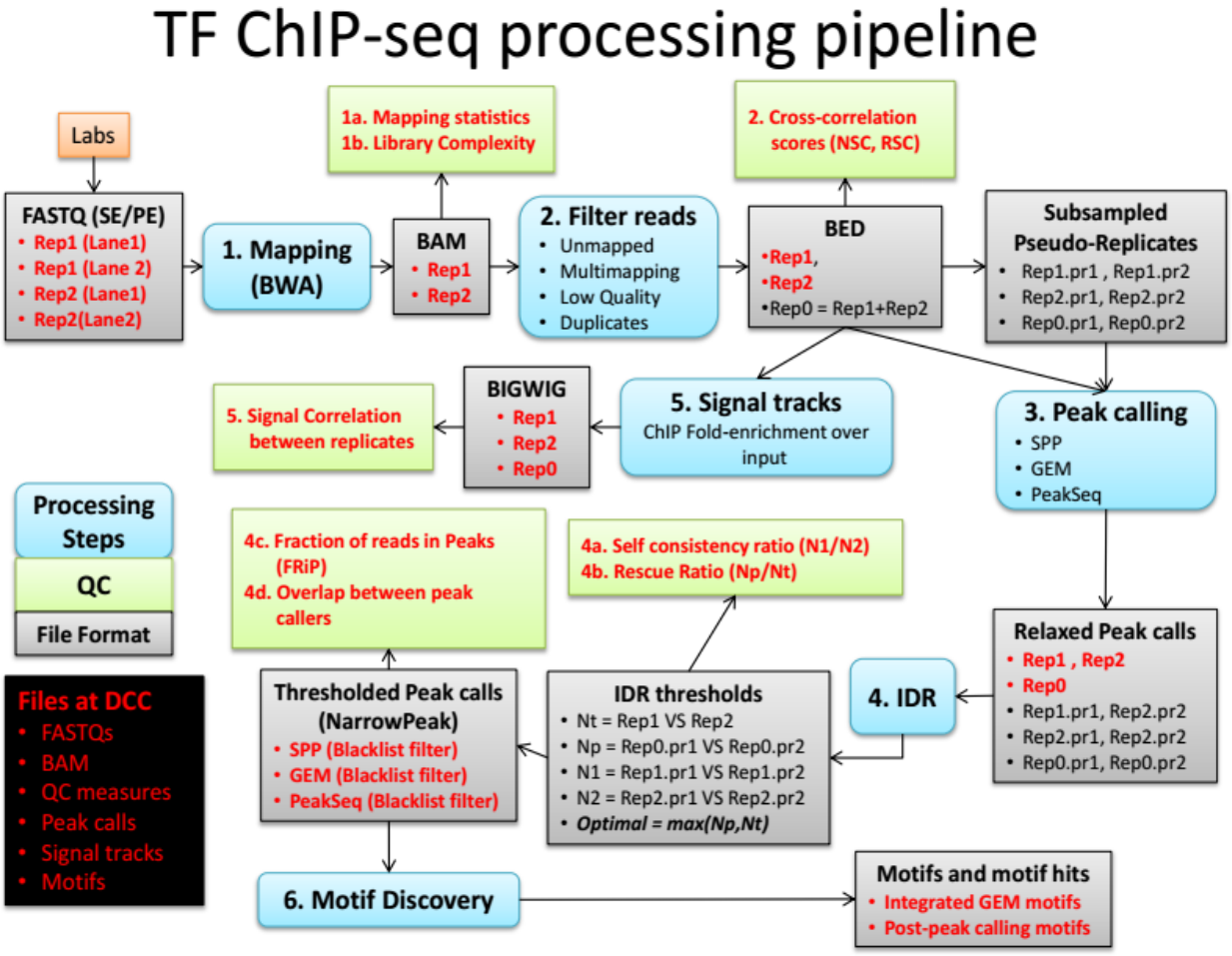
ENCODE3 pipeline v1 specifications

Pipeline Overview



0. FASTQ read quality filtering

Optional, was not specified

- FastQC can show the distribution of sequence quality scores and to help set an appropriate cutoff: http://www.bioinformatics.babraham.ac.uk/projects/fastqc/Help/3%20Analysis%20Modules/3%20Per%20Sequence%20Quality%20Scores.html)

1a. Read alignment (bowtie2 aligner)

Single-End ChIP-seq parameters

■

Program(s)	• Bowtie2 version 2.2.6 • SAMtools 1.7 • SAMstats (version 0.2.1)
Input(s)	• Input: $fastq • Bowtie2 index: $bwt2_idx
Output(s)	• BAM file $bam • mapping stats from flagstat (SAMstats) $flagstat_qc
Commands	bam = "$prefix.bam" log = "$prefix.align.log" flagstat_qc = “$prefix.flagstat.qc” bowtie2 --mm -x $bwt2_idx --threads $nth_bwt2 -U <(zcat -f $fastq) 2> $log | \ samtools view -Su /dev/stdin | samtools sort - $prefix samtools sort -n --threads 10 ${bam} -O SAM | SAMstats --sorted_sam_file - --outf ${flagstat_qc}
QC to report	Output from last command i.e. samtools flagstat
Status	Frozen

Paired-End ChIP-seq parameters

■

Program(s)	• Bowtie2 version 2.2.6 • SAMtools 1.7 • SAMstats (version 0.2.1)
Input(s)	• Input: $fastq1, $fastq2 • Bowtie2 index: $bwt2_idx
Output(s)	• BAM file $bam • mapping stats from flagstat (SAMstats) $flagstat_qc
Commands	bam = "$prefix.bam" log = "$prefix.align.log" flagstat_qc = "$prefix.flagstat.qc" bowtie2 -X2000 --mm --threads $nth_bwt2 -x $bwt2_idx \ -1 $fastq1 -2 $fastq2 \ 2>$log | \ samtools view -Su /dev/stdin | samtools sort - $prefix samtools sort -n --threads 10 ${bam} -O SAM | SAMstats --sorted_sam_file - --outf ${flagstat_qc}

QC to report	Output from last command i.e. samtools flagstat
Comment	
Status	Frozen

1a. Read alignment (BWA aligner)

Single-End ChIP-seq parameters

- With dynamic read trimming q = 5
- seed length l = 32
- max. mismatches in seed k = 2

Program(s)	• BWA version 0.7.13 • SAMtools (version 1.7) • SAMstats (version 0.2.1)
Input(s)	• FASTQ file ${FASTQ_FILE_1} • hg19 male or female or mm9 reference sequence ${BWA_INDEX_NAME}
Output(s)	• BAM file ${RAW_BAM_FILE} • mapping stats from flagstat (SAMstats) ${RAW_BAM_FILE_MAPSTATS}
Commands	# ========== # Map reads to create raw SAM file # ========== SAI_FILE_1="${OFPREFIX}.sai" RAW_BAM_PREFIX="${OFPREFIX}.raw.srt" RAW_BAM_FILE="${RAW_BAM_PREFIX}.bam" # To be stored RAW_BAM_FILE_MAPSTATS="${RAW_BAM_PREFIX}.flagstat.qc" # QC File module add bwa/0.7.13 module add samtools/1.7 bwa aln -q 5 -l 32 -k 2 -t ${NTHREADS} ${BWA_INDEX_NAME} ${FASTQ_FILE_1} > ${SAI_FILE_1} bwa samse ${BWA_INDEX_NAME} ${SAI_FILE_1} ${FASTQ_FILE_1} | samtools view -Su - | samtools sort -o ${RAW_BAM_FILE} - rm ${SAI_FILE_1} # Use bwa-mem for reads >= 70 bp # bwa mem -M -t ${NTHREADS} ${BWA_INDEX_NAME} ${FASTQ_FILE_1} | samtools sort -o ${RAW_BAM_FILE} - samtools sort -n --threads 10 ${RAW_BAM_FILE} -O SAM | SAMstats --sorted_sam_file - --outf ${RAW_BAM_FILE_MAPSTATS}
QC to report	Output from last command i.e. samtools flagstat

~~Status~~	~~Frozen~~

Paired-End ChIP-seq parameters

- With dynamic read trimming q = 5
- seed length l = 32
- max. mismatches in seed k = 2

