Polishing

Polishing an alignment is done to eliminate poor base calls, resolve possible insertions or deletions from sequencing, and in general fix errors in the alignment caused by poor reads from sequencing machines. It is best done with extremely high-quality reads, and it does not matter as much if the reads are short because they are being mapped to a reference and are not meant to bridge gaps, only correct a few bases at a time. For this step in sequencing, the short reads provided by most Illumina protocols and machines are perfect for polishing because they can be generated with good depth of coverage and a reliably low amount of error. Any error that creeps in is typically found at the ends of the reads and can be trimmed off.

The program we used for polishing was called PILON, and we used version 1.22. The basic settings for require a FASTA reference file and BAM files of reads aligned to the references. Pilon then tries to identify inconsistencies in the alignment and attempts to correct them.

PILON github page: <https://github.com/broadinstitute/pilon/wiki/Requirements-&-Usage>

The output from PILON is a FASTA file that has an improved genome and also a VCF file that contains documentation of all the changes made.

PILON also has an option to produce tracks that can be viewed in a genome viewer such as IGV.

In order to get all the input files in the correct format it may be necessary to run a file conversion tool such as SAMtools.

SCRIPT EXAMPLE:

pilon -Xmx128G --genome ${reference\_genome} --bam ${sorted\_bam1} --outdir ${output\_dir} --output ${output\_prefix} --changes --fix bases --diploid --threads 1

the first term, **-Xmx** is telling pilon how much memory has been allocated to this program, in this case 128G. Note that there is no space between the term and the defined number given for memory allocation

the second term, **--genome** tells pilon that the next term is the path/name of the reference genome. In this case the reference is a previously defined variable: ${reference\_genome}.

The reason that it may be helpful to include the term as a variable is that PILON can be run multiple times with the same reference and BAM files to produce even more polished results. This method does have an upper limit on how many times it is profitable to do this before further polishing attempts lead to the re-introduction of errors.

The third term is **--bam**, and this term tells pilon that the reads that have been aligned are contained in a BAM file. Note that this BAM file, ${sorted\_bam1}, has been sorted beforehand. This is a requirement in order for PILON to run properly. In addition, there must be an index file (.bai) in the same directory as the sorted BAM file, or again PILON will not function. Running samtools index /path/to\_bam\_file will solve this problem.

The fourth term is **--outdir**, which simply defines where PILON will put the output files.

The fifth term, **--output**, is further defined here with the variable ${output\_prefix}, but it can be any string that you may want added to the output to track the files later.

The next term, **--changes**, is optional, and if selected generates a file listing all of the changes made to the output.

If you want a more detailed changelog, you can add the **-vcf** term to generate a VCF file that can be evaluated in other programs or even visualized in graph form.

The option **-vcfqe** produces a VCF file with quality weighted evidence field.

Using **--tracks** generates track files that can be viewed in a genome viewer.

The term used in this particular script to polish, **--fix bases**, is just one of many possible options for polishing. It is highly recommended that you read through the github wiki for this particular section of the program to make sure you are getting the most out of using PILON.

The **--diploid** option tells PILON that the genome is from a diploid organism. This affects SNP calls and Heterozygosity calls.

The last term in the script is **--threads**, which simply tells PILON that the computer running the program how many threads are dedicated to it.

The number of threads and memory allocated here are not necessarily optimized. To find the right amount you need, you may need to run the program and see if you run out of memory or time. For reference sake, this particular script was given 4 hours of supercomputer walltime to go with the single tread and 128 G of memory. This script finished well within the time allowed, and the reference in question was ~116Mb, and the single BAM file was 2.7Gb in size.

GOOD LUCK!