# Genome annotation pipeline

This pipeline is intended to be a standard procedure to annotate genomes combining different sources of evidence (RNA-seq, *de novo* transcriptomes, inter-specific protein alignments, *ab initio* predictions). At QMUL, the core elements of the pipeline are in Apocrita and are loaded into the path after sourcing:

source /data/SBCS-Informatics/chema\_pipeline\_0519/init.sh

This results in:

$ Loading trinity/2.4.0

$  Loading requirement: intel/2017.1 bowtie2/2.3.2

$ checking paths to required modules

$ /data/SBCS-Informatics/chema\_pipeline\_0519/repeatModeler/RepeatModeler-open-1.0.11/BuildDatabase

$ /data/SBCS-Informatics/chema\_pipeline\_0519/repeatModeler/RepeatModeler-open-1.0.11/RepeatModeler

$ /data/SBCS-Informatics/chema\_pipeline\_0519/repeatModeler/RepeatMasker/RepeatMasker

$ /share/apps/centos7/star/2.5.3a/bin/STAR

$ /data/SBCS-Informatics/chema\_pipeline\_0519/stringtie-1.3.6.Linux\_x86\_64/stringtie

$ /share/apps/centos7/trinity/2.4.0/Trinity

$ /data/SBCS-Informatics/chema\_pipeline\_0519/env/bin/mikado

# Step 1: Mask repeat elements

To avoid including transposable elements (TEs) into the gene repertoire and excluding repetitive regions that could affect mapping and gene prediction algorithms, it is necessary to mask (either with Ns [hard masking] or with lower case letters [soft masking]) the genome. To do so, we use RepeatModeler (to generate a database of TEs and repetitive sequences of the species) and RepeatMasker (to mask the genome using the previously built database). In addition to being an essential first step for gene annotation, identification of repetitive elements is a project on its own, as TEs are an essential component of animal genomes.

**Note**: RepBase (a curated database used for initial TE/repetitive element prediction requires now a licence; we have an old version [2015] in the server, but it might be good to consider buying institute-wide licence)

## 1.1 RepeatModeler

The first step is to prepare a database of the unmasked genome and generate a species-specific prediction of repetitive elements. [RepeatModeler](http://www.repeatmasker.org/RepeatModeler/) uses sequence similarity approaches (with either BLAST or HMMR) to identify potential repeat elements in the genome of interest that are similar to those curated in a database (RepBase). After several rounds of search, it outputs a set of consensus repetitive sequences.

Install with conda:

module load anaconda2

conda create -n Repeats\_env

source activate Repeats\_env

conda install -c bioconda repeatmodeler

* repeatmodeler version 2.0.1

#!/bin/bash

#$ -wd /data/scratch/btx654/

#$ -o /data/scratch/btx654/

#$ -j y

#$ -pe smp 20

#$ -l h\_vmem=20G

#$ -l h\_rt=240:0:0

#$ -l highmem

echo "Working on owenia"

module load anaconda2

source activate Repeats\_env

cd owenia/september2020/

# Build a genomic database and run RepeatModeler to generate a Owenia-specific. The database is built using BLAST (ncbi flag).

BuildDatabase -name owenia -engine ncbi Owenia\_unmasked\_v082020.fa

RepeatModeler -engine ncbi -pa 8 -database owenia

RepeatModeler will create a folder starting with RM\_ and will generate two main files: consensi.fa.classified and families-classified.stk. The first is the set of consensus families predicted by RepeatModeler in fasta format. The second is the seed alignments of these families in Stockholm format. For downstream analysis, one needs the consensi.fa.classified, which can be renamed to include the name of the species.

## 1.2 Filtering *bona fide* genes out of the RepeatModeler library

Sometimes, RepeatModeler will interprete an expanded gene family as a repetitive element. To avoid masking out real genes (i.e. to remove false positives), one needs to check for the presence of real genes in the consensi.fa.classified library. To do so, one can get a curated proteome of a related species (e.g. *Capitella*, from [ENSEMBL Metazoa](https://metazoa.ensembl.org/Capitella_teleta/Info/Index)), remove potential TEs from *Capitella*’s proteome, and then use that proteome to identify and filter out potential genes from the consensi.fa.classified dataset. As a reference TE database, one can use the RepeatPeps.lib of RepeatMasker.

#make a diamond BLAST database of RepeatPeps.lib and BLAST Capitella proteome against it

module load diamond/0.9.22

diamond makedb --in RepeatPeps.lib -d RepeatPeps

diamond blastp -d RepeatPeps -q cte\_NRproteome.fasta -o cte\_NRproteome.vs.RepeatPeps.1e5.blastp -f 6 qseqid bitscore evalue stitle -k 25 -e 1e-5 -p 8

#Get the unique set of qseqids with significant hit against a TE and filter it out of the proteome. One can do that easily with shell commands (cut, grep, etc) tabulating the fasta file of the proteome (so that one can find at the same time the ID and sequence associated to it). Alternatively, if there are not many, one can do that by hand.

#make a diamond BLAST database of this proteome and BLAST the consensi.fa.classified dataset against it, to find potential bona fide genes. To make sure we only get the real genes, the e-value is very stringent.

diamond makedb --in Capitella\_filteredProt.fasta -d CapitellaProtNoTEs

diamond blastp -d CapitellaProtNoTEs -q consensi.fa.classified -o consensi.fa.classified.vs.CapitellaProtNoTEs.1e10.blastp -f 6 qseqid bitscore evalue stitle -k 25 -e 1e-10 -p 8

#Since there will not be many hits, I recommend removing the potential bona fide genesout of the consensi.fa.classified by hand, as one makes sure that each of those hits are actual genes and not a TEs.

With *Owenia*, this approach uncovered 7 potential genes incorporated into the consensi.fa.classified database.

rnd-1\_family-1155#Unknown ===== alcohol dehydrogenase

rnd-1\_family-59#Unknown ===== histone cluster?

rnd-6\_family-1486#Unknown ===== KR superfamily domain (plasminogen)

rnd-6\_family-235#Unknown ===== FReD superfamily domain

rnd-6\_family-3681#Unknown ===== amidase

rnd-6\_family-6593#Unknown ===== adenosine receptor (!) GPCR!

rnd-6\_family-889#DNA/Academ =====

I verified manually with the NCBI BLAST web server that these are real genes with clear homology to non-TEs and extracted them from the file. I saved the filtered consensi.fa.classified as Owenia\_RMconsensusDB\_noGenes.fa 

## 1.3 Annotate the TEs with TEclass

The annotation is made possible thanks to the online tool [TEclass](http://www.bioinformatics.uni-muenster.de/tools/teclass/index.hbi?lang=en). This software classifies unknown TEs consensus sequences into four categories, according to their mechanism of transposition: DNA transposons, LTRs, LINEs, SINEs

Feed the consensus filtered file (the file we obtained in step 1.2: Owenia\_RMconsensusDB\_noGenes.fa) to TEclass by uploading the whole file to the “[newrequest](http://www.bioinformatics.uni-muenster.de/tools/teclass/generate/index.pl?lang=en)” page, mind that this software is working only with fasta file weighting less than 5Mb.

