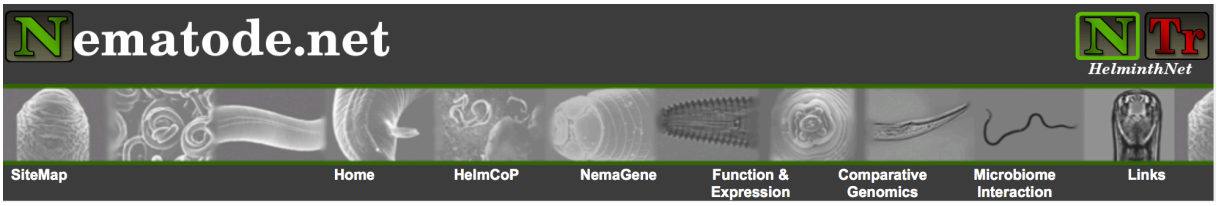
Washington University in St. Louis
School of Medicine
McDonnell Genome Institute



Bioinformatics Workshop for Helminth Genomics (2015)

Section 1: Genome

Sponsors:

The Elizabeth H. and James S. McDonnell III Genome Institute, Washington University, Campus Box 8501
4444 Forest Park Blvd., Saint Louis, MO 63108 **Phone:** 314.286.1800 **Fax:** 314.286.1810 **Web:** genome.wustl.edu

Table of contents – Curriculum

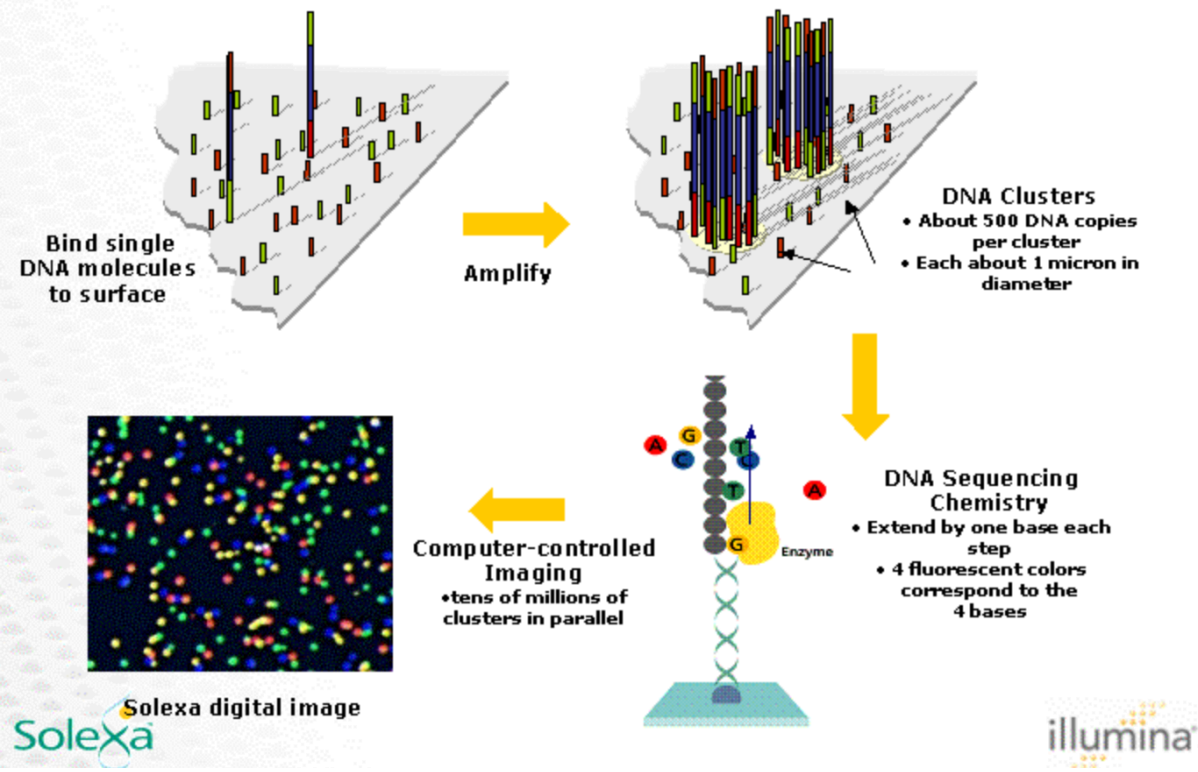
Section 1: Genome

Section 1: Genome
Module 1: Sequencing platforms

In this module, we'll introduce you to several of the sequencing platforms in use at our center and we'll look into what makes these systems unique, and what traits may be important to you when you are deciding which platform(s) to use for your project. We'll also talk about the specific case of the data we'll be using during the genomic section of this workshop. We'll describe the format in which it comes off the sequencing machine, and we'll look at one method we use for assessing the quality of raw data.

Illumina sequence-by-synthesis

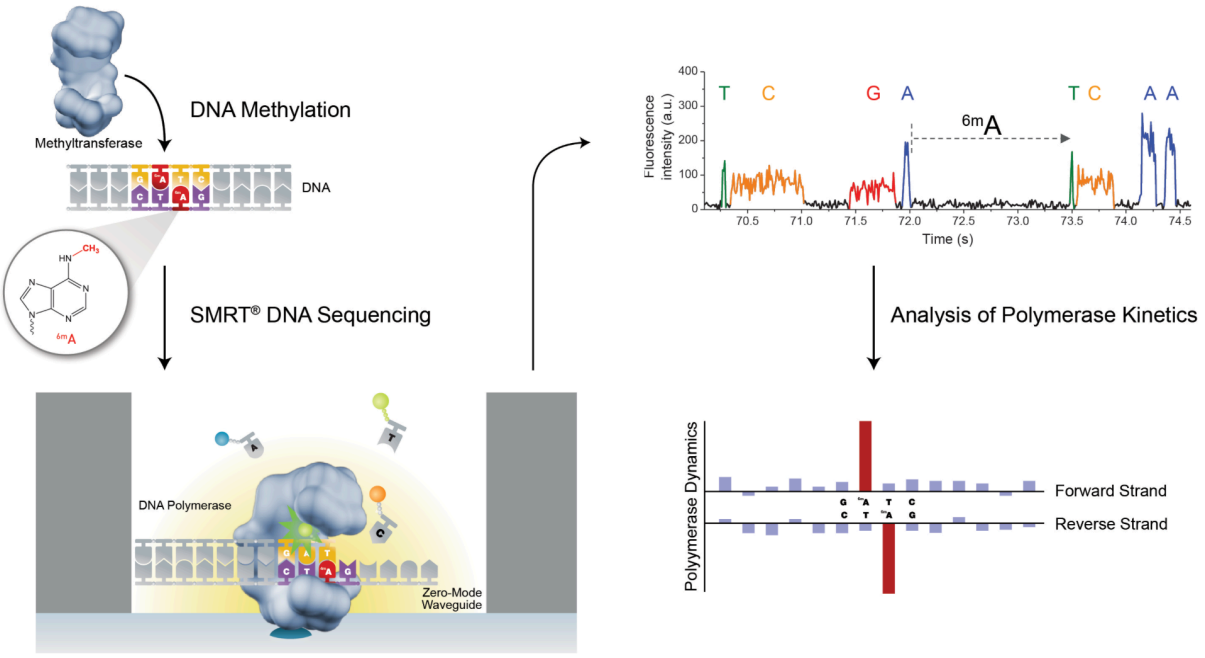
The Illumina sequencers primarily use a sequence-by-synthesis approach, using fluorescently labeled reversible-terminator nucleotides on clonally amplified DNA templates that are immobilized on an acrylamide coating on the surface of a glass flowcell. As nucleotides are incorporated onto the growing molecule attached to the flowcell, they release pulses of light that are captured by the sequencer and processed to derive base sequence.



Pacific Biosciences (PacBio) sequencing

PacBio's sequencing method is dubbed Single Molecule Real Time (SMRT) sequencing. DNA polymerase molecules, bound to a dna template, are attached to the bottom of 50nm wells termed Zero-Mode Waveguides (ZMWs). Each ZMW is small enough to see a single nucleotide being incorporated by the bound polymerase. Each of the four bases is attached to a unique

fluorescent dye, and when a nucleotide is incorporated the fluorescent tag is released and diffuses away from the observable area in the ZMW. A detector watches these fluorescent signals are records the fluorescence to determine the base incorporated. These fluorescences and their intensity are recorded over time, and these kinetics are used to calculate the base sequence.



