BIO 539 Annotating Cryptasterina

Jacob Green

5/8/2020

Why annotate two sea star genomes: Hystera and Pentagona

Echinoderms are unique in the tree of life when it comes to diversity of lifecycles and modes of reproduction. Across the phylum we find everything from planktotrophy to lecithotrophy and all the nuances in between. Within *Echinodermata*, *Asteroidea*, or Sea stars, exhibit a stunning array of processes when it comes to modes of development, parental investment, larval dispersal, and morphological diversity. On the Austrailian coastline of south east Queensland near Statue Bay two species of Sea star (*Cryptasterina hystera* and *Cryptasterina pentagona*) exhibit a striking and recent divergence in reproductive mode. These two species are morphologically identical, only distinguishable through molecular data and a reproductive period where you will find larvae brooding inside *C. hystera*. The ancestral line of this genus are broadcast spawners and have planktotrophic larvae, but very recently in evolutionary history *C. hystera* began retaining its larvae creating a reproductive rift between itself and *C. pentagona*. In 2019, Dr. Jon Puritz traveled to Statue Bay, Queensland to start a pilot study for defining the molecular dynamics of this shift in reproductive mode. We collected tissue samples for the two species and sequenced nanopore, DNA, and RNA reads. We generated a genome assembly for each species and the scope of this project is to annotate those two assemblies. These annotations will inform preliminary research into the molecular basis of shifting reproductive modes in closely related cogener species of sea stars.

Process of Annotation

Names are important, as names give meaning and in many cases derive the function of a thing. This is no different for genomes. Annotations are the descriptions of structural and functional regions of genomic sequences. Annotating open reading frames (ORF), coding and noncoding regions, high repetitive regions, transposable elements, and other functional elements of genomes are integral to translating genetic sequences into data points that can inform research. There are two stages to the process of annotation 1) alignment/mapping 2) gene prediction/naming. Alignment involves matching your seqeuence reads to the genome and mapping involves assigning quality to how the reads matched, where they match to on the genome, and the length of the matched fragment. For gene prediction and naming we take those aligned mapped fragments and align them to sequences in databases that have been curated. These curated databases hold a plethora of information that can then be collated and used in a variety of ways. There are a multitude of programs that can be leveraged to do both. In the scope of this project we will be using QUAST, MAKER, BRAKER2 for these processes.

Cryptasterina Workflow

How am I structuring this annotation process?

Looks a little something like this. Many of the annotation programs are depedent on the output of other programs.

