CUT&RUN-Flow

Release 0.10

Daniel Stribling, Rolf Renne

Sep 05, 2020

DOCUMENTATION CONTENTS:

Welcome to *CUT&RUN-Flow* (*CnR-flow*), a Nextflow pipeline for QC, tag trimming, normalization, and peak calling for paired-end sequencing data from CUT&RUN experiments.

This software is available via GitHub, at http://www.github.com/rennelab/CnR-flow .

Full project documentation is available at this documentation.

Pipeline Design:

CUT&RUN-Flow is built using Nextflow, a powerful domain-specific workflow language built to create flexible and efficient bioinformatics pipelines. Nextflow provides extensive flexibility in utilizing cluster computing environments such as PBS and SLURM, and in automated and compartmentalized handling of dependencies using Conda / Bioconda and Environment Modules.

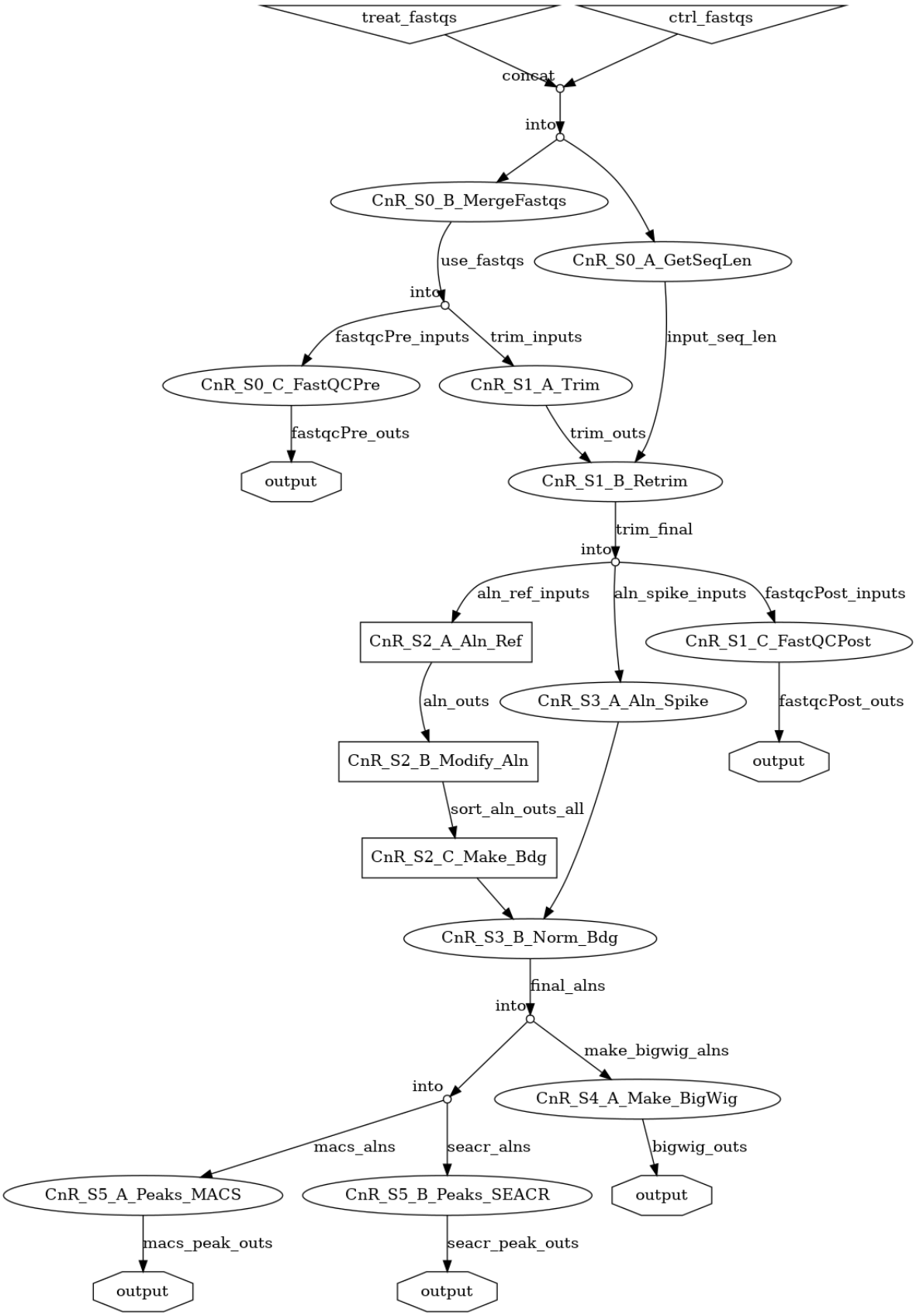
Dependencies:

In addition to standard local configurations, Nextflow allows handling of dependencies in separated working environments within the same pipeline using Conda or Environment Modules. **CnR-flow is pre-configured to acquire and utilize dependencies using conda environments with no additional required setup.**

CUT&RUN-Flow utilizes UCSC Genome Browser Tools and Samtools for reference library preparation, FastQC for tag quality control, Trimmomatic and CUT&RUN-Tools:kseq_test for tag trimming, Bowtie2 for tag alignment, Samtools, bedtools and UCSC Genome Browser Tools for alignment manipulation, and MACS2 and/or SEACR for peak calling, as well as their associated language subdependencies of Java, Python2/3, R, and C++.

Pipeline Features:

- One-step reference database prepration using a path (or URL) to a FASTA file.
- Ability to specify groups of samples containing both treatment (Ex: H3K4me3) and control (Ex: IgG) antibody groups, with automated association of each control sample with the respective treatment samples during the peak calling step
- Built-in normalization protocol to normalize to a sequence library of the user's choice when spike-in DNA is used in the CUT&RUN Protocol (Optional, includes an *E. coli* reference genome for utiliziation of *E. coli* as a spike-in control as described by *Meers et. al. (eLife 2019)* [see the *References* section of this documentation])
- SLURM, PBS… and many other job scheduling environments enabled natively by Nextflow
- Output of CRAM (alignment), bedgraph (genome coverage), and bigWig (genome coverage) file formats


treat_fastqs
ctrl_fastqs
concat
into
CnR_S0_B_MergeFastqs
use_fastqs
CnR_S0_A_GetSeqLen
into
fastqcPre_inputs
trim_inputs
input_seq_len
CnR_S0_C_FastQCPre
CnR_S1_A_Trim
fastqcPre_outs
trim_outs
output
CnR_S1_B_Retrim
trim_final
into
aln_ref_inputs
aln_spike_inputs
fastqcPost_inputs
CnR_S2_A_Aln_Ref
CnR_S1_C_FastQCPost
aln_outs
CnR_S3_A_Aln_Spike
fastqcPost_outs
CnR_S2_B_Modify_Aln
output
sort_aln_outs_all
CnR_S2_C_Make_Bdg
CnR_S3_B_Norm_Bdg
final_alns
into
make_bigwig_alns
into
CnR_S4_A_Make_BigWig
macs_alns
seacr_alns
bigwig_outs
CnR_S5_A_Peaks_MACS
CnR_S5_B_Peaks_SEACR
output
macs_peak_outs
seacr_peak_outs
output
output

For a full list of required dependencies and tested versions, see the *Dependencies* section of this documentation, and for dependency configuration options see the *Dependency Config* section.

CHAPTER

ONE

QUICKSTART

Here is a brief introduction on how to install and get started using the pipeline. For full details, see this documentation.

Prepare Task Directory:

Create a task directory, and navigate to it.

```
$ mkdir ./my_task  # (Example)
$ cd ./my_task     # (Example)
```

Install Nextflow (if necessary):

Download the nextflow executable to your current directory.
(You can move the nextflow executable and add to $PATH for future usage)

```
$ curl -s https://get.nextflow.io | bash

# For the following steps, use:
nextflow    # If nextflow executable on $PATH (assumed)
./nextflow  # If running nextflow executable from local directory
```

Download and Install CnR-flow:

Nextflow will download and store the pipeline in the user's Nextflow info directory (Default: `~/.nextflow/`)

```
$ nextflow run rennelab/CnR-flow --mode initiate
```

Configure, Validate, and Test:

If using Nextflow's builtin Conda dependency handling (recommended), install miniconda (if necessary).
Installation instructions
The CnR-flow configuration with Conda should then work "out-of-the-box."

If using an alternative configuration, see the *Dependency Config* section of this documentation for dependency configuration options.

