1. sort TSS regions

\*\* ajh

pipeline.add\_job(MakeSortedBed(input\_tss\_file, tss\_file))

tss\_file = os.path.join(PRECHECK\_PATH, 'tss\_file.sorted.bed.gz')

\*\* NextFlow

process: sortTssBedProcess

in channels/files

each tssBedMap from sortTssBedInChannel

out channels/files

file("precheck/${tssBedMap['outBed']}") into sortTssBedOutChannel

2. sort chromosome size file

\*\* ajh

pipeline.add\_job(MakeSortedBed(input\_chromosome\_sizes\_file, chromosome\_sizes\_file))

input\_chromosome\_sizes\_file = GENOME\_FILES[args.genome]['chromosome\_sizes']

chromosome\_sizes\_file = os.path.join(PRECHECK\_PATH, 'chromosome\_sizes.sorted.txt')

\*\* NextFlow

process: sortChromosomeSizeProcess

in channels/files

each chromosomeSizeMap from sortChromosomeSizeInChannel

out channels/files

file("${chromosomeSizeMap['outTxt']}") into sortChromosomeSizeOutChannel

3. align reads to genome

\*\* ajh

pipeline.add\_job(AlignReads(read1, read2, genome\_index, bam, whitelist\_regions, args.nthreads))

read1 = os.path.join(run\_directory, *'final\_outs'*, *'Undetermined\_S0\_L*%s*\_R1\_001.*%s*.trimmed.fastq.gz'* % (str(lane).zfill(3), sample))

read2 = os.path.join(run\_directory, *'final\_outs'*, *'Undetermined\_S0\_L*%s*\_R2\_001.*%s*.trimmed.fastq.gz'* % (str(lane).zfill(3), sample))

genome\_index = GENOME\_FILES[args.genome][*'bowtie\_index'*]

(bam)

whitelist\_regions = GENOME\_FILES[args.genome][*'whitelist\_regions'*]

\*\* NextFlow

process: runAlignProcess

in channels/files

each alignMap from runAlignInChannel

out channels/files

file("\*.bam") into runAlignOutChannel

4. MERGE BAMS

\*\* ajh

pipeline.add\_job(MergeBams(input\_bams, merged\_bam[i]))

(bams)

merged\_bam = [os.path.join(MERGE\_BAMS\_PATH, *'*%s*.q10.sorted.merged.bam'* % sample) for sample in all\_samples]

\*\* NextFlow

process: mergeBamsProcess

in channels/files

set file(inBams), outBam from mergeBamsInChannel

out channels/files

file("\*-merged\_bam.bam") into mergeBamsOutChannel (\*-merged\_bam.bam.bai)

5. DEDUP MERGED BAM

\*\* ajh

pipeline.add\_job(GetUniqueFragments(merged\_bam[i], fragments\_file[i], transposition\_sites\_file[i], duplicate\_report[i], insert\_size\_distributions[i]))

merged\_bam = [os.path.join(MERGE\_BAMS\_PATH, *'*%s*.q10.sorted.merged.bam'* % sample) for sample in all\_samples]

fragments\_file = [os.path.join(GET\_UNIQUE\_FRAGMENTS\_PATH, '%s.fragments.txt.gz' % sample) for sample in all\_samples]

transposition\_sites\_file = [os.path.join(GET\_UNIQUE\_FRAGMENTS\_PATH, '%s.bed.gz') % x for x in all\_samples]

duplicate\_report = [os.path.join(GET\_UNIQUE\_FRAGMENTS\_PATH, '%s.duplicate\_report.txt' % sample) for sample in all\_samples]

insert\_size\_distributions = [os.path.join(GET\_UNIQUE\_FRAGMENTS\_PATH, '%s.insert\_sizes.txt') % x for x in all\_samples]

\*\* NextFlow

process: getUniqueFragmentsProcess

in channels/files

set file(inBam), uniqueFragmentsMap from getUniqueFragmentsInChannel

out channels/files

file("\*-transposition\_sites.bed.gz\*") into getUniqueFragmentsOutChannelTranspositionSites // get both -transposition\_sites.bed.gz and -transposition\_sites.bed.gz.tbi files

file("\*-fragments.txt.gz\*") into getUniqueFragmentOutChannelFragments // get both -fragments.txt.gz and -fragments.txt.gz.tbi files

file("\*-insert\_sizes.txt") into getUniqueFragmentsOutChannelInsertSizeDistribution

file("\*-duplicate\_report.txt") into getUniqueFragmentsOutChannelDuplicateReport