Program(s)	• BWA version 0.7.10 • SAMtools (version 1.7) • SAMstats (version 0.2.1)
Input(s)	• FASTQ file 1 ${FASTQ_FILE_1} • FASTQ file 2 ${FASTQ_FILE_2} • hg19 male or female or mm9 indexes ${BWA_INDEX_NAME}
Output(s)	• Raw BAM file ${RAW_BAM_FILE} • mapping stats from flagstat (SAMstats) ${RAW_BAM_FILE_MAPSTATS}
Commands	SAI_FILE_1="${OFPREFIX}_1.sai" SAI_FILE_2="${OFPREFIX}_2.sai" RAW_SAM_FILE="${OFPREFIX}.raw.sam.gz" bwa aln -q 5 -l 32 -k 2 -t ${NTHREADS} ${BWA_INDEX_NAME} ${FASTQ_FILE_1} > ${SAI_FILE_1} bwa aln -q 5 -l 32 -k 2 -t ${NTHREADS} ${BWA_INDEX_NAME} ${FASTQ_FILE_2} > ${SAI_FILE_2} bwa sampe ${BWA_INDEX_NAME} ${SAI_FILE_1} ${SAI_FILE_2} ${FASTQ_FILE_1} ${FASTQ_FILE_2} | gzip -nc > ${RAW_SAM_FILE} rm ${SAI_FILE_1} ${SAI_FILE_2} # Use bwa-mem for reads >= 70 bp # bwa mem -M -t ${NTHREADS} ${BWA_INDEX_NAME} ${FASTQ_FILE_1} ${FASTQ_FILE_2} | gzip -nc > ${RAW_SAM_FILE} # ========== # Remove read pairs with bad CIGAR strings and sort by position # ========== RAW_BAM_PREFIX="${OFPREFIX}.raw.srt" RAW_BAM_FILE="${RAW_BAM_PREFIX}.bam" # To be stored BADCIGAR_FILE="${TMP}/badReads${RANDOM}.tmp" RAW_BAM_FILE_MAPSTATS="${RAW_BAM_PREFIX}.flagstat.qc" # QC File # Find bad CIGAR read names zcat ${RAW_SAM_FILE} | awk 'BEGIN {FS="\t" ; OFS="\t"} ! /^@/ && $6!="*" { cigar=$6; gsub("[0-9]+D","",cigar); n = split(cigar,vals,"[A-Z]"); s = 0; for (i=1;i<=n;i++) s=s+vals[i]; seqlen=length($10) ; if (s!=seqlen) print $1"\t" ; }' | sort | uniq > ${BADCIGAR_FILE} # Remove bad CIGAR read pairs if [[$(cat ${BADCIGAR_FILE} | wc -l) -gt 0]] then zcat ${RAW_SAM_FILE} | grep -v -F -f ${BADCIGAR_FILE} | samtools view -Su - | samtools sort - ${RAW_BAM_PREFIX} else samtools view -Su ${RAW_SAM_FILE} | samtools sort - ${RAW_BAM_PREFIX} fi rm ${BADCIGAR_FILE} ${RAW_SAM_FILE}

	~~samtools sort -n --threads 10 ${RAW_BAM_FILE} -O SAM | SAMstats --sorted_sam_file - --outf ${RAW_BAM_FILE_MAPSTATS}~~ ~~samtools flagstat ${RAW_BAM_FILE} > ${RAW_BAM_FILE_MAPSTATS}~~
~~QC to report~~	~~Output from last command i.e. samtools flagstat~~
~~Comment~~	~~We don't directly convert to BAM since BWA has some weird bugs where it sometimes generated Clipped CIGAR strings which are not compatible with samtools~~
~~Status~~	~~Frozen~~

1b. Post-alignment filtering

Single-End ChIP-seq parameters

- Remove reads unmapped, not primary alignment, reads failing platform, duplicates (-F 1796 or -F 1804 to keep it the same as PE)
- Remove multi-mapped reads (i.e. those with MAPQ < 30, using -q in SAMtools)
  - http://samtools.sourceforge.net/
  - Remove PCR duplicates (using Picard's MarkDuplicates or FixSeq)
  - PICARD: http://picard.sourceforge.net/command-line-overview.shtml#MarkDuplicates
  - FixSeq: https://bitbucket.org/thashim/fixseq (To be added at a later date)

Program(s)	• SAMtools (version 1.7) • MarkDuplicates (Picard - version 1.126) • bedtools 2.26.0
Input(s)	• Raw BAM file ${RAW_BAM_FILE}
Output(s)	• Filtered deduped position sorted BAM and index file ${FINAL_BAM_FILE} ${FINAL_BAM_INDEX_FILE} • Flagstat Metric for filtered BAM file ${FINAL_BAM_FILE_MAPSTATS} • Duplication metrics from MarkDuplicates ${DUP_FILE_QC} • Library complexity measures ${PBC_FILE_QC} • Subsampled tagAlign file for CC analysis ${SUBSAMPLED_TA_FILE}
Commands	# ============================ # Remove unmapped, mate unmapped # not primary alignment, reads failing platform # Remove low MAPQ reads # ================== FILT_BAM_PREFIX="${OFPREFIX}.filt.srt" FILT_BAM_FILE="${FILT_BAM_PREFIX}.bam" MAPQ_THRESH=30 samtools view -F 1804 -q ${MAPQ_THRESH} -b ${RAW_BAM_FILE} -o ${FILT_BAM_FILE} # ======================== # Mark duplicates # ====================== module add picard-tools/1.126 TMP_FILT_BAM_FILE="${FILT_BAM_PREFIX}.dupmark.bam" MARKDUP="/srv/gs1/software/picard-tools/1.126/picard MarkDuplicates" DUP_FILE_QC="${FILT_BAM_PREFIX}.dup.qc" # QC file java -Xmx4G -jar ${MARKDUP} INPUT=${FILT_BAM_FILE} OUTPUT=${TMP_FILT_BAM_FILE} METRICS_FILE=${DUP_FILE_QC} VALIDATION_STRINGENCY=LENIENT ASSUME_SORTED=true REMOVE_DUPLICATES=false mv ${TMP_FILT_BAM_FILE} ${FILT_BAM_FILE} # ============================ # Remove duplicates # Index final position sorted BAM

	# ============================ FINAL_BAM_PREFIX="${OFPREFIX}.filt.nodup.srt" FINAL_BAM_FILE="${FINAL_BAM_PREFIX}.bam" # To be stored FINAL_BAM_INDEX_FILE="${FINAL_BAM_PREFIX}.bai" # To be stored FINAL_BAM_FILE_MAPSTATS="${FINAL_BAM_PREFIX}.flagstat.qc" # QC file samtools view -F 1804 -b ${FILT_BAM_FILE} -o ${FINAL_BAM_FILE} # Index Final BAM file samtools index ${FINAL_BAM_FILE} ${FINAL_BAM_INDEX_FILE} samtools sort -n --threads 10 ${FINAL_BAM_FILE} -O SAM | SAMstats --sorted_sam_file - --outf ${FINAL_BAM_FILE_MAPSTATS} # ============================ # Compute library complexity # ============================ # sort by position and strand # Obtain unique count statistics module add bedtools/2.26.0 PBC_FILE_QC="${FINAL_BAM_PREFIX}.pbc.qc" # PBC File output # TotalReadPairs [tab] DistinctReadPairs [tab] OneReadPair [tab] TwoReadPairs [tab] NRF=Distinct/Total [tab] PBC1=OnePair/Distinct [tab] PBC2=OnePair/TwoPair bedtools bamtobed -i ${FILT_BAM_FILE} | awk 'BEGIN{OFS="\t"}{print $1,$2,$3,$6}' | grep -v 'chrM' | sort | uniq -c | awk 'BEGIN{mt=0;m0=0;m1=0;m2=0} ($1==1){m1=m1+1} ($1==2){m2=m2+1} {m0=m0+1} {mt=mt+$1} END{printf "%d\t%d\t%d\t%d\t%f\t%f\t%f\n",mt,m0,m1,m2,m0/mt,m1/m0,m1/m2}' > ${PBC_FILE_QC} rm ${FILT_BAM_FILE} # ================================ # make tagAlign for filtered (but not deduped) BAM # and subsample it # ================================ bedtools bamtobed -i ${FILT_BAM_FILE} | awk 'BEGIN{OFS="\t"}{$4="N";$5="1000";print $0}' | gzip -nc > ${TA_FILE} NREADS=15000000 SUBSAMPLED_TA_FILE="${OFPREFIX}.filt.sample.$((NREADS / 1000000)).SE.tagAlign.gz" zcat ${TA_FILE} | grep -v “chrM” | shuf -n ${NREADS} --random-source=<(openssl enc -aes-256-ctr -pass pass:$(zcat -f ${TA_FILE} | wc -c) -nosalt </dev/zero 2>/dev/null) | gzip -nc > ${SUBSAMPLED_TA_FILE}
QC to report	(1) Flagstat output from Final filtered deduped BAM file ${FINAL_BAM_FILE_MAPSTATS} (2) PICARD MarkDup output ${DUP_FILE_QC} http://sourceforge.net/apps/mediawiki/picard/index.php?title=Main_Page#Q:_What_is_meaning_of_the_histogram_produced_by_MarkDuplicates.3F (3) Library complexity measures ${PBC_FILE_QC} • Format of file TotalReads [tab] DistinctReads [tab] OneRead [tab] TwoRead [tab] NRF=Distinct/Total [tab] PBC1=OnePair/Distinct [tab] PBC2=OnePair/TwoPair • NRF (non redundant fraction) • PBC1 (PCR Bottleneck coefficient 1) • PBC2 (PCR Bottleneck coefficient 2) • PBC1 is the primary measure. Provisionally,

	○ 0-0.5 is severe bottlenecking ○ 0.5-0.8 is moderate bottlenecking ○ 0.8-0.9 is mild bottlenecking ○ 0.9-1.0 is no bottlenecking.
Status	Frozen

2a. Convert SE BAM to tagAlign (BED 3+3 format)

Program(s)	• bedtools 2.26.0 • gawk • shuf
Input(s)	• Filtered BAM file ${FINAL_BAM_FILE}
Output(s)	• tagAlign file ${FINAL_TA_FILE}
Commands	# ========== # Create tagAlign file # ========== module add bedtools/2.26.0 # Create SE tagAlign file FINAL_TA_FILE="${FINAL_BAM_PREFIX}.SE.tagAlign.gz" bedtools bamtobed -i ${FINAL_BAM_FILE} | awk 'BEGIN{OFS="\t"}{$4="N";$5="1000";print $0}' | gzip -nc > ${FINAL_TA_FILE}
QC to report	None
Status	Frozen