Once dependencies have been configured, validate all dependencies:

```
$ nextflow run CnR-flow --mode validate_all
```

Fill the required task input paramaters in "nextflow.config" For detailed setup instructions, see the *Task Setup* section of this documentation *Additionally, for usage on a SLURM, PBS, or other cluster systems, configure your system executor, time, and memory settings.*

```
# Configure:
$ <vim/nano...> nextflow.config   # Task Input, Steps, etc. Configuration

#REQUIRED values to enter (all others *should* work as default):
# ref_fasta                (or some other ref-mode/location)
# treat_fastqs             (input paired-end fastq[.gz] file paths)
#   [OR fastq_groups]      (mutli-group input paired-end .fastq[.gz] file paths)
```

Prepare and Execute Pipeline:

Prepare your reference databse (and normalization reference) from .fasta[.gz] file(s):

```
$ nextflow run CnR-flow --mode prep_fasta
```

Perform a test run to check inputs, paramater setup, and process execution:

```
$ nextflow run CnR-flow --mode dry_run
```

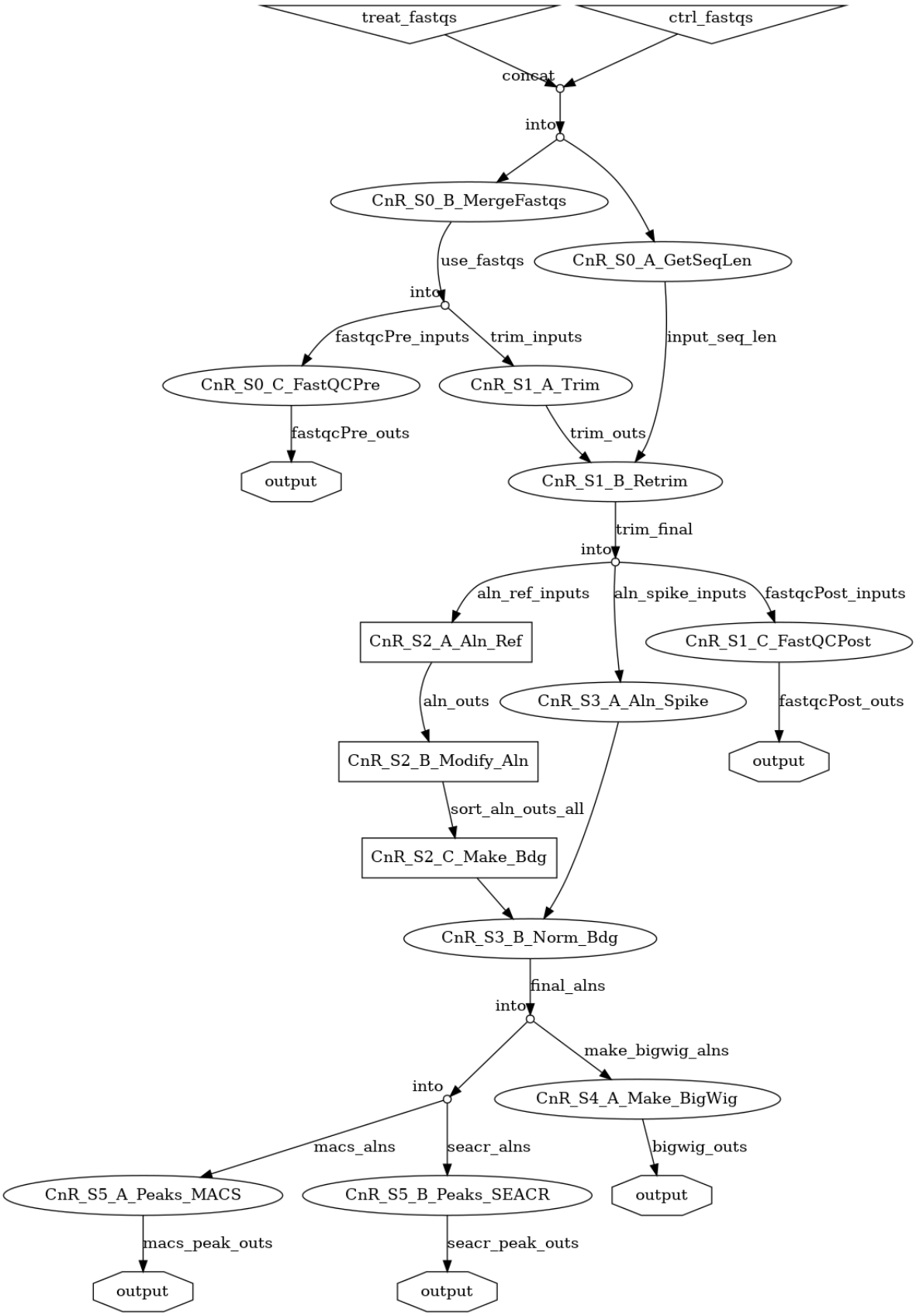
If satisifed with the pipeline setup, execute the pipeline:

```
$ nextflow run CnR-flow --mode run
```

Further documentation on CUT&RUN-Flow components, setup, and usage can be found in this documentation.

1.1 Workflow

The CUT&RUN-Flow (CnR-flow) workflow is designed to be simple to install and execute without sacrificing analysis capabilities. Nextflow [Nextflow_Citation] provides several features that facilitate this design, included automatic download of dependencies and i/o handling between workflow components.



Full Workflow:

1.1.1 Download and Installation

(If necessary, begin by installing Nextflow and Conda as described in *Quickstart*)

Nextflow allows the pipeline to be downloaded simply by using the "nextflow run" command, and providing the user/repository path to the relevant github repository. CnR-flow has a one-step installation mode that can be performed simultaneously (`--mode initiate`).
Together, this gives the command:

```
$ nextflow run rennelab/CnR-flow --mode initiate
```

This should download the pipeline, install the few required local files and dependencies, and prepare the pipeline for execution.

Note:

After the initial download, the pipeline can then be referred to by name, as with:
"nextflow run CnR-flow . . . "

1.1.2 Dependency Setup and Validation

cnr-flow comes preconfigured to utilize the conda environment management together with bioconda packages to handle dependency utilization.
for more details, or for an alternative configuration, see *Dependency Config*

either using the default dependency configuration, or with a user's custom configuration, the dependency setup can then be tested using the `--mode validate` or `--mode validate_all` parameters.

```
# validate all workflow dependencies (recommended)
$ nextflow run CnR-flow --mode validate_all
```

```
# validate only steps enabled in nextflow.config
$ nextflow run CnR-flow --mode validate
```

1.1.3 Reference Preparation

CnR-flow provides one-step preparation of alignment reference genome(s) for use by the pipeline. Either the local path or URL of a fasta file are provided to the `--ref_fasta (params.ref_fasta)` paramater, and the execution is performed with:

```
$ nextflow run CnR-flow --mode prep_fasta
```

This copies the reference fasta to the directory specified by `--ref_dir (params.ref_dir)`, creates a bowtie2 alignment reference, creates a fasta index using Samtools, and creates a ".chrom.sizes" file using UCSC faCount [faCount_Citation]. The effective genome size is also calculated with faCount, using the (Total - N's) method. Reference details are written to a ".refinfo.txt" in the same directory.

Note: If normalization is enabled, the same process will be repeated for the fasta file supplied to `--norm_ref_fasta (params.norm_ref_fasta)` for alignments to the spike-in control genome.

These referenes are then detected automatically, using the same parameter used for preparation setup. For more details, see *Task Setup*. The list of all detectable prepared databases can be provided using the `--mode list_refs` run mode:

```
$ nextflow run CnR-flow --mode list_refs
```

1.1.4 Experimental Condition

CUT&RUN-Flow allows automated handling of treatment (Ex: H3K4me3) and and control (Ex: IgG) input files, performing the analysis steps on each condition in parallel, and then associating the treatment with the control for the final peak calling step. This can be performed either with a single treatment/control group, or with multiple groups in parallel. For more information, see *Task Setup*.

1.1.5 Preprocessing Steps

GetSeqLen

This step is enabled with paramater `--do_retrim (params.do_retrim)` (default: `true`). This step takes one example input fastq[.gz] file and determines the sequence length, for use in later steps.

MergeFastqs

This step is enabled with paramater `--do_merge_lanes (params.do_merge_lanes)` (default: `true`). If multiple sets of paired end files are provided that differ only by the "_L00#_" component of the name, these sequences are concatenated for further analysis.