Comparing capabilities

Each of these systems brings unique strengths to the table, and careful thought should go into your choice of sequencing platform for any given project.

For example, the Illumina platform (HiSeq2500 1T) is good for *de novo* genome sequencing if large insert size libraries used to facilitate scaffolding. However, in case of highly repetitive genomes, polymorphic genomes, or sequencing a population of individuals, the short Illumina reads would not provide optimal results. In such cases, one would need to use long read sequencing platforms such as the PacBio sequencers, and generate *de novo* PacBio assembly or hybrid Illumina/PacBio assembly. Illumina platforms are suitable for cost-effective re-sequencing of isolates if a reference genome is already available and the rapid run of HiSeq2500 (27hrs vs 6 days) or MiSeq (21 days) could be used (depending on the amount of sequence data needed to be generated) as a time-efficient platform.

Data used in 'Section 1: Genome'

The data we'll be using for the genomic section of the workshop is from the pig whipworm *Trichuris suis* which was chosen for its relatively small size compared to other worm genomes (~80Mb). For expediency's sake, some of the demonstrations will only use a subset of the full dataset that would normally be involved in the genomic analysis of a standard helminth. We'll also fast-forward through some of the lengthier steps and simply move to finished data after showing you how to start the programs involved in each step.

Getting data off the sequencing machine

Our *T. suis* data was sequenced on a HiSeq 2000 machine. That machine (as with all Illumina platforms) first generates sequence data in a format called 'Bcl'. Bcl is a binary format that contains base calls and quality scores, but is only machine readable and not anything a typical user will interact with directly. Illumina's Real-Time Analysis (RTA) software calls and records the series of cycle-specific cluster images per spot on the flowcell and converts that image data into bases and quality values in the Bcl file. It then converts that Bcl file into paired end fastq format using another Illumina program called 'bcl2fastq'. These paired-end fastq files are the starting point for our analysis.

An introduction to the Fastq format

While we plan to cover the common sequencing data formats in module 2 of this section, its useful at this point to introduce the fastq data format.

Fastq is a plain-text-based file format that contains exactly four lines per sequence record. It starts with a header line, followed by the nucleotide sequence. Then is typically a line containing a plus sign (+), and finally a line containing encoded quality values:

```
@K5HV3:00029:00029
AAAAAGGGTAAAAACGATCGTCACAGG
+
AB>>(44*44;;:/:447444C765?@-
```

The sequence and quality lines must be of the same length (i.e. one quality value per base), and the third line (beginning with a '+') is allowed to contain test (sometimes you may see this third line repeat the sequence header line after the starting plus sign). The quality values in line 4 are encoded such that each numeric value can be represented by a single character. This coding involves converting these quality scores to ascii characters.

Fastq files only support nucleotide sequence data, the format is not meant to house amino acid sequence. 'Paired end' fastq usually refers to a set of two fastq files with each file containing the sequencing data for each end of each read fragment. Very importantly, these ends must be ordered identically. There is an alternate form of paired end fastq in which the sequence of each end of the read fragment are kept one after the other within a single fastq file. This format is called 'interleaved' fastq.

Assessing the quality of newly generated fastq

We'll use the program **FastQC** to check out the quality of the paired end fastq we'll be using for the next few modules of this section. The FastQC program works on fastq files (as well as sam and bam files, which we'll discuss in the next module) and runs a number of quality control metrics we can use to assess sequencing data. Its important to remember that while FastQC uses hard, fast rules to determine when to flag a quality metric with a warning or fail notice, those warning and fail notices do NOT always mean your data is bad. A simple example of this is if you have sequence data from a polyA primed sequencing library, FastQC will likely throw up a fail flag for Kmer Content and possibly for Overrepresented Sequences because many reads will have strings of the base 'A'. You need to review FastQC results thoughtfully and with awareness of the data you are checking.

Here is how to run **FastQC**:

1.1.1: `cd ~/WORKSHOP_RESOURCES/Section_1/module_1/QC_sequence_output`

```
1.1.2: mkdir FASTQC_OUTPUT
1.1.3: fastqc -o FASTQC_OUTPUT -extract -f fastq raw_data/6p_7kb_TSAC-
Adult1-g846_g847.1.raw.fastq.gz raw_data/6p_7kb_TSAC-Adult1-
g846_g847.2.raw.fastq.gz
```

Here is how to view the results

```
1.1.4: chrome FASTQC_OUTPUT/6p_7kb_TSAC-Adult1-
g846_g847.1.raw.fastq.gz/fastqc_report.html
1.1.5: chrome FASTQC_OUTPUT/6p_7kb_TSAC-Adult1-
g846_g847.2.raw.fastq.gz/fastqc_report.html
```

Useful information:
(IUPAC code) http://www.bioinformatics.org/sms/iupac.html
(Illumina HiSeq 2000 information)
http://www.illumina.com/documents/products/datasheets/datasheet_hiseq2000.pdf
(Illumina MiSeq information)
http://www.illumina.com/documents/products/datasheets/datasheet_miseq.pdf
(PacBio RS II information) http://www.pacificbiosciences.com/products/
(FastQC) http://www.bioinformatics.babraham.ac.uk/projects/fastqc/

Section 1: Genome
Module 2: Sequence data files

This module will be a review of the common formats used to store sequencing data. We'll look at: fasta, fastq, sam & bam. You will also be introduced to the Picard & Fastx toolkits and shown how to convert between these formats.

Fastq
We've already discussed the fastq data format above. Just as a reminder, this is a four-line-per-sequence-record, nucleotide-only data format that provides both base sequence as well as quality in a single file. Commonly this format will house paired-end data, with the read from each end of the DNA fragments housed in separate, paired fastq files.

Fasta
One of the most common sequence formats out there, fasta files, are simple text files with each sequence record represented by a header line, and then a variable number of lines containing the sequence data itself. The header lines must begin with the greater-than symbol (>), and after that the line is relatively free-form. Be aware that many programs will only recognize the first white-space delimited word on the header and use that as the identifier for that sequence. For this reason, you will often find the sequence IDs as the first string on these header lines. The sequence section of the fasta format is free-form. Sequence data is often listed using a fixed number of bases per line, but its completely valid to put an entire genome's worth of sequence on a single line. Some older fasta files used to split the sequence lines with a blank every 10 characters to help make longer sequences more human-readable. You can't make many assumptions about the specific format you will see inside a fasta file. The only safe assumption is that every sequence record will be separated by a header line:

>gi|5524378|gb|AAD44166.1| Cytochrome b [Elephas maximus maximus]
ATGATGATGATGATGATGAAGACAAGGTGAGCCTAAGTAAAACTATCAAA
CGACGTCAATCAATACTTCTGTGAGGTGCGTTACGTAATCAATCAAGCAA
TAATATGATAGAGGTGGATCAAAACGATTTCAAATTGCGCTAACAAAGAG
TTAATGCTTCTTCTTATCCT

The fasta format is valid for both nucleotide and protein data.

Sam (and Bam)

The sam format (Sequence Alignment/Map) is an information rich data format for hosting nucleotide (-only) base and quality values. This format also supports the storage of alignment information, but can be used as a simple sequence data format as well. The sam format consists of 11 tab-delimited columns per sequence record (or several lines worth of 11 columns for sequence alignment data in which the same sequence maps to multiple things), as well as a number of header lines. For sequence only sam files (no alignments), there will usually be very few header lines, but for sam files hosting alignment information, there will be at least one header line per reference used in the mapping. These alignment sam files tend to have many of these header lines, so it may be useful view the sam file without the headers (I'll show you how to do this in a bit).

The columns in a sam file are setup to contain a lot of information:

Col	Field	Type	Regexp/Range	Brief description
1	QNAME	String	`[!-?A-~]{1,254}`	Query template NAME
2	FLAG	Int	`[0,2^16-1]`	bitwise FLAG
3	RNAME	String	`\*|[!-()+-<>-~][!-~]*`	Reference sequence NAME
4	POS	Int	`[0,2^31-1]`	1-based leftmost mapping POSition
5	MAPQ	Int	`[0,2^8-1]`	MAPping Quality
6	CIGAR	String	`\*|([0-9]+[MIDNSHPX=])+`	CIGAR string
7	RNEXT	String	`\*|=|[!-()+-<>-~][!-~]*`	Ref. name of the mate/next read
8	PNEXT	Int	`[0,2^31-1]`	Position of the mate/next read
9	TLEN	Int	`[-2^31+1,2^31-1]`	observed Template LENgth
10	SEQ	String	`\*|[A-Za-z=.]+`	segment SEQuence
11	QUAL	String	`[!-~]+`	ASCII of Phred-scaled base QUALity+33

Much of this complexity is in place to support the storage of alignment information. If you are dealing with sam as a format solely for hosting sequence data, the important columns are the QNAME, SEQ and QUAL columns (columns 1, 10 and 11 respectively).