6. CALL PEAKS

\*\* ajh

class CallPeaks:

def \_\_init\_\_(*self*, bed, genome, sample\_name, outdir):

pipeline.add\_job(CallPeaks(transposition\_sites\_file[i], macs\_genome, all\_samples[i], PEAK\_CALLS\_PATH))

transposition\_sites\_file = [os.path.join(GET\_UNIQUE\_FRAGMENTS\_PATH, '%s.bed.gz') % x for x in all\_samples]

macs\_genome = GENOME\_FILES[args.genome]['macs\_genome']

PEAK\_CALLS\_PATH = os.path.join(args.outdir, *'call\_peaks'*)

\*\* NextFlow

process: callPeaksProcess

in channels/files

set file(inBed), file(inTbi), callPeaksMap from callPeaksInChannel

out channels/files

file("\*-peaks.narrowPeak.gz") into callPeaksOutChannel

7 . CHOOSE PEAKS TO USE

if args.exclude\_sample\_peaks is not None:

peaks\_to\_merge = []

samples\_to\_exclude = set(args.exclude\_sample\_peaks)

for i, sample in enumerate(samples):

if sample not in samples\_to\_exclude:

peaks\_to\_merge.append(peak\_call\_files[i])

else:

peaks\_to\_merge = peak\_call\_files

peak\_call\_files = [os.path.join(PEAK\_CALLS\_PATH, *'*%s*\_peaks.narrowPeak.gz'*) % sample for sample in all\_samples]

PEAK\_CALLS\_PATH = os.path.join(args.outdir, *'call\_peaks'*)

\*\* NextFlow

process: Note: this is part of function makePeakFileChannelSetup()

in channels/files

out channels/files

mergePeaksInChannel

8. MERGE PEAKS

\*\* ajh

pipeline.add\_job(MergePeaks(peaks\_to\_merge, merged\_peaks))

peaks\_to\_merge = peak\_call\_files

peak\_call\_files = [os.path.join(PEAK\_CALLS\_PATH, *'*%s*\_peaks.narrowPeak.gz'*) % sample for sample in all\_samples]

merged\_peaks = os.path.join(PEAK\_CALLS\_PATH, *'merged\_peaks.bed'*)

\*\* NextFlow

process: mergePeaksProcess

in channels/files

set file(inBed), mergePeaksMap from mergePeaksInChannel

out channels/files

file("\*-merged\_peaks.bed") into mergePeaksOutChannel

9. MAKE WINDOWED GENOME INTERVALS

\*\* ajh

pipeline.add\_job(MakeWindowedGenomeIntervals(chromosome\_sizes\_file, genomic\_windows\_file))

chromosome\_sizes\_file = os.path.join(PRECHECK\_PATH, 'chromosome\_sizes.sorted.txt')

genomic\_windows\_file = os.path.join(MAKE\_MATRICES\_PATH, *'genomic\_windows.bed'*)

\*\* NextFlow

process: makeWindowedGenomeIntervalsProcess

in channels/files

makeWindowedGenomeIntervalsProcess

out channels/files

file("\*-genomic\_windows.bed") into makeWindowedGenomeIntervalsOutChannel

10. MAKE GENE REGIONS

\*\* ajh

if not 'gene\_score\_bed' in GENOME\_FILES[args.genome]:

print('No gene\_score\_bed entry provided for genome... using nearby peaks.')

pipeline.add\_job(MakePromoterSumIntervals(tss\_file, merged\_peaks, peak\_promoter\_assignments))

else:

gene\_score\_bed = os.path.join(PRECHECK\_PATH, 'gene\_score\_bed.sorted.bed.gz')

pipeline.add\_job(MakeSortedBed(GENOME\_FILES[args.genome]['gene\_score\_bed'], gene\_score\_bed))

tss\_file = os.path.join(PRECHECK\_PATH, *'tss\_file.sorted.bed.gz'*)

merged\_peaks = os.path.join(PEAK\_CALLS\_PATH, *'merged\_peaks.bed'*)

peak\_promoter\_assignments = os.path.join(MAKE\_MATRICES\_PATH, 'peak\_promoter\_assignments.bed')

