**High-Throughput Sequencing: from Raw Reads to Variants**

Brice A. J. Sarver

University of Montana

Division of Biological Sciences

[brice.sarver@gmail.com](mailto:brice.sarver@gmail.com)

ConGen Lecture/Worksheet, 2 September 2015

**Introduction**

After sequencing, libraries are often split by their identifier (barcode) and compressed to save disk space. It is usually the user’s responsibility to obtain the raw data, usually via FTP, and store the data locally for analysis. Instructions differ from core to core but are straightforward.

A typical workflow involves several quality-control steps. The first is to look at some general per-library summary statistics using FastQC or a similar application. This lets the user see whether anything appears to be amiss with the library or sequencing run. Typical things to look at include the distribution of quality scores per read position, GC content, k-mer enrichment, etc. FastQC will generate HTML reports that indicate whether something might need your attention.

Next, the reads must be cleaned. Reads often have low-quality bases at the beginning or end; these can be confidently removed. If an entire read is poor, it can be discarded. It may be beneficial to keep the read’s mate even if the first is, itself, discarded – it’s still data, and these (effectively) single-end reads can be accommodated in many pipelines. This step usually involves adapter trimming, removing adapter sequence from reads that are not real data, as well. Next, pairs that overlap one another may be merged into a single read to reduce the computational burden. Finally, reads that are identified as PCR duplicates may be removed, though this sometimes happens later in a pipeline. Not all datasets should or can be deduped, however, such as double digest RADs (although the addition of a degenerate base region makes this possible now; see http://www.ncbi.nlm.nih.gov/pubmed/25411373), or experiments where the number of reads is of principal interest (e.g., RNA-seq).

If you are going to be assembling a genome *de novo*, I recommend all of the above.

Many applications can perform these steps, but I recommend the expHTS pipeline from Matt Settles (UC Davis): <https://github.com/msettles/expHTS>.

Popular alternatives include Trimmomatic, CutAdapt, and Picard’s MarkDuplicates.

**Exercises**

Several libraries were prepared as part of a targeted capture study built on the entire exome. For this course/workshop, I have selected a 20 Mbp region of mouse (*Mus*) chromosome 1. The start codons of ~50 transcripts are in this region.

The reads have been cleaned, adapter trimmed, and merged. I have included reads that map to this region plus a subset of others that do not.

To begin, navigate to: /home/congen/data/brice/

**Indexing**

Throughout this tutorial, we will index and sort reads using several programs. These steps are essential for quickly traversing large datasets. Failure to index or sort will usually throw an error with a message indicating what you failed to do.

We need to create a sequence dictionary using Picard, index using SAMtools, and index using BWA. These steps are fast. You only need to do this once for each dataset you might be working with; if you’re using the entire human genome for a project, for example, you can save time and disk space by using indices you generated previously.

Execute:

picard CreateSequenceDictionary R=chr1.20mbp.fa O=chr1.20mbp.dict

samtools faidx chr1.20mbp.fa

bwa index chr1.20mbp.fa

**Mapping**

Mapping uses an algorithm to determine the most probable map (alignment) location of a sequencing read with respect to the reference. For this exercise, we’ll be using BWA. Specifically, we’ll be using the MEM algorithm. It performs well on datasets of this size and is recommended when you have reads 70 bp or longer.

Execute:

bwa mem -M -t 4 chr1.20mbp.fa 10252.final.fq.gz > 10252.raw.sam

This will run BWA on four threads (the -@ argument). Use more if you can! The time to completion linearly decreases as the number of CPUs increases. On my computer (Mid 2014 MPB, 2.8 GHz Intel Core i7), this step took 8m 39s of real time.

The results are in the SAM (Sequence Alignment/Map) format. Because this is plain text, the files can be quite large. To reduce the size, let’s convert the SAM file to its binary counterpart, the BAM.

Execute:

samtools view -Sb 10252.raw.sam > 10252.raw.bam

If you are comfortable with pipes in UNIX, you can combine this command with the previous BWA call and eliminate the intermediate SAM.

How much space did you save? Let’s find out using du, which calculates the disk usage of a file.

Execute:

du –h 10252.raw.sam

du –h 10252.raw.bam

Storage is expensive. Compress whenever possible!

Take a look at the raw SAM file using less. Note the header, the first few lines of the file (starting with #) that gives some information about the file itself. You can view BAMs or their headers by using samtools view.

**Marking Duplicates and Indel Realignment**

It is possible to use information from mapping to identify PCR duplicates. If reads start at the same position and have the same match-mismatch structure (say, out of 100 bp, there are 100 matches to the reference), they are flagged as duplicates.

We need to do some indexing and sorting again. Execute:

samtools sort -@ 4 10252.raw.bam 10252.sorted

samtools index 10252.sorted.bam

Some downstream applications require read groups (@RG) to be set in the header of the BAM. Let’s do this now using Picard. Execute:

picard AddOrReplaceReadGroups I=10252.sorted.bam O=10252.sorted.RG.bam SO=coordinate LB=spret PL=illumina SM=10252 VALIDATION\_STRINGENCY=LENIENT

And, finally, we’ll flag reads as duplicates and index again. Execute:

picard MarkDuplicates I=10252.sorted.RG.bam O=10252.sorted.RG.dedup.bam VALIDATION\_STRINGENCY=LENIENT M=10252.dupmetrics

samtools index 10252.sorted.RG.dedup.bam

Identifying indels (insertions/deletions) is more challenging computationally. Often, mappers do an okay job at identifying true insertions and deletions. To be sure, you can perform indel realignment on your dataset. This is a two-step process using the Genome Analysis ToolKit (GATK). The first identifies any regions where there are indels by creating a list of targets. Execute:

gatk -T RealignerTargetCreator -R chr1.20mbp.fa -I 10252.sorted.RG.dedup.bam -o 10252.indel\_intervals.list

The second takes this information and performs local realignment to attempt to place indels in their correct place. Execute:

gatk -T IndelRealigner -R chr1.20mbp.fa -I 10252.sorted.RG.dedup.bam -targetIntervals 10252.indel\_intervals.list -o 10252.realigned.bam

We’re now ready to call variants.

**Variant Calling and Filtering**

For this exercise, in the interest of time, we’ll use the GATK’s UnifiedGenotyper. It is recently depreciated in favor of HaplotypeCaller but is faster. Execute:

gatk -T UnifiedGenotyper -R chr1.20mbp.fa -I 10252.realigned.bam --genotpying\_mode DISCOVERY -stand\_emit\_conf 10 -stand\_call\_conf 30 -o 10252.raw.vcf

This produces a raw set of variants. However, not all variants are good variants. Sometimes variants are called in regions of extremely low coverage or using reads with sub-optimal mapping quality. Therefore, the last step is a filter that removes the bad (or somewhat suspect) variants. Here, we’ll use VCFtools to filter on a minimum genotype quality and minimum depth, two commonly used filters. Execute:

vcftools --vcf 10252.raw.vcf --recode --out 10252.filtered -minGQ 20 --minDP 5

Note that this just sets the FILTER field to a PASS or describes why it failed. Some programs will include variants even if they don’t have a PASS in their filter field. To remove them variants that do not pass the filter, execute:

vcftools --vcf 10252.filtered.recode.vcf --out 10252.final --recode --remove-filtered-geno-all

Feel free to compare this VCF to the one generated previously. VCFs can also be compressed to produce a binary VCF (.bcf). This is useful for saving space on large analyses similar to compression of SAMs.