The bam file format is often mentioned interchangeably with the sam file format. A bam file is simply the binary (compressed) version of a sam file. It is convenient to keep sam files in their compressed bam format to save space. In fact, many programs prefer bam as input over sam files. The samtools package provides a number of convenient tools for manipulating sam and bam files.

Converting between formats

Here is how to view the contents of a bam file (with headers):

```
samtools view -h <bam>
```

Here is how to view it without headers:

```
samtools view <bam>
```

Here is how to convert a sam file into a bam file:

```
samtools view -bSh -o <bam output> <sam input>
```

Here is how to convert a bam file back into sam:

```
samtools view -h <input bam> > <output sam>
```

Next, we'll practice some ways to convert between fastq, fasta and sam/bam using the Picard and Fastx bioinformatics toolsets. First, we'll create a sorted bam file from a set of paired end fastq files using the Picard toolset

```
1.2.1: cd
~/WORKSHOP_RESOURCES/Section_1/module_2/Convert_between_formats
1.2.2: java -jar ~/bin/picard.jar FastqToSam F1=raw_data/6p_7kb_TSAC-
Adult1-g846_g847.1.raw.fastq.gz F2=raw_data/6p_7kb_TSAC-Adult1-
g846_g847.2.raw.fastq.gz SAMPLE_NAME=6p_7kb_TSAC-Adult1-g846_g847
SORT_ORDER=queryname OUTPUT=6p_7kb_TSAC-Adult1-
g846_g847.PE.name_sorted.bam
```

Here we introduce the idea of the 'sorted' bam file. A sorted bam file is simply a bam file that has been sorted either by 'name order' or by 'coordinate order'. Coordinate order sorting is meant for alignment bam files. It re-orders the sequence records within the bam file based on their alignment positions to each reference piece, with the references themselves being ordered alphabetically. Note that if you coordinate sort a bam file that is not aligned, it will work but the ordering will not be correct. Name ordering is the only valid ordering for alignment-free bam files, and it simply orders the sequences based on their names.
Here is how to extract paired end fastq from a bam file:

```
1.2.3: java -jar ~/bin/picard.jar SamToFastq INPUT=6p_7kb_TSAC-Adult1-
g846_g847.PE.name_sorted.bam F=6p_7kb_TSAC-Adult1-
g846_g847.end1.new_fastq F2=6p_7kb_TSAC-Adult1-
g846_g847.end2.new_fastq
```

Here is how to extract fasta from fastq using the Fastx toolset:

```
1.2.4: fastq_to_fasta -i 6p_7kb_TSAC-Adult1-g846_g847.end1.new_fastq -
o 6p_7kb_TSAC-Adult1-g846_g847.end1.new_fasta
```

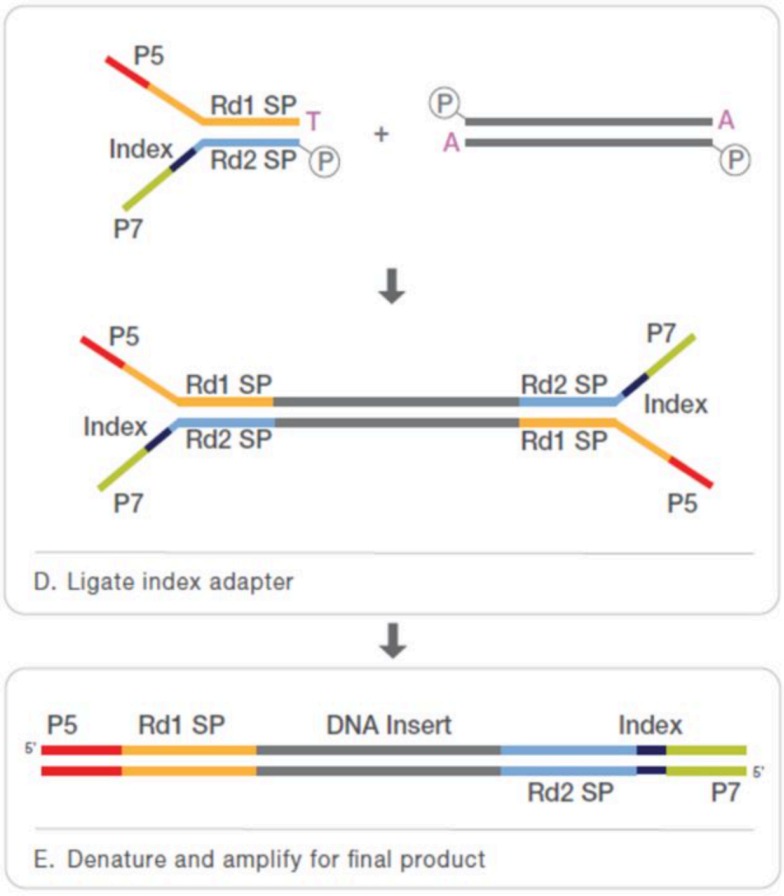
Useful information:
(Bam/Sam specification) https://samtools.github.io/hts-specs/SAMv1.pdf
(Picard Tools) http://broadinstitute.github.io/picard/command-line-overview.html#Overview
(Fastx toolkit) http://hannonlab.cshl.edu/fastx_toolkit/

Section 1: Genome
Module 3: Analytical processing of sequences

In this module, we'll demonstrate typical steps involved in processing raw sequence data from the HiSeq 2000 platform to an analysis-ready state for assembly. This tutorial will be done on a subset of the *T. suis* data that will be used in the full assembly. In a real world usage case you'd most likely be running this process on multiple pairs of fastq files.

Why data is not analysis-ready directly off the sequencing machine
Raw sequence-data from a sequencing machine is technically capable of being used directly in most downstream analyses, but there are a number if factors that make that very ill advised.

Sequencing adapters
To prepare DNA material for sequencing, a sequencing library must be made. In our example case, we used the TruSeq genomic library preparation kit for the HiSeq 2000. The DNA sample is first fragmented into pieces roughly 200bp in length. Then TruSeq universal adapters and a specific version of the TruSeq index adapter are ligated onto each end of the fragments via a single base (A) overhang.



The adapter–DNA fragment complex is then denatured and amplified to produce a final product containing the DNA insert, end-specific sequencing primers on either end, as well as a specific index for use in identifying this library out of a pool of libraries.

These TruSeq sequence adapters are normally not visible in the final, sequenced product, because the sequencing primers are immediately adjacent to the DNA insert. However, if some fraction of the DNA inserts are shorter than the expected length, it is possible that sequencing can go all the way across the insert and read into the adapter sequence on the far end. Many of the analyses we typically want can be negatively affected by having adapter sequence left within the reads. It decreases mapping efficiency, confuses assemblies, etc. It is a good practice to identify and trim off any adapter sequence that may be present in your reads.

Low sequence quality

During sequencing, each called base is typically assigned a quality score that refers to the likelihood that the base was correctly called by the sequencer. The common value used to represent these per-base confidences is the Phred score.

$Q = -10 * \log_{10} P$
Q → Phred score
P → probability of an error occurring
Eg. Phred 20 implies that you are likely to see 1 error per 100 bases, Phred 30 implies 1 error per 1000 bases, Phred 40 implies 1 error per 10000 bases

Poor quality sequence can interfere with downstream analysis as seriously as untrimmed adapters. It makes mapping less reliable, confuses assemblers, and is a major impediment to variant calling. As with adapter sequence, it is a good practice to trim off low quality sequence that may be present in your reads.

Length filtering

After trimming reads for adapter and low quality, its possible that some of the reads have been cut down to a very short size. We typically apply a length filter requiring that after the above trimming there be at least 60 bases of read left, otherwise we discard the sequence as a 'short read'. Note that the 60bp threshold is the value we will use for the *T. suis* dataset. If your reads are shorter or longer, you may need to adjust that cutoff.

Low sequence complexity

These are regions that have an unusual composition that can create problems in sequence similarity searching (as well as other kinds of analyses). These regions contain low information content and can be 'sticky' during alignments. It is a good practice to filter your sequence data for low complexity regions before running downstream analysis.