GENOME\_FILES[args.genome]['gene\_score\_bed']

gene\_score\_bed = os.path.join(PRECHECK\_PATH, *'gene\_score\_bed.sorted.bed.gz'*)

\*\* NextFlow

process: makePromoterSumIntervalsProcess

in channels/files

set file( inPath ), inMap from makePromoterSumIntervalsInChannel

out channels/files

file( "\*-gene\_regions.bed.gz" ) into makePromoterSumIntervalsOutChannel

11. GET COUNTS IN GENE REGIONS

\*\* ajh

pipeline.add\_job(GetRegionCounts(transposition\_sites\_file[i], merged\_peaks, peak\_counts[i], chromosome\_sizes\_file))

transposition\_sites\_file = [os.path.join(GET\_UNIQUE\_FRAGMENTS\_PATH, '%s.bed.gz') % x for x in all\_samples]

merged\_peaks = os.path.join(PEAK\_CALLS\_PATH, *'merged\_peaks.bed'*)

peak\_counts = [os.path.join(COUNT\_REPORT\_PATH, *'*%s*.peak\_counts.txt'* % sample) for sample in all\_samples]

chromosome\_sizes\_file = os.path.join(PRECHECK\_PATH, 'chromosome\_sizes.sorted.txt')

\*\* NextFlow

process: makeMergedPeakRegionCountsProcess

in channels/files

ajh file: transposition\_sites\_file

src process: getUniqueFragmentsProcess

out channel: file("\*-transposition\_sites.bed.gz\*") into getUniqueFragmentsOutChannelTranspositionSites // get both -transposition\_sites.bed.gz and -transposition\_sites.bed.gz.tbi files

Note: use getUniqueFragmentsOutChannelTranspositionSitesCopy02

ajh file: merged\_peaks

src process: mergePeaksProcess

out channel: file("\*-merged\_peaks.bed") into mergePeaksOutChannel

ajh file: chromosome\_sizes\_file

src process: sortChromosomeSizeProcess

out channel: file("precheck/${chromosomeSizeMap['outTxt']}") into sortChromosomeSizeOutChannel

out channels/files

ajh file: peak\_counts

src process: makeGeneRegionCountsProcess

out channel: file( "\*-peak\_counts.txt" ) into makeMergedPeakRegionCountsOutChannel

12. GET COUNTS in TSS REGIONS

\*\* ajh

pipeline.add\_job(GetRegionCounts(transposition\_sites\_file[i], tss\_file, tss\_counts[i], chromosome\_sizes\_file, flanking\_distance=1000))

transposition\_sites\_file = [os.path.join(GET\_UNIQUE\_FRAGMENTS\_PATH, '%s.bed.gz') % x for x in all\_samples]

tss\_file = os.path.join(PRECHECK\_PATH, *'tss\_file.sorted.bed.gz'*)

tss\_counts = [os.path.join(COUNT\_REPORT\_PATH, *'*%s*.tss\_counts.txt'* % sample) for sample in all\_samples]

chromosome\_sizes\_file = os.path.join(PRECHECK\_PATH, 'chromosome\_sizes.sorted.txt')

\*\* NextFlow

process: process makeTssCountsProcess

in channels/files

set file( inTranspositionSites ), file( inTbiTranspositionSites ), inTranspositionSitesMap from makeTssRegionCountsInChannelTranspositionSites

set file( inTssRegions ), inTssRegionMap from makeTssRegionCountsInChannelTssRegions

set file( inChromosomeSizes ), inChromosomeSizesMap from makeTssRegionCountsInChannelChromosomeSizes

out channels/files

file( "\*-tss\_counts.txt" ) into makeTssRegionCountsOutChannel

13. MAKE COUNT REPORT

\*\* ajh

pipeline.add\_job(MakeCountReport(duplicate\_report[i], peak\_counts[i], tss\_counts[i], count\_report[i]))

duplicate\_report = [os.path.join(GET\_UNIQUE\_FRAGMENTS\_PATH, *'*%s*.duplicate\_report.txt'* % sample) for sample in all\_samples]

peak\_counts = [os.path.join(COUNT\_REPORT\_PATH, *'*%s*.peak\_counts.txt'* % sample) for sample in all\_samples]

tss\_counts = [os.path.join(COUNT\_REPORT\_PATH, *'*%s*.tss\_counts.txt'* % sample) for sample in all\_samples]

count\_report = [os.path.join(COUNT\_REPORT\_PATH, *'*%s*.count\_report.txt'* % sample) for sample in all\_samples]