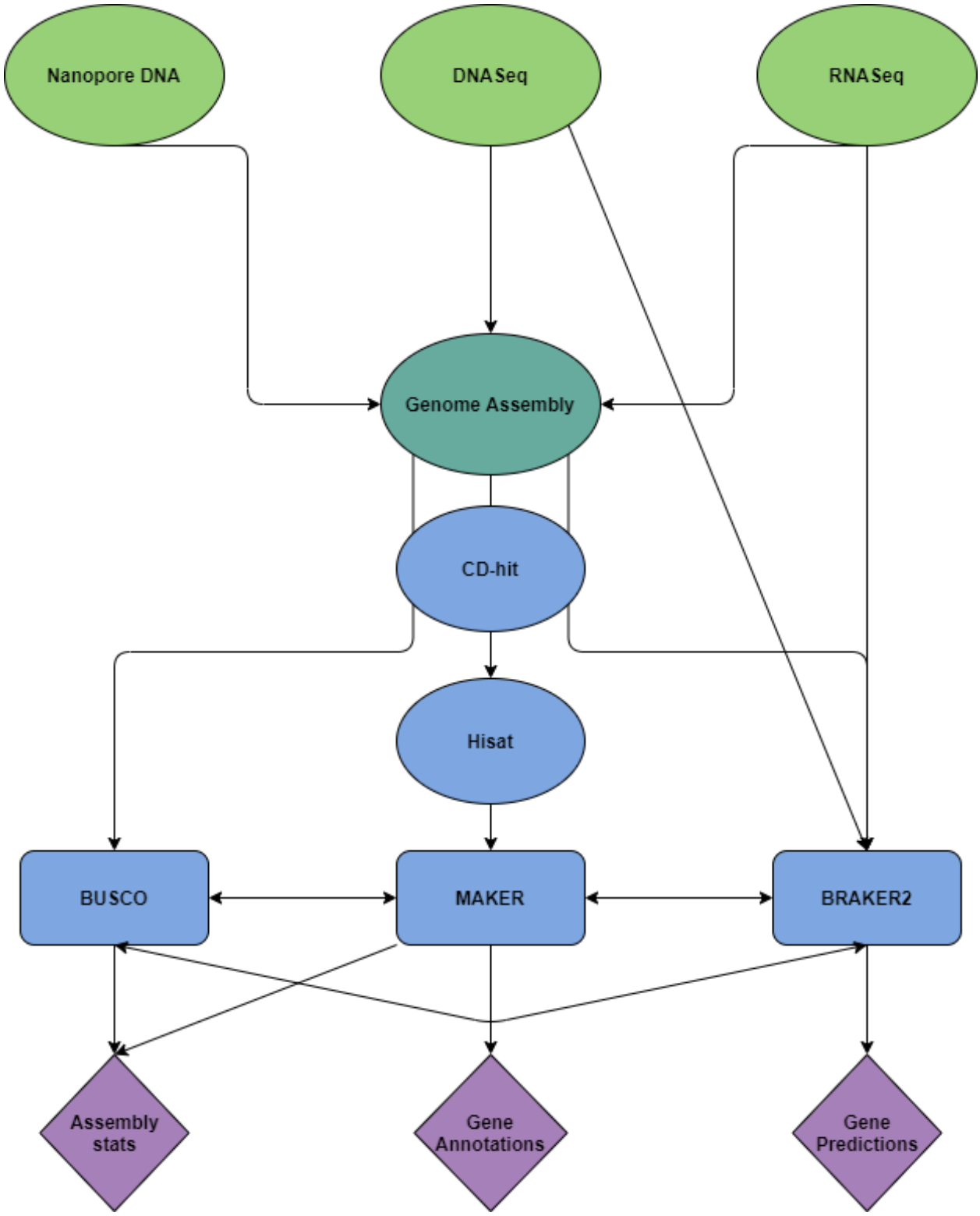

Figure 1: Programming workflow for annotating genomes

Making working directories, locating assemblies, and linking files

Jon provided me with the location of the sequencing reads and assemblies.

First I'll make a directory to house my work

```
mkdir cryptasterina
```

I will be working with linked versions of the assemblies.

```
ln -s /RAID_STORAGE2/Shared_Data/Cryptasterina/Hystera/
ln -s /RAID_STORAGE2/Shared_Data/Cryptasterina/Pentagona/
```

Now I have all the files located in my home.

What is in these directories?

```
ls /home/jgreen/cryptasterina/Pentagona
```

```
## maker_bopts.ctl
## maker_exe.ctl
## maker_opts.ctl
## nanopore.filtered.trimmed.fq.gz
## pentagona_assemblyV49.fasta
## pentagona_assemblyV49.fasta.dammit
## Pentagona.RNA.F.fq.gz
## Pentagona.RNA.R.fq.gz
## trimmed.F.fq.gz
## trimmed.R.fq.gz
```

```
ls /home/jgreen/cryptasterina/Hystera
```

```
## hystera_assemblyV9_nanopore_only.fasta
## hystera_assemblyV9_nanopore_only.fasta.dammit
## Hystera.RNA.F.fq.gz
## Hystera.RNA.R.fq.gz
## maker_bopts.ctl
## maker_exe.ctl
## maker_opts.ctl
## Nanopore.filtered.trimmed.total.fq.gz
## temp
```

In these assemblies we find there there are a few difference between the assemblies that we have. Specifically, the Hystera files do not have paired end reads. This is an artifact that our lab sent in the wrong samples and duplicated the Pentagona reads. This disparity caused some issues early in the project. We had to redo many of the initial analysis for hystera.

Using Bioconda: the good, the bad, the ugly

Bioconda is a channel for the conda package manager specializing in bioinformatics software. The conda package manager makes installing software a vastly more streamlined process. Im going to be using Bioconda as a way to install tools such as BUSCO, dammit, Quast, Diamond, Hisat2, CD-hit-est, Maker, and Braker2. This allows me to manage the various dependencies for each of these programs.

Typically I would want to create one environment for all of these programs, but they have different dependencies that Bioconda is unable to rectify. I'll be making individual environments for each.

We have to make sure that we have the correct channels installed for Bioconda to work.

```
cd ~

./Miniconda3-latest-Linux-x86_64.sh
conda config --add channels defaults
conda config --add channels conda-forge
conda config --add channels bioconda
```

BUSCO

BUSCO is already installed in my environments but i'll add the code here to install and update

```
conda create -n busco busco
conda update busco
```

Make a directory for our busco results

```
cd ~/cryptasterina/
mkdir busco
```

Run the BUSCO python script for the eukaryota and metazoa database

```
python /home/jgreen/miniconda3/pkgs/busco-1.2-py27_1/bin/BUSCO_v1.2.py -i /home/jgreen/cryptasterina/Hys

python /home/jgreen/miniconda3/pkgs/busco-1.2-py27_1/bin/BUSCO_v1.2.py -i /home/jgreen/cryptasterina/Hys
```