Contaminant filtering

Finally, we filter our reads to remove contaminant. By contaminant, we mean any read whose source is not what we expect (in our case, our reads should originate from *T. suis*). Typical sources of contamination are:

Host, bacteria, other (environmental contaminants). Also, for RNA-Seq work, it is often common to filter for 18/16s ribosomal data. This is because the amount of ribosomal sequence present can sometimes dwarf the amount of actual expressed transcript amongst your reads. So it is helpful for downstream analysis to get rid of it.

For this demonstration, we'll be screening for host contaminant only, in this case from pig. In general, you will want to pick and choose the contaminant db's you'll use based on the situation of each project. To save compute resources, we only want to screen for contaminants we expect might be a problem.

Finally, be aware that contamination screening should be done after filtering out adapters, low quality and low complexity sequence. Those earlier issues, if left unfixed, can impede the identification of a read as contaminant.

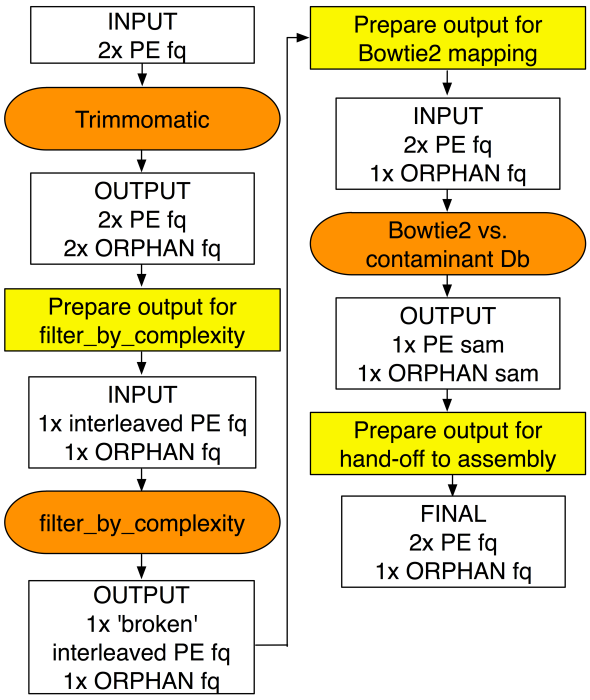
Discard both ends or only one?

One thing that must be considered when filtering and screening your data is whether to discard both ends of a paired end set, or only one. Because most sequence data generated is actually sequence from both ends of a single DNA insert, you need to think about whether a problem seen on one end should be considered to apply to both ends or not. In general, issues with adapter, sequencing error, and low complexity are not issues that necessarily affect both ends of a sequence insert. In those cases, we usually will just discard the problem end and retain the other. In the case of one end being identified as a contaminant, we normally will consider both ends contaminants and discard them both.

Processing raw reads into an analysis-ready state

Now we're going to walk through typical steps we'd use to prepare our *T. suis* reads for assembly. To do this we need to accomplish these things:

a) remove any adapters that may have been introduced during sequencing library preparation
b) remove low quality, terminal regions
c) apply a length filter to remove short reads after trimming
d) remove reads of low sequence complexity
e) remove reads that originate from host organism



We'll use the program Trimmomatic for adapter removal, quality trimming and length filtering. The filter_by_complexity script from the seq_crumbs package will remove reads of low sequence complexity, and we'll use the Bowtie2 aligner to map the trimmed reads against a host database. Between these steps some file manipulation is required to get the sequences into the format needed for the next step. This data shuffling will be done using parts of the samtools & KHMER packages, as well as some old fashioned command line unix.

The analysis-ready output

This processing will result in a set of paired end fastq, and an extra fastq of orphaned reads whose mates were discarded (due to filtering steps that removed only a single end from a pair). This process can be messy on a technical level due to the need to convert data between the bam & fastq formats, and the need to keep the paired-end data synchronized and free of orphans. In practice, we would normally assemble all these steps into a single pipeline script. For the purposes of this demonstration, we'll walk through each step manually.

Be aware that there are alternatives to the software we're showing for most of these steps. The programs we're using are generally robust, but you may want to experiment with other options for your data. No tool does a perfect job, and you may be able to find tools that perform better or more efficiently for your specific dataset.

Finally, it's often reasonable to simply work only with paired end data, and discard the small fraction of orphaned reads generated at each step. This simplifies the process at the expense of

a small fraction of your reads. This is actually a fairly common practice, especially if you find yourself doing an extra hour of coding work to preserve a few thousand reads out of 200 million reads.

Processing the data

Here are the steps involved in running Trimmomatic and preparing the output for the next step. This step trims off adapter, quality trims and filters the trimmed reads based on length.

```
1.3.1: cd
~/WORKSHOP_RESOURCES/Section_1/module_3/Processing_genomic_data_to_cle
aned_state
1.3.2: java -jar ~/bin/trimmomatic-0.33.jar PE -threads 8 -phred33 -
trimlog TRIMLOG.txt raw_data/6p_7kb_TSAC-Adult1-
g846_g847.1.raw.fastq.gz raw_data/6p_7kb_TSAC-Adult1-
g846_g847.2.raw.fastq.gz 6p_7kb_TSAC-Adult1-g846_g847.PE_end1.fastq
6p_7kb_TSAC-Adult1-g846_g847.ORPHANS_end1.fastq 6p_7kb_TSAC-Adult1-
g846_g847.PE_end2.fastq 6p_7kb_TSAC-Adult1-
g846_g847.ORPHANS_end2.fastq
ILLUMINACLIP:databases/TruSeq_adapters.fna:2:30:10 SLIDINGWINDOW:5:20
LEADING:20 TRAILING:20 MINLEN:60
1.3.3: cat 6p_7kb_TSAC-Adult1-g846_g847.ORPHANS_end1.fastq
6p_7kb_TSAC-Adult1-g846_g847.ORPHANS_end2.fastq >
Tsuis_genomic_7kb_insert.trimmomatic_ALL_ORPHANS.fastq
1.3.4: java -jar ~/bin/picard.jar FastqToSam F1=6p_7kb_TSAC-Adult1-
g846_g847.PE_end1.fastq F2=6p_7kb_TSAC-Adult1-g846_g847.PE_end2.fastq
SAMPLE_NAME=Tsuis_genomic_7kb_insert SORT_ORDER=coordinate
OUTPUT=Tsuis_genomic_7kb_insert.trimmomatic_PE_coord_sorted.bam
1.3.5: java -jar ~/bin/picard.jar SamToFastq
INPUT=Tsuis_genomic_7kb_insert.trimmomatic_PE_coord_sorted.bam
INTERLEAVE=true
FASTQ=Tsuis_genomic_7kb_insert.trimmomatic_PE_interleaved.fastq
```

Next we filter out low complexity data using filter_by_complexity from the seq_crumbs package, and then prepare the output for the final Bowtie2 mapping step

```
1.3.6: filter_by_complexity -o
Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.brokenPE_interleav
ed.fastq --paired_reads --fail_drags_pair False
Tsuis_genomic_7kb_insert.trimmomatic_PE_interleaved.fastq
1.3.7: filter_by_complexity -o
Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.ORPHANS.fastq
Tsuis_genomic_7kb_insert.trimmomatic_ALL_ORPHANS.fastq
1.3.8: source /home/ec2-user/bin/KHMER/khmerEnv/bin/activate
1.3.9: extract-paired-reads.py -f
Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.brokenPE_interleav
ed.fastq
1.3.10: deactivate
```

1.3.11: cat
Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.ORPHANS.fastq
Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.brokenPE_interleaved.fastq.se >
Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.ALL_ORPHANS.fastq
1.3.12: paste - - - - - - - - < Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.brokenPE_interleaved.fastq.pe | tee >(cut -f 1-4 | tr "\t" "\n" > Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.PE_end1.fastq) | cut -f 5-8 | tr "\t" "\n" > Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.PE_end2.fastq

Finally, we will map the cleaned reads against a host database, pig in this case, and remove all reads and read pairs that have either end detected as a contaminant. We'll use Bowtie2 for this mapping, and we'll prepare the final, cleaned & contaminant free data for assembly

1.3.13: bowtie2-build Sus_scrofa.Sscrofa10.2.dna_rm.toplevel.fa Sus_scrofa.Sscrofa10.2.dna_rm.toplevel
1.3.14: bowtie2 -q -x databases/Sus_scrofa.Sscrofa10.2.dna_rm.toplevel -1 Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.PE_end1.fastq -2 Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.PE_end2.fastq -S Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.mapped_to_host.PE.sam
1.3.15: bowtie2 -q -x databases/Sus_scrofa.Sscrofa10.2.dna_rm.toplevel -U Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.ALL_ORPHANS.fastq -S Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.mapped_to_host.ORPHANS.sam