For example, these files will be merged into the common name: 'my_sample_CTRL_R(1/2)_001.fastq.gz':

```
./my_sample_CTRL_L001_R1_001.fastq.gz ./my_sample_CTRL_L001_R2_001.fastq.gz
./my_sample_CTRL_L002_R1_001.fastq.gz ./my_sample_CTRL_L002_R2_001.fastq.gz
#... -->
./my_sample_CTRL_R1_001.fastq.gz ./my_sample_CTRL_R2_001.fastq.gz
```

FastQCPre

This step is enabled with paramater `--do_fastqc (params.do_fastqc)` (default: `true`). FastQC [FastQC_Citation] is utilized to perform quality control checks on the input (presumably untrimmed) fastq[.gz] files.

Trim

This step is enabled with paramater `--do_trim (params.do_trim)` (default: `true`). Trimming of input fastq[.gz] files for read quality and adapter content is performed by Trimmomatic [Trimmomatic_Citation].

Default trimming parameters:

```
trimmomatic_settings = "ILLUMINACLIP:${params.trimmomatic_adapterpath}/
↪Truseq3.PE.fa:2:15:4:4:true LEADING:20 TRAILING:20 SLIDINGWINDOW:4:15␣
↪MINLEN:25"
```

Default flags:

```
trimmomatic_flags = "-phred33"
```

Retrim

This step is enabled with paramater `--do_retrim (params.do_retrim)` (default: `true`). Trimming of input fastq[.gz] files is performed by the kseq_test executable from the CUTRUNTools toolkit [CUTRUNTools_Citation]. It is designed to identify and remove very short adapter sequences from tags that were potentially missed by previous trimming steps.

FastQCPost

This step is enabled with paramater `--do_fastqc (params.do_fastqc)` (default: `true`). FastQC [FastQC_Citation] is utilized to perform quality control checks on sequences after any/all trimming steps are performed.

1.1.6 Alignment Steps

Aln_Ref

Sequence reads are aligned to the reference genome using Bowtie2 [Bowtie2_Citation].
Default alignment parameters were selected using concepts presented in work by the Henikoff Lab [Meers2019] and the Yuan Lab [CUTRUNTools_Citation].

Default flags:

```
aln_ref_flags = "--local --very-sensitive-local --phred33 -I 10 -X 700 --
↪dovetail --no-unal --no-mixed --no-discordant"
```

> **Warning:** None of the output alignments (.sam/.bam/.cram) files produced in this step (or indeed, anywhere else in the pipeline) are normalized. The only normalized outputs are are genome genome coverage tracks produced if normalization is enabled.

Modify_Aln

Output alignments are then subjected to several cleaning, filtering, and preprocessing steps utilizing Samtools [Samtools_Citation].
These are:

1. Removal of unmapped reads (samtools view)
2. Sorting by name (samtools sort [required for fixmate])
3. Adding/correcting mate pair information (samtools fixmate -m)
4. Sorting by genome coordinate (samtools sort)
5. Marking duplicates (samtools mkdup)
6. (Optional Processing Steps [see below])
7. Alignment compression BAM -> CRAM (samtools view)
8. Alignment indexing (samtools index)

Optional processing steps include:

- Removal of Duplicates
- Filtering to reads <= 120 bp in length

The desired category (or categories) of output are selected with `--use_aln_modes` (`params.use_aln_modes`). Multiple categores can be specifically selected using `params.use_aln_modes` as a list, and the resulting selections are analyzed and output in parallel. (Example: `params.use_aln_modes = ['all', 'less_120_dedup']`)

Option	Deduplicated	Length <= 120bp
all	false	false
all_dedup	true	false
less_120	false	true
less_120_dedup	true	true

Default mode:

```
use_aln_modes = ["all"]  // Options: ["all", "all_dedup", "less_120", "less_
↪120_dedup"]
```

Make_Bdg

Further cleaning steps are then performed on the outputs, to prepare the alignments for (optional) normalization and peak calling.
These modifications are performed as suggested by the Henikoff lab in the documentation for SEACR, https://github.com/FredHutch/SEACR/blob/master/README.md [SEACR_Citation] , and are performed utilizing Samtools [Samtools_Citation] and bedtools [bedtools_Citation].

These are:

1. Sorting by name and uncompress CRAM -> BAM (samtools sort)
2. Covert BAM to bedgraph (bedtools bamtobed)
3. Filter discordant tag pairs (awk)
4. Change bedtools bed format to BEDPE format (cut | sort)
5. Convert BEDPE to (non-normalized) bedgraph (bedtools genomecov)

Note: Genome coverage tracks output by this step are NOT normalized.

1.1.7 Normalization Steps

Aln_Spike

This step is enabled with paramater `--do_norm_spike (params.do_norm_spike)` (default: `true`).
This step calculates a normalization factor for scaling output genome coverage tracks.

Strategy: A dual-alignment strategy is used to filter out any reads that cross-map to both the primary reference and the normalization references. Sequence pairs that align to the normalization reference are then re-aligned to the primary reference. The number of read pairs that align to both references is then subtracted from the normalization factor output by this step, depending on the value of `--norm_mode (params.norm_mode)` (default: obj:*true*).

Details:

Sequence reads are first aligned to the normalization reference genome using Bowtie2 [Bowtie2_Citation]. Default alignment parameters are the same as with alignment to the primary reference genome.

Default flags:

```
aln_norm_flags = params.aln_ref_flags
```

All reads that aligned to the normalization reference are then again aligned to the primary reference using Bowtie2 [Bowtie2_Citation].

Counts are then performed of **pairs** of sequence reads that align (and re-align, respectively) to each reference using Samtools [Samtools_Citation] (via `samtools view`). The count of aligned pairs to the spike-in genome reference is then returned, with the number of cross-mapped pairs subtracted depending on the value of `--norm_mode (params.norm_mode)`.

norm_mode	Normalization Factor Used
all	norm_ref_aligned (pairs)
adj	norm_ref_aligned - cross_map_aligned (pairs)

```
norm_mode = 'adj' // Options: ['adj', 'all']
seacr_norm_mode = "auto" // Options: "auto", "norm", "non"
```

Norm_Bdg

This step is enabled with paramater `--do_norm_spike (params.do_norm_spike)` (default: `true`).
This step uses a normalization factor to create scaled genome coverage tracks. The calculation as provided by the Henikoff Lab:
https://github.com/Henikoff/Cut-and-Run/blob/master/spike_in_calibration.csh [Meers2019] is:

$$count_{norm} = (count_{site} * norm\_scale)/norm\_factor$$

Thus, the scaling factor used is calucated as:

$$scale\_factor = norm\_scale / norm\_factor$$

Where `norm_factor` is calculated in the previous step, and the arbitrary `norm_scale` is provided by the parameter: `--norm_scale (params.norm_scale)`.

Default `norm_scale` value:

```
norm_scale = 1000  // Arbitrary value for scaling of normalized counts.
```

The normalized genome coverage track is then created by bedtools [bedtools_Citation] using the `-scale` option.

1.1.8 Conversion Steps

Make_BigWig

This step is enabled with paramater `--do_make_bigwig (params.do_make_bigwig)` (default: `true`).
This step converts the output genome coverage file from the previous steps as in the UCSC bigWig file format using UCSC bedGraphToBigWig, a genome coverage format with significantly decreased file size [bedGraphToBigWig_Citation].

> **Warning:** The bigWig file format is a "lossy" file format that cannot be reconverted to bedGraph with all information intact.

1.1.9 Peak Calling Steps

One or more peak callers can be used for peak calling. The peak caller used is determined by `--peak_callers (params.peak_callers)`. This can either be provided a single argument, as with:

```
--peak_callers seacr (params.peak_callers = "seacr")
```

...or can be configured using a list:

```
params.peak-callers = ['macs', 'seacr']
```

Default `peak_callers` value:

```
peak_callers = ['macs', 'seacr']  // Options: ['macs', 'seacr']
```

Peaks_MACS2

This step is enabled if `macs` is included in `params.peak-callers`.
This step calls peaks using the **non-normalized** alignment data produced in previous steps, using the MACS2 peak_caller [MACS2_Citation]

Default MACS2 Settings:

```
//Macs2 Settings
macs_qval = '0.01'
macs_flags = ''
```

Peaks_SEACR

This step is enabled if `seacr` is included in `params.peak_callers`.
This step calls peaks using the **normalized** alignment data produced in previous steps (if normalization is enabled, using the SEACR peak caller [SEACR_Citation].

Special thanks to the Henikoff group for their permission to distribute SEACR bundled with this pipeline.

Parameters: `--seacr_norm_mode (params.seacr_norm_mode)` passes either `norm` or `non` to SEACR. Options:

- **`'auto'`:**
  - if `do_norm = true` - Passes `'non'` to SEACR
  - if `do_norm = false` - passes `'norm'` to SEACR
- `'norm'`
- `'non'`

`--seacr_fdr (params.seacr_fdr)` is passed directly to SEACR.