Here are the results from the BUSCO runs.

```
cat ~/cryptasterina/busco/run_hystera_busco_meta/short_summary_hystera_busco_meta

cat ~/cryptasterina/busco/run_hystera_busco_euk/short_summary_hystera_busco
```

```
## #Summarized BUSCO benchmarking for file: hystera_assemblyV9_nanopore_only.fasta
## #BUSCO was run in mode: genome
##
## Summarized benchmarks in BUSCO notation:
##  C:91%[D:1.9%],F:2.3%,M:6.3%,n:978
##
##  893 Complete BUSCOs
##  874 Complete and single-copy BUSCOs
##  19  Complete and duplicated BUSCOs
##  23  Fragmented BUSCOs
##  62  Missing BUSCOs
##  978 Total BUSCO groups searched
## #Summarized BUSCO benchmarking for file: hystera_assemblyV9_nanopore_only.fasta
## #BUSCO was run in mode: genome
##
## Summarized benchmarks in BUSCO notation:
##  C:93%[D:1.6%],F:2.3%,M:4.6%,n:303
##
##  282 Complete BUSCOs
##  277 Complete and single-copy BUSCOs
##  5   Complete and duplicated BUSCOs
##  7   Fragmented BUSCOs
##  14  Missing BUSCOs
##  303 Total BUSCO groups searched
```

```
cat ~/cryptasterina/busco/run_pentagona_busco_meta/short_summary_pentagona_busco_meta

cat ~/cryptasterina/busco/run_pentagona_busco_euk/short_summary_pentagona_busco
```

```
## #Summarized BUSCO benchmarking for file: pentagona_assemblyV49.fasta
## #BUSCO was run in mode: genome
##
## Summarized benchmarks in BUSCO notation:
##  C:91%[D:3.2%],F:2.1%,M:6.0%,n:978
##
##  898 Complete BUSCOs
##  866 Complete and single-copy BUSCOs
##  32  Complete and duplicated BUSCOs
##  21  Fragmented BUSCOs
##  59  Missing BUSCOs
##  978 Total BUSCO groups searched
## #Summarized BUSCO benchmarking for file: pentagona_assemblyV49.fasta
## #BUSCO was run in mode: genome
##
## Summarized benchmarks in BUSCO notation:
##  C:93%[D:2.3%],F:1.9%,M:4.6%,n:303
##
##  283 Complete BUSCOs
##  276 Complete and single-copy BUSCOs
##  7   Complete and duplicated BUSCOs
##  6   Fragmented BUSCOs
##  14  Missing BUSCOs
##  303 Total BUSCO groups searched
```

Although this is a nice text format we would like an image, which can be generated by the BUSCO program

Link all the short_summary files into one directory. Move into that directory and then run the following code.

```
generate_plot.py -wd .
```

Which produces this lovely graph.