Paired-End ChIP-seq parameters

- Remove reads unmapped, mate unmapped, not primary alignment, reads failing platform, duplicates (-F 1804).
- Retain properly paired reads -f 2
- Remove multi-mapped reads (i.e. those with MAPQ < 30, using -q in SAMtools)
  - http://samtools.sourceforge.net/
- Remove PCR duplicates (using Picard's MarkDuplicates or FixSeq)
  - PICARD: http://picard.sourceforge.net/command-line-overview.shtml#MarkDuplicates

Program(s)	• SAMtools (version 1.7) • MarkDuplicates (Picard - version 1.126) • bedtools 2.26.0
Input(s)	• Raw BAM file ${RAW_BAM_FILE}
Output(s)	• Filtered deduped position sorted BAM and index file ${FINAL_BAM_FILE} ${FINAL_BAM_INDEX_FILE} • Filtered deduped name sorted BAM file ${FINAL_NMSRT_BAM_FILE} • Flagstat Metric for filtered BAM file ${FINAL_BAM_FILE_MAPSTATS} • Duplication metrics from MarkDuplicates ${DUP_FILE_QC} • Library complexity measures ${PBC_FILE_QC}
Commands	# ============================ # Remove unmapped, mate unmapped # not primary alignment, reads failing platform # Remove low MAPQ reads # Only keep properly paired reads # Obtain name sorted BAM file # ================== FILT_BAM_PREFIX="${OFPREFIX}.filt.srt" FILT_BAM_FILE="${FILT_BAM_PREFIX}.bam" TMP_FILT_BAM_PREFIX="tmp.${FILT_BAM_PREFIX}.nmsrt" TMP_FILT_BAM_FILE="${TMP_FILT_BAM_PREFIX}.bam" MAPQ_THRESH=30 samtools view -F 1804 -f 2 -q ${MAPQ_THRESH} -u ${RAW_BAM_FILE} | samtools sort -n - ${TMP_FILT_BAM_PREFIX} # Will produce name sorted BAM # Remove orphan reads (pair was removed) # and read pairs mapping to different chromosomes # Obtain position sorted BAM samtools fixmate -r ${TMP_FILT_BAM_FILE} ${OFPREFIX}.fixmate.tmp samtools view -F 1804 -f 2 -u ${OFPREFIX}.fixmate.tmp | samtools sort - ${FILT_BAM_PREFIX} rm ${OFPREFIX}.fixmate.tmp rm ${TMP_FILT_BAM_FILE} # ============= # Mark duplicates # ============= module add picard-tools/1.126

```
TMP_FILT_BAM_FILE="${FILT_BAM_PREFIX}.dupmark.bam"
MARKDUP="/srv/gs1/software/picard-tools/1.126/picard MarkDuplicates"
DUP_FILE_QC="${FILT_BAM_PREFIX}.dup.qc"

java -Xmx4G -jar ${MARKDUP} INPUT=${FILT_BAM_FILE} OUTPUT=${TMP_FILT_BAM_FILE}
METRICS_FILE=${DUP_FILE_QC} VALIDATION_STRINGENCY=LENIENT
ASSUME_SORTED=true REMOVE_DUPLICATES=false

mv ${TMP_FILT_BAM_FILE} ${FILT_BAM_FILE}

# ============================
# Remove duplicates
# Index final position sorted BAM
# Create final name sorted BAM
# ============================
FINAL_BAM_PREFIX="${OFPREFIX}.filt.srt.nodup"
FINAL_BAM_FILE="${FINAL_BAM_PREFIX}.bam" # To be stored
FINAL_BAM_INDEX_FILE="${FINAL_BAM_PREFIX}.bai"
FINAL_BAM_FILE_MAPSTATS="${FINAL_BAM_PREFIX}.flagstat.qc" # QC file
FINAL_NMSRT_BAM_PREFIX="${OFPREFIX}.filt.nmsrt.nodup"
FINAL_NMSRT_BAM_FILE="${FINAL_NMSRT_BAM_PREFIX}.bam" # To be stored

samtools view -F 1804 -f 2 -b ${FILT_BAM_FILE} > ${FINAL_BAM_FILE}

samtools sort -n ${FINAL_BAM_FILE} ${FINAL_NMSRT_BAM_PREFIX}

# Index Final BAM file
samtools index ${FINAL_BAM_FILE} ${FINAL_BAM_INDEX_FILE}

samtools sort -n --threads 10 ${FINAL_BAM_FILE} -O SAM  | SAMstats --sorted_sam_file -
--outf ${FINAL_BAM_FILE_MAPSTATS}


# ============================
# Compute library complexity
# ============================
# Sort by name
# convert to bedPE and obtain fragment coordinates
# sort by position and strand
# Obtain unique count statistics

module add bedtools/2.26.0

PBC_FILE_QC="${FINAL_BAM_PREFIX}.pbc.qc"

# TotalReadPairs [tab] DistinctReadPairs [tab] OneReadPair [tab] TwoReadPairs [tab]
NRF=Distinct/Total [tab] PBC1=OnePair/Distinct [tab] PBC2=OnePair/TwoPair


samtools sort -n ${FILT_BAM_FILE} ${OFPREFIX}.srt.tmp
bedtools bamtobed -bedpe -i ${OFPREFIX}.srt.tmp.bam | awk 'BEGIN{OFS="\t"}{print
$1,$2,$4,$6,$9,$10}' | grep -v 'chrM' | sort | uniq -c | awk 'BEGIN{mt=0;m0=0;m1=0;m2=0}
($1==1){m1=m1+1} ($1==2){m2=m2+1} {m0=m0+1} {mt=mt+$1} END{printf
"%d\t%d\t%d\t%d\t%f\t%f\t%f\n",mt,m0,m1,m2,m0/mt,m1/m0,m1/m2}' > ${PBC_FILE_QC}
rm ${OFPREFIX}.srt.tmp.bam


rm ${FILT_BAM_FILE}
```

QC to report	(1) Flagstat output from Final filtered deduped BAM file ${FINAL_BAM_FILE_MAPSTATS} (2) PICARD MarkDup output ${DUP_FILE_QC} http://sourceforge.net/apps/mediawiki/picard/index.php?title=Main_Page#Q:_What_is_meaning_of_the_histogram_produced_by_MarkDuplicates.3F (3) Library complexity measures ${PBC_FILE_QC} ● Format of file TotalReadPairs [tab] DistinctReadPairs [tab] OneReadPair [tab] TwoReadPairs [tab] NRF=Distinct/Total [tab] PBC1=OnePair/Distinct [tab] PBC2=OnePair/TwoPair ● NRF (non redundant fraction) ● PBC1 (PCR Bottleneck coefficient 1) ● PBC2 (PCR Bottleneck coefficient 2) ● PBC1 is the primary measure. Provisionally, ○ 0-0.5 is severe bottlenecking ○ 0.5-0.8 is moderate bottlenecking ○ 0.8-0.9 is mild bottlenecking ○ 0.9-1.0 is no bottlenecking.
Status	Frozen

2a. Convert SE BAM to tagAlign (BED 3+3 format)

Program(s)	• bedtools 2.26.0 • gawk • shuf
Input(s)	• Filtered BAM file ${FINAL_BAM_FILE}
Output(s)	• tagAlign file ${FINAL_TA_FILE}
Commands	# ==================== # Create tagAlign file # ==================== module add bedtools/2.26.0 # Create SE tagAlign file FINAL_TA_FILE="${FINAL_BAM_PREFIX}.SE.tagAlign.gz" bedtools bamtobed -i ${FINAL_BAM_FILE} | awk 'BEGIN{OFS="\t"}{$4="N";$5="1000";print $0}' | gzip -nc > ${FINAL_TA_FILE}
QC to report	None
Status	Frozen

2a. Convert PE BAM to tagAlign (BED 3+3 format)

Program(s)	• bedtools 2.26.0 • gawk • shuf
Input(s)	• Filtered BAM file ${FINAL_BAM_FILE}
Output(s)	• tagAlign file (virtual single end) ${FINAL_TA_FILE} • BEDPE file (with read pairs on each line) ${FINAL_BEDPE_FILE} •
Commands	# ==================== # Create tagAlign file # ==================== module add bedtools/2.26.0 # ================ # Create BEDPE file # ================ FINAL_BEDPE_FILE="${FINAL_NMSRT_BAM_PREFIX}.bedpe.gz" bedtools bamtobed -bedpe -mate1 -i ${FINAL_NMSRT_BAM_FILE} | gzip -nc > ${FINAL_BEDPE_FILE} zcat ${FINAL_BEDPE_FILE} | awk 'BEGIN{OFS="\t"}{printf "%s\t%s\t%s\tN\t1000\t%s\n%s\t%s\t%s\tN\t1000\t%s\n",$1,$2,$3,$9,$4,$5,$6,$10}' | gzip -nc > ${FINAL_TA_FILE}
QC to report	None
Status	Frozen

2b. Calculate Cross-correlation QC scores

- Code package: https://code.google.com/p/phantompeakqualtools/ (Updated version is imminent)
- Dependencies: unix, bash, R-3.20 and above, gawk, samtools, boost C++ libraries, R packages: SPP, caTools, snow
- **Cross-correlation analysis is done on a filtered (but not-deduped) and subsampled BAM. There is a special fastq trimming for cross-correlation analysis. Read1 fastq is trimmed to 50bp first using** trimfastq.py (last modified `2017/11/08`, https://github.com/ENCODE-DCC/chip-seq-pipeline2/blob/master/src/trimfastq.py). And then **it is separately mapped as SE. Reads are filtered but duplicates are not removed. Then 15 million reads are randomly sampled and used for cross-correlation analysis.**