1.3.16: samtools view -bSh -o Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.mapped_to_host.ORPHANS.bam Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.mapped_to_host.ORPHANS.sam
1.3.17: samtools sort Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.mapped_to_host.ORPHANS.bam Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.mapped_to_host.ORPHANS.sorted
1.3.18: bamtools filter -in Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.mapped_to_host.ORPHANS.sorted.bam -out Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.host_free.ORPHANS.bam -isMapped false
1.3.19: java -jar ~/bin/picard.jar SamToFastq INPUT=Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.host_free.ORPHANS.bam FASTQ=Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.host_free.ORPHANS.fastq

```
1.3.20: samtools view -bSh -o
Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.mapped_to_host.PE.
bam
Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.mapped_to_host.PE.
sam
1.3.21: samtools sort
Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.mapped_to_host.PE.
bam
Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.mapped_to_host.PE.
sorted
1.3.22: bamtools filter -in
Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.mapped_to_host.PE.
sorted.bam -out
Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.host_free.PE.bam -
isMapped false -isMateMapped false
1.3.23: java -jar ~/bin/picard.jar SamToFastq
INPUT=Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.host_free.PE
.bam
FASTQ=Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.host_free.PE
_end1.fastq
SECOND_END_FASTQ=Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.h
ost_free.PE_end2.fastq
```

Evaluating our analysis-ready data

Now that we've processed our data to an analysis-ready state, lets run FastQC again on the final output and compare it back to the FastQC results from the original, raw data

```
1.3.24: cd /home/ec2-
user/WORKSHOP_RESOURCES/Section_1/module_3/Processing_genomic_data_to_
cleaned_state
1.3.25: mkdir FASTQC_OUTPUT
1.3.26: fastqc -o FASTQC_OUTPUT -extract -f
Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.host_free.PE_end1.
fastq
Tsuis_genomic_7kb_insert.trimmomatic_and_complexity.host_free.PE_end2.
fastq
```

We'll then use the chrome browser (as before) to compare the final paired fastq files to the original, raw paired fastq files.

Useful information:
(Trimmomatic) http://www.usadellab.org/cms/?page=trimmomatic
(Bowtie2) http://bowtie-bio.sourceforge.net/bowtie2/index.shtml
(seq_crumbs) https://bioinf.comav.upv.es/seq_crumbs/available_crumbs.html
(bamtools) https://github.com/pezmaster31/bamtools

Section 1: Genome
Module 4: Genome assembly

There are a lot of choices when deciding on a genome assembler. Considerations include the predicted genome size, the technology type, and the cost (computational, and paying for the assembler). Today's demonstration will be using ALLPATHS-LG, which is a de Bruijn Graph assembler for large genomes. ALLPATHS-LG requires paired end reads from at least one fragment and one jumping library sequenced on the Illumina platform. The use of multiple libraries enables ALLPATHS-LG to build a higher quality assembly. When using ALLPATHS_LG, our recommended sequence coverage requirements are: 45x fragments, 45x 3-8kb and 10x-20x lrg insert ie. 5kb+. In our assembly, we will be using 11,898,407 fragment read pairs, 4,960,173 3kb read pairs and 2,975,142 7kb read pairs

ALLPATHS-LG requires a specific format for input sequence data files in order to run the assembler. PrepareAllPathsInputs.pl, an ALLPaths script, will be run after we begin by setting up two dependency files:

Dependency File #1: `in_groups.csv`

```
100,Illumina_011,/home/ec2-
user/WORKSHOP_RESOURCES/Section_1/module_4/Tsuis/all_path_data/13p_fra
gment.*.trimPaired.fastq.gz
200,Illumina_012,/home/ec2-
user/WORKSHOP_RESOURCES/Section_1/module_4/Tsuis/all_path_data/33p_3-
5kb_.*.trimPaired.fastq.gz
300,Illumina_013,/home/ec2-
user/WORKSHOP_RESOURCES/Section_1/module_4/Tsuis/all_path_data/6p_7kb.
*.trimPaired.fastq.gz
```

Notes: This file does not require a header with each field type.
Group name: unique name for data set (free form)
Library name: library name for data set (free form but good practice to use some identifying nomenclature)
File name: absolute path to data file. Wildcard characters * and ? are accepted in the name of the file but NOT the file extension.
Supported extensions are .bam, .fasta, .fa, .fq, .fastq.gz, and fq.gz (all case specific). Also, if you use .fasta or .fa, the script expects a corresponding .quala or .qa file to exist for each respective file.

Dependency File #2: `in_libs.csv`

```
library_name,project_name,organism_name,type,paired,frag_size,frag_std
dev,insert_size,insert_stddev,read_orientation,genomic_start,genomic_e
nd
Illumina_011,Awesome,T.suis,fragment,1,205,10,,,inward,,
Illumina_012,Awesome,T.suis,jumping,1,,,7475,500,outward,,
Illumina_013,Awesome,T.suis,jumping,1,,,2833,500,outward,,
```

Notes: Every field is required in this file. A header is used, and each field must be represented in the data entered (a comma separated field can be left blank; see example above).
Library_name: must match same field in in_group.csv
Project_name: free form name
Organism_name: organism
Type: informative field only ie fragment, jumping EcoP15, etc.
Paired: 0 = unpaired reads; 1: paired reads
Frag_size: average # of bases in the fragment library
Frag_stddev: estimated standard deviation of the fragment sizes
Insert_size: average # of bases in the jumping library inserts (if larger than 20kb the library is considered Long Jumping)
Insert_stddev: estimated standard deviation of the insert sizes
Read_orientation: inward or outward
Genomic_start: index of the FIRST genomic base in the reads. If not zero, then all bases before the genomic start will be trimmed off
Genomic_end: index of the LAST genomic base in the reads. If not zero, then all bases after the genomic end will be trimmed off

With these two files prepared, you can now run:

1.4.1: `PrepareAllPathsInputs.pl DATA_DIR=/home/ec2-user/WORKSHOP_RESOURCES/Section_1/module_4/Tsuis/all_path_data PLOIDY=2`

Other optional settings include:

PICARD_TOOLS_DIR (use version 1.101) if you are using .bam files in the .csv files made above.
INCLUDE_NON_PF_READS=1 allows you to use the orphan reads kept in the previous module.
GENOME_SIZE, FRAG_COVEAGE, JUMP_COVERAGE, and LONG_JUMP_COVERAGE used together you can set the desired coverage percentage based on the estimated size set for GENOME_SIZE

Now we can start the assembly:

1.4.2: `RunAllPathsLG PRE=/home/ec2-user/WORKSHOP_RESOURCES/Section_1/module_4 REFERENCE_NAME=Tsuis DATA_SUBDIR=all_path_data RUN=myrun SUBDIR=attempt1`

Notes: All of the **ALLPATHS** arguments are to set the pipeline directory names. If your ALLPATHS run fails at any point, you can troubleshoot the issue and then restart ALLPATHS and it will restart on the stage that failed (as long as you don't delete any of the directories/data that was produced up to that point).
Use the following command which adds "OVERWRITE=True":

1.4.3: `RunAllPathsLG PRE=/home/ec2-user/WORKSHOP_RESOURCES/Section_1/module_4 REFERENCE_NAME=Tsuis DATA_SUBDIR=all_path_data RUN=myrun SUBDIR=attempt1 OVERWRITE=True`

This assembly took 5.3 hours. When the assembly finishes it will be found at the following location:

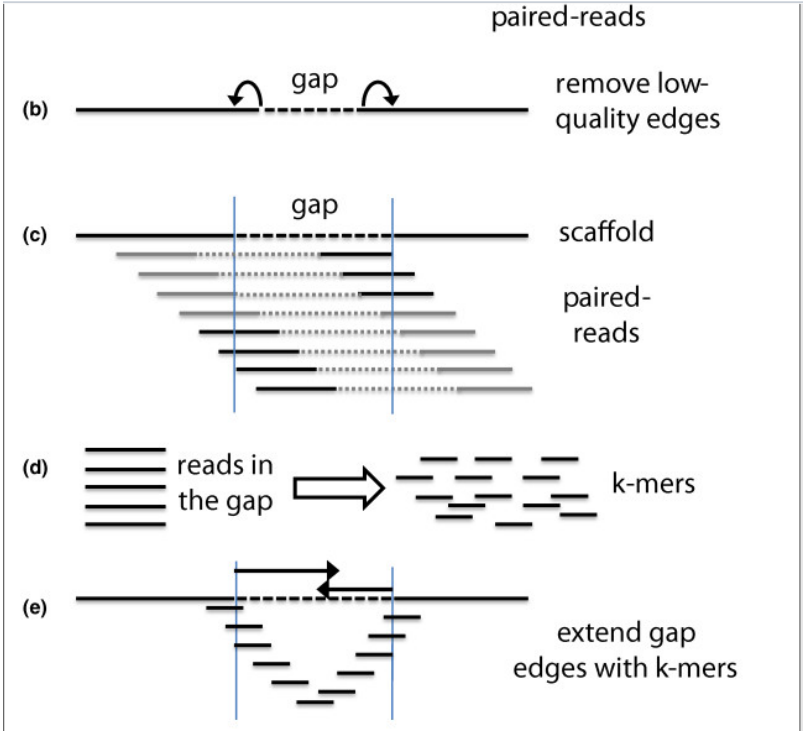
/home/ec2-user/WORKSHOP_RESOURCES/Section_1/module_4/Tsuis/all_path_data/myrun/ASSEMBLY/attempt1/final.assembly.fasta

Useful information:
ftp://ftp.broadinstitute.org/pub/crd/ALLPATHS/Release-LG/AllPaths-LG_Manual.pdf

Quality Assessment
Assembly improvement and QC of de novo assemblies go hand in hand since high-quality draft genomes lead to more successful and accurate annotation. We use a combination of CEGMA, N50, and RNA mapping to assess the quality of an assembly.

CEGMA (Core Eukaryotic Genes Mapping Approach) uses a defined set of 458 single-copy, conserved eukaryotic genes, and searches for orthologs of these proteins in the de novo genome assembly. Since these proteins are conserved across eukaryotes ranging from yeast to plants to humans, the completeness of this protein set in a draft genome is a useful indicator of the genome quality. CEGMA produces a completeness report, but we prefer to parse the cegma.gff file against the core proteins to get a count of the CEGs (Core Eukaryotic Genes) and lcCEGs found in the assembly.

1.4.4: `cegma --genome final.assembly.fasta -threads 8 &`



Schematic overview of the GapFiller algorithm. (a) The input data consist of a set of scaffold sequences containing gapped nucleotides and one or more sets of paired-end and/or mate-pair reads. **(b)** As a pre-processing step low quality nucleotides are removed from the sequence edges, thus enlarging the gap of ten nucleotides from each side. It should be stressed that the contig ends resulting from a draft assembly often contain misassemblies. **(c)** Paired-reads are aligned to the scaffolds and retained if one pair aligns to a scaffold sequence (dark grey) and one pair to a gapped region (black). **(d)** All pairs that are estimated to fall in the gapped regions are split into *k*-mers and used for gap filling. **(e)** The gap is closed from each edge by using *k*-mers that present a sequence overlap of size (*k*-mer - 1) and one nucleotide overhang. Gaps are closed if the right and left extensions can be merged and correspond to the estimated sequence gap.

Boetzer and Pirovano *Genome Biology* 2012 **13**:R56 doi:10.1186/gb-2012-13-6-r56
Download authors' original image

N50 is a basic statistic for describing how contiguous an assembly is. The longer the N50 is, the better the assembly.

RNA mapping looks at the percent of gene contained within the assembly

Assembly Improvement
After assessing an assembly, we can take advantage of numerous assembly improvement tools. Two open source options that we use are **GapFiller** and **PBJelly**. PBJelly (part of PBSuites) is able to fill gaps and merge scaffolds utilizing long reads (which is particularly useful for PacBio data). For this project, we do not have any available PacBio data, so we will be utilizing GapFiller instead. The image below illustrates how GapFiller fills the contig gaps:

We will start by creating the libraries.txt file:

```
Libraries File: libraries.txt
lib100 bowtie
/gscmnt/gc2546/mitrevalab/research/t_suis_class/25p_fragment_lib_TSAC-
Adult1-g846_g847.1.raw.fastq
/gscmnt/gc2546/mitrevalab/research/t_suis_class/25p_fragment_lib_TSAC-
Adult1-g846_g847.2.raw.fastq 205 0.3 FR
lib200 bowtie /gscmnt/gc2546/mitrevalab/research/t_suis_class/65p_3-
5kb_TSAC-Adult1-g846_g847.1.raw.fastq
/gscmnt/gc2546/mitrevalab/research/t_suis_class/65p_3-5kb_TSAC-Adult1-
g846_g847.2.raw.fastq 7475 0.5 RF
lib300 bowtie
/gscmnt/gc2546/mitrevalab/research/t_suis_class/12p_7kb_TSAC-Adult1-
g846_g847.1.raw.fastq
/gscmnt/gc2546/mitrevalab/research/t_suis_class/12p_7kb_TSAC-Adult1-
g846_g847.2.raw.fastq 2833 0.5 RF
```

Notes:
Library Name: free form
Mapper: bowtie or bwa
Path to both mate pairs files
Insert size
Error
Read orientation

We then run **GapFiller** using:

```
1.4.5: GapFiller.pl -l libraries.txt -s final.assembly.fasta -T 8 -b
Tsuis -i 5
```

Notes: –l is the file made above; –s assembly file; –T threads; –b directory and root file name; –i iterations. Runtime varies based on number of gaps and amount of data used

Useful links:
http://korflab.ucdavis.edu/datasets/cegma/README

Section 1: Genome
Module 5: Genome annotation

"A beginner's guide to eukaryotic genome annotation" http://www.nature.com/nrg/journal/v13/n5/full/nrg3174.html is a great resource. The first step when annotating a genome is to identify repeat sequences, because they can interfere with gene predictors and evidence alignment.

Masking repeats
Tandem Repeat Finder (TRF): Start by using TRF to mask short interspersed tandem repeats:

1.5.1: `trf Tsuis.gapfilled.final.fa2 7 7 80 10 50 500 -d -m -h  >> TRF.stdout`

Now we need to create a blast database for **RepeatModeler**:

1.5.2: `makeblastdb -in final.assembly.fasta.2.7.7.80.10.50.500.mask -dbtype nucl`

Running **RepeatModeler** (run time is 24-36 hours):

```
RepeatModeler -database Tsuis.gapfilled.final.fa.masked.fasta >> RM_stdout
```

Repeatmodeler will create an RM_[PID].[DATE]/ directory,
(e.g. RM_10825.ThuAug271528572015/)

Once RepeatModeler has completed, you will need to QC the output to check for repeats that are really genes (gene families) or RNA features.

The following are the screening steps for QC:

Blastx vs nr for protein coding genes:

1.5.3: `blastx nr consensi.fa.classified E=10e-5 -o consensi.fa.classified.nrcheck.blast.out`

Blastn vs RNA database for ribosomal or other RNA genes (Rfam.fasta comes with the Rfam download):

1.5.4: `blastn Rfam.fasta consensi.fa.classified 10e-5 -o $1.rnacheck.blast.out`

Retrotransposon check:

1.5.5: `blastx transposonDb consensi.fa.classified E=10e-5 -o $1.retrocheck.blast.out`

The final file output from RM is consensi.fa.classified file in the RM directory (e.g. .M_10825.ThuAug271528572015/consensi.fa.classified). We then screen the blast.out files with tools that look at P >=0.01 identity/coverage (50% PID/20% Identity) and naming that is known to be acceptable and database types that lead us to believe the protein has been checked:

"unknown", "hypothetical", "oxidase", "histone", "kinase", "protease", "reductase", "RNA", "synthase", "ATPase", "phosphatase", "cytochrome", "ribosomal", "titin", "extensin", "abductin", "tRNA", "drosophila", "nucleosome", "transferase", "unnamed", "polyprotein", "putative", "peptide", "resolvase", "alpha", "beta", "fusion", "lactamase", "galact", "integrase", "ref", "emb", "dbj", "gb", "pir", "prf", "sp", "pdb", "intron","synthetase"

1.5.6: `mkdir RepeatMasker`
1.5.7: `cd RepeatMasker`

1.5.8: `RepeatMasker -lib repeats.lib trf.masked.fasta >>RepeatMasker.stdout`

Note: The input sequence can be split into chunks to expedite.