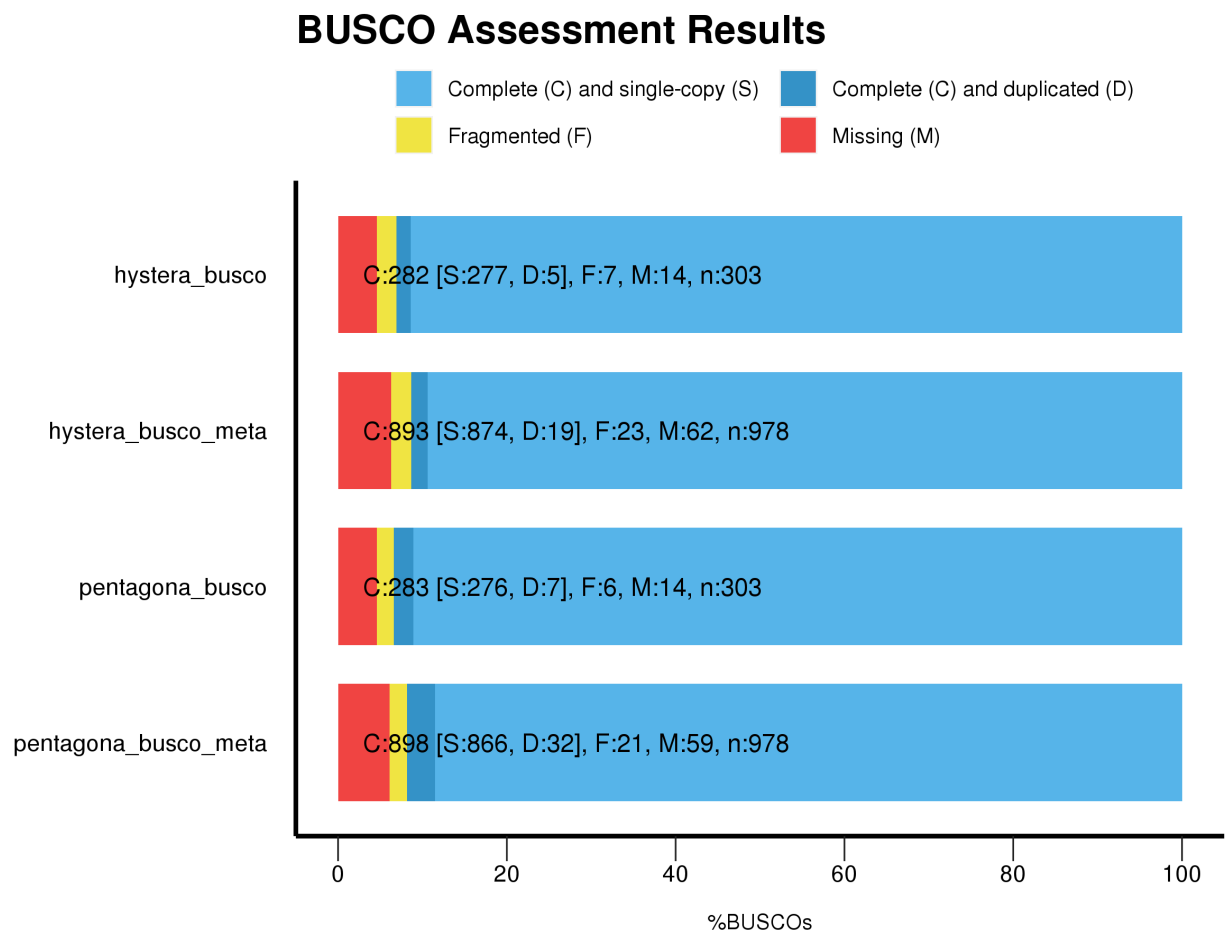

Figure 2: Hystera and Pentagona Metazoa and Eukaryota BUSCO Results

For our assemblies we have done a few things through leveraging BUSCO. When applied in the genome mode BUSCO will leverage two important programs: Augustus and hmmer. Both of these begin to predict or find genes. In this case they are subset for orthologs defined by the lineage dataset. These predictions can be used down the line in other programs such as Maker and Braker2. One of my goals is to figure out how that works and use these gene predictions to prime these other programs. We also have checked our assemblies for quality. Having many single copy complete orthologs and few missing orthologs signals that a genome is well built. There are few duplication and fragments, which will make singular annotation easier. One issue annotation runs into is that if genes are duplicated overannotation occurs and you get many different variation of the same gene or protein. In this case all of our assemblies had > 90 % complete orthologs. Moving forward we can be confident in our assemblies.

QUASTing for quality

QUAST (QUality ASsessment Tool) is a neat program that I initially found to assess the assemblies. By digging through the manual I discovered that there is a ton more functionality to this program from gene finding to generation circos plots to visualize your data.

QUAST uses the following for analysis:

- BWA
- bedtools
- Minimap2

- Genemark-ES
- Barrnap
- BUSCO

For visualization: * Icarus * Circos

I started by installing Quast from bioconda.

```
conda install -c bioconda quast
conda create -n quast quast
```

```
python /home/jgreen/miniconda3/envs/quast/bin/quast.py pentagona_assemblyV50.fasta --eukaryote --lar
ge --k-mer-stats --rna-finding --pe1 trimmed.F.fq.gz --pe2 trimmed.R.fq.gz --nanopore nanopore.filtered
```

It was taking quite a long time, which is expected because it is aligning not only the trimmed forward and reverse reads but also the nanopore reads. Making so many .sam and.bam files quickly filled up space. It didn't help that QUAST had to rename the forward and reverse reads (= 64 GB) and also generating two directories that store the same information in a quast/latest/ and quast/results directory. I had two whole directories filled with 1000 GB of .sam and .bam files, so the program ran out of memory.

To circumvent this issue I dug into the quast manual a bit more. There are three arguments of interest that I will be adding (–min-contig 1000 –space-efficient –conserved-genes-finding).

```
python /home/jgreen/miniconda3/envs/quast/bin/quast.py pentagona_assemblyV50.fasta --eukaryote --large
```

Yet, again though I ran into another space issue. This is a problem when workiong on servers with limited space. The inability to run programs that either are not optimized for large genomes or do not properly take into consideration the memory footprint of smaller working labs decreases the efficiency of a program. I will try taking out the nanopore reads and running.

```
python /home/jgreen/miniconda3/envs/quast/bin/quast.py pentagona_assemblyV49.fasta --eukaryote --large
```

No luck on this run either. For the time being we will suspend using QUAST for quality assesment.

Weird thought of using dammit?

So I had this odd idea of using dammit. It is a de novo transcriptome annotation pipeline and I was wodnering if it would work in this case. We actually get some real interesting results and create some files that we need for subsequent analysis.

