

Tutorial 1: Processing RNA-seq Illumina Paired End Data through Trinity De Novo

Trinity partitions the sequence data into many individual de Bruijn graphs, each representing the transcriptional complexity at a given gene or locus, and then processes each graph independently to extract full-length splicing isoforms and to tease apart transcripts derived from paralogous genes. Briefly, the process works like so:

- **Inchworm** assembles the RNA-Seq data into the unique sequences of transcripts, often generating full-length transcripts for a dominant isoform, but then reports just the unique portions of alternatively spliced transcripts.
- **Chrysalis** clusters the Inchworm contigs into clusters and constructs complete de Bruijn graphs for each cluster. Each cluster represents the full transcriptional complexity for a given gene (or sets of genes that share sequences in common). Chrysalis then partitions the full read set among these disjoint graphs.
- **Butterfly** then processes the individual graphs in parallel, tracing the paths that reads and pairs of reads take within the graph, ultimately reporting full-length transcripts for alternatively spliced isoforms, and teasing apart transcripts that corresponds to paralogous genes.

To run Trinity, we can use either paired or unpaired reads. When using paired, you should be able to see /1 in one fasta file and /2 in the other.

<Example Left.fq>

```
@61DFRAAXX100204:1:100:10494:3070/1
ACTGCATCCTGGAAAGAATCAATGGTGGCCGGAAAGTGTTTTTCAAATACAAGAGTGACAATGTGCCCTGTTGTTTT
+
ACCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCBC?CCCCCCCC@@CACCCCCACCCCCCCCCCCCCCCCCCCCCCCC
@61DFRAAXX100204:1:100:10497:13422/1
GTAATTTCCGTACCTGCCACAGTGTGGGCTCACCTGCTTAGAGGACAGGGAAGGACCCTAAAGGTAGGCTGATGC
+
CCCCCCCCCCCCCCCCCCCCDCDCDCDCDCDCDCDCDCDCDCDCDCDCDCDCDCDCDCDCDCDCDCDCDCDCDCDCDC
@61DFRAAXX100204:1:100:10546:4478/1
CTGGGCTGCAGCTAAGTTCTCTGCATCCTCCTTCTTGCTTGTGGCTGGGAAGAAGACAATGTTGTGCGATGGTCTGG
+
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC7CB@CA:>AB?C=C@@@?A@?5:88:
```

<Example Right.fq>

```
@61DFRAAXX100204:1:100:10494:3070/2
CTCAAATGGTTAATTCTCAGGCTGCAAATATTCGTTAGGATGGAAGAACATTTTCTCAGTATTCCATCTAGCTGC
+
C<CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC=
@61DFRAAXX100204:1:100:10497:13422/2
GAGTTACTGGTAAGACGCTTACACCTATAACTCAAGGTCGGAATAGTCCCTCCAGTCCCTTTAGTAACCCAGTGGC
+
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

If you have strand-specific data, specify the library type. There are four library types:

- Paired reads:
 - **RF**: first read (/1) of fragment pair is sequenced as anti-sense (reverse(**R**)), and second read (/2) is in the sense strand (forward(**F**)); typical of the dUTP/UDG sequencing method.
 - **FR**: first read (/1) of fragment pair is sequenced as sense (forward), and second read (/2) is in the antisense strand (reverse)
- Unpaired (single) reads:
 - **F**: the single read is in the sense (forward) orientation
 - **R**: the single read is in the antisense (reverse) orientation

Once we have transferred the files to the server, running trinity is relatively simple since it runs stepwise through the “Trinity” pipeline. Navigate to the directory where you have your sequence files (*tutorial1and2* folder), and run the following command:

Example Paired Run:

```
Trinity.pl --seqType fq --left reads.left.fq --right
reads.right.fq --SS_lib_type RF --paired_fragment_length 280 --
min_contig_length 305 --CPU 4 --bfly_opts "-V 10 --stderr"
```

Note: `--bfly_opts "-V 10 --stderr"` is set so that the Verbose level is high and will print to the screen.

Where `--left <FILENAME>` is your filename of you want to process one type of paired end reads, and `--right <FILENAME>` is the filename of the second type of paired end reads. Page 1 of the tutorial should describe the difference.

By setting the `--SS_lib_type` parameter to one of the above, you are indicating that the reads are strand-specific. By default, reads are treated as not strand-specific.

if strand-specific data, set:

```
--SS_lib_type <string> :if paired: RF or FR, if single: F or R
```

Butterfly-related options:

```
--bfly_opts <string> :parameters to pass through to butterfly (see butterfly documentation).
```

```
--bflyHeapSpace <string> :java heap space setting for butterfly (default: 1000M) => yields
command java -Xmx1000M -jar Butterfly.jar ... $bfly_opts
```

```
--no_run_butterfly :stops after the Chrysalis stage. You'll need to run the Butterfly
computes separately, such as on a computing grid.
```

Inchworm-related options:

```
--no_meryl :do not use meryl for computing the k-mer catalog (default: uses meryl,
providing improved runtime performance)
```

```
--min_kmer_cov <int> :min count for K-mers to be assembled by Inchworm (default: 1)
```

Misc:

```
--CPU <int> :number of CPUs to use, default: 2
```

```

--min_contig_length <int> :minimum assembled contig length to report (def=200)

--paired_fragment_length <int> :maximum length expected between fragment pairs (aim for 90%
percentile) (def=300)

--jaccard_clip :option, set if you have paired reads and you expect high gene density with
UTR overlap (use FASTQ input file format for reads).

```

Other important considerations:

- Trinity performs best with strand-specific data, in which case sense and antisense transcripts can be resolved. If you do know this, use the `-SS_lib_type` flag to describe the data.
- Whether you use Fastq or Fasta formatted input files, be sure to keep the reads oriented as they are reported by Illumina, if the data are strand-specific. This is because, Trinity will properly orient the sequences according to the specified library type. If the data are not strand-specific, now worries because the reads will be parsed in both orientations.
- If you do not have strand-specific data, and you do not plan to use the `--jaccard_clip` option, you can combine all your reads into a single fastq or fasta file and use the `--single` option. You can also combine paired reads and single reads, as long as the paired reads are recognized by having the same accession prefix with /1 and /2 to discriminate between paired ends.
- If you have multiple paired-end library fragment sizes, set the `--paired_fragment_length` according to the larger insert library. Pairings that exceed that distance will be treated as if they were unpaired by the Butterfly process. Trinity's defaults are tuned to a library with an ~300 base fragment length.
- by setting the `--CPU` option, you are indicating:
 - the number of threads for Inchworm to use (in most cases, Inchworm multithreading does not currently lead to performance gains. In future releases, this may change).
 - most importantly, the number of Butterfly executions that will occur simultaneously.

For loblolly, take note that the `--cpu` option can be set to 4 instead of the default of 2.

Tutorial 2: Processes for Read Alignment, Visualization, and Abundance Estimation with Paired End

Once you have finished running Trinity.pl, you can process this output to visualize and get abundance estimations.

1. Align reads to the Trinity transcripts using the `util/alignReads.pl` script, which can leverage Bowtie, BLAT, or BWA as the aligner.

Caution should be taken in using this wrapper and the modified tools, because there are advantages and disadvantages to each, as described below:

- a. Bowtie: Abundance estimation using RSEM (as described below) currently leverages Bowtie gap-free alignments. Running bowtie (original, not the newer bowtie 2...still investigating) with paired fragment reads will exclude alignments where only one of the mate pairs aligns. Since Trinity doesn't perform scaffolding across sequencing gaps yet, there will be cases (more so in fragmented

transcripts corresponding to lowly expressed transcripts) where only one of the mate-pairs aligns. The alignReads.pl script operates similarly to TopHat in that it runs Bowtie to align each of the paired fragment reads separately, and then groups them into pairs afterwards. We capture both the paired and the unpaired fragment read alignments from Bowtie for visualization and examining read support for the transcript assemblies. The properly-mapped pairs are further extracted and can be used as a substrate for RSEM-based abundance estimation (see below).

- b. BLAT: we've found BLAT to be particularly useful in generating spliced short-read alignments to targets where short introns exist. We include BLAT here only for exploratory purposes.
 - c. BWA: the modified version of BWA provides SAM entries for each of the multiply mapped reads alternative mappings, but grouping of pairs is performed by the alignReads.pl script, and the total number of alignments reported tends to be substantially less than running the latest version of BWA in paired mode without having the multiply mapped individual reads. BWA is recommended specifically for SNP-calling exercises, and we're continuing to explore the various options available, including further tweaks here.
2. Run from the *tutorial1and2* directory:

```
/opt/trinityrnaseq_r2011-10-29/util/alignReads.pl --left
reads.left.fq --right reads.right.fq --seqType fq --target
trinity_out_dir/Trinity.fasta --aligner bowtie --
SS_lib_type RF
```

Note: if your data are strand-specific, be sure to set --SS_lib_type as done with Trinity.pl

3. This alignment generates a lot of output files. The **bowtie_out.coordSorted.bam** file contains both properly-mapped pairs and single unpaired fragment reads. This file can be used for visualizing the alignments and coverage data using IGV. The ***nameSorted*PropMapPairsForRsem.bam** contains only the properly-mapped pairs for use with the RSEM software. We will be using the **bowtie_out.coordSorted.bam** file to visualize the data with IGV.
4. To use IGV, get it from : <http://www.broadinstitute.org/igv/>.
5. Once you have the program running, use the *Import Genome* tool to load the Trinity.fasta file as a genome. Also, load the **bowtie_out.coordSorted.bam** file containing the aligned reads. You will need to transfer the associated **bowtie_out.coordSorted.bam.bai** file (the index), so that IGV will load the bam file.



Note: If after loading the genome and the bam file you still can not see any data, use the zoom tool in the top right corner to zoom in. Also, clicking on the nucleotides in the bottom sequence window will toggle a 3 frameshift translation. This could then be flipped by right clicking in this same window to get the other 3 frameshift translation.

6. RSEM is enormously useful for abundance estimation in the context of transcriptome assemblies. RSEM can be downloaded here: <http://deweylab.biostat.wisc.edu/rsem/>. However, currently RSEM is included with Trinity since they have a slightly modified version.

7. Run