\*\* NextFlow

process: makeCountReportsProcess

in channels/files

set file( inDuplicateReport ), inDuplicateReportMap from makeCountReportsInChannelDuplicateReport

set file( inMergedPeakRegionCounts ), inMergedPeakRegionCountsMap from makeCountReportsInChannelMergedPeakRegionCounts

set file( inTssRegionCounts ), inTssRegionCountsMap from makeCountReportsInChannelTssRegionCounts

out channels/files

file( "\*-count\_report.txt" ) into makeCountReportsOutChannel

14. CALL CELLS + MAKE SUMMARY PLOT

\*\* ajh

pipeline.add\_job(CallCells(count\_report[i], args.reads\_threshold, called\_cells\_counts[i], cell\_whitelist[i], call\_cells\_stats\_files[i]))

count\_report = [os.path.join(COUNT\_REPORT\_PATH, *'*%s*.count\_report.txt'* % sample) for sample in all\_samples]

called\_cells\_counts = [os.path.join(CALL\_CELLS\_PATH, *'*%s*.called\_cells.txt'* % sample) for sample in all\_samples]

cell\_whitelist = [os.path.join(CALL\_CELLS\_PATH, *'*%s*.cell\_whitelist.txt'* % sample) for sample in all\_samples]

call\_cells\_stats\_files = [os.path.join(CALL\_CELLS\_PATH, *'*%s*.stats.json'* % sample) for sample in all\_samples]

\*\* NextFlow

process: callCellsProcess

in channels/files

set file( inCountReport ), inCountReportMap from callCellsInChannel

out channels/files

file( "\*-called\_cells.txt" ) into callCellsOutChannelCalledCellsCounts

file( "\*-called\_cells\_whitelist.txt" ) into callCellsOutChannelCalledCellsWhitelist

file( "\*-called\_cells\_stats.json" ) into callCellsOutChannelCalledCellsStats

15. Per base coverage around TSS's

\*\* ajh

pipeline.add\_job(PerBaseTSSRegionCoverage(transposition\_sites\_file[i], tss\_file, per\_base\_tss\_region\_coverage\_files[i], chromosome\_sizes\_file))

transposition\_sites\_file = [os.path.join(GET\_UNIQUE\_FRAGMENTS\_PATH, '%s.bed.gz') % x for x in all\_samples]

tss\_file = os.path.join(PRECHECK\_PATH, *'tss\_file.sorted.bed.gz'*)

per\_base\_tss\_region\_coverage\_files = [os.path.join(PER\_BASE\_TSS\_REGION\_COVERAGE\_PATH, *'*%s*.tss\_region\_coverage.txt.gz'*) % x for x in all\_samples]

chromosome\_sizes\_file = os.path.join(PRECHECK\_PATH, 'chromosome\_sizes.sorted.txt')

\*\* NextFlow

process: getPerBaseCoverageTssProcess

in channels/files

set file( inTranspositionSites ), file( inTbiTranspositionSites ), inTranspositionSitesMap from getPerBaseCoverageTssInChannelTranspositionSites

set file( inTssRegions ), inTssRegionMap from getPerBaseCoverageTssInChannelTssRegions

set file( inChromosomeSizes ), inChromosomeSizesMap from getPerBaseCoverageTssInChannelChromosomeSizes

out channels/files

file( "\*-tss\_region\_coverage.txt.gz" ) into getPerBaseCoverageTssOutChannel

16. OPTIONAL BANDING SCORES QC

\*\* ajh

if args.calculate\_banding\_scores:

pipeline.add\_job(GetPerCellInsertSizes(fragments\_file[i], per\_cell\_insert\_sizes[i], cell\_whitelist[i]))

pipeline.add\_job(GetBandingScores(per\_cell\_insert\_sizes[i], banding\_scores[i], cell\_whitelist[i]))

fragments\_file = [os.path.join(GET\_UNIQUE\_FRAGMENTS\_PATH, *'*%s*.fragments.txt.gz'* % sample) for sample in all\_samples]

per\_cell\_insert\_sizes = [os.path.join(BANDING\_SCORES\_PATH, *'*%s*.per\_cell\_insert\_sizes.txt'*) % x for x in all\_samples]

cell\_whitelist = [os.path.join(CALL\_CELLS\_PATH, *'*%s*.cell\_whitelist.txt'* % sample) for sample in all\_samples]

banding\_scores = [os.path.join(BANDING\_SCORES\_PATH, *'*%s*.banding\_scores.txt'*) % x for x in all\_samples]