`--seacr_call_stringent (params.seacr_call_stringent)` - SEACR is called in "stringent" mode.
`--seacr_call_relaxed (params.seacr_call_relaxed)` - SEACR is called in "relaxed" mode.
(If both of these are true, both outputs will be produced)

Default SEACR Settings:

```
//SEACR Settings
seacr_fdr_threshhold = "0.01"
seacr_norm_mode = "auto" // Options: "auto", "norm", "non"
seacr_call_stringent = true
seacr_call_relaxed = true
```

1.2 Task Setup

1.2.1 Task Setup Overview

After running using `--mode initiate`, *CUT&RUN-Flow* will copy the task configuration template into your current working directory. For a full example of this file, see *Task nextflow.config*.

```
# Configure:
$ <vim/nano...> ./my_task/nextflow.config   # Task Input, Steps, etc.␣
↪Configuration
```

Task-level inputs such as input files and reference fasta files must be configured here (see *Input File Setup*). Additional task-specific settings are also configured here, such as output read naming rules and output file locations (see *Output Setup*).

Note: These settings are provided for user customizability, but in the majority of cases the default settings should work fine.

Many pipeline settings can justifiably be configured either on a task-specifc basis (in *Task nextflow.config*) or as defaults for the pipeline (in *Pipe nextflow.config*). These include nextflow "executor" settings for use of SLURM and Environment Modules and associated settings such as memory and cpu usage. These settings are described here, in *Executor Setup*, but can also be set in the *Pipe nextflow.config*.

Likewise, settings for individual pipeline components such as Trimmomatic tag trimming paramaters, or the `qval` used for MACS2 peak calling can be provided in either config file, or both (for a description of these parameters, see *Workflow*).

Note: If any settings are provided in both the above *Task nextflow.config* file and the *Pipe nextflow.config* file located in the pipe directory, the task-directory settings will take precedence. For more information on Nextflow configuration precedence, see `config`.

1.2.2 Reference Files Setup

CUT&RUN-Flow handles reference database preparation with a series of steps utilizing `--mode prep_fasta`. The location of the fasta used for preparation is provided to the `--ref_fasta (params.ref_fasta)` paramater as either a file path or URL.

Reference preparation is then performed using:

```
$ nextflow CnR-flow --mode prep_fasta
```

This will place the prepared reference files in the directory specified by `--refs_dir (params.refs_dir)` (see *Output Setup*). Once prepared, the this parameter can be dynamically used during

pipeline execution to detect the reference name and location, depending on the value of the `--ref_mode (params.ref_mode)` parameter.

Ref Modes:

- `'fasta'` : Get reference name from `--ref_fasta (params.ref_fasta)` (which must then be set)
- `'name'` : Get reference name from `--ref_name (params.ref_name)` (which must then be set)
- `'manual'` : Set required paramaters manually:

Ref Required Manual Paramaters:

- `--ref_name (params.ref_name)` : Reference Name
- `--ref_bt2db_path (params.ref_bt2db_path)` : Reference Bowtie2 Alignment Reference Path
- `--ref_chrom_sizes_path (params.ref_chrom_sizes_path)` : Path to <reference>.chrom_sizes file
- `--ref_eff_genome_size (params.ref_eff_genome_size)` : Effective genome size for reference.

The `--ref_mode (params.ref_mode)` parameter also applies to the preparation and location of the fasta used for the normalization reference if `--do_norm (params.do_norm)`. These paramaters are named in parallel using a `norm_[ref...]` prefix and are autodetected from the value of `--norm_ref_fasta (params.norm_ref_fasta)` or `--norm_ref_name (params.norm_ref_name)` depending on the value of `--ref_mode (params.ref_mode)`. For details on normalization steps, see *Normalization Steps*.

1.2.3 Input File Setup

Two (mutually-exclusive) options are provided for supplying input sample fastq[.gz] files to the workflow.

Single Sample Group:

A single group of samples with zero or one (post-combination) control sample(s) for all treatment samples.

- `--treat_fastqs (params.treat_fastqs)`
- `--ctrl_fastqs (params.ctrl_fastqs)`

```
params {
// CnR-flow Input Files:
//   Provided fastqs must be in glob pattern matching pairs.
//     Example: ['./relpath/to/base*R{1,2}*.fastq']
//     Example: ['/abs/path/to/other*R{1,2}*.fastq']
//
treat_fastqs = []  // REQUIRED, Single-group Treatment fastq Pattern
ctrl_fastqs  = []  //           Single-group Control   fastq pattern

}
```

Multiple Sample Group:

A multi-group layout, with groups of samples provided where each group has a control sample. (All groups are required to have a control sample in this mode.)

- `params.fastq_groups`

```
params {
// Can specify multiple treat/control groups as Groovy mapping.
//   Specified INSTEAD of treat_fasts/ctrl_fastqs paramaters.
//   Note: There should be only one ctrl sample per group
//     (after optional lane combination)
//Example:
//fastq_groups = [
//  'group_1_name': ['treat': 'relpath/to/treat1*R{1,2}*',
//                   'ctrl':  'relpath/to/ctrl1*R{1,2}*',
//                  ]
//  'group_2_name': ['treat': ['relpath/to/g2_treat1*R{1,2}*'
//                             '/abs/path/to/g2_treat2*R{1,2}*'
//                            ],
//                   'ctrl':  'relpath/to/g2_ctrl1*R{1,2}*'
//                  ]
//]
//fastq_groups = []

}
```

Multiple pairs of files representing the same sample/replicate that were sequenced on different lanes can be automatically recognized and combined (default: `true`). For more information see: *MergeFastqs*.

Note: Note, for convenience, if the same file is found both as a treatment and control, the copy passed to treatment will be ignored (facilitates easy pattern matching).

Warning: Input files must be paired-end, and in fastq[.gz] format. Nextflow requires the use of this (strange-looking) `R{1,2}` naming construct, (matches either R1 or R2) which ensures that files are fed into the pipeline as pairs.

1.2.4 Executor Setup

Nextflow provides extensive options for using cluster-based job scheduling, such as SLURM, PBS, etc. These options are worth reviewing in the nextflow docs: *executor*. The specific executor is selected with the configuration setting: `process.executor = 'option'`. The default value of `process.executor = 'local'` runs the execution on the local filesystem.

Specific settings of note:

Option	Example
`process.executor`	`'slurm'`
`process.memory`	`'4 GB'`
`process.cpus`	`4`
`process.time`	`'1h'`
`process.clusterOptions`	`'--qos=low'`

To facilitate process efficiency (and for adequate capacity) for different parts of the process, memory-related process labels have been applied to the processes: `'small_mem'`, `'norm_mem'`, and

`'big_mem'`. These are specified using `process.withLabel: my_label { key = value }` Example: `process.withLabel: big_mem { memory = '16 GB' }`.

A `1n/2n/4n` or `1n/2n/8n` strategy is recommended for the respective `small_mem/norm_mem/big_mem` options. (for details on nextflow process labels, see process). Additionally, mutliple cpu usage is disabled for processes that do not support (or aren't significanlly more effective) with multiple processes, and so the `process.cpus` setting only applies to processes within the pipeline with multiple CPUS enabled.

```
// Process Settings (For use of PBS, SLURM, etc)
process {
    // --Executor, see: https://www.nextflow.io/docs/latest/executor.html
    //executor = 'slurm'  // for running processes using SLURM (Default:
↪'local')
    // Process Walltime, See https://www.nextflow.io/docs/latest/process.html
↪#process-time
    //time = '12h'
    // Process CPUs, See https://www.nextflow.io/docs/latest/process.html
↪#cpus
    //cpus = 8
    //
    // Memory: See https://www.nextflow.io/docs/latest/process.html#process-
↪memory
    // Set Memory for specific task sizes (1n/2n/4n scheme recommended)
    //withLabel: big_mem   { memory = '16 GB' }
    //withLabel: norm_mem  { memory = '4 GB'  }
    //withLabel: small_mem { memory = '2 GB'  }
    // -*OR*- Set Memory for all processes
    //memory = "16 GB"

    ext.ph = null //Placeholder to prevent errors.
}
```

1.2.5 Output Setup

Output options can control the quantity, naming, and location of output files from the pipeline.

publish_files: Three modes are available for selecting the number of output files from the pipeline:

- `minimal` : Only the final alignments are output. (Trimmed Fastqs are Excluded)
- `default` : Multiple types of alignments are output. (Trimmed Fastqs are included)
- `all` : All files produced by the pipline (excluding deleted intermediates) are output.