This will output a txt file in fasta format containing the information about the TE type at the end of the header line, that will be used later to obtain plot about the diversity of TEs. Rename this file teclass\_owenia.txt and upload it to Apocrita.

## 1.4 RepeatMasker-Kimura

Now that we have a curated database of repetitive elements, we can run RepeatMasker to mask those regions in the genome. To avoid mistaken Ns (gaps in the genome) from masked regions (TEs, etc), we run RepeatMasker with the -xsmall flag to soft-mask the genome. To generate a track of repeat elements that we can explore thereafter locally in IGV, we add the -gff flag.

**Note**: I run this in Ofus\_unmasked\_v072019.fa, which is the version of the assembly with the scaffolds ordered from largest to shortest.

Created a new anaconda3 env named Repeats\_env3:

module load anaconda3

conda create -n Repeats\_env3

source activate Repeats\_env3

conda install -c bioconda repeatmodeler

* repeatmodeler version 2.0.1

!!!!!!!!! IMPORTANT !!!!!!! otherwise it’s not working! set some path with the script /data/home/btx604/.conda/envs/Repeats\_env/share/RepeatMasker/configure. I Chose RMBlast as default (n2) and I specified the path to the bin of the conda env:  /data/home/btx604/.conda/envs/Repeats\_env/bin

/data/home/btx654/.conda/envs/Repeats\_env3/share/RepeatMasker/

/data/home/btx654/.conda/envs/Repeats\_env3/bin

The first thing we will need in this step is the genome size. To get this information one can run quast:

module load anaconda3

conda create -n quast

source activate quast

conda install -c bioconda quast

#!/bin/bash

#$ -wd /data/scratch/btx654/

#$ -o /data/scratch/btx654/

#$ -j y

#$ -pe smp 4

#$ -l h\_vmem=5G

#$ -l h\_rt=10:0:0

module load anaconda3

source activate quast

cd owenia/september2020/

mkdir quast

cd quast

quast \

 /data/scratch/btx654/owenia/september2020/Owenia\_unmasked\_v082020.fa \

 -o /data/scratch/btx654/owenia/september2020//quast/ \

 --eukaryote \

With the info about the genome size we can run the following script. It should be submitted like this:

qsub kimura\_sept2020\_owenia\_v1.sh owenia 500139420

* $1 is the name of the species
* $2 is the genome size in bp

kimura\_sept2020\_owenia\_v1.sh

#!/bin/bash

#$ -wd /data/scratch/btx654/

#$ -o /data/scratch/btx654/

#$ -j y

#$ -pe smp 15

#$ -l h\_vmem=15G

#$ -l h\_rt=72:0:0

#$ -l highmem

genome\_size=$2 #in bp

species\_fasta=Owenia\_unmasked\_v082020.fa

species\_fasta\_path=/data/scratch/btx654/owenia/september2020/$species\_fasta

species\_unmasked="$1"\_unmasked.fa

species\_unmasked\_align="$species\_unmasked".align

teclass\_name=teclass\_"$1".txt

teclass\_path=/data/SBCS-MartinDuranLab/03-Giacomo/data/01-owenia/sept2020/$teclass\_name

output=TEclass\_"$1"

divsum="$output".divsum

html="$output".html

echo "Working on "$1

module load anaconda3

source activate Repeats\_env3

cd owenia/september2020/

mkdir kimura\_highmem

cd RM\*

cp consensi.fa.classified ../kimura\_highmem/ #I have found no genes in step 1.2 so I am using the consensi file obtained in step 1.1

cd ../kimura\_highmem

cp $species\_fasta\_path ./

mv $species\_fasta $species\_unmasked

cp $teclass\_path ./

#the next section will update the filter consensi file obtained in step 1.2 with the annotations from TEclass

grep '>' $teclass\_name | sed 's/|/:/g' | sed 's/#/:/' | awk  ' BEGIN { FS = ":" } ; { OFS="\t" ; print $1,$4}' > table\_of\_changes.tsv

grep '>' consensi.fa.classified | sed 's/|/:/g' | sed 's/#/:/' | awk  ' BEGIN { FS = ":" } ; { OFS="\t" ; print $1}' > fasta\_entries

grep '>' $teclass\_name | sed 's/|/:/g' | sed 's/#/:/' | awk  ' BEGIN { FS = ":" } ; { OFS="\t" ; print $1}' > text\_entries

comm -13 <(sort text\_entries) <(sort fasta\_entries) > missing\_entries

cat missing\_entries | awk '{ OFS="\t" ; print $1,"unclear"}' >>table\_of\_changes.tsv

while read -r line

do

Target=$(echo $line | awk '{ print $1}')

rep=$(echo $line | awk '{ print $2}')

sed -iE 's/'"$Target"'#.\* (/'"$Target"'#'"$rep"' (/' consensi.fa.classified

done <table\_of\_changes.tsv

#end of the section

RepeatMasker -pa 15 -xsmall -gff -a -lib consensi.fa.classified $species\_unmasked

#Kimura distances analyses

/data/home/btx654/.conda/envs/Repeats\_env3/share/RepeatMasker/util/calcDivergenceFromAlign.pl *-s $divsum $*species\_unmasked\_align

/data/home/btx654/.conda/envs/Repeats\_env3/share/RepeatMasker/util/createRepeatLandscape\_TEclass.pl *-div $divsum*  -g *$genome\_size > $*html

Repeat masker generates four main outputs:

* Ofus\_unmasked\_v072019.fa.masked: the masked genome
* Ofus\_unmasked\_v072019.fa.out: a tabular collection of all masked elements, with IDs, etc
* Ofus\_unmasked\_v072019.fa.tbl: a summary table of the masking process, good for publications!
* Ofus\_unmasked\_v072019.fa.out.gff: a GFF file of the masked genome, useful to add as track in IGV.

The Kimura distances analyses will output:

* TEclass\_owenia.html: This file is containing the info that we will need to plot Kimura values. Otherwise one can move this file to his personal pc and click the link to see the plots in a browser.

## 1.5 LTR\_Finder

RepeatModeler is a general prediction tool, and thus it might underperform while predicting specific TE families. LTR transposons are often abundant, specially in plant genomes, and there are tools that deal specifically with its prediction. To improve TE annotation, one can run structural based algorithms such as [LTR\_Finder](https://github.com/xzhub/LTR_Finder). Different from RepeatModeler, LTR\_Finder does not search for sequence similarity, but for the structural sequence elements that characterise LTR transposons.