\*\* NextFlow

process:

in channels/files

out channels/files

17. PEAK MATRIX

\*\* ajh

pipeline.add\_job(MakeMatrices(transposition\_sites\_file[i], merged\_peaks, cell\_whitelist[i], peak\_matrices[i]))

transposition\_sites\_file = [os.path.join(GET\_UNIQUE\_FRAGMENTS\_PATH, '%s.bed.gz') % x for x in all\_samples]

merged\_peaks = os.path.join(PEAK\_CALLS\_PATH, *'merged\_peaks.bed'*)

cell\_whitelist = [os.path.join(CALL\_CELLS\_PATH, *'*%s*.cell\_whitelist.txt'* % sample) for sample in all\_samples]

peak\_matrices = [os.path.join(MAKE\_MATRICES\_PATH, *'*%s*.peak\_matrix.mtx.gz'*) % x for x in all\_samples]

\*\* NextFlow

process: makePeakMatrixProcess

in channels/files

set file( inTranspositionSites ), inTranspositionSitesMap from makePeakMatrixInChannelTranspositionSites

set file( inMergedPeaks ), inMergedPeaksMap from makePeakMatrixInChannelMergedPeaks

set file( inCellWhitelist ), inCellWhitelistMap from makePeakMatrixInChannelCellWhitelist

out channels/files

file( "\*-peak\_matrix.mtx.gz" ) into makePeakMatrixOutChannel

18. WINDOW MATRIX

\*\* ajh

pipeline.add\_job(MakeMatrices(transposition\_sites\_file[i], genomic\_windows\_file, cell\_whitelist[i], window\_matrices[i]))

transposition\_sites\_file = [os.path.join(GET\_UNIQUE\_FRAGMENTS\_PATH, '%s.bed.gz') % x for x in all\_samples]

genomic\_windows\_file = os.path.join(MAKE\_MATRICES\_PATH, *'genomic\_windows.bed'*)

cell\_whitelist = [os.path.join(CALL\_CELLS\_PATH, *'*%s*.cell\_whitelist.txt'* % sample) for sample in all\_samples]

window\_matrices = [os.path.join(MAKE\_MATRICES\_PATH, *'*%s*.window\_matrix.mtx.gz'*) % x for x in all\_samples]

\*\* NextFlow

process: makeWindowMatrixProcess

in channels/files

set file( inTranspositionSites ), file( inTbiTranspositionSites ), inTranspositionSitesMap from makeWindowMatrixInChannelTranspositionSites

set file( inWindowedIntervals ), inWindowedIntervalsMap from makeWindowMatrixInChannelWindowedGenomeIntervals

set file( inCellWhitelist ), inCellWhitelistMap from makeWindowMatrixInChannelCellWhitelist

out channels/files

file( "\*-window\_matrix.mtx.gz" ) into makeWindowMatrixOutChannel

19. PROMOTER MATRIX

\*\* ajh

if not 'gene\_score\_bed' in GENOME\_FILES[args.genome]:

pipeline.add\_job(MakeMatrices(transposition\_sites\_file[i], peak\_promoter\_assignments, cell\_whitelist[i], promoter\_matrices[i]))

else:

pipeline.add\_job(MakeMatrices(transposition\_sites\_file[i], gene\_score\_bed, cell\_whitelist[i], promoter\_matrices[i]))

transposition\_sites\_file = [os.path.join(GET\_UNIQUE\_FRAGMENTS\_PATH, '%s.bed.gz') % x for x in all\_samples]

peak\_promoter\_assignments = os.path.join(MAKE\_MATRICES\_PATH, 'peak\_promoter\_assignments.bed’

cell\_whitelist = [os.path.join(CALL\_CELLS\_PATH, *'*%s*.cell\_whitelist.txt'* % sample) for sample in all\_samples]

promoter\_matrices = [os.path.join(MAKE\_MATRICES\_PATH, *'*%s*.promoter\_matrix.mtx.gz'*) % x for x in all\_samples]

gene\_score\_bed = os.path.join(PRECHECK\_PATH, *'gene\_score\_bed.sorted.bed.gz'*)

