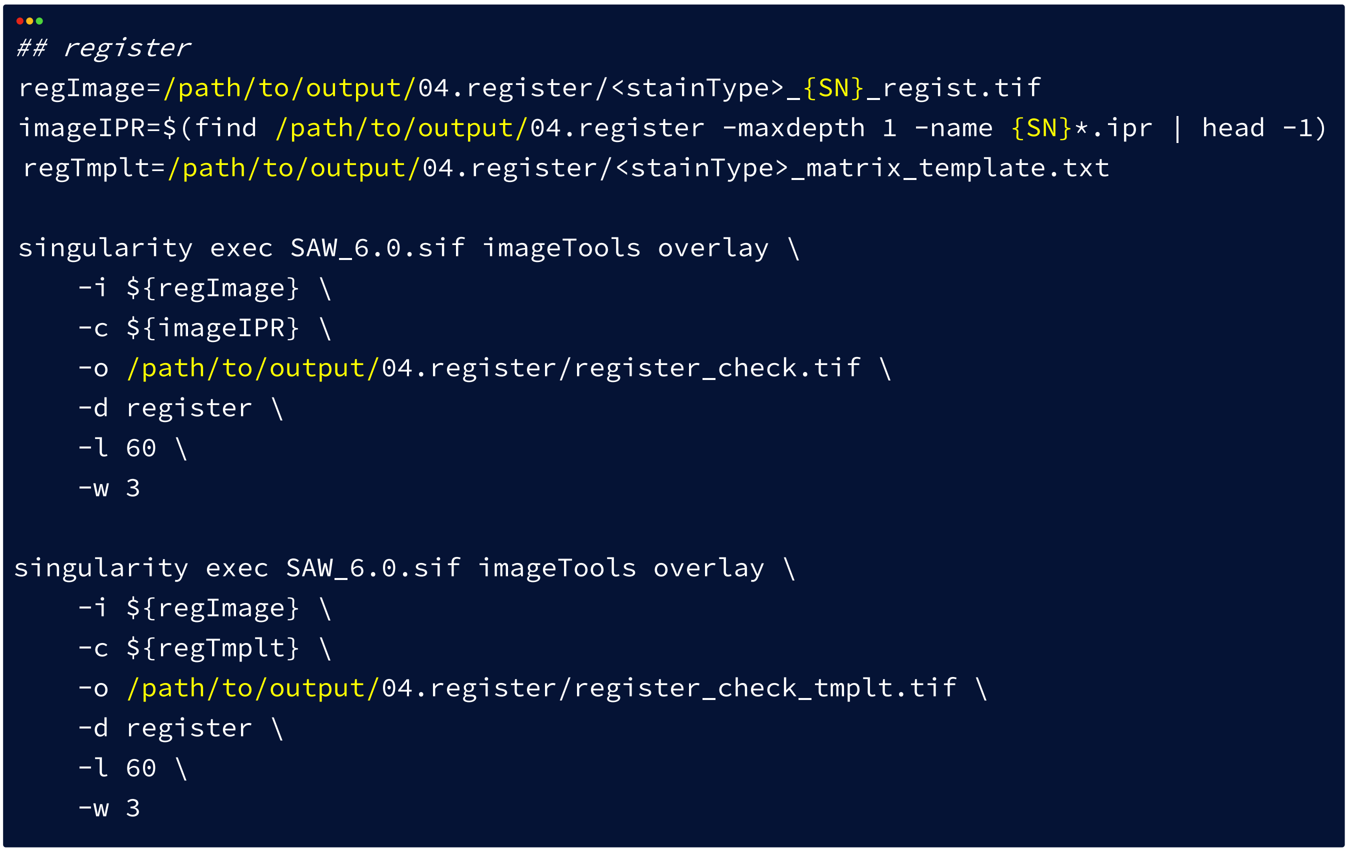
Overlay stitching template in IPR or in a TXT format with the pre-registered panoramic image to check the stitching result.

```
## stitch
preRegImage=/path/to/output/04.register/<stainType>_fov_stitched_transformed.tif
imageIPR=$(find /path/to/output/04.register -maxdepth 1 -name {SN}*.ipr | head -1)
 stitchTmplt=/path/to/output/04.register/<stainType>_transform_template.txt

singularity exec SAW_6.0.sif imageTools overlay \
    -i ${preRegImage} \
    -c ${imageIPR} \
```



```
    -o /path/to/output/04.register/<stainType>_stitch_check.tif \
    -d stitch \
    -l 60 \
    -w 3

singularity exec SAW_6.0.sif imageTools overlay \
    -i ${preRegImage} \
    -c ${stitchTmplt} \
    -o /path/to/output/04.register/<stainType>_stitch_check_tmplt.tif \
    -d stitch \
    -l 60 \
    -w 3
```