**Note**: because RepeatModeler/RepeatMasker and LTR\_Finder follow different approaches, they can be run in parallel.

Installation of ltr\_finder with Conda:

module load anaconda2

conda create -n ltr\_finder

source activate ltr\_finder

conda install -c bioconda ltr\_finder

Create another conda environment that will be used to install Perl dependencies:

module load anaconda2

conda create -n myperl perl perl-app-cpanminus

* Once the environment is activated, modules can be installed via cpan or cpanm with no special configuration needed.

source activate myperl

cpanm My::Module

cpanm IO::Scalar

#!/bin/bash

#$ -wd /data/scratch/btx654/

#$ -o /data/scratch/btx654/

#$ -j y

#$ -pe smp 4

#$ -l h\_vmem=5G

#$ -l h\_rt=72:0:0

species\_unmasked=/data/scratch/btx654/owenia/september2020/Owenia\_unmasked\_v082020.fa

txt\_output=Owenia\_Sept2020\_LTRfinder\_FullOutput.txt

gff\_output=Owenia\_Sept2020\_LTRfinder\_FullOutput.gff

cd owenia/september2020/

mkdir ltr\_finder

cd ltr\_finder

module load anaconda2

source activate ltr\_finder

ltr\_finder $species\_unmasked -w 0 -C 2>&1 > $txt\_output

conda deactivate

source activate myperl

perl /data/SBCS-MartinDuranLab/03-Giacomo/src/dawgpaws/MODIFIED\_cnv\_ltrfinder2gff.pl -i $txt\_output -o $gff\_output

LTR\_Finder generates a .txt file, which needs to be converted into .GFF prior merging with RepeatCraft. To do so, we can use a script from the [Dawgpaw](http://dawgpaws.sourceforge.net) package([cnv\_ltrfinder2gff.pl](https://github.com/jestill/dawgpaws/blob/master/scripts/cnv_ltrfinder2gff.pl)). I am using a modified version of it where the first line isted of being “#!/usr/bin/perl -w” is “#!/usr/bin/env perl”. When you specify env the script will look into the anaconda env for the perl modules.

And this generates Owenia\_LTRfinder\_FullOutput.gff file with all the predicted LTRs, which can be used together with the .gff output of RepeatMasker to run RepeatCraft and generate a curated set of repetitive elements.

## 1.6 RepeatCraft

To merge different sources of evidence and curate the potential repetitive regions of the genome, reducing artefacts, we run [RepeatCraft](https://github.com/niccw/repeatcraftp). The configuration file Owenia.repeatcraft.cgf looked like this, with default parameters except for LTR\_finder, for which I calculated from the .gff file the largest LTR found:

calculate the longest ltr:

awk 'BEGIN {FS="\t"}; {print $5 -= $4}' Owenia\_Sept2020\_LTRfinder\_FullOutput.gff | sort -nr | head -n1

#!/bin/bash

#$ -wd /data/scratch/btx654/

#$ -o /data/scratch/btx654/

#$ -j y

#$ -pe smp 1

#$ -l h\_vmem=6G

#$ -l h\_rt=12:0:0

LTR\_gff=Owenia\_Sept2020\_LTRfinder\_FullOutput.gff

LTR\_gff\_original=/data/scratch/btx654/owenia/september2020/ltr\_finder/$LTR\_gff

RM\_out="$1"\_unmasked.fa.out

RM\_out\_original=/data/scratch/btx654/owenia/september2020/kimura\_highmem/$RM\_out

RM\_out\_gff="$1"\_unmasked.fa.out.gff

RM\_out\_gff\_original=/data/scratch/btx654/owenia/september2020/kimura\_highmem/$RM\_out\_gff

output="$1"\_masked\_RepeatCraft

cd owenia/september2020/

mkdir repeatcraft

cd repeatcraft

#create a configuration file that we can modify accordingly to the longest ltr size we have found

echo "#configuration file for RepeatCraft

# Label short TEs

shortTE\_size: 100

mapfile: None

# LTR grouping (based on LTR\_FINDER result).

ltr\_finder\_gff: $LTR\_gff

max\_LTR\_size: 25000

LTR\_flanking\_size: 200

# TEs grouping

gap\_size: 150" > repeatcraft.cgf

# Load python

module load python

# Copy required files into the working directory

cp $RM\_out\_original ./

cp $RM\_out\_gff\_original ./

cp $LTR\_gff\_original ./

# Run RepeatCraft

/data/SBCS-MartinDuranLab/02-Chema/src/repeatcraftp/repeatcraft.py \

 -r $RM\_out\_gff \

 -u $RM\_out \

 -c repeatcraft.cgf \

 -o $output

RepeatCraft will generate several outputs:

* Ofus\_masked\_RM\_RC\_v072019.rclabel.gff: a GFF file with all the evidences, renamed by RepeatCraft.
* Ofus\_masked\_RM\_RC\_v072019.rmerge.gff: the GFF file with all the repeats, merging all evidences.
* Ofus\_masked\_RM\_RC\_v072019.summary.txt: a summary of the run.

## 1.7 Soft-masking

To soft-mask the genome with the final, polished annotation of TEs and repeats, we use [BEDtools](https://bedtools.readthedocs.io/en/latest/).

#!/bin/bash

#$ -wd /data/scratch/btx654/

#$ -o /data/scratch/btx654/

#$ -j y

#$ -pe smp 1

#$ -l h\_vmem=3G

#$ -l h\_rt=0:30:0

unmasked="$1"\_unmasked.fa

unmasked\_original=/data/scratch/btx654/owenia/september2020/kimura\_highmem/$unmasked

masked="$1"\_masked\_RepeatCraft.rmerge.gff

masked\_original=/data/scratch/btx654/owenia/september2020/repeatcraft/$masked

output="$1"\_softmasked.fa

cd owenia/september2020/

mkdir softmasking

cd softmasking

# Load the application module

module load bedtools/2.26.0

# Copy required files into working directory

cp $unmasked\_original ./

cp $masked\_original ./

bedtools maskfasta -soft -fi $unmasked -bed $masked -fo $output

This generates the soft-masked genome assembly Owenia\_softmasked\_v072019.fa, which will be used in all downstream analyses.

## 1.8 Kimura plots

In the html file obtained during the Kimura distances analyses in step 1.4 we can find the informations required to plot these data with R.