Program(s)	• phantompeakqualtools (v1.2.1)
Input(s)	• Read1 FASTQ for PE ${FASTQ_R1} or FASTQ for SE
Output(s)	• outFile containing NSC/RSC results in tab-delimited file of 11 columns (same file can be appended to from multiple runs) ${CC_SCORES_FILE} • cross-correlation plot ${CC_PLOT_FILE
Commands	#### for both PE and SE samples # Trim R1 fastq to 50bp python trimfastq.py $FASTQ_R1 50 | gzip -nc > $TRIMMED_FASTQ_R1 # Align $TRIMMED_FASTQ_R1 (not paired) with bowtie2 (step 1a SE) and use it for filtering step (1b) and then get $FILT_BAM_FILE (not the deduped $FINAL_BAM_FILE), which is filtered but not deduped. # ================================== # make tagAlign for filtered (but not deduped) BAM # and subsample it for cross-correlation analysis # ================================== bedtools bamtobed -i ${FILT_BAM_FILE} | awk 'BEGIN{OFS="\t"}{$4="N";$5="1000";print $0}' | gzip -nc > ${TA_FILE} NREADS=15000000 SUBSAMPLED_TA_FILE="${OFPREFIX}.filt.sample.$((NREADS / 1000000)).SE.tagAlign.gz" zcat ${TA_FILE} | grep -v “chrM” | shuf -n ${NREADS} --random-source=<(openssl enc -aes-256-ctr -pass pass:$(zcat -f ${TA_FILE} | wc -c) -nosalt </dev/zero 2>/dev/null) | gzip -nc > ${SUBSAMPLED_TA_FILE} ### cross-correlation analysis CC_SCORES_FILE="${SUBSAMPLED_TA_FILE}.cc.qc" CC_PLOT_FILE="${SUBSAMPLED_TA_FILE}.cc.plot.pdf" # CC_SCORE FILE format # Filename <tab> numReads <tab> estFragLen <tab> corr_estFragLen <tab> PhantomPeak <tab> corr_phantomPeak <tab> argmin_corr <tab> min_corr <tab> phantomPeakCoef <tab> relPhantomPeakCoef <tab> QualityTag Rscript $(which run_spp.R) -c=${SUBSAMPLED_TA_FILE} -p=${NTHREADS} -filtchr=chrM -savp=${CC_PLOT_FILE} -out=${CC_SCORES_FILE} sed -r 's/,[^\t]+//g' ${CC_SCORES_FILE} > temp mv temp ${CC_SCORES_FILE}

QC to report	format:Filename<tab>numReads<tab>estFragLen<tab>corr_estFragLen<tab>PhantomPeak<tab>corr_phantomPeak<tab>argmin_corr<tab>min_corr<tab>phantomPeakCoef<tab>relPhantomPeakCoef<tab>QualityTag • Normalized strand cross-correlation coefficient (NSC) = col9 in outFile • Relative strand cross-correlation coefficient (RSC) = col10 in outFile • Estimated fragment length = col3 in outFile, take the top value • Important columns highlighted, but all/whole file can be stored for display
Status	Frozen

2c. Generate self-pseudoreplicates for each replicate (SE datasets)

Program(s)	• UNIX shuf • UNIX split • gawk
Input(s)	• TagAlign file ${FINAL_TA_FILE}
Output(s)	• 2 pseudoreplicate virtual SE tagAlign files ${PR1_TA_FILE} ${PR2_TA_FILE}
Commands	# ======================== # Create pseudoReplicates # ======================== PR_PREFIX="${OFPREFIX}.filt.nodup" PR1_TA_FILE="${PR_PREFIX}.SE.pr1.tagAlign.gz" PR2_TA_FILE="${PR_PREFIX}.SE.pr2.tagAlign.gz" # Get total number of read pairs nlines=$(zcat ${FINAL_TA_FILE} \| wc -l) nlines=$(((nlines + 1) / 2)) # Shuffle and split BED file into 2 equal parts zcat ${FINAL_TA_FILE} \| shuf --random-source=<(openssl enc -aes-256-ctr -pass pass:$(zcat -f ${FINAL_TA_FILE} | wc -c) -nosalt </dev/zero 2>/dev/null) | split -d -l ${nlines} - ${PR_PREFIX} # Will produce ${PR_PREFIX}00 and ${PR_PREFIX}01 # Convert reads into standard tagAlign file gzip -nc “${PR_PREFIX}00" > ${PR1_TA_FILE} rm "${PR_PREFIX}00" gzip -nc “${PR_PREFIX}01" > ${PR2_TA_FILE} rm "${PR_PREFIX}01"
QC to report	None
Status	Frozen

2c. Generate self-pseudoreplicates for each replicate (PE datasets)

Program(s)	• UNIX shuf • UNIX split • gawk
Input(s)	• TAGALIGN file ${FINAL_TA_FILE}
Output(s)	• 2 pseudoreplicate virtual SE tagAlign files ${PR1_TA_FILE} ${PR2_TA_FILE}
Commands	# ========================

	# Create pseudoReplicates # ======================= PR_PREFIX="${OFPREFIX}.filt.nodup" PR1_TA_FILE="${PR_PREFIX}.PE2SE.pr1.tagAlign.gz" PR2_TA_FILE="${PR_PREFIX}.PE2SE.pr2.tagAlign.gz" joined="temp.bedpe" # Make temporary fake BEDPE file from FINAL_TA_FILE zcat ${FINAL_TA_FILE} | sed 'N;s/\n/\t/' > $joined # Get total number of read pairs nlines=$(zcat ${joined} \| wc -l) nlines=$(((nlines + 1) / 2)) # Shuffle and split BEDPE file into 2 equal parts zcat ${joined} \| shuf --random-source=<(openssl enc -aes-256-ctr -pass pass:$(zcat -f ${FINAL_TA_FILE} | wc -c) -nosalt </dev/zero 2>/dev/null) | split -d -l ${nlines} - ${PR_PREFIX} # Will produce ${PR_PREFIX}00 and ${PR_PREFIX}01 # Convert fake BEDPE into standard tagAlign file awk 'BEGIN{OFS="\t"}{printf "%s\t%s\t%s\t%s\t%s\t%s\n%s\t%s\t%s\t%s\t%s\t%s\n",$1,$2,$3,$4,$5,$6,$7,$8,$9,$10,$11,$12} "${PR_PREFIX}00" | gzip -nc > ${PR1_TA_FILE} rm "${PR_PREFIX}00" awk 'BEGIN{OFS="\t"}{printf "%s\t%s\t%s\t%s\t%s\t%s\n%s\t%s\t%s\t%s\t%s\t%s\n",$1,$2,$3,$4,$5,$6,$7,$8,$9,$10,$11,$12}'"${PR_PREFIX}01" | gzip -nc > ${PR2_TA_FILE} rm "${PR_PREFIX}01" rm -f ${joined}
QC to report	None
Status	Frozen

2d. Generate pooled dataset and pooled-pseudoreplicates

Program(s)	• gzip
Input(s)	• Final tagalign files for all replicates ${REP1_TA_FILE} ${REP2_TA_FILE} obtained from ${FINAL_TA_FILE} of the step 2a. • Self-consistency pseudoreplicates for all replicates REP*_PR1_TA_FILE and REP*_PR2_TA_FILE obtained from ${PPR1_TA_FILE} ${PPR2_TA_FILE} of step 2c.
Output(s)	• Pooled tagAlign file ${POOLED_TA_FILE} • 2 pooled-pseudoreplicate tagAlign files
Commands	# ======================= # Create pooled datasets # ======================= REP1_TA_FILE="${DATASET_PREFIX}.Rep1.tagAlign.gz" REP2_TA_FILE="${DATASET_PREFIX}.Rep2.tagAlign.gz" POOLED_TA_FILE="${DATASET_PREFIX}.Rep0.tagAlign.gz" zcat ${REP1_TA_FILE} ${REP2_TA_FILE} | gzip -nc > ${POOLED_TA_FILE} # =======================

	# Create pooled pseudoreplicates # ====================== REP1_PR1_TA_FILE="${DATASET_PREFIX}.Rep1.pr1.tagAlign.gz" REP1_PR2_TA_FILE="${DATASET_PREFIX}.Rep1.pr2.tagAlign.gz" REP2_PR1_TA_FILE="${DATASET_PREFIX}.Rep2.pr1.tagAlign.gz" REP2_PR2_TA_FILE="${DATASET_PREFIX}.Rep2.pr2.tagAlign.gz" PPR1_TA_FILE="${DATASET_PREFIX}.Rep0.pr1.tagAlign.gz" PPR2_TA_FILE="${DATASET_PREFIX}.Rep0.pr2.tagAlign.gz" zcat ${REP1_PR1_TA_FILE} ${REP2_PR1_TA_FILE} | gzip -nc > ${PPR1_TA_FILE} zcat ${REP1_PR2_TA_FILE} ${REP2_PR2_TA_FILE} | gzip -nc > ${PPR2_TA_FILE}
QC to report	None
Status	Frozen

2f. Calculate Jensen-Shannon distance (JSD)