This option is selected with `--publish_files (params.publish_files)`.

publish_mode: This mode selects the value for the Nextflow `process.publishDir` mode used to output files (for details, see: publishDir). Available options are:

- `'copy'` : Copy output files (from the nextflow working directory) to the output folder.
- `'symlink'` : Link to the output files located in the nextflow working directory.

trim_name_prefix & trim_name_suffix:

`--trim_name_prefix (params.trim_name_prefix)` & `--trim_name_suffix (params.trim_name_suffix)`

These options allow trimming of a prefix or suffix from sample names (after any merging steps).

out_dir: `--out_dir (params.out_dir)` : Location for output of the files

refs_dir: `--refs_dir (params.refs_dir)` : Location for placing and searching for refernce directories

```
params {
    // ------- General Pipeline Output Paramaters --------
    publish_files    = 'default' // Options: ["minimal", "default", "all"]
    publish_mode     = 'copy'    // Options: ["symlink", "copy"]

    //Name trim guide: ( regex-based )
    //   ~/groovy-slashy-string/  ;  "~" denotes groovy pattern type.
    //   ~/^/ matches beginning   ;   ~/$/ matches end
    trim_name_prefix = ''        // Example: ~/^myprefix./ removes "myprefix.
↪" prefix.
    trim_name_suffix = ''        // Example: ~/_mysuffix$/ removes "_mysuffix
↪" suffix.

    // Workflow Output Default Naming Scheme:
    //   Absolute paths for output:
    out_dir          = "${launchDir}/cnr_output"
    refs_dir         = "${launchDir}/cnr_references"
}
```

1.3 Pipe Setup

1.3.1 Pipe Settings Location

The Default *CUT&RUN-Flow* settings used by the system are located in *CUT&RUN-Flow's* installation directory. This is by default located in the user's `$HOME` directory as such:

```
# PIpe Defaults Configuration:
$NXF_HOME/assets/rennelab/CnR-flow/nextflow.config  # Pipe Executor, Dependency,␣
↪Resource, etc. Configuration
#Default: $HOME/.nextflow/assets/rennelab/CnR-flow/nextflow.config
```

It is recommended that dependency configuration and other pipe-level settings be configured here. An example of this file is provided in *Pipe nextflow.config*.

Note: If any settings are provided in both the above *Pipe nextflow.config* file and the *Task nextflow.config* file located in the task directory, the task-directory settings will take precedence. For more information on Nextflow configuration precedence, see *`config`*.

1.3.2 Dependencies

The following external dependencies are utilized by *CUT&RUN-Flow*:

Table 1: Dependencies

Dependency	Def. Ver.	Lowest Ver.	Highest Ver.	Citation	Default Conda
Nextflow	20.07.1	20.07.1	20.07.1	[Nextflow_Citation]	N/A
UCSC-faCount	366	366	366	[faCount_Citation]	bioconda::ucsc-facount=366=*_0'
Bowtie2	2.4.1	2.4.1	2.4.1	[Bowtie2_Citation]	bioconda::bowtie2=2.4.1
Samtools	1.9	1.9	1.9	[Samtools_Citation]	bioconda::samtools=1.9=*_11
FastQC	0.11.9	0.11.9	0.11.9	[FastQC_Citation]	bioconda::fastqc=0.11.9
Trimmomatic	0.39	0.39	0.39	[Trimmomatic_Citation]	bioconda::trimmomatic=0.39
CUTRUNTools	20190822	20190822	20190822	[CUTRUNTools_Citation]	N/A
bedtools	2.29.2	2.29.2	2.29.2	[bedtools_Citation]	bioconda::bedtools=2.29.2
UCSC-bedGraphToBigWig	377	377	377	[bedGraphToBigWig_Citation]	libgcc-ng libpng conda-forge::libuuid mysql-connector-c openssl zlib bioconda::ucsc-bedgraphtobigwig=377
MACS2	2.2.6	2.2.6	2.2.6	[MACS2_Citation]	bioconda::macs2=2.2.6
R	3.6.0	3.6.0	3.6.0	[R_Citation]	r=3.6.0
SEACR	1.3	1.3	1.3	[SEACR_Citation]	r=3.6.0 bioconda::bedtools=2.29.2

1.3.3 Dependency Config

Conda/Bioconda

CUT&RUN-FLow comes preconfigured to use the Conda package manager, along with tools from the Bioconda [Bioconda_Citation] package suite for automated dependency handling. Nextflow automatically acquires, stores, and activates each conda environment as it is required in the pipeline. For more information on Nextflow's, usage of conda, see `conda`.

```
// -- External Conda Environments:
facount_conda          = 'bioconda::ucsc-facount=366=*_0'
bowtie2_conda          = 'bioconda::bowtie2=2.4.1'
fastqc_conda           = 'bioconda::fastqc=0.11.9'
trimmomatic_conda      = 'bioconda::trimmomatic=0.39'
bedtools_conda         = 'bioconda::bedtools=2.29.2'
macs2_conda            = 'bioconda::macs2=2.2.6'
R_conda                = 'r=3.6.0'
samtools_conda         = 'bioconda::samtools=1.9=*_11'
bedgraphtobigwig_conda = 'libgcc-ng libpng conda-forge::libuuid mysql-
↪connector-c openssl zlib bioconda::ucsc-bedgraphtobigwig=377'

// -- Packaged/Installed Tool Conda Environments (Changes should not be␣
↪necessary):
seacr_conda         = "${params.R_conda} ${params.bedtools_conda}"
```

(continues on next page)

(continued from previous page)

```
//kseqtest_conda      = ""  // (uses binary, should not be needed)

// -- Comprehensive Conda Environment (If provided, is used for all␣
↪execution)
//all_conda           = ""
```

Modules

CUT&RUN-FLow comes with an alternative "easy" configuration option utilizng Environment Modules. Each conda dependency paramater has an equivalent "module" paramater. Each module will then be loaded at runtime for the appropriate steps of the pipeline. For more information on Nextflow's use of Environment Modules, see `process` ("modules section").

```
// Dependency Configuration Using Environment Modules
// (values will vary depending on system)
// To enable, comment ("//") / delete the conda-configuration and uncomment␣
↪this configuration.
// -- External Tool Modules:
//facount_module          = ""  // Ex: "ucsc/20200320"
//bowtie2_module          = ""  // Ex: "bowtie2/2.3.5.1"
//fastqc_module           = ""  // Ex: "fastqc/0.11.7"
//trimmomatic_module      = ""  // Ex: "trimmomatic/0.39"
//bedtools_module         = ""  // Ex: "bedtools/2.29.2"
//macs2_module            = ""  // Ex: "macs/2.2.7.1"
//R_module                = ""  // Ex: "R/4.0"
//bedgraphtobigwig_module = ""  // Ex: "ucsc/20200320"

// -- Packaged/Installed Tool Modules (Changes should not be necessary):
//seacr_module          = "${params.R_module}:${params.bedtools_module}"
//kseqtest_module       = ""  // (uses binary, should not be needed)

// -- Comprehensive Tool Modules (If provided, is used for all execution)
//all_module            = ""  // Ex: "cutruntools/20200104"
```

Executable System Calls

To accommodate custom module or local setups, each required dependency system call can be customized:

```
// System Call Settings
//   Call executed on the system, defaults assume that each tool is available
//     on the system PATH (as with conda setup)
//   Can replace with direct path as desired:
//   Ex:
//        samtools_call = "samtools"
//     or samtools_call = "/path/to/samtools/dir/samtools"
java_call             = "java"
bowtie2_build_call    = "bowtie2-build"
facount_call          = "faCount"
samtools_call         = "samtools"
```

(continues on next page)

(continued from previous page)

```
fastqc_call           = "fastqc"
trimmomatic_call      = "trimmomatic"
bowtie2_call          = "bowtie2"
bedtools_call         = "bedtools"
macs2_call            = "macs2"
bedgraphtobigwig_call = "bedGraphToBigWig"
kseqtest_call         = "${projectDir}/CUTRUNTools/kseq_test"  //Installed␣
↪with --mode initiate
seacr_call            = "${projectDir}/SEACR/SEACR_1.3.sh"     //Packaged␣
↪with download
seacr_R_script        = "${projectDir}/SEACR/SEACR_1.3.R"      //Packaged␣
↪with download
```

1.4 Example Files

- *Task nextflow.config*
- *Pipe nextflow.config*

1.4.1 Task nextflow.config

This nextflow.config file is copied to the task directory by the command:

```
#mkdir ./my_task_dir
#cd ./my_task_dir
nextflow run rennelab/CnR-flow --mode initiate
```

This configuration file contains paramaters for input file and task workflow setup.