Dammit is a program that I have used in the past for de novo transcriptome assemblies. It provides a thorough annotation by leveraging a myriad of programs that you can find on their site.

Install dammit environment

```
conda create -n dammit python=3
source activate dammit
conda install dammit
dammit databases --install --busco-group eukaryota
```

Commands to run the dammit program

```
dammit annotate pentagona_assemblyV59.fasta --busco-group eukaryota
dammit annotate hystera_assemblyV9_nanopore_only.fasta --busco-group eukaryota
```

This is going to take a while. So as Camille says "Go grab some coffee" and check your email or read a book.

We have our dammit results for both the Pentagona and Hystera assemblies, which are extensive as shown below

Pentagona

```
ls /home/jgreen/cryptasterina/Pentagona/pentagona_assemblyV49.fasta.dammit/
```

```
## annotate.doit.db
## dammit.log
## pentagona_assemblyV49.fasta
## pentagona_assemblyV49.fasta.dammit.fasta
## pentagona_assemblyV49.fasta.dammit.gff3
## pentagona_assemblyV49.fasta.dammit.namemap.csv
## pentagona_assemblyV49.fasta.dammit.stats.json
## pentagona_assemblyV49.fasta.transdecoder.bed
## pentagona_assemblyV49.fasta.transdecoder.cds
## pentagona_assemblyV49.fasta.transdecoder_dir
## pentagona_assemblyV49.fasta.transdecoder_dir.__checkpoints
## pentagona_assemblyV49.fasta.transdecoder_dir.__checkpoints_longorfs
## pentagona_assemblyV49.fasta.transdecoder.gff3
## pentagona_assemblyV49.fasta.transdecoder.pep
## pentagona_assemblyV49.fasta.x.OrthoDB.best.csv
## pentagona_assemblyV49.fasta.x.OrthoDB.best.gff3
## pentagona_assemblyV49.fasta.x.OrthoDB.maf
## pentagona_assemblyV49.fasta.x.pfam-A.csv
## pentagona_assemblyV49.fasta.x.pfam-A.gff3
## pentagona_assemblyV49.fasta.x.rfam.gff3
## pentagona_assemblyV49.fasta.x.rfam.tbl
## pentagona_assemblyV49.fasta.x.rfam.tbl.cmscan.out
## pentagona_assemblyV49.fasta.x.sprot.best.csv
## pentagona_assemblyV49.fasta.x.sprot.best.gff3
## pentagona_assemblyV49.fasta.x.sprot.maf
## pipeliner.28272.cmds
## pipeliner.54884.cmds
## pipeliner.56170.cmds
```

Hystera

```
ls /home/jgreen/cryptasterina/Hystera/hystera_assemblyV9_nanopore_only.fasta.dammit/
```

```
## annotate.doit.db
## dammit.log
## hystera_assemblyV9_nanopore_only.fasta
## hystera_assemblyV9_nanopore_only.fasta.dammit.namemap.csv
## hystera_assemblyV9_nanopore_only.fasta.dammit.stats.json
## hystera_assemblyV9_nanopore_only.fasta.transdecoder.bed
## hystera_assemblyV9_nanopore_only.fasta.transdecoder_dir
## hystera_assemblyV9_nanopore_only.fasta.transdecoder_dir.__checkpoints
## hystera_assemblyV9_nanopore_only.fasta.transdecoder_dir.__checkpoints_longorfs
## hystera_assemblyV9_nanopore_only.fasta.transdecoder.gff3
## hystera_assemblyV9_nanopore_only.fasta.transdecoder.pep
## hystera_assemblyV9_nanopore_only.fasta.x.pfam-A.csv
## hystera_assemblyV9_nanopore_only.fasta.x.pfam-A.gff3
## pipeliner.10780.cmds
## pipeliner.49356.cmds
## pipeliner.50072.cmds
```

With these files we are going to be moving onto some other gene prediction programs.

Diamond

Diamond is a rapid annotation program that leverages any database you would like to feed it. To make annotation meaningful, using a database that has well described genes is important. We will use the UniProt SwissProt, a database where all the genes annotated will be thororoughly annotated. Follow the links for (Diamond)[https://github.com/bbuchfink/diamond] or (UniProt)[https://www.uniprot.org/] for inqueries.