RepeatMasker outputs the following files:
D918.newname.fsa.masked - Masked fasta
D918.newname.fsa.tbl - Summary table gives total %masked and breakdown of types
D918.newname.fsa.log - run output
Other output files with details of repeats/positions etc.
D918.newname.fsa.cat D918.newname.fsa.out D918.newname.fsa.ref

Useful links:
https://tandem.bu.edu/trf/trf.html
http://www.repeatmasker.org/RepeatModeler.html
http://www.repeatmasker.org/webrepeatmaskerhelp.html

We also annotate non-coding RNAs using the Rfam and tRNA scan. We mask these predictions before running the predictor programs, in order to further simplify the regions the predictors have to look at.

Rfam - http://nar.oxfordjournals.org/content/43/D1/D130

1.5.9: `rfam_scan -f tab -o Rfam.out File.fasta`

-f specifies format
-o specifies output location
The last argument is just the sequence file to use

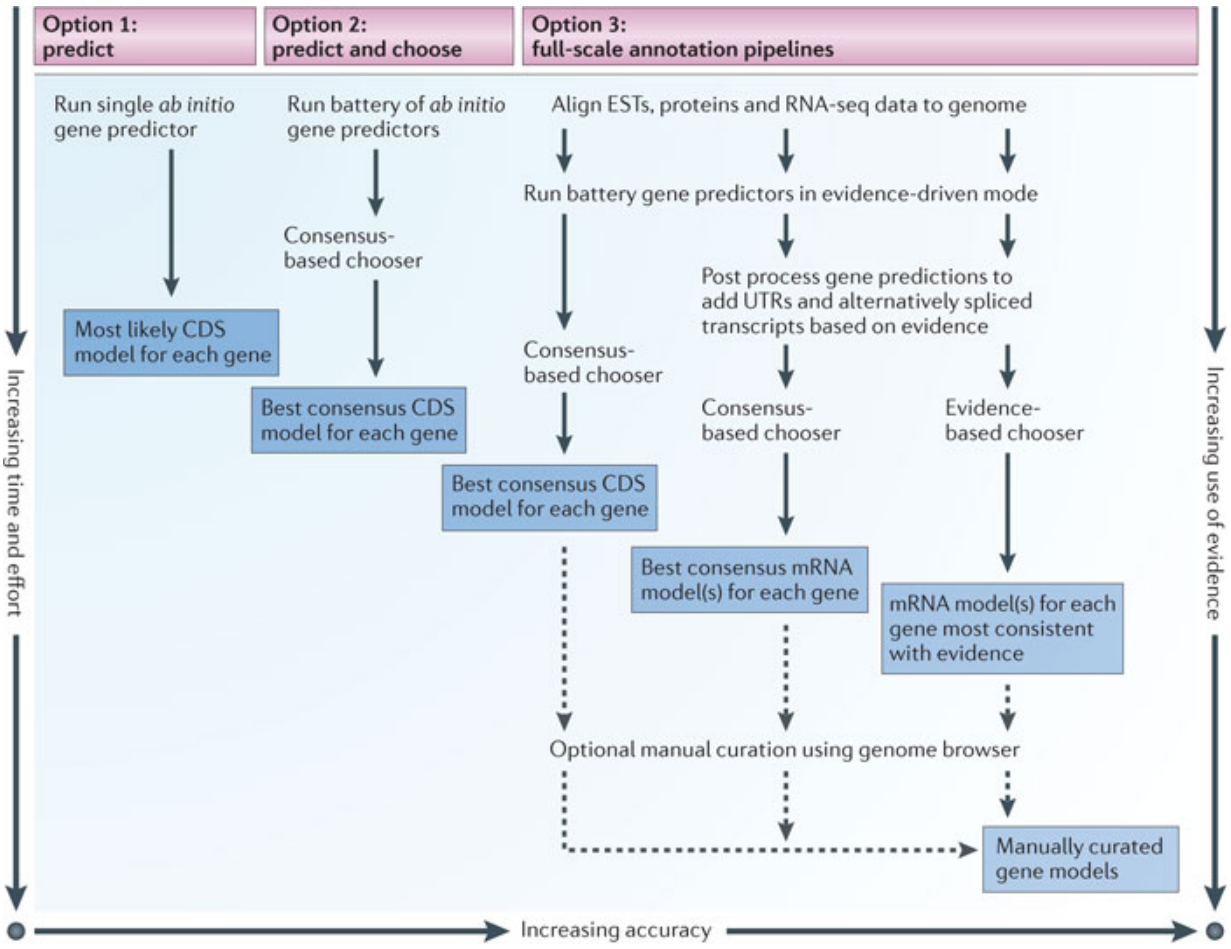
Notes: For rfam scan, we modified the script so that it skips the rare group II introns, because if greatly reduces the run time.

We can scan a sequence file for tRNAs using tRNAscan, EufindtRNA & tRNA covariance.

tRNAscan - http://lowelab.ucsc.edu/tRNAscan-SE/Manual

1.5.10: `tRNAscan-SE -o tRNAscan.output File.fsa`

Annotation
Producing gene predictions to produce a high quality final set of gene annotations.



A beginner's guide to eukaryotic genome annotation. M Yandell & D Ence *Nature Reviews Genetics* 13, 329-342 (May 2012).

We will perform annotation using **Maker** (M. Yandell et. al., 2007)
http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2134774/pdf/188.pdf

First, we generate config files for **Maker**:

```
1.5.11: maker -CTL
```

This creates 3 default config files:

maker_bopts.ctl : blast type and cut-off
maker_exe.ctl : program paths
maker_opts.ctl : all other parameters

The -CTL option will give you default parameters. You will need to set up paths in each file to match the system you are on (paths to blast databases, etc). For our maker runs, we only need to do the -CTL once, and then we copy the ctl files to the new directories so we don't have to update paths for blast exe's etc. We only need to change the maker_opts.ctl file for blast db's.

Change any parameters and/or paths which are different from the working copy, including:

path to sequence file
protein database path
EST database path
alt-est database path if needed
ab initio predictors being run
ab initio corresponding model files
model_gff and/or pred_gff or other _gff files
evidence predictors

Open maker_opts.ctl and add path to these lines

Find this line:
#-----EST Evidence (for best results provide a file for at least one):

```
1.5.12: est= #set of ESTs or assembled mRNA-seq in fasta format
```

Beneath this EST Evidence section, also change:
#-----Protein Homology Evidence (for best results provide a file for at least one)

```
1.5.13: protein=  #protein sequence file in fasta format (i.e. from
mutiple oransisms)
```

To run any of the predictors: **Snap**, **Fgenesh**, **Augustus** you need to train them. **Fgenesh** is a commercial predictor that you would need to purchased, but snap is free and maker also works with **GeneMark** and others, but those are the most common. We are not going to go perform overpredictor training today.

To run maker:

```
1.5.14: maker --RM_off -g File.fasta maker_bopts.ctl maker_exe.ctl
maker_opts.ctl
```

Here is some information on the directory structure and the files that maker outputs:

Path/Maker

maker_bopt.ctl
maker_ext.ctl
maker_opts.ctl

GENOME.maker.output/ - contains all output for a given run of MAKER

maker_bopts.log : These are logs of the control files used for this run of MAKER
maker_opts.log
maker_exe.log

seen.dbm

sequnce_maker_length_99.db
sequnce_maker_length_99_master_datastore_index.log - log of MAKER run progress as well as an index for traversing through the output

mpi_blastdb/ - Contains fasta indexes and error corrected fasta files built from the EST and protein database provided by the user.

*.mpi.10/ - contains indexed database files
nematode_protein_new.mpi.10/ - contains indexed database files

<Sequence_name>._datastore/ - contains subdirectories that hold the output for each individual contig of the input fasta file. See DATASTORE DIRECTORY STRUCTURE section in README for more information

08/ 25/Contig#/ - first two directories; numbers/letters vary

run.log
<Sequence_name>.gff - a gff file that can be loaded into GMOD, GBROWSE, or Apollo
<Sequence_name>.maker.snap.proteins.fasta - a fasta file of ab-inito predicted protein sequences from program

<Sequence_name>.maker.snap.transcripts.fasta - a fasta file of ab-inito predicted transcript sequences from program

<Sequence_name>.maker.transcripts.fasta - a fasta file of the MAKER annotated transcript sequences

<Sequence_name>.maker.proteins.fasta - a fasta file of the MAKER annotated protein sequences

<Sequence_name>.maker.non_overlapping_ab_initio.proteins.fasta - a fasta file of filtered ab-inito protein sequences that don't overlap maker annotations

<Sequence_name>.maker.non_overlapping_ab_initio.transcripts.fasta - a fasta file of filtered ab-inito transcript sequences that don't overlap maker annotations

theVoid.Contig#/ - a directory containing all of the raw output files produced by MAKER, including BLAST reports, SNAP output, exonnerate output and the masked genomic sequence.