First we need to copy-paste these info in order to generate two files that look like this:

table:

Divergence unclear nonLTR DNA LTR LINE Retro SINE

0, 0.206642179894558, 0.00297577023622733, 1.61080164406957, 0.316742279582761, 0.494263179654985, 0.149770238066817, 0.00494202196659483

1, 0.29488937304722, 0.00398628846332489, 2.82234021865343, 0.269521446639819, 0.574690153397627, 0.271607864862962, 0.00544348213944024

2, 0.317798585042547, 0.0040632669986301, 2.74432437259195, 0.269168145154405, 0.484371937728884, 0.226616010391662, 0.00437218086108869

3, 0.397777483726438, 0.00280301840634757, 2.45014800073148, 0.204195662081585, 0.477379087615209, 0.186876091470654, 0.00636262584540927

4, 0.355636834225145, 0.00214020322573254, 2.20615643533957, 0.148996653773062, 0.3568137060662, 0.168448429839823, 0.0103765066148955

5, 0.301087644721146, 0.00204822887186137, 1.97399417146523, 0.144916791401885, 0.302978117581694, 0.149525706252069, 0.00934619390729089

6, 0.262411229252835, 0.00260507360127702, 1.79007305602906, 0.138096093285348, 0.299755616143994, 0.124288943271058, 0.00953194211326114

7, 0.239363255949711, 0.00186527988535677, 1.48807726453556, 0.122190328448815, 0.344873835379743, 0.112410055580102, 0.00886492810344764

8, 0.242699125775769, 0.00108749676240277, 1.37310192425944, 0.123240835525422, 0.313344626984212, 0.116041243059785, 0.0087887493451326

9, 0.193704987301341, 0.000526653148036202, 1.25317876363355, 0.119865776626845, 0.232635132019788, 0.108142645504727, 0.00787880307455069

10, 0.177116012970943, 0.000409885707469329, 1.22066243048788, 0.0983861660014722, 0.178551012835581, 0.114094785809925, 0.0104884753935213

11, 0.154375753864792, 0.000486464354279453, 1.09288665948387, 0.0964435076923151, 0.164249200752862, 0.104452474472018, 0.0123475570072041

12, 0.137114766918392, 0.000577039098417797, 1.04859281038075, 0.0893678806601567, 0.158548990199573, 0.097118719416278, 0.0122175932462992

13, 0.134265961279357, 0.000494462124181293, 0.924270476420355, 0.0887288588449997, 0.164660686014312, 0.0890695638428181, 0.0127408473421271

14, 0.162618455469877, 0.000565642276307674, 0.854410756104768, 0.0942173284401377, 0.164392960666848, 0.0812147540779729, 0.0133312827051305

15, 0.199831878878893, 0.000802176321154609, 0.778722061140472, 0.0925769858332702, 0.152439493771557, 0.0804991536160057, 0.0117717175742716

16, 0.20533514434835, 0.000530452088739576, 0.683648771376589, 0.0843664752520407, 0.132561836457522, 0.0759090335250919, 0.0107180113897041

17, 0.160126350368463, 0.000905147608640807, 0.616042022842351, 0.0803999812692229, 0.123426183842897, 0.0701362432099433, 0.00851922449944058

18, 0.118752687000757, 0.000533451252452766, 0.577206851641488, 0.0727209224979707, 0.122502641363482, 0.0641169216375706, 0.00858040743918966

19, 0.108250615398402, 0.000453273609186814, 0.538037773547224, 0.0721268881385115, 0.121480726314275, 0.0585328786921055, 0.00714660724003719

20, 0.0852244360182607, 0.000310313472191414, 0.51577038258652, 0.0651308389168764, 0.124223361557863, 0.0546997475224009, 0.00676931244491786

21, 0.0644452300920411, 0.00027992194656442, 0.464038007641949, 0.0648825081614243, 0.124558268172503, 0.0533285298727303, 0.00647319501430221

22, 0.0533409264160781, 0.000654417522218105, 0.405780052290219, 0.0639939559253298, 0.121314572644564, 0.0490565210796621, 0.00563922755778779

23, 0.0496717495293612, 0.000647219529306448, 0.366113912796556, 0.0583603268064733, 0.105190068801215, 0.0476709074441683, 0.00611229564748166

24, 0.048097788412679, 0.000496861455151845, 0.323490797825934, 0.0501552147199275, 0.0905249580207055, 0.0447853120635842, 0.00424301687717397

25, 0.040797823934774, 0.000382093457060433, 0.284645629412695, 0.0431381713523001, 0.0969303719350896, 0.0455602959670725, 0.0032454950261669

26, 0.0388883563707096, 0.000302315702289574, 0.260197046655511, 0.0384094898978369, 0.129244161557991, 0.0347936981252148, 0.00288619521332672

27, 0.0302935529456966, 0.000119966548527609, 0.228435702988579, 0.035218179762755, 0.128233843275141, 0.0304874988658163, 0.00279761991166383

28, 0.0267311462871693, 7.35794830969332e-05, 0.194351407053657, 0.0289337321181362, 0.0829110810741533, 0.0294119987582662, 0.0014673908327402

29, 0.0257762125608895, 5.69841105506141e-05, 0.156749691915906, 0.0276886792886671, 0.0533801154885972, 0.0312020996065457, 0.00108149843497639

30, 0.0220198599822426, 8.99749113957064e-05, 0.129004028516688, 0.0224695345949735, 0.0615684322583491, 0.0266597661907954, 0.000780382398172094

31, 0.0188185526347833, 1.05970451199388e-05, 0.11316144606238, 0.0194077883323014, 0.0617273879351482, 0.0184102664812944, 0.00116727451717363

32, 0.0163428429616686, 2.57928079334358e-05, 0.0956987153702062, 0.0211980891248284, 0.0641169216375706, 0.0257248268892702, 0.000736994496454609

33, 0.0161700911317888, 0, 0.0754889506609977, 0.0227956436627211, 0.0611481494500074, 0.0219538783805524, 0.000502060005588042

34, 0.0141248614236406, 7.39793715920253e-06, 0.0667012010371028, 0.0284214749559233, 0.0613606901851488, 0.0177364543670643, 0.00047306808969387

35, 0.0118005095459182, 0, 0.0488969655701204, 0.0205026830318634, 0.0691667135535927, 0.0197624894274481, 0.000333307060659206

36, 0.00975527983777004, 0, 0.0396305494176004, 0.0236881947837665, 0.0555577082886208, 0.0172511896782701, 0.000330707785441108

37, 0.0103531131379326, 0, 0.0314224381673414, 0.0229522000085496, 0.0522682255279938, 0.0136315989649446, 8.83753574153383e-05

38, 0.00961771819545838, 0, 0.0270224650558438, 0.0215477916137864, 0.0512495095867468, 0.0111178998847961, 0.000179749878543867

39, 0.00990303863670654, 0, 0.0187461728171717, 0.0229651963846401, 0.0396107549370933, 0.0109699411416121, 0.000122565823745707

40, 0.00724497980982983, 0, 0.0157386114455845, 0.020753013229791, 0.0354063272996958, 0.00981366355805347, 2.69924734187119e-05

41, 0.00501500161694913, 0, 0.0139589076981774, 0.0292706381752512, 0.0297421067109647, 0.00934199507809243, 0