Listing 1: . . . /my_task_dir/nextflow.config (Task Settings)

```
//Daniel Stribling
//Renne Lab, University of Florida
//CnR-flow Configuration File
//
// Config syntax:
// This config file uses the syntax described here:
//   https://www.nextflow.io/docs/latest/config.html
// Primarily, "//" acts as a comment and allows removal/deactivation of lines.
//
// Settings within scopes, Ex:
//   params {
//     setting = value
//     ...
//   }
// are equivalent to:
//   params.setting = value"
//
// Configuration Hierarchy:
```

(continues on next page)

(continued from previous page)

```
//   Different Nextflow configuration files have the precedence order:
//     task-dir  >  pipe-dir  >  user-default  >  pipe-default
//
//   This configruation is (presumably) in the task directory:
//   (after running "nextflow CnR-flow --mode initiate" )
//   and will therefore superceed pipe-setup, user, and pipe hardcoded defaults.
//   For more detail visit: https://www.nextflow.io/docs/latest/config.html
//
//   This configuration only includes task-specific paramaters and paramaters
//   designed to be modified at runtime. Other parameters are provided in the
//   pipe configuration, at CnR-flow/nextflow.config. Parameters
//   provided here will override system defaults.
//

// Process Settings (For use of PBS, SLURM, etc)
process {
    // --Executor, see: https://www.nextflow.io/docs/latest/executor.html
    //executor = 'slurm'  // for running processes using SLURM (Default: 'local')
    // Process Walltime, See https://www.nextflow.io/docs/latest/process.html#process-
↪time
    //time = '12h'
    // Process CPUs, See https://www.nextflow.io/docs/latest/process.html#cpus
    //cpus = 8
    //
    // Memory: See https://www.nextflow.io/docs/latest/process.html#process-memory
    // Set Memory for specific task sizes (1n/2n/4n scheme recommended)
    //withLabel: big_mem   { memory = '16 GB' }
    //withLabel: norm_mem  { memory = '4 GB'  }
    //withLabel: small_mem { memory = '2 GB'  }
    // -*OR*- Set Memory for all processes
    //memory = "16 GB"

    ext.ph = null //Placeholder to prevent errors.
}

params {
    //REQUIRED values to enter (all others should work as default):
    // ref_fasta               (or some other ref-mode/location)
    // treat_fastqs            (input paired-end fastq[.gz] file paths)
    //   [OR fastq_groups]     (mutli-group input paired-end .fastq[.gz] file paths)

    // Automatic alignment reference preparation/usage settings:
    ref_mode       = 'fasta'  // Options: ['name', 'fasta', 'manual']
    ref_fasta      = ''       // REQUIRED: Ex: '/path/to/my/reference.fasta[.gz]'
    // Default Pre-Supplied Normalization library is Ecoli:
    norm_ref_fasta = "${projectDir}/ref_dbs/gcf_000005845.2_asm584v2_genomic.fasta.gz"

    // CnR-flow Input Files:
    //   Provided fastqs must be in glob pattern matching pairs.
    //     Example: ['./relpath/to/base*R{1,2}*.fastq']
    //     Example: ['/abs/path/to/other*R{1,2}*.fastq']
    //
    treat_fastqs = []  // REQUIRED, Single-group Treatment fastq Pattern
    ctrl_fastqs  = []  //           Single-group Control   fastq pattern

    // Can specify multiple treat/control groups as Groovy mapping.
    //   Specified INSTEAD of treat_fasts/ctrl_fastqs paramaters.
```

(continues on next page)

(continued from previous page)

```
    //   Note: There should be only one ctrl sample per group
    //     (after optional lane combination)
    //Example:
    //fastq_groups = [
    //  'group_1_name': ['treat': 'relpath/to/treat1*R{1,2}*',
    //                   'ctrl':  'relpath/to/ctrl1*R{1,2}*',
    //                  ]
    //  'group_2_name': ['treat': ['relpath/to/g2_treat1*R{1,2}*'
    //                             '/abs/path/to/g2_treat2*R{1,2}*'
    //                            ],
    //                   'ctrl':  'relpath/to/g2_ctrl1*R{1,2}*'
    //                  ]
    //]
    //fastq_groups = []

    // ------- Step-Specific Pipeline Paramaters --------
    // Step Settings:
    do_merge_lanes = true  // Merge sample names differing only by lane-ID (Ex: L001,␣
↪L002)
    do_fastqc      = true  // Perform FastQC Analysis
    do_trim        = true  // Trim tags using Trimmomatic
    do_retrim      = true  // Retrim tags using kseq_test (CUTRUNTools)
    do_norm_spike  = true  // Normalize using aligment count to a spike-in reference
    do_make_bigwig = true  // Create UCSC bigWig files from final alignments


    //FastQC Settings:
    fastqc_flags = ''

    // Alignment Mode Options (params.use_aln_modes) :
    //   "all"            : Use all reads
    //   "all_dedup"      : Use all reads, and remove duplicates.
    //   "less_120"       : Use trimmed reads <= 120 bp
    //   "less_120_dedup" : Use trimmed reads <= 120 bp and remove duplicates.
    //   (Multiple modes can be performed in parallel. Ex: ['all', 'less_120'])

    use_aln_modes  = ["all"]  // Options: ["all", "all_dedup", "less_120", "less_120_
↪dedup"]

    // Peak Caller Options (params.peak_callers):
    //   "macs2" : Call Peaks with Macs2
    //   "seacr" : Call Peaks with SEACR
    //   (Multiple callers can be used in parallel. Ex: ['macs2', 'seacr'])

    peak_callers   = ['macs', 'seacr']  // Options: ['macs', 'seacr']

    //Trimmomatic Trim Settings

    //--Trimmomatic Adapter Path, Downloaded after "nextflow CnR-flow --mode initiate"
    trimmomatic_adapterpath = "${projectDir}/ref_dbs/trimmomatic_adapters/TruSeq3-PE-
↪2.fa"
    trimmomatic_settings    = "ILLUMINACLIP:${params.trimmomatic_adapterpath}/Truseq3.
↪PE.fa:2:15:4:4:true LEADING:20 TRAILING:20 SLIDINGWINDOW:4:15 MINLEN:25"
    trimmomatic_flags       = "-phred33"
    //kseq_test (CUTRUNTools) Retrim Settings
    input_seq_len = 'auto'  // Length of untrimmed sequence reads (Ex: '150') or 'auto
↪'
```

(continues on next page)

(continued from previous page)

```
    //Bowtie2 Alignment Settings
    aln_ref_flags  = "--local --very-sensitive-local --phred33 -I 10 -X 700 --
↪dovetail --no-unal --no-mixed --no-discordant"

    //Normalization Settings
    aln_norm_flags = params.aln_ref_flags
    norm_scale     = 1000  // Arbitrary value for scaling of normalized counts.
    norm_mode      = 'adj' // Options: ['adj', 'all']

    //Macs2 Settings
    macs_qval  = '0.01'
    macs_flags = ''

    //SEACR Settings
    seacr_fdr_threshhold = "0.01"
    seacr_norm_mode      = "auto" // Options: "auto", "norm", "non"
    seacr_call_stringent = true
    seacr_call_relaxed   = true

    // ------- General Pipeline Output Paramaters --------
    publish_files    = 'default' // Options: ["minimal", "default", "all"]
    publish_mode     = 'copy'    // Options: ["symlink", "copy"]

    //Name trim guide: ( regex-based )
    //   ~/groovy-slashy-string/  ;  "~" denotes groovy pattern type.
    //   ~/^/ matches beginning   ;   ~/$/ matches end
    trim_name_prefix = ''        // Example: ~/^myprefix./ removes "myprefix." prefix.
    trim_name_suffix = ''        // Example: ~/_mysuffix$/ removes "_mysuffix" suffix.

    // Workflow Output Default Naming Scheme:
    //   Absolute paths for output:
    out_dir          = "${launchDir}/cnr_output"
    refs_dir         = "${launchDir}/cnr_references"
}
```

1.4.2 Pipe nextflow.config

This nextflow.config file is contains configuration paramaters for pipeline setup.