Explaining GFF3 files

http://www.broadinstitute.org/annotation/argo/help/gff3.html

Field Descriptions (Note: Except for the last field [9], all gff flavors are the same):

1. **seqname** - The name of the sequence. Typically a chromosome or a contig. Argo does not care what you put here. It will superimpose gff features on any sequence you like.
2. **source** - The program that generated this feature. Argo displays the value of this field in the inspector but does not do anything special with it.
3. **feature** - The name of this type of feature. The official GFF3 spec states that this should be a term from the SOFA ontology, but Argo does not do anything with this value except display it.
4. **start** - The starting position of the feature in the sequence. The first base is numbered 1.
5. **end** - The ending position of the feature (inclusive).
6. **score** - A score between 0 and 1000. If there is no score value, enter ".".
7. **strand** - Valid entries include '+', '-', or '.' (for don't know/don't care).
8. **frame** - If the feature is a coding exon, *frame* should be a number between 0-2 that represents the reading frame of the first base. If the feature is not a coding exon, the value should be '.'. Argo does not do anything with this field except display its value.
9. ***GFF3: grouping attributes*** *Attribute keys and values are separated by '=' signs. Values must be URI encoded.quoted. Attribute pairs are separated by semicolons. Certain, special attributes are used for grouping and identification (See below). This field is the one important difference between GFF flavors.*

Special Field 9 Attributes:

The first special thing about field 9 attributes is that they can be associated with transcripts. Previous flavors of GFF restricted attributes to the lowest level subfeature (exons).

Any key=value attribute pair will be displayed by argo, but the following have special meaning:

1. ID - unique identifier for this feature.
2. Parent - identifier of parent feature.
3. Name - used as the feature label in the feature map.

Section 1: Genome
Module 6: Functional annotation

This module will review two standard methods for assigning functional annotations to a de novo geneset. We'll run a protein vs. protein alignment using the NCBI's BLASTP+, and we'll discuss the interproscan program and take a look at typical interproscan output and how to make use of it.

Using NCBI's BLASTP+ to assign functional annotation

One common method of assigning a function to a set of de novo gene calls is simply by mapping them to an already annotated set of genes from closely related organisms. We'll go over the details of actually running a blastp in a bit, but first we'll review how to locate and prepare a database for this mapping.

If you happen to have a highly conserved organism's gene set handy that happens to already be well annotated, you may not need to do anymore digging. For example, if you are working with a non-parasitic nematode, you can't do much better than to simply use the highly curated and well annotated *C.elegans* gene set for this mapping. Bu t if you are working with an organism not in that happy circumstance (as most of us are, all the time), the next best thing is to walk the lineage of your species using GenBank's Entrez Records which are avaible when using a taxonomy search. Lets do this now:

1. Open a browser on your laptop
2. Go to the NCBI website at http://www.ncbi.nlm.nih.gov
3. Enter "*Trichuris suis*" in the search box at the top of the screen, and set the search menu to "Taxonomy", then click "Search"
4. Click through the "*Trichuris suis*" link
5. Notice the "Entrez records" table on the right side of the screen. What we want is to find a level of taxonomy above our species for which GenBank has a good number of "Protein" available
6. Click on the genus level link in the lineage (Trichuris)
7. Click the "Trichuris" link at the top of the list of species to get back to the "Entrez records" table at that level in the taxonomy
8. Notice that GenBank has 48,510 proteins available for this taxa, click on the "48.510" number which is a link that will prepare an output set of those proteins
9. Now we will download this protein set. Open the "Send to:" menu in the upper right corner of the page
10. Choose Destination "File"
11. Set the Format to "Fasta"
12. Click on "Create File" to download the file

For the purposes of this workshop I've already prepared a somewhat smaller db for use in our demonstration, which is already available in the EC2 instance (i.e. you don't really need to download the above). But the above process is very useful for when you don't have a specific protein db in mind, yet you want to assign blastp annotations to your gene set basic on homology to related organisms.

Running NCBI's BLASTP+

Now we're going to actually map our gene set to our protein database and filter based on alignment strength. First we need to prepare the blast database for use.

```
1.6.1: cd /home/ec2-
user/WORKSHOP_RESOURCES/Section_1/module_6/NCBI_Blast+/database
1.6.2: makeblastdb -in Ttrichuira_geneset.fna -dbtype prot
```

Next we can start the blastp alignment

1.6.3: `cd /home/ec2-user/WORKSHOP_RESOURCES/Section_1/module_6/NCBI_Blast+`
1.6.4: `blastp -db database/Ttrichuira_geneset.fna -query Tsuis.protein.faa -num_threads 8 -outfmt 6 -max_target_seqs 1  -out Tsuis_vs_Ttrichiura.raw_blastp.tsv`

Then we would typically parse the results using some alignment scoring threshold to filter out only the solid hits

1.6.5: `awk '{if ($11<1e-05) print $0;}' < Tsuis_vs_Ttrichiura.raw_blastp.tsv > Tsuis_vs_Ttrichiura.raw_blastp.tsv.hits_at_1e-05`
The output format we selected using the –outfmt 6 argument produces results in this tab-delimited format:

Query id
Subject id
Percent identity
Alignment length
Mismatch count
Gap open count
Start of alignment in query
End of alignment in query
Start of alignment in subject
End of alignment in subject
E-value
Bitscore

The results at this point will provide associations between our de novo gene set and the genes from our database. We would then use a lookup script to go back and extract the full line annotations from our database and add them to our new genes. While we won't cover that in this workshop, we'd be happy to provide scripts for this on request after the class.

Interproscan

Interproscan is a program that searches a collection of databases and reports associations to all these databases for each gene searched. For our purposes we are mainly interested in the IPR and GO annotations provided by this software. But here is a full listing of what is searched:

PANTHER, PFAM, PIRSF, PRINTS, PRODOM, PROSITE, PROFILE, SMART, TIGRFAMs, GENE3D, SSF, SWISSPROT, TREMBL, INTERPRO, GO, MEROPS, UniProt, HAMAP, PFAMB

Due to its resource intensive nature, and the size of the databases needed in its execution we are not able to demonstrate this software live in our workshop. So we've short-cut this section and deposited pre-built interproscan output for the *T.suis* gene set in the EC2 instance. First lets take a look at the raw output

1.6.6: `cd /home/ec2-user/WORKSHOP_RESOURCES/Section_1/module_6/Interproscan`
1.6.7: `more trichuris_suis_interpro_results`

That command will scroll through that file one page at a time. The length of each line will cause the output to wrap on your screen, making it look messy. But the output of interproscan is actually organized into tab-delimited columns:

1. Protein Accession
2. Sequence MD5 digest
3. Sequence Length
4. Analysis (i.e. the db that was searched on this line)
5. Signature Accession
6. Signature Description
7. Start location
8. Stop location
9. Score (i.e. usually the e-value of the match reported by member database method, although sometimes a specific search engine will report a non- e-value based score)
10. Status
11. Date
12. InterPro annotations – accession (optional column)
13. InterPro annotations – description (optional column)
14. GO annotations (optional column)
15. Pathways annotation (optional column)

Parsing Interproscan results for downstream use

In order to prepare these annotations for downstream analysis (primarily the building of the gene summary table, and the expression analysis that will be shown in Section 2) we need to parse our raw interproscan output into a pre-arranged format that we typically use for that later work. This requires the use of a locally generated perl script (that we're happy to share on request), and would normally build files for both GO and IPR annotations. As a demonstration we'll show how we use this script to generate the GO index

1.6.8: scripts/prepare_files_for_FUNC.no_parents.pl -iprscan_file trichuris_suis_interpro_results -GO_description GO.terms_and_ids.obo.120531 -gene_fof tsuis_full_gene_list.txt -output Tsuis.GO_annotatioN_index

If you 'more' the output file, you'll see that this is a much simpler format than trying to work with the native interproscan result file. This parsed annotation file will be used in Section 2 to help populate the gene summary table in Excel. This format (3 simple columns) should be easy to work with within the spreadsheet.

Useful information:
(NCBI BLAST+ UNIX tutorial) https://molevol.mbl.edu/wiki/index.php/BLAST_UNIX_Tutorial
(NCBI BLAST+ command line arguments) http://www.ncbi.nlm.nih.gov/books/NBK279675/
(Interproscan) https://github.com/ebi-pf-team/interproscan/wiki
(Interpro Db) https://www.ebi.ac.uk/interpro/about.html
(Gene Ontology) http://geneontology.org