Overlay registration template in IPR or in a TXT format with the registered panoramic image.



```
## register
regImage=/path/to/output/04.register/<stainType>_{SN}_regist.tif
imageIPR=$(find /path/to/output/04.register -maxdepth 1 -name {SN}*.ipr | head -1)
regTmplt=/path/to/output/04.register/<stainType>_matrix_template.txt

singularity exec SAW_6.0.sif imageTools overlay \
    -i ${regImage} \
    -c ${imageIPR} \
    -o /path/to/output/04.register/register_check.tif \
    -d register \
    -l 60 \
    -w 3

singularity exec SAW_6.0.sif imageTools overlay \
    -i ${regImage} \
    -c ${regTmplt} \
    -o /path/to/output/04.register/register_check_tmplt.tif \
    -d register \
    -l 60 \
    -w 3
```

2.14.5 manualRegister

SAW manualRegister pipeline is performed when automatic registration result does not meet the requirement. Users can use **StereoMap**, an offline visualization software, to manually register images with the gene expression matrix. The operation parameters are recorded in the JSON file and can be input into the SAW `manualRegister` pipeline to modify registration record in IPR. Remember to run `imageTools ipr2img` again to acquire transformed images.

Running `manualRegister` requires the following files and information:

- count output gene expression matrix file **(.raw.gef)**
- An image processing output directory, including several `*fov_stitched_transformed.tif` files, usually in `/path/to/output/04.register`
- `register/rapidRegister` output IPR file **(.ipr)**, supporting v0.1.0
- **StereoMap** manual registration output JSON file, recording manual operations **(.regist.json)**

🕒 Expected running time for 1 cm x 1 cm Stereo-seq Chip T image: ~3 min, Memory: 7 G

2.14.5.1 Arguments and Options

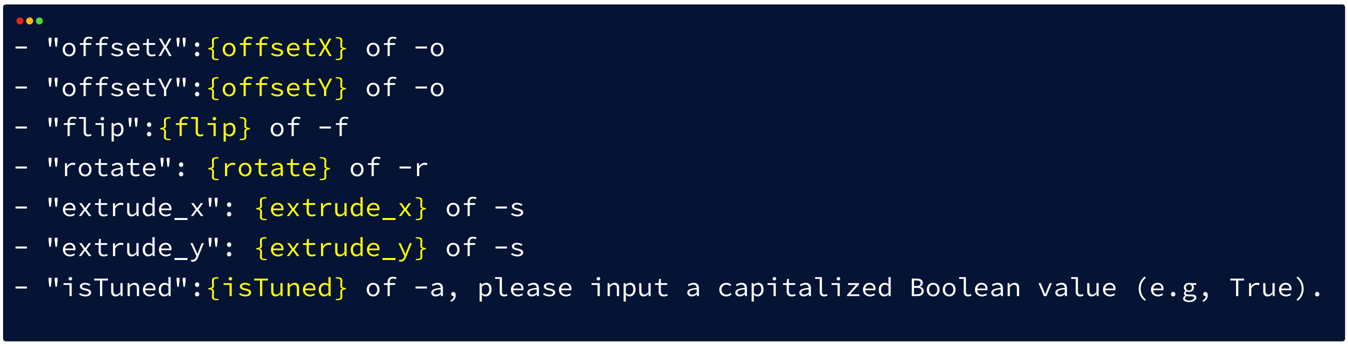
Table 2-19 `manualRegister` Arguments and Options

Parameters	Function
-i	(Required) `register/rapidRegister` output directory which includes a `fov_stitched_transformed.tif` file.
-c	(Required) `register/rapidRegister` output IPR file. `manualRegister` would overwrite registration parameters in the IPR to the manually adjusted parametes acquired from **StereoMap**.
-v	(Required) `count` output gene expression matrix GEF file.
-o	(Optional; default to 0 0) Offset in the x and y direction from the center of the `fov_stitched_transformed.tif` image. These two values are specified as "offsetX" and "offsetY" in the **StereoMap** output JSON file created after manual registration, and will be recorded as "OffsetX" and "OffsetY" in the IPR "Register" group. The offset x and offset y are separate by space.
-f	(Optional; default to 0) Whether flip the image horizontally along the y-axis. This value is specified as "flip" in the **StereoMap** output JSON file created after manual registration. Valid options: 1 (flip), 0 (not flip). Flip information will be recorded as "Flip" in the IPR "Register" group.
-r	(Optional; default to 0) The manual rotation degree from the center of the fov_stitched_transformed.tif image. This value is specified as "rotate" in the **StereoMap** output JSON file created after manual registration, and will be recorded as "ManualRotation" in the IPR "Register" group.
-s	(Optional; default to 0 0) The manual scale in the x and y direction from the center off the `fov_stitched_transformed.tif` image. These two values are specified as "extrude_x" and "extrude_y" in the **StereoMap** output JSON file created after manual registration, and will be recorded as "ManualScaleX" and "ManualScaleY" in the IPR "Register" group. The scale x and scale y are separated by space.