For diamond we are going to use the uniprot swiss prot database.

```
wget ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/complete/uniprot_sprot.fa
```

We will create an environment for the Diamond program

```
conda create -n diamond diamond
```

This will generate an annotation file containing proteins from the Uniprot database aligned from the assembly.

```
diamond blastx -p 2 -k 1 -e 0.00001 -d /home/jgreen/databases/uniprot_sprot.dmnd -q pentagona_assemblyV4
```

HiSAT2

Create Hisat2 environment

```
conda create -n hisat2 hisat2
```

Create Hisat2 environment

```
nano hisat2_script.sh
```

Make hisat2 script to build database, align genome, convert sam file to bam, and sort

```
#!/bin/bash
PROC=16
R1_FQ="$1"
R2_FQ="$2"
DBDIR="$3"
GENOME="$4"
hisat2-build ${GENOME} ${GENOME%.*}
hisat2 -p ${PROC} -x ${GENOME%.*} -1 ${R1_FQ%%.*}_val_1.fq.gz -2 ${R2_FQ%%.*}_val_2.fq.gz -S ${R1_FQ%.*
samtools view --threads ${PROC} -b -o ${R1_FQ%.*}.bam ${R1_FQ%.*}.sam
samtools sort -m 7G -o ${R1_FQ%.*}_sorted.bam -T ${R1_FQ%.*}_temp --threads ${PROC} ${R1_FQ%.*}.bam
java -jar picard.jar CollectAlignmentSummaryMetrics  REFERENCE_SEQUENCE=${DBDIR}/${GENOME} INPUT=${R1_FQ
```

Run the hisat2 script

```
./hisat2_script.sh
```

CD-HIT-EST

Create CD-hit environment

```
conda create -n cdhit cd-hit
```

Run cd-hit-est for pentagona and hystera. Then run a cd-hit-est-2d comparison to cluster similarities between hystera and pentagona

```
cd-hit-est -i -o -c 0.95 -n 10 -d 0 -M 16000 -T 8
cd-hit-est -i -o -c 0.95 -n 10 -d 0 -M 16000 -T 8
cd-hit-est-2d -i -i2 -o  -c 0.95 -n 10 -d 0 -M 16000 -T 8
```

Maker

MAKER is a pipeline for genome annotation that uses existing databases to generate expressed sequence tag (EST) alignments and protein homology. To complete that goal MAKER follows some steps:

- Identifies and masks out repeat elements
- Aligns ESTs to the genome
- Aligns proteins to the genome
- Produces *ab initio* gene predictions
- Synthesizes into final annotations
- Produces quality-values for downstream annotation

MAKER uses the following for annotation:

- Perl
- SNAP
- RepeatMasker
- Exonerate
- NCBI BLAST
- Augustus
- GeneMark-ES
- FGENESH

```
conda create -n maker maker genemark-es
```

For Maker to run we need a few files

1. fasta file (from Jon)
2. est file (from CD-hit)
3. protein file (from diamond run)

We need to make three control files in the directory the we will run the analysis in.

```
maker -CTRL
```