```
/opt/trinityrnaseq_r2011-10-29/util/RSEM_util/run_RSEM.pl --transcripts
trinity_out_dir/Trinity.fasta --name_sorted_bam
bowtie_out/bowtie_out.nameSorted.sam+.sam.PropMapPairsForRSEM.bam --paired --
group_by_component
```

This will run RSEM to estimate read abundance.

8. Execute

```
/opt/trinityrnaseq_r2011-10-29/util/RSEM_util/summarize_RSEM_fpkms.pl --
transcripts trinity_out_dir/Trinity.fasta --RSEM RSEM.isoforms.results --
fragment_length 300 --group_by_component | tee Trinity.RSEM.fpkms
```

This will summarize the RSEM FPKM values into an easy to read text file named Trinity.RSEM.fpkms.

Trinity.RSEM.fpkms File

```
#Total fragments mapped to transcriptome: 24114.01
transcript      length  eff_length  count  fraction  fpkm  %comp_fpkm
comp20_c0_seq1  349    50          3.00   5.67e-03  2488.18  100.00

comp0_c0_seq1   3739   3440        531.56 2.03e-02  6408.03  11.02
comp0_c0_seq2   3697   3398        4240.44 1.64e-01  51750.92  88.98

comp9_c0_seq1   5528   5229        192.07 4.83e-03  1523.25  12.45
comp9_c0_seq2   5399   5100        1317.93 3.40e-02  10716.49  87.55

comp19_c0_seq1  433    134         2.00   1.87e-03  618.95   100.00

comp1_c0_seq1   6716   6417        699.32 1.43e-02  4519.33  17.66
comp1_c0_seq2   6665   6366        2949.41 6.10e-02  19213.17  75.07
comp1_c0_seq3   3969   3670         6.08   2.18e-04  68.70    0.27
comp1_c0_seq4   3918   3619        123.99 4.51e-03  1420.79  5.55
comp1_c0_seq5   3152   2853         0.42   1.93e-05  6.10     0.02
comp1_c0_seq6   3101   2802         24.79  1.16e-03  366.89   1.43

comp32_c0_seq1  562    263         7.00   3.45e-03  1103.76  100.00

comp10_c0_seq1  3823   3524        610.19 2.28e-02  7180.58  90.22
comp10_c0_seq2  3715   3416         50.42  1.94e-03  612.09   7.69
comp10_c0_seq3  2749   2450         0.00   1.29e-07  0.00     0.00
comp10_c0_seq4  2641   2342         9.39   5.27e-04  166.27   2.09
```

Column	Description
1. transcript	Transcript name is given based off of the Trinity.fasta file generated by the Trinity.pl pipeline script
2. length	The contig length made by running Trinity.pl based on the --min_contig_length set during this step.
3. eff_length	The effective length is the length of the contig that has a high density of reads.
4. count	The count is the estimated average depth of read coverage across the transcript.
5. fraction	The fraction of fragments mapped to the particular transcript.
6. fpkm	FPKM stands for Fragments Per Kilobase of transcript per Million mapped reads. In RNA-Seq, the relative expression of a transcript is proportional to the number of cDNA fragments that originate from it. Paired-end RNA-Seq experiments produce two reads per fragment, but that doesn't necessarily mean that both reads will be mappable. For example, the second read is of poor quality. If we were to count reads rather than fragments, we might double-count some fragments but not others, leading to a skewed expression value. Thus, FPKM is calculated by counting fragments, not reads.
7. %comp_fpkm	The calculated FPKM percentage for a given contig (comp##) per transcript. The %comp_fpkm for a particular comp## will sum to 100%.

Tutorial 3: Processing RNA-seq Illumina Single End Data through Trinity De Novo

This dataset is larger than the one we used for Tutorial 1, which means it will take much longer to run. Navigate to the directory where we have the single end illumine data (tutorial3 folder), and execute the command below.

Example Single Run:

```
Trinity.pl --seqType fq --min_contig_length 250 --CPU 4 --single s_6_sequence.txt
```

Be sure to note, --single operator is used when working on single end illumine data.

When this has finished running, visualize your results by following the steps in Tutorial 2.