Listing 2: . . . /CnR-flow/nextflow.config (Pipe Settings)

```
//Daniel Stribling
//Renne Lab, University of Florida
//CnR-flow Configuration File
//
// Config syntax:
// This config file uses the syntax described here:
//   https://www.nextflow.io/docs/latest/config.html
// Primarily, "//" acts as a comment and allows removal/deactivation of lines.
//
// Settings within scopes, Ex:
//   params {
//     setting = value
```

(continues on next page)

(continued from previous page)

```
//      ...
//    }
// are equivalent to:
//   params.setting = value"
//
// Configuration Hierarchy:
//   Different Nextflow configuration files have the precedence order:
//     task-dir  >  pipe-dir  >  user-default  >  pipe-default
//
//   (presumably) This configruation is in the CnR-flow/nextflow.config,
//   and will therefore superceed user and pipe defaults,
//   but be overridden by the task-specific configuration file
//   in the case of conflicting settings.
//   For more detail visit: https://www.nextflow.io/docs/latest/config.html
//
//   Users wishing to modify dependency configuration should specifically
//   direct attention to the conda/module section.
//   Other default paramaters are provided below.

// Pipeline Details
manifest {
    author = 'Daniel Stribling, Rolf Renne'
    defaultBranch = 'master'
    description = """\
    CUT&RUN-Flow, A Nextflow pipeline for QC, tag trimming, normalization, and peak
    calling for paired-end sequence data from CUT&RUN experiments.
    """.stripIndent()
    //doi
    homePage = 'http://www.rennelab.com'
    mainScript = 'CnR-flow.nf'
    name = 'CUT&RUN-Flow'
    nextflowVersion = '20.07.1'
    version = '0.10'
}

// Process Settings (For use of PBS, SLURM, etc)
process {
    // --Executor, see: https://www.nextflow.io/docs/latest/executor.html
    //executor = 'slurm'  // for running processes using SLURM (Default: 'local')
    // Process Walltime, See https://www.nextflow.io/docs/latest/process.html#process-
↪time
    //time = '12h'
    // Process CPUs, See https://www.nextflow.io/docs/latest/process.html#cpus
    //cpus = 8
    //
    // Memory: See https://www.nextflow.io/docs/latest/process.html#process-memory
    // Set Memory for specific task sizes (1n/2n/4n scheme recommended)
    //withLabel: big_mem   { memory = '16 GB' }
    //withLabel: norm_mem  { memory = '4 GB'  }
    //withLabel: small_mem { memory = '2 GB'  }
    // -*OR*- Set Memory for all processes
    //memory = "16 GB"

    ext.ph = null //Placeholder to prevent errors.
}

params {
```

(continues on next page)

(continued from previous page)

```
    // ------- Dependency Configuration --------
    // Configuration using conda is recommended for most systems.
    // Each dependency can only have one type of resource configured:
    // (Ex: bowtie2_module OR bowtie2_conda)

    // Dependency Configuration Using Anaconda
    // Miniconda Install Instructions:
    //     https://docs.conda.io/en/latest/miniconda.html
    //
    // -- External Conda Environments:
    facount_conda          = 'bioconda::ucsc-facount=366=*_0'
    bowtie2_conda          = 'bioconda::bowtie2=2.4.1'
    fastqc_conda           = 'bioconda::fastqc=0.11.9'
    trimmomatic_conda      = 'bioconda::trimmomatic=0.39'
    bedtools_conda         = 'bioconda::bedtools=2.29.2'
    macs2_conda            = 'bioconda::macs2=2.2.6'
    R_conda                = 'r=3.6.0'
    samtools_conda         = 'bioconda::samtools=1.9=*_11'
    bedgraphtobigwig_conda = 'libgcc-ng libpng conda-forge::libuuid mysql-connector-c␣
↪openssl zlib bioconda::ucsc-bedgraphtobigwig=377'

    // -- Packaged/Installed Tool Conda Environments (Changes should not be␣
↪necessary):
    seacr_conda         = "${params.R_conda} ${params.bedtools_conda}"
    //kseqtest_conda      = ""  // (uses binary, should not be needed)

    // -- Comprehensive Conda Environment (If provided, is used for all execution)
    //all_conda           = ""

    // Dependency Configuration Using Environment Modules
    // (values will vary depending on system)
    // To enable, comment ("//") / delete the conda-configuration and uncomment this␣
↪configuration.
    // -- External Tool Modules:
    //facount_module          = ""  // Ex: "ucsc/20200320"
    //bowtie2_module          = ""  // Ex: "bowtie2/2.3.5.1"
    //fastqc_module           = ""  // Ex: "fastqc/0.11.7"
    //trimmomatic_module      = ""  // Ex: "trimmomatic/0.39"
    //bedtools_module         = ""  // Ex: "bedtools/2.29.2"
    //macs2_module            = ""  // Ex: "macs/2.2.7.1"
    //R_module                = ""  // Ex: "R/4.0"
    //bedgraphtobigwig_module = ""  // Ex: "ucsc/20200320"

    // -- Packaged/Installed Tool Modules (Changes should not be necessary):
    //seacr_module         = "${params.R_module}:${params.bedtools_module}"
    //kseqtest_module      = ""  // (uses binary, should not be needed)

    // -- Comprehensive Tool Modules (If provided, is used for all execution)
    //all_module           = ""  // Ex: "cutruntools/20200104"

    // System Call Settings
    //   Call executed on the system, defaults assume that each tool is available
    //     on the system PATH (as with conda setup)
    //   Can replace with direct path as desired:
    //   Ex:
    //        samtools_call = "samtools"
    //     or samtools_call = "/path/to/samtools/dir/samtools"
```

(continues on next page)

(continued from previous page)

```
    java_call             = "java"
    bowtie2_build_call    = "bowtie2-build"
    facount_call          = "faCount"
    samtools_call         = "samtools"
    fastqc_call           = "fastqc"
    trimmomatic_call      = "trimmomatic"
    bowtie2_call          = "bowtie2"
    bedtools_call         = "bedtools"
    macs2_call            = "macs2"
    bedgraphtobigwig_call = "bedGraphToBigWig"
    kseqtest_call         = "${projectDir}/CUTRUNTools/kseq_test"  //Installed with --
↪mode initiate
    seacr_call            = "${projectDir}/SEACR/SEACR_1.3.sh"     //Packaged with␣
↪download
    seacr_R_script        = "${projectDir}/SEACR/SEACR_1.3.R"      //Packaged with␣
↪download

    // ------- Step-Specific Pipeline Paramaters --------
    // Step Settings:
    do_merge_lanes = true  // Merge sample names differing only by lane-ID (Ex: L001,␣
↪L002)
    do_fastqc      = true  // Perform FastQC Analysis
    do_trim        = true  // Trim tags using Trimmomatic
    do_retrim      = true  // Retrim tags using kseq_test (CUTRUNTools)
    do_norm_spike  = true  // Normalize using aligment count to a spike-in reference
    do_make_bigwig = true  // Create UCSC bigWig files from final alignments


    //FastQC Settings:
    fastqc_flags = ''

    // Alignment Mode Options (params.use_aln_modes) :
    //   "all"            : Use all reads
    //   "all_dedup"      : Use all reads, and remove duplicates.
    //   "less_120"       : Use trimmed reads <= 120 bp
    //   "less_120_dedup" : Use trimmed reads <= 120 bp and remove duplicates.
    //   (Multiple modes can be performed in parallel. Ex: ['all', 'less_120'])

    use_aln_modes  = ["all"]  // Options: ["all", "all_dedup", "less_120", "less_120_
↪dedup"]

    // Peak Caller Options (params.peak_callers):
    //   "macs2" : Call Peaks with Macs2
    //   "seacr" : Call Peaks with SEACR
    //   (Multiple callers can be used in parallel. Ex: ['macs2', 'seacr'])

    peak_callers   = ['macs', 'seacr']  // Options: ['macs', 'seacr']

    //Trimmomatic Trim Settings

    //--Trimmomatic Adapter Path, Downloaded after "nextflow CnR-flow --mode initiate"
    trimmomatic_adapterpath = "${projectDir}/ref_dbs/trimmomatic_adapters/TruSeq3-PE-
↪2.fa"
    trimmomatic_settings    = "ILLUMINACLIP:${params.trimmomatic_adapterpath}/Truseq3.
↪PE.fa:2:15:4:4:true LEADING:20 TRAILING:20 SLIDINGWINDOW:4:15 MINLEN:25"
    trimmomatic_flags       = "-phred33"
    //kseq_test (CUTRUNTools) Retrim Settings
```

(continues on next page)