Parameters	Function
-a	(Optional; default to False) Whether perform fine-tuning for manual registration results. Turn on the fine-tuning switch to perform a precise adjustment that relies on a self-developed algorithm to register manually operated image and spatial gene expression matrix according to the tracklines. Since the result of fine-tuning depends highly on the accuracy of image stitching, which aligns tracklines, please check the stitching result before turning on the fine tune switch. The "OffsetX" and "OffsetY" records in the IPR "Register" group for a fine-tuned result would differ from the input offsets. Valid options: True (perform fine tuning) and False (transform image based solely on manual operation).
-p	(Required) Output directory.

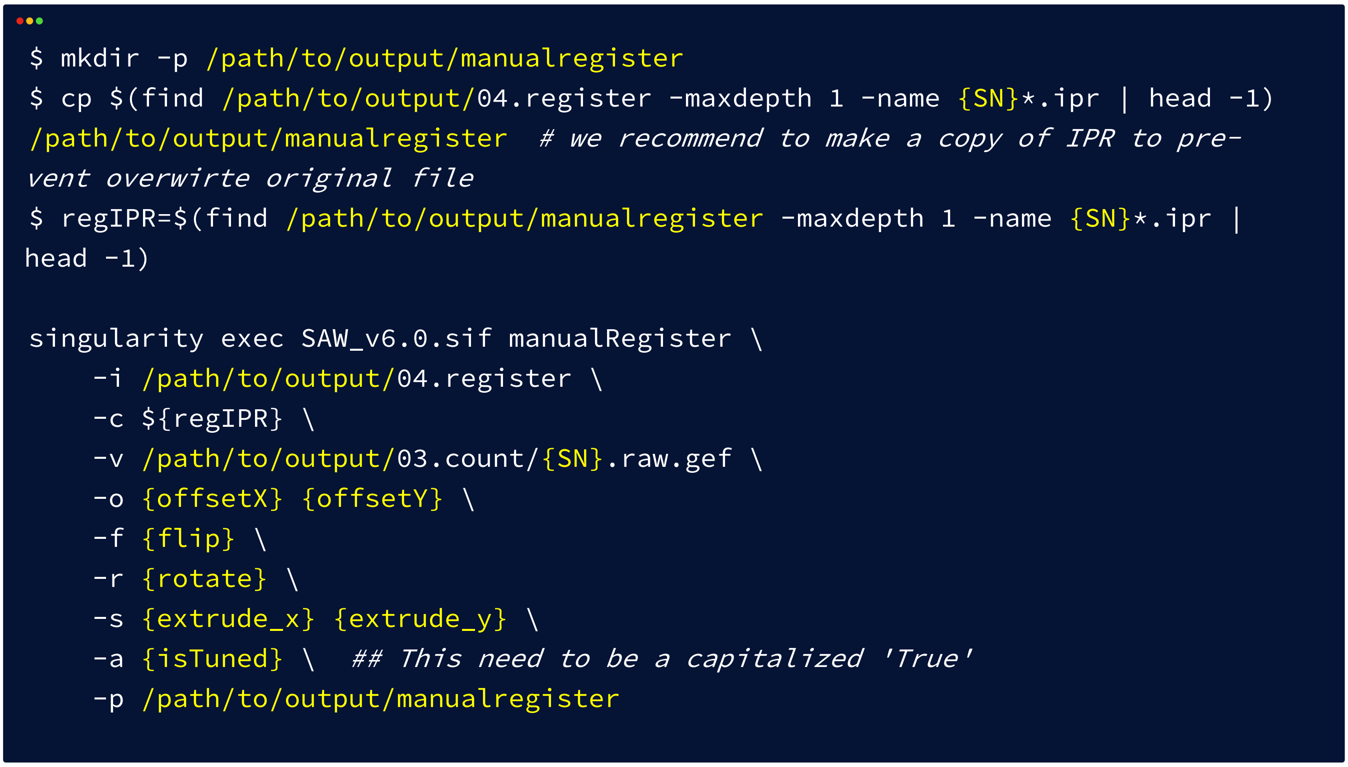
2.14.5.2 Usage Examples

Tip: get the inputs for `manualRegister` from StereoMap manual registration output JSON file:



```
- "offsetX":{offsetX} of -o
- "offsetY":{offsetY} of -o
- "flip":{flip} of -f
- "rotate": {rotate} of -r
- "extrude_x": {extrude_x} of -s
- "extrude_y": {extrude_y} of -s
- "isTuned":{isTuned} of -a, please input a capitalized Boolean value (e.g, True).
```