42, 0.00417603555424605, 0, 0.0131005470434624, 0.036546009510708, 0.0271676245795622, 0.0061654808173289, 2.21938114776076e-05

43, 0.00439857350176477, 0, 0.00904947664393261, 0.0228564267139751, 0.020711624770549, 0.00572680313821294, 0

44, 0.00258487923227487, 0, 0.00997721795254611, 0.0198278711963956, 0.0133648733387182, 0.0036017956752939, 0

45, 0.00342124601975985, 0, 0.00613129035099853, 0.0148008729245937, 0.00786740625244057, 0.00300676159459696, 0

46, 0.00157935961136597, 0, 0.00429420260454575, 0.00970769310685408, 0.00422702133737029, 0.00252429612526843, 0

47, 0.000431879574699391, 0, 0.00419842930997121, 0.00447615186981262, 0.00266225765607518, 0.00116087630125216, 0

48, 0.00114268137472547, 0, 0.00105030713235921, 0.00421242540729943, 0.00183668785795769, 0.00064561997532608, 0

49, 0.00030451508901258, 0, 0.000792578997272401, 0.00231875343879113, 0.00115787713753897, 0.00086335926090369, 0

50, 0.000339305388085586, 0, 0.000576239321427613, 0.000939737963466267, 0.000646019863821172, 7.51790370773014e-05, 0

piechart

Simple\_repeat 3121129

SINE 856876

Retro 14621610

LINE 32471055

LTR 16839526

DNA 147110666

nonLTR 148057

unclear 21891503

Unmasked 263078998

Then we have to modify these two files with the commands:

sed -e 's/ /\t/g' table | sed -e 's/[,]//g' > table\_owenia\_sept2020

to obtain the data for the piechart with percentage

genome\_size=500139420 #obtained with quast in step 1.4

awk '{ print $1, 100\*$2/500139420 }' pie > pie\_owenia\_sept2020

* then we should manually add a line containing “name percentage” as the first line of the file pie\_owenia\_sept2020

We can finally plot our data using Rstudio. The datasets should be imported from text files

*library(ggplot2)*

*library(plyr)*

l*ibrary(reshape2)*

library("RColorBrewer")

#this will generate the Kimura plot using pie\_owenia\_sept2020

lm <- melt(table\_owenia\_sept2020, id.vars=0:1)

ggplot(data=lm, aes(x=lm$Div, y=lm$value, fill=lm$variable))+geom\_bar(stat="identity")+labs(x="Kimura Substitution Level (%)", y="Genome Proportion")+ scale\_fill\_brewer(name=element\_blank(), palette="Set3")+theme\_bw(base\_size = 14)

#this will generate the piechart using pie\_owenia\_sept2020

ggplot(pie\_owenia\_sept2020, aes(x = "", y=percentage, fill=name, order=pie\_owenia\_sept2020$name)) +

    geom\_bar(width = 1, stat = "identity", color = "white") +

    coord\_polar("y", start = 0) +

    scale\_fill\_brewer (name=element\_blank(), palette="Set3") +

    theme\_void()

* The plots have been exported in pdf 5x6.5 inch

# Step 2: Generate gene evidences

This step will align different sources of evidence (e.g. short Illumina reads from RNA-seq data, a *de novo* transcriptome assembly generated with Trinity, spliced-alignments of proteomes of closely-related species, etc) to the soft-masked genome assembly. Each source of evidence can be process independently and in parallel, so that jobs run simultaneously and this step is done faster.

## 2.1 *De novo* transcriptome assembly with Trinity

Independently assembled transcript evidences will be the base for genome annotation. For *Owenia*, we use different de novo transcriptomes:

* Stage-specific RNA-seq data from oocyte to larva (Illumina sequenced)
* Tissue-specific RNA-seq data from the adult (BGI sequenced)

**2.1.1 *De novo* assembly of Illumina strand-specific embryonic data**

ace     ace\_R1  owe\_ace\_R1\_r1\_\_paired.fastq.gz  owe\_ace\_R1\_r2\_paired.fastq.gz

1cell   1cell\_R1        owe\_1cell\_R1\_r1\_\_paired.fastq.gz        owe\_1cell\_R1\_r2\_paired.fastq.gz

2cell   2cell\_R1        owe\_2cell\_R1\_r1\_\_paired.fastq.gz        owe\_2cell\_R1\_r2\_paired.fastq.gz

4cell   4cell\_R1        owe\_4cell\_R1\_r1\_\_paired.fastq.gz        owe\_4cell\_R1\_r2\_paired.fastq.gz

8cell   8cell\_R1        owe\_8cell\_R1\_r1\_paired.fastq.gz owe\_8cell\_R1\_r2\_paired.fastq.gz

3h      3h\_R1   owe\_3h\_R1\_r1\_\_paired.fastq.gz   owe\_3h\_R1\_r2\_paired.fastq.gz

4h      4h\_R1   owe\_4h\_R1\_r1\_\_paired.fastq.gz   owe\_4h\_R1\_r2\_paired.fastq.gz

5h      5h\_R1   owe\_5h\_R1\_r1\_\_paired.fastq.gz   owe\_5h\_R1\_r2\_paired.fastq.gz

9h      9h\_R1   owe\_9h\_R1\_r1\_\_paired.fastq.gz   owe\_9h\_R1\_r2\_paired.fastq.gz

13h     13h\_R1  owe\_13h\_R1\_r1\_\_paired.fastq.gz  owe\_13h\_R1\_r2\_paired.fastq.gz

18h     18h\_R1  owe\_18h\_R1\_r1\_\_paired.fastq.gz  owe\_18h\_R1\_r2\_paired.fastq.gz

27h     27h\_R1  owe\_27h\_R1\_r1\_\_paired.fastq.gz  owe\_27h\_R1\_r2\_paired.fastq.gz

#!/bin/bash

#$ -pe smp 20

#$ -l highmem

#$ -l h\_vmem=10G

#$ -l h\_rt=240:0:0

#$ -cwd

#$ -j y

module load trinity/2.4.0

Trinity --seqType fq --max\_memory 200G --samples\_file tissueLibraries\_R1.txt --SS\_lib\_type RF --CPU 10 --output Oxford\_Illumina\_trinity\_R1

Rename the trinity.fasta file to Oxford.trinity.fasta and run transcriptome stats script:

#!/bin/bash

#$ -cwd

#$ -j y

#$ -pe smp 1

#$ -l h\_vmem=1G

#$ -l h\_rt=1:0:0

module load perl

/share/apps/trinity/r20140717/util/TrinityStats.pl Oxford\_Illumina\_trinity\_R1/Trinity.fasta