(continued from previous page)

```
    input_seq_len = 'auto'  // Length of untrimmed sequence reads (Ex: '150') or 'auto
↪'

    //Bowtie2 Alignment Settings
    aln_ref_flags  = "--local --very-sensitive-local --phred33 -I 10 -X 700 --
↪dovetail --no-unal --no-mixed --no-discordant"

    //Normalization Settings
    aln_norm_flags = params.aln_ref_flags
    norm_scale     = 1000  // Arbitrary value for scaling of normalized counts.
    norm_mode      = 'adj' // Options: ['adj', 'all']

    //Macs2 Settings
    macs_qval  = '0.01'
    macs_flags = ''

    //SEACR Settings
    seacr_fdr_threshhold = "0.01"
    seacr_norm_mode      = "auto" // Options: "auto", "norm", "non"
    seacr_call_stringent = true
    seacr_call_relaxed   = true

    // ------- General Pipeline Output Paramaters --------
    publish_files    = 'default' // Options: ["minimal", "default", "all"]
    publish_mode     = 'copy'    // Options: ["symlink", "copy"]

    //Name trim guide: ( regex-based )
    //   ~/groovy-slashy-string/  ;  "~" denotes groovy pattern type.
    //   ~/^/ matches beginning   ;   ~/$/ matches end
    trim_name_prefix = ''        // Example: ~/^myprefix./ removes "myprefix." prefix.
    trim_name_suffix = ''        // Example: ~/_mysuffix$/ removes "_mysuffix" suffix.

    // Workflow Output Default Naming Scheme:
    //   Absolute paths for output:
    out_dir          = "${launchDir}/cnr_output"
    refs_dir         = "${launchDir}/cnr_references"
    //   Subdirectory Settigns:
    log_dir          = 'logs'
    merge_fastqs_dir = 'S0_B_merged_reads'
    fastqc_pre_dir   = 'S0_C_FastQC_pre'
    trim_dir         = 'S1_A_fastq_trimomatic'
    retrim_dir       = 'S1_B_fastq_kseqtest'
    fastqc_post_dir  = 'S1_C_FastQC_post'
    aln_dir_ref      = 'S2_A_aln_ref'
    aln_dir_mod      = 'S2_B_aln_mod'
    aln_dir_bdg      = 'S2_C_aln_bdg'
    aln_dir_spike    = 'S3_A_aln_spikein'
    aln_dir_norm     = 'S3_B_aln_norm'
    aln_bigwig_dir   = 'S4_A_aln_bigWig'
    peaks_dir_macs   = 'S5_A_peaks_macs'
    peaks_dir_seacr  = 'S5_B_peaks_seacr'
    prep_bt2db_suf   = 'bt2_db'

}

// Location to store conda environments for reuse.
conda.cacheDir = "${projectDir}/conda_envs/"
```

(continues on next page)

(continued from previous page)

```
// ------- Individual Task Settings, Included for Completeness --------

    //params {
    //REQUIRED values to enter (all others should work as default):
    // ref_fasta               (or some other ref-mode/location)
    // treat_fastqs            (input paired-end fastq[.gz] file paths)
    //   [OR fastq_groups]     (mutli-group input paired-end .fastq[.gz] file paths)

    // Automatic alignment reference preparation/usage settings:
    //ref_mode       = 'fasta'  // Options: ['name', 'fasta', 'manual']
    //ref_fasta      = ''       // REQUIRED: Ex: '/path/to/my/reference.fasta[.gz]'
    // Default Pre-Supplied Normalization library is Ecoli:
    //norm_ref_fasta = "${projectDir}/ref_dbs/gcf_000005845.2_asm584v2_genomic.fasta.
↪gz"

    // CnR-flow Input Files:
    //   Provided fastqs must be in glob pattern matching pairs.
    //     Example: ['./relpath/to/base*R{1,2}*.fastq']
    //     Example: ['/abs/path/to/other*R{1,2}*.fastq']
    //
    //treat_fastqs = []  // REQUIRED, Single-group Treatment fastq Pattern
    //ctrl_fastqs  = []  //           Single-group Control   fastq pattern

    // Can specify multiple treat/control groups as Groovy mapping.
    //   Specified INSTEAD of treat_fasts/ctrl_fastqs paramaters.
    //   Note: There should be only one ctrl sample per group
    //     (after optional lane combination)
    //Example:
    //fastq_groups = [
    //  'group_1_name': ['treat': 'relpath/to/treat1*R{1,2}*',
    //                   'ctrl':  'relpath/to/ctrl1*R{1,2}*',
    //                  ]
    //  'group_2_name': ['treat': ['relpath/to/g2_treat1*R{1,2}*'
    //                             '/abs/path/to/g2_treat2*R{1,2}*'
    //                            ],
    //                   'ctrl':  'relpath/to/g2_ctrl1*R{1,2}*'
    //                  ]
    //]
    //fastq_groups = []
```

1.5 References

1.6 About

1.6.1 Renne Lab

Principal Investigator: Rolf Renne
Henry E. Innes Professor of Cancer Research
University of Florida
UF Health Cancer Center
UF Department of Molecular Genetics and Microbiology
UF Genetics Institute
http://www.rennelab.com

1.6.2 Lead Developer

- Dan Stribling <ds@ufl.edu>
  https://github.com/dstrib
  https://www.linkedin.com/in/DanielStribling
  University of Florida, Renne Lab

1.6.3 Changelog

v0.10:

Refinements
Changed verbose task logging to implementation with "beforeScript"
Complete Initial Documentation
Moved macs2 peak-calling out of alpha testing
"Reordered" output step directories
Tuned resource usage defaults
Added process memory usage categories
Move UCSC/Kent tools to external dependency setup
Added bigWig track format creation step
Overhauled alignment modification step
Removed Picard dependency
Changed (non-track) alignment output files to CRAM (compressed BAM)

v0.09:

Refinements
Added one-step database preparation
Implemented 'list_refs' mode
Implemented automatic reference paramater finding
Shifted paramaters to config files
Implemented initiate mode
Added minimal documentation
Added Kent's-Util (faCount) automated installation
Added automated acquisition of Trimmomatic adapters
Implemented MACS2 peak calling

Added autodetection of tag sequence length

v0.08

Initial Github Upload

BIBLIOGRAPHY

[Meers2019] Meers, Michael P., et al. "Improved CUT&RUN chromatin profiling tools." Elife 8 (2019): e46314.

[Nextflow_Citation] Di Tommaso, Paolo, et al. "Nextflow enables reproducible computational workflows." Nature biotechnology 35.4 (2017): 316-319.

[Bioconda_Citation] Grüning, Björn, et al. "Bioconda: sustainable and comprehensive software distribution for the life sciences." Nature methods 15.7 (2018): 475-476.

[CUTRUNTools_Citation] Zhu, Qian, et al. "CUT&RUNTools: a flexible pipeline for CUT&RUN processing and footprint analysis." Genome biology 20.1 (2019): 192.

[SEACR_Citation] Meers, Michael P., Dan Tenenbaum, and Steven Henikoff. "Peak calling by Sparse Enrichment Analysis for CUT&RUN chromatin profiling." Epigenetics & chromatin 12.1 (2019): 42.

[R_Citation] R Core Team (2017). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/

[Bowtie2_Citation] Langmead, Ben, and Steven L. Salzberg. "Fast gapped-read alignment with Bowtie 2." Nature methods 9.4 (2012): 357.

[faCount_Citation] Kent WJ, Sugnet CW, Furey TS, Roskin KM, Pringle TH, Zahler AM, Haussler D. The human genome browser at UCSC. Genome Res. 2002 Jun;12(6):996-1006.

[Samtools_Citation] Li H.*, Handsaker B.*, Wysoker A., Fennell T., Ruan J., Homer N., Marth G., Abecasis G., Durbin R. and 1000 Genome Project Data Processing Subgroup (2009) The Sequence alignment/map (SAM) format and SAMtools. Bioinformatics, 25, 2078-9. [PMID: 19505943]

[FastQC_Citation] Andrews, S. (2010). FastQC: A Quality Control Tool for High Throughput Sequence Data [Online]. Available online at: http://www.bioinformatics.babraham.ac.uk/projects/fastqc/

[Trimmomatic_Citation] Bolger, A. M., Lohse, M., & Usadel, B. (2014). Trimmomatic: A flexible trimmer for Illumina Sequence Data. Bioinformatics, btu170.

[bedtools_Citation] Quinlan AR and Hall IM, 2010. BEDTools: a flexible suite of utilities for comparing genomic features. Bioinformatics. 26, 6, pp. 841–842.

[bedGraphToBigWig_Citation] BigWig and BigBed tools: Kent WJ, Zweig AS, Barber G, Hinrichs AS, Karolchik D. BigWig and BigBed: enabling browsing of large distributed data sets. Bioinformatics. 2010 Sep 1;26(17):2204-7.

[MACS2_Citation] Zhang et al. Model-based Analysis of ChIP-Seq (MACS). Genome Biol (2008) vol. 9 (9) pp. R137