This provides us with these three files. We need to pen them up and check the locations of the programs to make sure this is all correct for our run.

```
cat /home/jgreen/cryptasterina/Pentagona/maker_bopts.ctl
```

```
## #-----BLAST and Exonerate Statistics Thresholds
## blast_type=ncbi+ #set to 'ncbi+', 'ncbi' or 'wublast'
##
## pcov_blastn=0.8 #Blastn Percent Coverage Threhold EST-Genome Alignments
## pid_blastn=0.85 #Blastn Percent Identity Threshold EST-Genome Aligments
## eval_blastn=1e-10 #Blastn eval cutoff
## bit_blastn=40 #Blastn bit cutoff
## depth_blastn=0 #Blastn depth cutoff (0 to disable cutoff)
##
## pcov_blastx=0.5 #Blastx Percent Coverage Threhold Protein-Genome Alignments
## pid_blastx=0.4 #Blastx Percent Identity Threshold Protein-Genome Aligments
## eval_blastx=1e-06 #Blastx eval cutoff
## bit_blastx=30 #Blastx bit cutoff
## depth_blastx=0 #Blastx depth cutoff (0 to disable cutoff)
```

```
##
## pcov_tblastx=0.8 #tBlastx Percent Coverage Threhold alt-EST-Genome Alignments
## pid_tblastx=0.85 #tBlastx Percent Identity Threshold alt-EST-Genome Aligments
## eval_tblastx=1e-10 #tBlastx eval cutoff
## bit_tblastx=40 #tBlastx bit cutoff
## depth_tblastx=0 #tBlastx depth cutoff (0 to disable cutoff)
##
## pcov_rm_blastx=0.5 #Blastx Percent Coverage Threhold For Transposable Element Masking
## pid_rm_blastx=0.4 #Blastx Percent Identity Threshold For Transposbale Element Masking
## eval_rm_blastx=1e-06 #Blastx eval cutoff for transposable element masking
## bit_rm_blastx=30 #Blastx bit cutoff for transposable element masking
##
## ep_score_limit=20 #Exonerate protein percent of maximal score threshold
## en_score_limit=20 #Exonerate nucleotide percent of maximal score threshold
```

```
cat /home/jgreen/cryptasterina/Pentagona/maker_exe.ctl
```

```
## #-----Location of Executables Used by MAKER/EVALUATOR
## makeblastdb=/home/jgreen/miniconda3/envs/maker/bin/makeblastdb #location of NCBI+ makeblastdb executa
## blastn=/home/jgreen/miniconda3/envs/maker/bin/blastn #location of NCBI+ blastn executable
## blastx=/home/jgreen/miniconda3/envs/maker/bin/blastx #location of NCBI+ blastx executable
## tblastx=/home/jgreen/miniconda3/envs/maker/bin/tblastx #location of NCBI+ tblastx executable
## formatdb= #location of NCBI formatdb executable
## blastall= #location of NCBI blastall executable
## xdformat= #location of WUBLAST xdformat executable
## blasta= #location of WUBLAST blasta executable
## RepeatMasker=/home/jgreen/miniconda3/envs/maker/bin/RepeatMasker #location of RepeatMasker executable
## exonerate=/home/jgreen/miniconda3/envs/maker/bin/exonerate #location of exonerate executable
##
## #-----Ab-initio Gene Prediction Algorithms
## snap=/home/jgreen/miniconda3/envs/maker/bin/snap #location of snap executable
## gmhmme3= #location of eukaryotic genemark executable
## gmhmmp= #location of prokaryotic genemark executable
## augustus=/home/jgreen/miniconda3/envs/maker/bin/augustus #location of augustus executable
## fgenesh= #location of fgenesh executable
## tRNAscan-SE=/home/jgreen/miniconda3/envs/maker/bin/tRNAscan-SE #location of trnascan executable
## snoscan=/home/jgreen/miniconda3/envs/maker/bin/snoscan #location of snoscan executable
##
## #-----Other Algorithms
## probuild= #location of probuild executable (required for genemark)
```

```
cat /home/jgreen/cryptasterina/Pentagona/maker_opts.ctl
```

```
## #-----Genome (these are always required)
## genome= #genome sequence (fasta file or fasta embeded in GFF3 file)
## organism_type=eukaryotic #eukaryotic or prokaryotic. Default is eukaryotic
##
## #-----Re-annotation Using MAKER Derived GFF3
## maker_gff= #MAKER derived GFF3 file
## est_pass=0 #use ESTs in maker_gff: 1 = yes, 0 = no
## altest_pass=0 #use alternate organism ESTs in maker_gff: 1 = yes, 0 = no
## protein_pass=0 #use protein alignments in maker_gff: 1 = yes, 0 = no
## rm_pass=0 #use repeats in maker_gff: 1 = yes, 0 = no
## model_pass=0 #use gene models in maker_gff: 1 = yes, 0 = no
## pred_pass=0 #use ab-initio predictions in maker_gff: 1 = yes, 0 = no
```

```
## other_pass=0 #passthrough anyything else in maker_gff: 1 = yes, 0 = no
##
## #-----EST Evidence (for best results provide a file for at least one)
## est= #set of ESTs or assembled mRNA-seq in fasta format
## altest= #EST/cDNA sequence file in fasta format from an alternate organism
## est_gff= #aligned ESTs or mRNA-seq from an external GFF3 file
## altest_gff= #aligned ESTs from a closly relate species in GFF3 format
##
## #-----Protein Homology Evidence (for best results provide a file for at least one)
## protein=  #protein sequence file in fasta format (i.e. from mutiple oransisms)
## protein_gff=  #aligned protein homology evidence from an external GFF3 file
##
## #-----Repeat Masking (leave values blank to skip repeat masking)
## model_org=all #select a model organism for RepBase masking in RepeatMasker
## rmlib= #provide an organism specific repeat library in fasta format for RepeatMasker
## repeat_protein= #provide a fasta file of transposable element proteins for RepeatRunner
## rm_gff= #pre-identified repeat elements from an external GFF3 file
## prok_rm=0 #forces MAKER to repeatmask prokaryotes (no reason to change this), 1 = yes, 0 = no
## softmask=1 #use soft-masking rather than hard-masking in BLAST (i.e. seg and dust filtering)
##
## #-----Gene Prediction
## snaphmm= #SNAP HMM file
## gmhmm= #GeneMark HMM file
## augustus_species= #Augustus gene prediction species model
## fgenesh_par_file= #FGENESH parameter file
## pred_gff= #ab-initio predictions from an external GFF3 file
## model_gff= #annotated gene models from an external GFF3 file (annotation pass-through)
## est2genome=0 #infer gene predictions directly from ESTs, 1 = yes, 0 = no
## protein2genome=0 #infer predictions from protein homology, 1 = yes, 0 = no
## trna=0 #find tRNAs with tRNAscan, 1 = yes, 0 = no
## snoscan_rrna= #rRNA file to have Snoscan find snoRNAs
## unmask=0 #also run ab-initio prediction programs on unmasked sequence, 1 = yes, 0 = no
##
## #-----Other Annotation Feature Types (features MAKER doesn't recognize)
## other_gff= #extra features to pass-through to final MAKER generated GFF3 file
##
## #-----External Application Behavior Options
## alt_peptide=C #amino acid used to replace non-standard amino acids in BLAST databases
## cpus=1 #max number of cpus to use in BLAST and RepeatMasker (not for MPI, leave 1 when using MPI)
##
## #-----MAKER Behavior Options
## max_dna_len=100000 #length for dividing up contigs into chunks (increases/decreases memory usage)
## min_contig=1 #skip genome contigs below this length (under 10kb are often useless)
##
## pred_flank=200 #flank for extending evidence clusters sent to gene predictors
## pred_stats=0 #report AED and QI statistics for all predictions as well as models
## AED_threshold=1 #Maximum Annotation Edit Distance allowed (bound by 0 and 1)
## min_protein=0 #require at least this many amino acids in predicted proteins
## alt_splice=0 #Take extra steps to try and find alternative splicing, 1 = yes, 0 = no
## always_complete=0 #extra steps to force start and stop codons, 1 = yes, 0 = no
## map_forward=0 #map names and attributes forward from old GFF3 genes, 1 = yes, 0 = no
## keep_preds=0 #Concordance threshold to add unsupported gene prediction (bound by 0 and 1)
##
## split_hit=10000 #length for the splitting of hits (expected max intron size for evidence alignments)
```

```
## single_exon=0 #consider single exon EST evidence when generating annotations, 1 = yes, 0 = no
## single_length=250 #min length required for single exon ESTs if 'single_exon is enabled'
## correct_est_fusion=0 #limits use of ESTs in annotation to avoid fusion genes
##
## tries=2 #number of times to try a contig if there is a failure for some reason
## clean_try=0 #remove all data from previous run before retrying, 1 = yes, 0 = no
## clean_up=0 #removes theVoid directory with individual analysis files, 1 = yes, 0 = no
## TMP= #specify a directory other than the system default temporary directory for temporary files
```