Program(s)	• deeptools (v3.3.0) plotFingerprint
Input(s)	• Filtered/deduped BAM ${NODUP_BAM_REP1}, ${NODUP_BAM_REP2} • (TF only) Filtered/deduped control BAM ${CTL_NODUP_BAM} (only one BAM file is allowed, pick 1st BAM if you have multiple controls) • Blacklist BED file ${BLACKLIST} • mapping quality threshold = 30 (${MAPQ_THRESH})
Output(s)	• JSD Plot ${JSD_PLOT} • JSD log ${JSD_LOG}: column description: https://github.com/deeptools/deepTools/blob/master/deeptools/plotFingerprint.py#L454 ○ TF: 11 col including JS distance with a control ■ auc, syn_auc, x_intercept, syn_x_intercept, elbow_pt, syn_elbow_pt, jsd, syn_jsd, pct_genome_enrich, diff_enrich, ch_div ○ histone: 7 col including synthetic JS distance without a control ■ auc, syn_auc, x_intercept, syn_x_intercept, elbow_pt, syn_elbow_pt, syn_jsd
Commands	NTH=4 # number of threads # BAMs are blacklist-filtered first for each replicate and control NODUP_BFILT_BAM_REP1=${NODUP_BAM_REP1}.bfilt.bam NODUP_BFILT_BAM_REP2=${NODUP_BAM_REP2}.bfilt.bam CTL_NODUP_BFILT_BAM=${CTL_NODUP_BFILT_BAM}.bfilt.bam bedtools intersect -nonamecheck -v -abam ${NODUP_BAM_REP1} -b ${BLACKLIST} > ${NODUP_BFILT_BAM_REP1}' bedtools intersect -nonamecheck -v -abam ${NODUP_BAM_REP2} -b ${BLACKLIST} > ${NODUP_BFILT_BAM_REP2}' bedtools intersect -nonamecheck -v -abam ${CTL_NODUP_BFILT_BAM} -b ${BLACKLIST} > ${CTL_NODUP_BFILT_BAM}' # For TF chip-seq,

	C_ALL=en_US.UTF-8 LANG=en_US.UTF-8 plotFingerprint -b ${NODUP_BFILT_BAM_REP1} ${NODUP_BFILT_BAM_REP2} --JSDsample ${CTL_NODUP_BFILT_BAM} --labels rep1 rep2 ctl1 --outQualityMetrics ${JSD_LOG} --minMappingQuality ${MAPQ_THRESH} -T "Fingerprints of different samples" --numberOfProcessors ${NTH} --plotFile ${JSD_PLOT} # For histone chip-seq, (--JSDsample is absent) C_ALL=en_US.UTF-8 LANG=en_US.UTF-8 plotFingerprint -b ${NODUP_BFILT_BAM_REP1} ${NODUP_BFILT_BAM_REP2} --labels rep1 rep2 --outQualityMetrics ${JSD_LOG} --minMappingQuality ${MAPQ_THRESH} -T "Fingerprints of different samples" --numberOfProcessors ${NTH} --plotFile ${JSD_PLOT}
QC to report	None
Status	Frozen

2g. Calculate GC bias

Program(s)	• CollectGcBiasMetrics (Picard - version 1.126) • python packages: pandas, matplotlib
Input(s)	• Filtered/deduped BAM for each true replicate ${NODUP_BAM_REPX} • Reference genome fasta ${REF_FA}
Output(s)	• GC bias Plot ${GC_BIAS_PLOT} • GC bias log ${GC_BIAS_LOG}
Commands	# we don't use plot directly generated from picard # we process picard's text output and make a plot java -Xmx6G -XX:ParallelGCThreads=1 -jar \ picard.jar \ CollectGcBiasMetrics R=${REF_FA} I=${NODUP_BAM_REPX} O=${GC_BIAS_LOG} \ USE_JDK_DEFLATER=TRUE USE_JDK_INFLATER=TRUE \ VERBOSITY=ERROR QUIET=TRUE \ ASSUME_SORTED=FALSE \ CHART=${GC_BIAS_PLOT} S=summary.txt # use ${GC_BIAS_LOG} into the following pyhton script # data_file: ${GC_BIAS_LOG} # prefix: any good prefix for output file name def plot_gc(data_file, prefix): ''' Replot the Picard output as png file to put into the html ''' # Load data data = pd.read_table(data_file, comment="#") # Plot the data fig = plt.figure() ax = fig.add_subplot(111) plt.xlim((0, 100))

	lin1 = ax.plot(data['GC'], data['NORMALIZED_COVERAGE'], label='Normalized coverage', color='r') ax.set_ylabel('Normalized coverage') ax2 = ax.twinx() lin2 = ax2.plot(data['GC'], data['MEAN_BASE_QUALITY'], label='Mean base quality at GC%', color='b') ax2.set_ylabel('Mean base quality at GC%') ax3 = ax.twinx() lin3 = ax3.plot(data['GC'], data['WINDOWS']/np.sum(data['WINDOWS']), label='Windows at GC%', color='g') ax3.get_yaxis().set_visible(False) lns = lin1 + lin2 + lin3 labs = [l.get_label() for l in lns] ax.legend(lns, labs, loc='best') # plot_img = BytesIO() # fig.savefig(plot_img, format='png') prefix = data_file.rstrip('.gc.txt') plot_png = prefix + '.gc_plot.png' fig.savefig(plot_png, format='png')
QC to report	None
Status	Frozen

3. Call peaks on replicates, self-pseudoreplicates, pooled data and pooled-pseudoreplicates

Call peaks on all replicates, pooled data, self-pseudoreplicates of each replicate and the pooled-pseudoreplicates using 3 peak callers SPP, GEM and PeakSeq

Pooling controls: If control datasets (input DNA or Igg) have replicates as far as possible match ChIP replicates to appropriate control replicates. However, under some conditions listed below, its best to pool the control replicates.

- If the no. of reads per replicate is < the no. of reads per ChIP replicate then pool the control replicate reads into a single control
- If the no. of reads between control replicates differ by > a factor of 1.2, then pool replicates (this is to avoid artificial differences in peak scores due to sequencing depth differences in different control replicates)

Pooled-replicates or pooled-pseudoreplicates should always be compared to pooled controls
Self-pseudoreplicates for a particular ReplicateN should be compared to the same control that was used for ReplicateN.

3a. Peak calling - SPP

- Use the estimated fragment length from column 3 from ${CC_SCORES_FILE}

Program(s)	SPP (v1.14) in phantompeakqualtool https://github.com/kundajelab/phantompeakqualtools

Input(s)	RepN ChIP ${REP1_TA_FILE} ${REP2_TA_FILE} vs. appropriate control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz Pooled replicate ${POOLED_TA_FILE} vs. pooled control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz RepN pseudoreplicate1 ${REP*_PR1_TA_FILE} vs. appropriate control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz RepN pseudoreplicate2 ${REP*_PR2_TA_FILE} vs. appropriate control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz Pooled-pseudoreplicate 1 ${PPR1_TA_FILE} vs. pooled control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz Pooled-pseudoreplicate 2 ${PPR2_TA_FILE} vs. pooled control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz a blacklist ${BLACKLIST} hg19: http://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeMapability/wgEncodeDacMapabilityConsensusExcludable.bed.gz hg38: http://mitra.stanford.edu/kundaje/genome_data/hg38/hg38.blacklist.bed.gz
Output(s)	Narrowpeak file ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.regionPeak.gz Cross-correlation plot (for diagnosis only) ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.pdf Cross-correlation score output (for diagnosis only) ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.ccscores
Commands	Rscript run_spp.R -c=${CHIP_TA_PREFIX}.tagAlign.gz -i=${CONTROL_TA_PREFIX}.tagAlign.gz -npeak=300000 -odir=${PEAK_OUTPUT_DIR} -speak=${FRAGLEN} -savr -savp -rf -out=${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.ccscores | grep -P 'chr[\dXY]+[\t]' | awk 'BEGIN{OFS="\t"} {if ($5>1000) $5=1000; print $0} # filter out peaks in blacklisted region rpeakfile_raw=${CHIP_TA_PREFIX}.tagAlign_VS_${CONTROL_TA_PREFIX}.tagAlign.regionPeak.gz rpeakfile=${CHIP_TA_PREFIX}.tagAlign_x_${CONTROL_TA_PREFIX}.tagAlign.regionPeak.gz rpeakfile_filt=${CHIP_TA_PREFIX}.tagAlign_x_${CONTROL_TA_PREFIX}.tagAlign.filt.regionPeak.gz zcat $rpeakfile_raw | awk 'BEGIN{OFS="\t"}{ if ($2<0) $2=0; print $1,int($2),int($3),$4,$5,$6,$7,$8,$9,$10;}' | gzip -f -nc > $rpeakfile bedtools intersect -v -a <(zcat -f $rpeakfile) -b <(zcat -f $blacklist) | awk 'BEGIN{OFS="\t"} {if ($5>1000) $5=1000; print $0}' | grep -P 'chr[\dXY]+[\t]' | gzip -nc > $filt_rpeakfile;
QC to report	Number of peaks called
Status	Frozen