Total trinity 'genes':  516585

Total trinity transcripts:      1111027

Percent GC: 37.16

Contig N50: 620

**2.1.2 *De novo* assembly of BGI strand-specific adult data**

BGI provides cleaned data (adaptors removed and quality trimmed reads). Sequencing is based on a rolling circle approach, and paired reads are in a RF orientation. Because we start from several libraries, each for a different tissue, it is better to first define a sample text file (tissueLibraries.txt). In the file, the first column is the tissue, then the replicate (in this case there is only one replicate), then the read 1 and the read 2. Columns are tabulated (separated by tabs)

BV      BV\_R1   Blood-vessel\_1.fq.gz    Blood-vessel\_2.fq.gz

BW      BW\_R1   Body-Wall\_1.fq.gz       Body-Wall\_2.fq.gz

G       G\_R1    Gut\_1.fq.gz     Gut\_2.fq.gz

H       H\_R1    Head\_1.fq.gz    Head\_2.fq.gz

HC      HC\_R1   Head+Chaetae\_1.fq.gz    Head+Chaetae\_2.fq.gz

O       O\_R1    Ovary\_1.fq.gz   Ovary\_2.fq.gz

R       R\_R1    Retractor-muscle\_1.fq.gz        Retractor-muscle\_2.fq.gz

T       T\_R1    Tail\_1.fq.gz    Tail\_2.fq.gz

Ts      Ts\_R1   Testis\_1.fq.gz  Testis\_2.fq.gz

We can launch Trinity with the following command. **Important!!** Because of the way BGI sequences (DNA nanoballs and rolling circle replication), the orientation of pair-end reads is FR, differently from the strand-specific system of Illumina, which is in RF orientation.

#!/bin/bash

#$ -pe smp 20

#$ -l h\_vmem=3G

#$ -l h\_rt=120:0:0

#$ -cwd

#$ -j y

module load trinity/2.4.0

Trinity --seqType fq --max\_memory 60G --samples\_file tissueLibraries.txt --SS\_lib\_type FR --CPU 20 --output BGI\_tissues\_trinity

Rename the trinity.fasta file to BGI.trinity.fasta and run transcriptome stats script:

#!/bin/bash

#$ -cwd

#$ -j y

#$ -pe smp 1

#$ -l h\_vmem=1G

#$ -l h\_rt=1:0:0

module load perl

/share/apps/trinity/r20140717/util/TrinityStats.pl BGI.trinity.fasta

Total trinity 'genes':  213541

Total trinity transcripts:      574180

Percent GC: 36.74

Contig N50: 687

## 2.2 Alignment of d*e novo* transcriptomes to genome with GMAP

In those cases in which there is a pre-existing *de novo* transcriptome available (e.g. assembled with Trinity, as it is the case with *Owenia*), one can directly generate spliced-alignments of the transcripts to the genome with [GMAP](http://research-pub.gene.com/gmap/). As an example for the BGI assembly:

#!/bin/bash

#$ -cwd

#$ -j y

#$ -pe smp 5

#$ -l h\_vmem=10G

#$ -l h\_rt=10:0:0

# Activate the genome annotation pipeline environment installed by James and add all modules into the $PATH

source /data/SBCS-Informatics/chema\_pipeline\_0519/init.sh

# Bring input files to working directory

cp /data/SBCS-MartinDuranLab/02-Chema/00-OweniaGenome/00-DATA/Owenia\_softmasked\_v072019.fa ./

# Make a GMAP index and align transcripts to the genome

gmap\_build -D /data/scratch/btx333/07-GMAP/00-Trinity\_BGI/gmapdb -d Owenia\_softmasked Owenia\_softmasked\_v072019.fa

gmap -D /data/scratch/btx333/07-GMAP/00-Trinity\_BGI/gmapdb -d Owenia\_softmasked -f 3 -n 0 -x 50 -t 5 -B 4 --gff3-add-separators=0 ./BGI\_tissues\_trinity/Trinity.fasta > BGItissuesVsGenome\_gmap.gff3

The output is BGItissuesVsGenome\_gmap.gff3, a GFF file with the aligned transcripts (94.03% alignment rate). The statistics of the aligning process are saved as stdout in the associated job file. Repeat the same with the Oxford-based assembly and the existing transcriptome generated by Harald Hausen (output HaraldRefTransVsGenome\_gmap.gff3).

|  |  |  |  |
| --- | --- | --- | --- |
| Transcriptome | No queries | Unaligned (no paths found) | Alignment rate |
| BGI adult tissues | 574,180 | 34,497 | 93.99% |
| Oxford embryos (R1) | 1,111,027 | 111,172 | 89.99% |
| Ofus\_RefTrans (Sars) | 226,138 | 8,837 | 96.09% |

## 2.3 Illumina short read alignments with STAR

Simultaneously, we can align Illumina short reads (paired end, PE and single end, SE) to the soft masked genome employing [STAR](https://github.com/alexdobin/STAR). To avoid generating a very large single BAM file that will require an enormous amount of RAM (thus either stopping the job or making a very long queuing), it is better to align each library alone. We can do that with an array job. Thereafter, we can use [StringTie](https://ccb.jhu.edu/software/stringtie/index.shtml?t=manual) to generate a GTF file with the assembled transcripts created by STAR.

**2.3.1 Downloading additional libraries from SRA**

In order to maximise the amount of data for genome annotation, it is advisable to check at the SRA for already published Illumina libraries of the species of interest. In the case of *Owenia*, there is the dataset that was published in the annelid phylogeny. To download, generate a .fastq file of the reads, and trim the reads, one can run a code like so:

#!/bin/bash

#$ -cwd

#$ -j y

#$ -pe smp 4

#$ -l h\_vmem=4G

#$ -l h\_rt=12:0:0

# Load sratools

module load sratools/2.8.2-1

module load trimmomatic/0.36

# Download SRA dataset, convert it to fastq and run trimmomatic

prefetch SRR1222288

cp /data/home/btx333/ncbi/public/sra/SRR1222288.sra /data/scratch/btx333/08-STAR

cd /data/scratch/btx333/08-STAR

fastq-dump --gzip --split-files SRR1222288.sra

java -jar /share/apps/trimmomatic/0.36/trimmomatic-0.36.jar PE -threads 1 -phred33 -trimlog trimSRR1222288 ./SRR1222288\_1.fastq.gz ./SRR1222288\_2.fastq.gz SRR1222288\_1\_paired.fastq.gz SRR1222288\_1\_unpaired.fastq.gz SRR1222288\_2\_paired.fastq.gz SRR1222288\_2\_unpaired.fastq.gz ILLUMINACLIP:/share/apps/trimmomatic/0.36/adapters/TruSeq3-PE-2.fa:2:30:10 LEADING:28 TRAILING:28 SLIDINGWINDOW:4:15 MAXINFO:40:0.5 MINLEN:36 > SRR1222288.log 2>&1

**Note2**: before aligning the libraries identify whether they are strand-specific or not. This is an important value that has to be specified at certain steps. For Oceane’s RNA-seq, they are not stranded.