\*\* NextFlow

process: makePromoterMatrixProcess

in channels/files

set file( inTranspositionSites ), file( inTbiTranspositionSites ), inTranspositionSitesMap from makeWindowMatrixInChannelTranspositionSites

set file( inGeneRegions ), inGeneRegionsMap from makePromoterMatrixInChannelGeneRegions

set file( inCellWhitelist ), inCellWhitelistMap from makeWindowMatrixInChannelCellWhitelist

out channels/files

file( "\*-promoter\_matrix.mtx.gz" ) into makePromoterMatrixOutChannel

20. MOTIF CALLING IN PEAKS + MOTIF MATRIX THAT IS PER PEAK SET NOT PER SAMPLE

\*\* ajh

motif\_calling\_outputs = []

run\_motif\_calling = 'motifs' in GENOME\_FILES[args.genome] and 'fasta' in GENOME\_FILES[args.genome]

if run\_motif\_calling:

for gc\_bin in range(0, MOTIF\_CALLING\_GC\_BINS):

motifs = GENOME\_FILES[args.genome]['motifs']

fasta = GENOME\_FILES[args.genome]['fasta']

# Note these are really just temp, so not a huge deal that they aren't declared up top

# with everything else

output\_file = peak\_motif\_files[gc\_bin]

motif\_calling\_outputs.append(output\_file)

pipeline.add\_job(PeakMotifs(fasta, merged\_peaks, motifs, output\_file, gc\_bin=gc\_bin))

pipeline.add\_job(MakeMotifMatrix(motif\_calling\_outputs,

GENOME\_FILES[args.genome]['fasta'],

merged\_peaks,

GENOME\_FILES[args.genome]['motifs'],

peak\_tf\_matrix))

fasta = GENOME\_FILES[args.genome][*'fasta'*]

merged\_peaks = os.path.join(PEAK\_CALLS\_PATH, *'merged\_peaks.bed'*)

motifs = GENOME\_FILES[args.genome][*'motifs'*]

output\_file = peak\_motif\_files[gc\_bin]

for gc\_bin in range(0, MOTIF\_CALLING\_GC\_BINS)

peak\_motif\_files = [os.path.join(PEAK\_MOTIFS\_PATH, *'gc\_binned.*%s*.peak\_calls.bb'* % gc\_bin) for gc\_bin in range(1, MOTIF\_CALLING\_GC\_BINS + 1)]

MOTIF\_CALLING\_GC\_BINS = 25

output\_file = peak\_motif\_files[gc\_bin]

motif\_calling\_outputs.append(output\_file)

peak\_motif\_files = [os.path.join(PEAK\_MOTIFS\_PATH, *'gc\_binned.*%s*.peak\_calls.bb'* % gc\_bin) for gc\_bin in range(1, MOTIF\_CALLING\_GC\_BINS + 1)]

peak\_tf\_matrix = os.path.join(MOTIF\_MATRICES\_PATH, *'peak\_motif\_matrix.mtx.gz'*)

\*\* NextFlow

process:

in channels/files

out channels/files

21. Produce summary plots and stats files on basic stats and duplication rates

\*\* ajh

pipeline.add\_job(SummarizeCellCalls(count\_report,

call\_cells\_stats\_files,

insert\_size\_distributions,

peak\_call\_files, merged\_peaks,

per\_base\_tss\_region\_coverage\_files,

call\_cells\_summary\_plot,

call\_cells\_summary\_stats,

window\_matrices,

barnyard))

count\_report = [os.path.join(COUNT\_REPORT\_PATH, *'*%s*.count\_report.txt'* % sample) for sample in all\_samples]

call\_cells\_stats\_files = [os.path.join(CALL\_CELLS\_PATH, *'*%s*.stats.json'* % sample) for sample in all\_samples]

insert\_size\_distributions = [os.path.join(GET\_UNIQUE\_FRAGMENTS\_PATH, *'*%s*.insert\_sizes.txt'*) % x for x in all\_samples]

peak\_call\_files = [os.path.join(PEAK\_CALLS\_PATH, *'*%s*\_peaks.narrowPeak.gz'*) % sample for sample in all\_samples]

merged\_peaks = os.path.join(PEAK\_CALLS\_PATH, *'merged\_peaks.bed'*)