3b. Peak calling - GEM

Program(s)	GEM v2.4.1 http://cgs.csail.mit.edu/gem/
Input(s)	RepN ChIP ${REP1_TA_FILE} ${REP2_TA_FILE} vs. appropriate control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz Pooled replicate ${POOLED_TA_FILE} vs. pooled control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz RepN pseudoreplicate1 ${REP*_PR1_TA_FILE} vs. appropriate control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz RepN pseudoreplicate2 ${REP*_PR2_TA_FILE} vs. appropriate control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz Pooled-pseudoreplicate 1 ${PPR1_TA_FILE} vs. pooled control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz Pooled-pseudoreplicate 2 ${PPR2_TA_FILE} vs. pooled control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz **Genome sequence** In order to run the motif discovery of GEM algorithm, a genome sequence is needed. The path to directory containing the genome sequence files (by chromosome, *.fa or *.fasta files, with the prefix "chr") can be specified using option --genome (for example, --genome your_path/mm8/). Note that the chromosome name should match those in the "--g" genome_info file, as well as those in your read alignment file. For hg19 these can be obtained from Female: http://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/referenceSequences/femaleByChrom/ Male: Add chromosome Y from http://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/referenceSequences/maleByChrom/ **Read distribution file** Obtain from http://cgs.csail.mit.edu/gem/
Output(s)	1. Narrowpeak file ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.narrowPeak.gz 2. K-mer set motifs (KSM.txt) 3. PFM file of PWM motifs (PFM.txt) 4. HTML file summarizing the GEM event and motif results See http://cgs.csail.mit.edu/gem/ for more details
Commands	# ============================ # See http://wiki.encodedcc.org/index.php/GPS/GEM of additional information # ============================ gunzip ${CHIP_TA_PREFIX}.tagAlign.gz java -Xmx15G -jar gem.jar --g hg19.info --d Read_Distribution_default.txt --s 2400000000 --expt ${CHIP_TA_PREFIX}.tagAlign --ctrl ${CONTROL_TA_PREFIX}.tagAlign.gz --f BED --out ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX} --genome ${SEQ_DIR} --k_min 6 --k_max 13 --outNP --q 0

	rm ${CHIP_TA_PREFIX}.tagAlign # ============================== # Sort peaks by signal value and truncate peaks to top 300K # ============================== sort -k7nr,7nr ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}/${CHIP_TA_PREFIX}_GEM_events.narrowPeak | head -n 300000 | gzip -nc > temp mv temp ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.narrowPeak.gz
QC to report	Number of peaks called
Status	Frozen

3c. Peak calling - PeakSeq

- Use the estimated fragment length from column 3 from ${CC_SCORES_FILE}

Program(s)	PeakSeq v 1.25 http://wiki.encodedcc.org/index.php/PeakSeq
Input(s)	RepN ChIP ${REP1_TA_FILE} ${REP2_TA_FILE} vs. appropriate control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz Pooled replicate ${POOLED_TA_FILE} vs. pooled control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz RepN pseudoreplicate1 ${REP*_PR1_TA_FILE} vs. appropriate control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz RepN pseudoreplicate2 ${REP*_PR2_TA_FILE} vs. appropriate control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz Pooled-pseudoreplicate 1 ${PPR1_TA_FILE} vs. pooled control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz Pooled-pseudoreplicate 2 ${PPR2_TA_FILE} vs. pooled control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz Mappability file ${PEAKSEQ_MAP_FILE}.txt (Obtain from http://archive.gersteinlab.org/proj/PeakSeq/Mappability_Map/H.sapiens/Mapability_HG.txt)
Output(s)	Narrowpeak file ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.regionPeak.gz
Commands	# ============================ # The chip and input reads (chip.bam and input.bam) should be preprocessed before running: # ============================ mkdir ${TMP_CHIP_DIR} mkdir ${TMP_CONTROL_DIR} zcat ${CHIP_TA_PREFIX}.tagAlign.gz | PeakSeq -preprocess tagAlign stdin ${TMP_CHIP_DIR} zcat ${CONTROL_TA_PREFIX}.tagAlign.gz | PeakSeq -preprocess tagAlign stdin ${TMP_CONTROL_DIR} # ============================ # Then it is necessary to setup the configuration file (config.dat). An example configuration file is included with the PeakSeq download. An example: # ============================ Experiment_id $(basename ${CHIP_TA_PREFIX}) Enrichment_mapped_fragment_length ${FRAGLEN} target_FDR 0.05 N_Simulations 10 Minimum_interpeak_distance ${FRAGLEN} Mappability_map_file ${PEAKSEQ_MAP_FILE}.txt ChIP_Seq_reads_data_dirs ${TMP_CHIP_DIR} Input_reads_data_dirs max_Qvalue 0.1 Background_model Simulated # ============================

	#Finally, the peaks are called using the configuration file: # ============================ PeakSeq -peak_select config.dat
QC to report	Number of peaks called
Status	Frozen

4. Run IDR on all pairs of replicates, self-pseudoreplicates and pooled pseudoreplicates

Use IDR to compare all pairs of matched replicates
(1) True replicates narrowPeak files: ${REP1_PEAK_FILE} vs. ${REP2_PEAK_FILE} IDR results transferred to Pooled-replicates narrowPeak file ${POOLED_PEAK_FILE}
(2) Pooled-pseudoreplicates: ${PPR1_PEAK_FILE} vs. ${PPR2_PEAK_FILE} IDR results transferred to Pooled-replicates narrowPeak file ${POOLED_PEAK_FILE}
(3) Rep1 self-pseudoreplicates: ${REP1_PR1_PEAK_FILE} vs. ${REP1_PR2_PEAK_FILE} IDR results transferred to Rep1 narrowPeak file ${REP1_PEAK_FILE}
(4) Rep2 self-pseudoreplicates: ${REP2_PR1_PEAK_FILE} vs. ${REP2_PR2_PEAK_FILE} IDR results transferred to Rep2 narrowPeak file ${REP2_PEAK_FILE}

IDR Threshold: Use IDR threshold of 5% for all pairwise analyses

4a. For True Replicates
Below we show the use for true replicates. The same steps can be applied for all other pairs.

Program(s)	IDR (https://github.com/kundajelab/idr) 2.0.4 / Installation instructions (https://github.com/kundajelab/idr#installation). NOTE: Works only with Python3
Input(s)	• a pair of narrowPeak files for replicates ${REP1_PEAK_FILE} ${REP2_PEAK_FILE} • a pooled-replicate narrowPeak file ${POOLED_PEAK_FILE} • a blacklist ${BLACKLIST} ○ hg19: http://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeMapability/wgEncodeDacMapabilityConsensusExcludable.bed.gz ○ mm9: http://mitra.stanford.edu/kundaje/akundaje/release/blacklists/mm9-mouse/mm9-blacklist.bed.gz ○ GRCh38: http://mitra.stanford.edu/kundaje/akundaje/release/blacklists/hg38-human/hg38.blacklist.bed.gz ○ mm10: http://mitra.stanford.edu/kundaje/akundaje/release/blacklists/mm10-mouse/mm10.blacklist.bed.gz • For SPP use signal.value for ranking with a parameter '--rank signal.value' • For GEM use signal.value for ranking with a parameter '--rank signal.value' • For PeakSeq use q.value for ranking with a parameter '--rank q.value'
Output(s)	• The output from EM fitting: suffixed by overlapped-peaks.txt.png • The full set of peaks that overlap between the replicates with local and global IDR: suffixed by overlapped-peaks.txt ${IDR_OUTPUT} • IDR output file ${IDR_OUTPUT} ○ # Columns 1-10 are same as pooled common peaks narrowPeak columns ○ # Col 11: -log10(local IDR value) ○ # Col 12: -log10(global IDR value) ○ # Col 15: ranking measure from Rep1 ○ # Col 19: ranking measure from Rep2

	• Final IDR thresholded file ${REP1_VS_REP2}.IDR0.05.narrowPeak.gz • Final IDR thresholded file filtered using a blacklist ${REP1_VS_REP2}.IDR0.05.filt.narrowPeak.gz
Commands	IDR_THRESH=0.05 # ============================== # Perform IDR analysis. # Generate a plot and IDR output with additional columns including IDR scores. # ============================== idr --samples ${REP1_PEAK_FILE} ${REP2_PEAK_FILE} --peak-list ${POOLED_PEAK_FILE} --input-file-type narrowPeak --output-file ${IDR_OUTPUT} --rank signal.value --soft-idr-threshold ${IDR_THRESH} --plot --use-best-multisummit-IDR # ============================== # Get peaks passing IDR threshold of 5% # ============================== IDR_THRESH_TRANSFORMED=$(awk -v p=${IDR_THRESH} 'BEGIN{print -log(p)/log(10)}') awk 'BEGIN{OFS="\t"} $12>="'${IDR_THRESH_TRANSFORMED}'" {print $1,$2,$3,$4,$5,$6,$7,$8,$9,$10}' ${IDR_OUTPUT} | sort | uniq | sort -k7n,7n | gzip -nc > ${REP1_VS_REP2}.IDR0.05.narrowPeak.gz NPEAKS_IDR=$(zcat ${REP1_VS_REP2}.IDR0.05.narrowPeak.gz | wc -l) # ============================== # Filter using black list # ============================== bedtools intersect -v -a ${REP1_VS_REP2}.IDR0.05.narrowPeak.gz -b ${BLACKLIST} | grep -P 'chr[\dXY]+[\t]' | awk 'BEGIN{OFS="\t"} {if ($5>1000) $5=1000; print $0}' | gzip -nc > ${REP1_VS_REP2}.IDR0.05.filt.narrowPeak.gz
Parameters	• --samples : [REP1_PEAK_FILE] and [REP2_PEAK_FILE] are the peak calls for the pair of replicates in narrowPeak format. They must be compressed files. e.g. /peaks/reps/chipSampleRep1_VS_controlSampleRep0.narrowPeak.gz AND /peaks/reps/chipSampleRep2_VS_controlSampleRep0.narrowPeak.gz • --input-file-type : the peak file format (narrowPeak or broadPeak). Set to narrowPeak if it is narrowPeak/regionPeak or broadPeak if it is broadPeak. BroadPeak files do not contain Column 10. • --rank : the ranking measure to use. It can take only one of the following values signal.value , p.value or q.value • --soft-idr-threshold : IDR threshold, Set to ${IDR_THRESH}
QC to report	• Number of peaks passing IDR thresholds of 5% ${NPEAKS_IDR} • For each pairwise analysis, we have a *overlapped-peaks.txt file. The 12th column of the overlapped-peaks.txt file has the global IDR score for each pair of overlapping peaks. • Also store ${POOLED_COMMON_PEAKS_IDR} • To get the number of peaks that pass an IDR threshold of T (e.g. 0.01) you simply find the number of lines in ${POOLED_COMMON_PEAKS_IDR} that have Column 14 <= T
Status	Frozen