Make our script to run the maker program

```
nano maker_script.sh
```

This will contain the following

```
#!/bin/bash
MAKERDIR="Pentagona"
maker -base ${MAKERDIR} -fix_nucleotides -dsindex
gff3_merge  -d ${MAKERDIR}.maker.output/${MAKERDIR}_master_datastore_index.log
fasta_merge -d ${MAKERDIR}.maker.output/${MAKERDIR}_master_datastore_index.log
```

Run the maker script

```
./maker_script.sh
```

Braker

BRAKER2 is another “annotation pipeline”" but relies on fully automated training of gene prediction tools.

```
conda create -n braker2 braker2
```

Create repeat database and set up repeat mode

```
BuildDatabase -name pentagona_db -engine ncbi -pa 16 pentagona_assemblyV49.fasta
RepeatModeler -database pentagona_db -engine ncbi -pa 16
ln -s RM/consensi.fa.classified
RepeatMasker -pa 16 -gff -nolow -lib consensi.fa.classified pentagona_assemblyV49.fasta
```

```
#!/bin/bash
#Makes a database and  aligns your sequences into reference.
#sh runGmap.sh <database name> <folder of database file ending with a "/"> <Fasta file> <query file>
dbname=$1
dbloc=$2
dbfasta=$3
query=$4
#Build the database!
#build the gmap database out of the script if multiple alignments are to be done
#gmap_build -d $dbname -D $dbloc $dbfasta
#Align the transcripts!
gmap -D $dbloc -d $dbname -B 5 -t 16 --input-buffer-size=1000000 --output-buffer-size=1000000 -f samse
#Convert the sam to bam
samtools view --threads 16 -b -o ${dbname%.*}.${query%.*}.bam ${dbname%.*}.${query%.*}.sam
#sort the bam file
samtools sort -m 7G -o ${query%.*}_sorted.bam -T ${R1_FQ%.*}_temp --threads 16 ${query%.*}.bam
```

```
sh runGmap.sh genome /home/jgreen/cryptasterina/Pentagona pentagona_assemblyV49.fasta pentagona.est.fas
```

```
gmap_build  -d genome pentagona_assemblyV49.fasta
```

Download genemark-es key

```
wget http://exon.gatech.edu/GeneMark/license_download.cgi
mv gm_key_64 ~/.gm_key
```

Run Braker 2

```
braker.pl --species=pentagona --genome=pentagona_assemblyV49.fasta.masked --bam=  --prot_seq= pentagonal
```