per\_base\_tss\_region\_coverage\_files = [os.path.join(PER\_BASE\_TSS\_REGION\_COVERAGE\_PATH, *'*%s*.tss\_region\_coverage.txt.gz'*) % x for x in all\_samples]

call\_cells\_summary\_plot = os.path.join(SUMMARIZE\_CELL\_CALLS\_PATH, *'called\_cells\_summary.pdf'*)

call\_cells\_summary\_stats = os.path.join(SUMMARIZE\_CELL\_CALLS\_PATH, *'called\_cells\_summary.stats.txt'*)

window\_matrices = [os.path.join(MAKE\_MATRICES\_PATH, *'*%s*.window\_matrix.mtx.gz'*) % x for x in all\_samples]

barnyard = GENOME\_FILES[args.genome].get(*'barnyard'*, False)

\*\* NextFlow

process: summarizeCellCallsProcess

in channels/files

set file( inCountReports ), inCountReportsMap from summarizeCellCallsInChannelCountReports

set file( inCalledCellStats ), inCalledCellStatsMap from summarizeCellCallsInChannelCalledCellStats

set file( inInsertSizes ), inInsertSizesMap from summarizeCellCallsInChannelInsertSizeDistribution

set file( inNarrowPeaks ), inNarrowPeaksMap from summarizeCellCallsInChannelNarrowPeak

set file( inNarrowPeaks ), inMergedPeaksMap from summarizeCellCallsInChannelMergedPeaks

set file( inPerBaseCoverageTss ), inPerBaseCoverageTss from summarizeCellCallsInChannelPerBaseCoverageTss

set file( inWindowMatrix ), inWindowMatrixMap from summarizeCellCallsInChannelWindowMatrix

val inBarnyardMap from summarizeCellCallsInChannelTestBarnyard

out channels/files

file( "\*-called\_cells\_summary.pdf" ) into summarizeCellCallsOutChannelCallCellsSummaryPlot

file( "\*-called\_cells\_summary.stats.txt" ) into summarizeCellCallsOutChannelCallCellsSummaryStats

def outSummaryPlot = aSample + '-called\_cells\_summary.pdf'

def outSummaryStats = aSample + '-called\_cells\_summary.stats.txt'

22. REDUCE DIMENSION WITH LSI

\*\* ajh

pipeline.add\_job(ReduceDimension(peak\_matrices[i],

promoter\_matrices[i],

svd\_coords[i],

umap\_coords[i],

tsne\_coords[i],

tfidf\_matrices[i],

seurat\_objects[i]))

peak\_matrices = [os.path.join(MAKE\_MATRICES\_PATH, *'*%s*.peak\_matrix.mtx.gz'*) % x for x in all\_samples]

promoter\_matrices = [os.path.join(MAKE\_MATRICES\_PATH, *'*%s*.promoter\_matrix.mtx.gz'*) % x for x in all\_samples]

svd\_coords = [os.path.join(REDUCE\_DIMENSION\_PATH, *'*%s*.svd\_coords.txt'* % x) for x in all\_samples]

umap\_coords = [os.path.join(REDUCE\_DIMENSION\_PATH, *'*%s*.umap\_coords.txt'* % x) for x in all\_samples]

tsne\_coords = [os.path.join(REDUCE\_DIMENSION\_PATH, *'*%s*.tsne\_coords.txt'* % x) for x in all\_samples]

tfidf\_matrices = [os.path.join(REDUCE\_DIMENSION\_PATH, *'*%s*.tfidf\_matrix.mtx.gz'* seurat\_objects = [os.path.join(REDUCE\_DIMENSION\_PATH, *'*%s*.seurat\_object.rds'* % x) for x in all\_samples]

% x) for x in all\_samples]

\*\* NextFlow

process:

in channels/files

out channels/files

23. CISTOPIC MODELS IF REQUESTED (TIME CONSUMING)

\*\* ajh

pipeline.add\_job(CisTopicModels(peak\_matrices[i], topic\_models[i], args.topics))

peak\_matrices = [os.path.join(MAKE\_MATRICES\_PATH, *'*%s*.peak\_matrix.mtx.gz'*) % x for x in all\_samples]

topic\_models = [os.path.join(TOPIC\_MODELS\_PATH, *'*%s*.topic\_model.rds'*) % x for x in all\_samples]

\*\* NextFlow

process:

in channels/files

out channels/files