- If you have more than 2 true replicates select the longest peak list from all pairs that passes the IDR threshold.
- **Nt = Best no. of peaks passing IDR threshold by comparing true replicates**

4b. IDR analysis - self-pseudoreplicates

- Perform as with real replicates, but comparing pseudoreplicate 1 vs pseudoreplicate 2 made from each of the real biological replicate peaks
- **Rep1 self-pseudoreplicates:** ${REP1_PR1_PEAK_FILE} vs. ${REP1_PR2_PEAK_FILE} and use ${REP1_PEAK_FILE} as pooled file
- **Rep2 self-pseudoreplicates:** ${REP2_PR1_PEAK_FILE} vs. ${REP2_PR2_PEAK_FILE} and use ${REP2_PEAK_FILE} as pooled file
- This gives the self-consistent IDR peaks
- **N1 and N2 = No. of peaks passing IDR threshold by comparing self-pseudoReplicates for Rep1 and Rep2 respectively**

4c. IDR analysis - pooled pseudoreplicates

- Perform as with real replicates, but comparing pooled-pseudoreplicate 1 vs pooled-pseudoreplicate 2 made from the pooled biological replicate peaks
- ${PPR1_PEAK_FILE} vs. ${PPR2_PEAK_FILE} and use ${POOLED_PEAK_FILE} as pooled file
- **Np = No. of peaks passing IDR threshold by comparing pooled pseudo-replicates**

4d. Select final peak calls - conservative set

- If you have more than 2 true replicates select the longest peak list from all pairs that passes the 5% IDR threshold. This is the conservative peak set.
- **Nt = Best no. of peaks passing IDR threshold by comparing true replicates**
- Filter using black list:
  hg19:
  http://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeMapability/wgEncodeDacMapabilityConsensusExcludable.bed.gz
  hg38:
  http://mitra.stanford.edu/kundaje/genome_data/hg38/hg38.blacklist.bed.gz

4e. Select final peak calls - optimal set

- **Longest of the Nt and Np peak lists**
- Filter using black list:
  http://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeMapability/wgEncodeDacMapabilityConsensusExcludable.bed.gz

4f. Compute IDR QC scores

- **Rescue Ratio = max(Np,Nt) / min(Np,Nt)**
  Nt and Np should be within a factor of 2 of each other
- **Self-consistency Ratio = max(N1,N2) / min(N1,N2)**
  N1 and N2 should be within a factor of 2 of each other
- If Rescue Ratio AND self-consistency Ratio are both > 2, Flag the file for reproducibility FAIL (-1)
- If Rescue Ratio OR self-consistency Ratio are > 2, Flag the file for reproducibility Borderline (0)

Rescue Ratio v2 <TODO>
Self-Consistency Ratio v2 <TODO>

4f. Compute Fraction of Reads in Peaks (FRiP) : function frip(ta_file, cc_qc_log, idr_peak_file)
You compute the fraction of reads from each replicate tagAlign and pooled tagAlign that fall within the Np and Nt peak sets.

Program(s)	bedtools 2.26.0
Input(s)	Final tagAlign file for Rep1 ${REP1_TA_FILE} vs. IDR peak from pseudo replicates of Rep1 ${REP1_PR_IDR_PEAK_FILE} with estimated fragment length for replicate 1 Final tagAlign file for Rep2 ${REP2_TA_FILE} vs. IDR peak from pseudo replicates of Rep2 ${REP2_PR_IDR_PEAK_FILE} with estimated fragment length for replicate 2 Pooled tagAlign file ${POOLED_TA_FILE} vs. IDR peak from true replicates (Nt) ${CONS_IDR_PEAK_FILE} with mean estimated fragment length of rep1 and rep2 Pooled tagAlign file ${POOLED_TA_FILE} vs. IDR peak from pooled pseudo replicates (Np) ${OPTIMAL_IDR_PEAK_FILE} with mean estimated fragment length of rep1 and rep2
Output(s)	FRiP text file ${FRiP}
Commands	# get estimated fragment length from cross-corr. analysis log FRAGLEN=$(cat ${CC_QC_LOG} | awk '{print $3}') HALF_FRAGLEN=$(((FRAGLEN+1)/2)) # rounding to integer CHRSIZEFILE=<path_of_file_containing_chromosome_sizes> # This file is a tab delimited file with 2 columns Col1 (chromosome name), Col2 (chromosome size in bp). val1=$(bedtools slop -i ${TA_FILE} -g $CHRSIZEFILE -s -l -$HALF_FRAGLEN -r $HALF_FRAGLEN | \ awk '{if ($2>=0 && $3>=0 && $2<=$3) print $0}' | \ bedtools intersect -a stdin -b ${IDR_PEAK_FILE} -wa -u \| wc -l) val2=$(zcat $TA_FILE \| wc -l) awk 'BEGIN {print '${val1}'/'${val2}'}' > ${FRIP}
QC to report	Fraction of reads in peaks
Status	Frozen

5. Create signal tracks

Peak calling and signal tracks using MACSv2 for TFs and histone marks

- Use the estimated fragment length from column 3 from ${CC_SCORES_FILE}

Program(s)	MACSv2 https://github.com/taoliu/MACS/ Installation Instructions (https://github.com/taoliu/MACS/blob/master/INSTALL.rst). NOTE: Works only with Python 2.7 (>=2.7.5). Does not work with Python 3. Also requires slopBed, bedClip and bedGraphToBigWig from KentTools
Input(s)	RepN ChIP ${REP1_TA_FILE} ${REP2_TA_FILE} vs. appropriate control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz Pooled replicate ${POOLED_TA_FILE} vs. pooled control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz RepN pseudoreplicate1 ${REP*_PR1_TA_FILE} vs. appropriate control tagAlign

	${CONTROL_TA_PREFIX}.tagAlign.gz RepN pseudoreplicate2 ${REP*_PR2_TA_FILE} vs. appropriate control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz Pooled-pseudoreplicate 1 ${PPR1_TA_FILE} vs. pooled control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz Pooled-pseudoreplicate 2 ${PPR2_TA_FILE} vs. pooled control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz
Output(s)	Narrowpeak file ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.narrowPeak.gz ~~Gappedpeak file ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.gappedPeak.gz~~ Fold-enrichment bigWig file ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.fc.signal.bw -log10(pvalue) bigWig file ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.pval.signal.bw
Commands	GENOMESIZE='hs' # for human GENOMESIZE='mm' #for mouse NPEAKS=500000 # capping number of peaks called from MACS2 ========== # Generate narrow peaks and preliminary signal tracks ========== macs2 callpeak -t ${REP1_TA_FILE}.tagAlign.gz -c ${CONTROL_TA_PREFIX}.tagAlign.gz -f BED -n ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX} -g ${GENOMESIZE} -p 1e-2 --nomodel --shift 0 --extsize ${FRAGLEN} --keep-dup all -B --SPMR # Sort by Col8 in descending order and replace long peak names in Column 4 with Peak_<peakRank> sort -k 8gr,8gr ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_peaks.narrowPeak | awk 'BEGIN{OFS="\t"}{$4="Peak_"NR ; print $0}' | head -n ${NPEAKS} | gzip -nc > ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.narrowPeak.gz # remove additional files rm -f ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_peaks.xls ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_peaks.narrowPeak ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_summits.bed ~~==========~~ ~~# Generate Broad and Gapped Peaks~~ ~~==========~~ ~~macs2 callpeak -t ${REP1_TA_FILE}.tagAlign.gz -c ${CONTROL_TA_PREFIX}.tagAlign.gz -f BED -n ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX} -g ${GENOMESIZE} -p 1e-2 --broad --nomodel --shift 0 --extsize ${FRAGLEN} --keep-dup all~~ ~~# Sort by Col8 (for broadPeak) or Col 14(for gappedPeak) in descending order and replace long peak names in Column 4 with Peak_<peakRank>~~ ~~sort -k 8gr,8gr ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_peaks.broadPeak | awk 'BEGIN{OFS="\t"}{$4="Peak_"NR ; print $0}' | gzip -nc > ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.broadPeak.gz~~ ~~sort -k 14gr,14gr ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_peaks.gappedPeak | awk 'BEGIN{OFS="\t"}{$4="Peak_"NR ; print $0}' | gzip -nc > ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.gappedPeak.gz~~ ~~# remove additional files~~ ~~rm -f ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_peaks.xls ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_peaks.broadPeak~~

~~${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_peaks.gappedPeak~~
~~${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_summits.bed~~

~~==========~~
For Fold enrichment signal tracks
~~==========~~
CHRSIZEFILE=<path_of_file_containing_chromosome_sizes>
This file is a tab delimited file with 2 columns Col1 (chromosome name), Col2 (chromosome size in bp).