Scenario 1: perform fine-tuning



```
 $ mkdir -p /path/to/output/manualregister
 $ cp $(find /path/to/output/04.register -maxdepth 1 -name {SN}*.ipr | head -1)
 /path/to/output/manualregister  # we recommend to make a copy of IPR to pre-
vent overwirte original file
 $ regIPR=$(find /path/to/output/manualregister -maxdepth 1 -name {SN}*.ipr |
head -1)

 singularity exec SAW_v6.0.sif manualRegister \
     -i /path/to/output/04.register \
     -c ${regIPR} \
     -v /path/to/output/03.count/{SN}.raw.gef \
     -o {offsetX} {offsetY} \
     -f {flip} \
     -r {rotate} \
     -s {extrude_x} {extrude_y} \
     -a {isTuned} \  ## This need to be a capitalized 'True'
     -p /path/to/output/manualregister
```

Scenario 2: transform image based solely on manual operation

```
$ mkdir -p /path/to/output/manualregister
$ cp $(find /path/to/output/04.register -maxdepth 1 -name {SN}*.ipr | head -1)
/path/to/output/manualregister  # we recommend to make a copy of IPR to pre-
vent overwirte original file
$ regIPR=$(find /path/to/output/manualregister -maxdepth 1 -name {SN}*.ipr |
head -1)

singularity exec SAW_v6.0.sif manualRegister \
    -i /path/to/output/04.register \
    -c ${regIPR} \
    -v /path/to/output/03.count/{SN}.raw.gef \
    -o {offsetX} {offsetY} \
    -f {flip} \
    -r {rotate} \
    -s {extrude_x} {extrude_y} \
    -p /path/to/output/manualregister
```

2.14.6 lasso

SAW `lasso` pipeline is performed to extract cell or spatial gene expression matrix(ces) of one or multiple manually delineated closed regions acquired from **StereoMap**.

Running **`lasso`** requires the following files and information:

- GEF file that includes **`wholeExp`** group or **`cellCut`** output cell bin GEF file **(.gef or .cellbin.gef)**
- **StereoMap** output coordinates list file **(.geojson)**

🕓 Running time and memory are highly dependent on the selected bin size and data volume.

2.14.6.1 Arguments and Options

Table 2-20 `lasso` Arguments and Options

Parameters	Function
-i	(Required) GEF file that includes wholeExp group or cellCut output cell bin GEF file (.gef or .cellbin.gef)
-m	(Required) StereoMap output coordinates list file (.geojson)
-n	(Required) The Stereo-seq Chip T serial number.
-o	(Required) Path to the directory where to store the lasso outputs. It has to be an existing path.
-s	(Required for square bin) Bin size.

2.14.6.2 Usage Examples

Please Note! Replace `{SN}` with your Stereo-seq Chip T serial number (SN, e.g. SS200000135TL_D1), `{bin}` with the bin size you want (e.g. 1, 10, 200), and `/upload/path/{taskID}` with the true path and task ID.

Scenario 1: square bin lasso

```
$ mkdir -p /path/to/output/lasso

$ singularity exec SAW_v6.0.sif lasso \
    -i /path/to/output/05.tissuecut/{SN}.gef \
    -m /upload/path/{taskID}.anno.geojson \
    -o /path/to/output/lasso \
    -s {bin} \
    -n {SN}
```

Scenario 2: cell bin lasso

```
$ mkdir -p /path/to/output/lasso

$ singularity exec SAW_v6.0.sif lasso \
    -i /path/to/output/051.tissuecut/{SN}.cellbin.gef \
    -m /upload/path/{taskID}.anno.geojson \
    -o /path/to/output/lasso \
    -n {SN}
```

2.14.6.3 Outputs

Square bin `lasso` output files are:

```
tree /path/to/output/lasso
/path/to/output/lasso
└── segmentation
    ├── SN.lasso.mask.tif
    └── SN.lasso.label.gem.gz
```

Cellbin `lasso` output files are:

```
tree /path/to/output/lasso
/path/to/output/lasso
└── segmentation
    ├── SN.lasso.cellbin.gef
```