macs2 bdgcmp -t ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_treat_pileup.bdg -c ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_control_lambda.bdg --outdir ${PEAK_OUTPUT_DIR} -o ${CHIP_TA_PREFIX}_FE.bdg -m FE

Remove coordinates outside chromosome sizes (stupid MACS2 bug)
slopBed -i ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_FE.bdg -g ${CHRSIZEFILE} -b 0 | awk '{if ($3 != -1) print $0}' | bedClip stdin ${CHRSIZEFILE} ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.fc.signal.bedgraph

rm -f ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_FE.bdg

Convert bedgraph to bigwig
bedGraphToBigWig ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.fc.signal.bedgraph ${CHRSIZEFILE} ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.fc.signal.bw

rm -f ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.fc.signal.bedgraph

~~==========~~
For -log10(p-value) signal tracks
~~==========~~

Compute sval = min(no. of reads in ChIP, no. of reads in control) / 1,000,000
chipReads=$(zcat ${REP1_TA_FILE}.tagAlign.gz | wc -l | awk '{printf "%f", $1/1000000}');

controlReads=$(zcat ${CONTROL_TA_FILE}.tagAlign.gz | wc -l | awk '{printf "%f", $1/1000000}');

sval=$(echo "${chipReads} ${controlReads}" | awk '$1>$2{printf "%f",$2} $1<=$2{printf "%f",$1}');

macs2 bdgcmp -t ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_treat_pileup.bdg -c ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_control_lambda.bdg --outdir ${PEAK_OUTPUT_DIR} -o ${CHIP_TA_PREFIX}_ppois.bdg -m ppois -S ${sval}

Remove coordinates outside chromosome sizes (stupid MACS2 bug)
slopBed -i ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_ppois.bdg -g ${CHRSIZEFILE} -b 0 | awk '{if ($3 != -1) print $0}' | bedClip stdin ${CHRSIZEFILE} ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.pval.signal.bedgraph

rm -rf ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_ppois.bdg

Convert bedgraph to bigwig
bedGraphToBigWig ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.pval.signal.bedgraph ${CHRSIZEFILE} ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.pval.signal.bw

rm -f ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.pval.signal.bedgraph
rm -f ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_treat_pileup.bdg ${peakFile}_control_lambda.bdg

QC to report	
Status	Beta

Count signal track generation

Program(s)	• bedtools 2.26.0 • bedGraphToBigWig
Input(s)	RepX ChIP ${REPX_TA_FILE}, or Pooled replicate ${POOLED_TA_FILE} Chromosome sizes file ${CHRSZ}
Output(s)	Positive bigWig file prefix.positive.bigwig Negative bigWig file prefix.negative.bigwig
Commands	zcat -f ${TA_FILE} | sort -k1,1 -k2,2n | bedtools genomecov -5 -bg -strand + -g ${CHRSZ} -i stdin > TMP.POS.BED bedGraphToBigWig TMP.POS.BED ${CHRSZ} ${TA_FILE_PREFIX}.positive.bigwig zcat -f ${TA_FILE} | sort -k1,1 -k2,2n | bedtools genomecov -5 -bg -strand - -g ${CHRSZ} -i stdin > TMP.NEG.BED bedGraphToBigWig TMP.NEG.BED ${CHRSZ} ${TA_FILE_PREFIX}.negative.bigwig
QC to report	
Status	Beta

6. Peak calling for Histone Marks

Naive overlap thresholding for histone peak calls

NOTE: We haven't yet finalized an IDR protocol for histone marks. For now this is a simple overlap version that works reasonably well. IDR protocol for histone marks is in development

But here we do a similar analysis as IDR described in Section 4. Repeat the same procedure for the following set of combination of (Rep1, Rep2 and Pooled) to do reproducibility QC.

(Rep1, Rep2, Pooled_rep)
(Rep1-PR1, Rep1-PR2, Rep1)
(Rep2-PR1, Rep2-PR2, Rep2)
(PPR1,PPR2,Pooled_rep)

(I've just split the piped commands on separate lines for clarity)

```
# ======================
# For narrowPeak files
# ======================

# Find pooled peaks that overlap Rep1 and Rep2 where overlap is defined as the fractional overlap wrt any one of the overlapping peak pairs  >= 0.5

intersectBed -wo -a Pooled.narrowPeak.gz -b Rep1.narrowPeak.gz |
awk 'BEGIN{FS="\t";OFS="\t"}{s1=$3-$2; s2=$13-$12; if (($21/s1 >= 0.5) || ($21/s2 >= 0.5)) {print $0}}' |
cut -f 1-10 | sort | uniq |
intersectBed -wo -a stdin -b Rep2.narrowPeak.gz |
awk 'BEGIN{FS="\t";OFS="\t"}{s1=$3-$2; s2=$13-$12; if (($21/s1 >= 0.5) || ($21/s2 >= 0.5)) {print $0}}' |
cut -f 1-10 | sort | uniq > PooledInRep1AndRep2.narrowPeak.gz


# filter through blacklist
zcat PooledInRep1AndRep2.narrowPeak.gz PooledInPsRep1AndPsRep2.narrowPeak.gz | sort | uniq | awk
'BEGIN{OFS="\t"} {if ($5>1000) $5=1000; print $0}' \
| grep -P 'chr[\dXY]+[ \t]' > PooledInRep1AndRep2.filt.narrowPeak.gz

# ======================
For BroadPeak files (there is just a difference is the awk commands wrt the column numbers)
# ======================

# Find pooled peaks that overlap Rep1 and Rep2 where overlap is defined as the fractional overlap wrt any one of the overlapping peak pairs  >= 0.5

intersectBed -wo -a Pooled.broadPeak.gz -b Rep1.broadPeak.gz |
awk 'BEGIN{FS="\t";OFS="\t"}{s1=$3-$2; s2=$12-$11; if (($19/s1 >= 0.5) || ($19/s2 >= 0.5)) {print $0}}' |
cut -f 1-9 | sort | uniq |
intersectBed -wo -a stdin -b Rep2.broadPeak.gz |
```

awk 'BEGIN{FS="\t";OFS="\t"}{s1=$3-$2; s2=$12-$11; if (($19/s1 >= 0.5) || ($19/s2 >= 0.5)) {print $0}}' | cut -f 1-9 | sort | uniq > PooledInRep1AndRep2.broadPeak.gz

Find pooled peaks that overlap PooledPseudoRep1 and PooledPseudoRep2 where overlap is defined as the fractional overlap wrt any one of the overlapping peak pairs >= 0.5

intersectBed -wo -a Pooled.broadPeak.gz -b PsRep1.broadPeak.gz |
awk 'BEGIN{FS="\t";OFS="\t"}{s1=$3-$2; s2=$12-$11; if (($19/s1 >= 0.5) || ($19/s2 >= 0.5)) {print $0}}' | cut -f 1-9 | sort | uniq |
intersectBed -wo -a stdin -b PsRep2.broadPeak.gz |
awk 'BEGIN{FS="\t";OFS="\t"}{s1=$3-$2; s2=$12-$11; if (($19/s1 >= 0.5) || ($19/s2 >= 0.5)) {print $0}}' | cut -f 1-9 | sort | uniq > PooledInPsRep1AndPsRep2.broadPeak.gz

Combine peak lists

zcat PooledInRep1AndRep2.broadPeak.gz PooledInPsRep1AndPsRep2.broadPeak.gz | sort | uniq > finalPeakList.broadPeak.gz

======================
For gappedPeak files (there is just a difference is the awk commands wrt the column numbers)
======================

Find pooled peaks that overlap Rep1 and Rep2 where overlap is defined as the fractional overlap wrt any one of the overlapping peak pairs >= 0.5

intersectBed -wo -a Pooled.gappedPeak.gz -b Rep1.gappedPeak.gz |
awk 'BEGIN{FS="\t";OFS="\t"}{s1=$3-$2; s2=$18-$17; if (($31/s1 >= 0.5) || ($31/s2 >= 0.5)) {print $0}}' | cut -f 1-15 | sort | uniq |
intersectBed -wo -a stdin -b Rep2.gappedPeak.gz |
awk 'BEGIN{FS="\t";OFS="\t"}{s1=$3-$2; s2=$18-$17; if (($31/s1 >= 0.5) || ($31/s2 >= 0.5)) {print $0}}' | cut -f 1-15 | sort | uniq > PooledInRep1AndRep2.gappedPeak.gz

Find pooled peaks that overlap PooledPseudoRep1 and PooledPseudoRep2 where overlap is defined as the fractional overlap wrt any one of the overlapping peak pairs >= 0.5

intersectBed -wo -a Pooled.gappedPeak.gz -b PsRep1.gappedPeak.gz |
awk 'BEGIN{FS="\t";OFS="\t"}{s1=$3-$2; s2=$18-$17; if (($31/s1 >= 0.5) || ($31/s2 >= 0.5)) {print $0}}' | cut -f 1-15 | sort | uniq |
intersectBed -wo -a stdin -b PsRep2.gappedPeak.gz |
awk 'BEGIN{FS="\t";OFS="\t"}{s1=$3-$2; s2=$18-$17; if (($31/s1 >= 0.5) || ($31/s2 >= 0.5)) {print $0}}' | cut -f 1-15 | sort | uniq > PooledInPsRep1AndPsRep2.gappedPeak.gz

Combine peak lists

zcat PooledInRep1AndRep2.gappedPeak.gz PooledInPsRep1AndPsRep2.gappedPeak.gz | sort | uniq > finalPeakList.gappedPeak.gz