**2.3.2 Mapping short read libraries to the genome**

First generate a genome STAR index. Since this will be common to all libraries, I do not include it in the array job.

#!/bin/bash

#$ -cwd

#$ -j y

#$ -pe smp 4

#$ -l h\_vmem=4G

#$ -l h\_rt=1:0:0

# Activate the genome annotation pipeline environment

source /data/SBCS-Informatics/chema\_pipeline\_0519/init.sh

# Bring input files into working directories

cp /data/SBCS-MartinDuranLab/02-Chema/00-OweniaGenome/00-DATA/Owenia\_softmasked\_v072019.fa ./

# Make genome directory

mkdir ./Owenia\_genome\_STAR

# Build STAR genome index and align the reads generating a BAM sorted by coordinate and a stranded wiggle file to visualise the RNA-seq

STAR --runMode genomeGenerate --runThreadN 5 --genomeDir ./Owenia\_genome\_STAR --genomeFastaFiles ./Owenia\_softmasked\_v072019.fa

Then generate an array job that maps every PE library to *Owenia*’s genome and transforms the BAM into a GTF. To simplify the array job, I rename all libraries to consecutive numbers:

Blood-vessel    1

Body-Wall       2

Gut     3

Head    4

Head+Chaetae    5

Ovary   6

Retractor-muscle        7

Tail    8

Testis  9

owe\_ace\_R1      10

owe\_ace\_R2      11

owe\_1cell\_R1    12

owe\_1cell\_R2    13

owe\_2cell\_R1    14

owe\_2cell\_R2    15

owe\_4cell\_R1    16

owe\_4cell\_R2    17

owe\_8cell\_R1    18

owe\_8cell\_R2    19

owe\_3h\_R1       20

owe\_3h\_R2       21

owe\_4h\_R1       22

owe\_4h\_R2       23

owe\_5h\_R1       24

owe\_5h\_R2       25

owe\_9h\_R1       26

owe\_9h\_R2       27

owe\_13h\_R1      28

owe\_13h\_R2      29

owe\_18h\_R1      30

owe\_18h\_R2      31

owe\_27h\_R1      32

owe\_27h\_R2      33

SRR1222288      34

#!/bin/bash

#$ -cwd

#$ -pe smp 5

#$ -l h\_vmem=5G

#$ -j y

#$ -l h\_rt=8:00:00

#$ -t 1-9

# Activate the genome annotation pipeline environment

source /data/SBCS-Informatics/chema\_pipeline\_0519/init.sh

# Build STAR genome index and align the reads generating a BAM sorted by coordinate, and a stranded wiggle file to visualise the RNA-seq

STAR --runThreadN 5 --genomeDir /data/scratch/btx333/00-BGI\_mapping/Owenia\_genome\_STAR --readFilesCommand zcat --readFilesIn ${SGE\_TASK\_ID}\_1.fq.gz ${SGE\_TASK\_ID}\_2.fq.gz --outSAMtype BAM SortedByCoordinate --outWigType wiggle --outWigStrand Stranded --outFileNamePrefix ./BGI\_library\_${SGE\_TASK\_ID}\_

# Assemble a GTF file from the BAM file

stringtie ./BGI\_library\_${SGE\_TASK\_ID}\_Aligned.sortedByCoord.out.bam -p 5 -o BGI\_library\_${SGE\_TASK\_ID}\_transcripts.gtf -v

The output is a series of BGI/Oxford\_library\_<n>\_transcripts.gtf files, together with the sorted BAM files and wiggle for each strand, which can be uploaded to the genome browser to visualise the RNA-seq data. Before uploading the wiggle files, re-convert the naming to the specific library (e.g. library 9 back to testis, etc)

|  |  |  |  |
| --- | --- | --- | --- |
| Library | No reads | Uniquely mapped reads % | % of reads unmapped |
| Blood-vessel | 26,229,562 | 69.63 | 22.65 |
| Body-Wall | 25,030,358 | 83.16 | 12.01 |
| Gut | 26,223,150 | 84.17 | 11.94 |
| Head | 25,135,465 | 80.84 | 14.5 |
| Head+Chaetae | 25,164,246 | 86.52 | 9.15 |
| Ovary | 25,127,050 | 79.03 | 17.01 |
| Retractor-muscle | 26,221,434 | 85.39 | 6.95 |
| Tail | 25,090,293 | 79.08 | 16.14 |
| Testis | 26,332,262 | 91.92 | 2.62 |
| owe\_ace\_R1 | 42,760,709 | 70.64 | 24.92 |
| owe\_ace\_R2 | 41,755,797 | 67.93 | 27.6 |
| owe\_1cell\_R1 | 41,650,434 | 72.11 | 22.44 |
| owe\_1cell\_R2 | 47,164,566 | 74.43 | 20.8 |
| owe\_2cell\_R1 | 44,590,774 | 71.93 | 22.85 |
| owe\_2cell\_R2 | 43,188,381 | 72.87 | 22.36 |
| owe\_4cell\_R1 | 45,902,725 | 72.74 | 22.38 |
| owe\_4cell\_R2 | 49,945,828 | 72.58 | 22.49 |
| owe\_8cell\_R1 | 45,680,904 | 72.80 | 22.08 |
| owe\_8cell\_R2 | 49,401,610 | 73.74 | 21.57 |
| owe\_3h\_R1 | 47,976,852 | 69.97 | 24.9 |
| owe\_3h\_R2 | 44,035,927 | 70.40 | 24.65 |
| owe\_4h\_R1 | 45,997,615 | 71.61 | 23.27 |
| owe\_4h\_R2 | 52,692,310 | 73.76 | 21.41 |
| owe\_5h\_R1 | 54,631,086 | 71.55 | 23.07 |
| owe\_5h\_R2 | 46,947,697 | 74.61 | 20.45 |
| owe\_9h\_R1 | 52,833,442 | 67.74 | 26.35 |
| owe\_9h\_R2 | 49,640,037 | 74.22 | 20.34 |
| owe\_13h\_R1 | 51,921,579 | 72.55 | 21.91 |
| owe\_13h\_R2 | 51,959,906 | 72.48 | 21.95 |
| owe\_18h\_R1 | 41,396,469 | 70.46 | 24.15 |
| owe\_18h\_R2 | 49,610,840 | 76.32 | 18.26 |
| owe\_27h\_R1 | 41,087,129 | 75.59 | 18.27 |
| owe\_27h\_R2 | 59,243,774 | 76.69 | 18.13 |
| SRR1222288 | 23,309,147 | 75.57 | 18.44 |

**2.3.1 Convert Wig tracks to BigWig**

wig tracks are a useful way to visualise RNA-seq data, but occupy quite a lot of space. Therefore, it is better to convert them to BigWig format, which is a binary format. We can use the [UCSC tool](https://genome.ucsc.edu/goldenPath/help/bigWig.html) to do so. For BGI libraries:

#!/bin/bash

#$ -cwd

#$ -j y

#$ -l highmem

#$ -pe smp 4

#$ -l h\_vmem=5G

#$ -l h\_rt=12:0:0

#$ -t 1-34

wigToBigWig BGI\_library\_${SGE\_TASK\_ID}\_Signal.UniqueMultiple.str1.out.wig ChromSizes BGI\_library\_${SGE\_TASK\_ID}\_Signal.UniqueMultiple.str1.out.bw

wigToBigWig BGI\_library\_${SGE\_TASK\_ID}\_Signal.UniqueMultiple.str2.out.wig ChromSizes BGI\_library\_${SGE\_TASK\_ID}\_Signal.UniqueMultiple.str2.out.wig

## 2.4 Generate curated intron junctions with Portcullis

In order to improve prediction of intron junctions, it is important to generate a well-curated dataset from the RNA-seq alignments. We can do so with [Portcullis](https://portcullis.readthedocs.io), which can be easily installed via conda.

module load anaconda2

conda config --add channels bioconda

conda create --yes --name portcullis portcullis

Once created, one can load conda, and activate the environment with portcullis to run the pipeline. It is better (and faster) to run portcullis on each BAM file independently and thereafter merge the junction BED  files.

#!/bin/bash

#$ -cwd

#$ -j y

#$ -l highmem

#$ -pe smp 4

#$ -l h\_vmem=5G

#$ -l h\_rt=12:0:0

#$ -t 1-34

module load anaconda2

source activate portcullis

#Run portcullis

portcullis full -t 4 -v --bam\_filter --orientation FR --strandedness firststrand -o portcullis\_library\_${SGE\_TASK\_ID} Owenia\_softmasked\_v082020.fa Library\_${SGE\_TASK\_ID}\_Aligned.sortedByCoord.out.bam

#Deactivate environment

source deactivate

Portcullis generates multiple files (e.g. a filtered BAM), but also the curated set of junctions in portcullis\_out/3-filt/portcullis\_filtered.pass.junctions.bed, which are the ones we will use in downstream analyses ([Step3](https://paper.dropbox.com/doc/Genome-annotation-pipeline--Agw0qUzYaN3tDnDBwFigm1upAg-8u4ZjaBxKMwaaUFRG0F8m#:uid=071853820811796006522229&h2=Step-3:-Merging-all-gene-evide)).

To merge the bed files, we can use junctools, which comes with portcullis:

#!/bin/bash

#$ -cwd

#$ -j y

#$ -l highmem

#$ -pe smp 1

#$ -l h\_vmem=20G

#$ -l h\_rt=12:0:0

module load anaconda2

source activate portcullis

junctools set -m 1 --operator max -o Owenia\_junctions\_consensus.bed consensus portcullis\_library\_1/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_2/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_3/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_4/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_5/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_6/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_7/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_8/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_9/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_10/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_11/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_12/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_13/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_14/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_15/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_16/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_17/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_18/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_19/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_20/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_21/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_22/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_23/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_24/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_25/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_26/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_27/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_28/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_29/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_30/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_31/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_32/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_33/3-filt/portcullis\_filtered.pass.junctions.bed portcullis\_library\_34/3-filt/portcullis\_filtered.pass.junctions.bed

source deactivate

This generates 749,824 distinct junctions.

# Step 3: Merging all gene evidences with Mikado

[Mikado](https://mikado.readthedocs.io/en/latest/index.html) is a pipeline that allows generating the “best” set of transcripts from multiple transcript assemblies. Employing multiple input files and evidences (e.g. RNA-seq alignments, BLAST homology, curated splice junctions, etc), Mikado will select the best-scoring transcripts as the primary transcript of their respective gene loci. The final output will be a curated transcriptome-based genomic annotation that can thereafter be used as experimental hints for gene prediction  ([Step4](https://paper.dropbox.com/doc/Genome-annotation-pipeline--Agw0qUzYaN3tDnDBwFigm1upAg-8u4ZjaBxKMwaaUFRG0F8m#:uid=331663410328713962126515&h2=Step-4:-Generating-ab-initio-g)).

**Note:** It is important to use the last version of Mikado, as it has significant improvements in performance. James has installed an environment (see [below](https://paper.dropbox.com/doc/Genome-annotation-pipeline-8u4ZjaBxKMwaaUFRG0F8m#:uid=057514426609565932153110&h2=3.1-Install-Mikado)).

## 3.1 Install Mikado in your user

To avoid conflicting environments with the Mikado copy installed in the initial bundle, install the newest Mikado starting from a clean shell. Then:

module load anaconda3

cd /data/SBCS-Informatics/chema\_pipeline\_0519/mikado

conda env create -f environment.yml

source activate mikado2.1

pip install dist/\*whl

This should have installed the environment, which can be called with conda activate mikado2.1, and removed with conda deactivate. However, to make Mikado run, you need to copy the scoring file into a folder that Mikado can access (the home folder will change from user to user):

cp human.yaml /data/home/btx333/.conda/envs/mikado2/lib/python3.7/site-packages/Mikado/configuration/scoring\_files/

After doing that, Mikado should be ready to go.

## 3.2 Generate configuration file and merge all input files into a single “all-contained” GTF file

As a first step, one needs to generate a configuration file that tells Mikado where all input evidences are, what type they are, whether they are stranded or not, and what value should Mikado add to each of them. For Owenia, that Owenia\_mikadoInput\_list file looks like so:

/data/scratch/btx333/10-Mikado/mikado\_input/BGI\_library\_1\_transcripts.gtf       stringtie1      True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/BGI\_library\_2\_transcripts.gtf       stringtie2      True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/BGI\_library\_3\_transcripts.gtf       stringtie3      True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/BGI\_library\_4\_transcripts.gtf       stringtie4      True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/BGI\_library\_5\_transcripts.gtf       stringtie5      True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/BGI\_library\_6\_transcripts.gtf       stringtie6      True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/BGI\_library\_7\_transcripts.gtf       stringtie7      True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/BGI\_library\_8\_transcripts.gtf       stringtie8      True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/BGI\_library\_9\_transcripts.gtf       stringtie9      True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/Oxford\_library\_10\_transcripts.gtf   stringtie10     True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/Oxford\_library\_11\_transcripts.gtf   stringtie11     True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/Oxford\_library\_12\_transcripts.gtf   stringtie12     True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/Oxford\_library\_13\_transcripts.gtf   stringtie13     True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/Oxford\_library\_14\_transcripts.gtf   stringtie14     True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/Oxford\_library\_15\_transcripts.gtf   stringtie15     True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/Oxford\_library\_16\_transcripts.gtf   stringtie16     True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/Oxford\_library\_17\_transcripts.gtf   stringtie17     True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/Oxford\_library\_18\_transcripts.gtf   stringtie18     True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/Oxford\_library\_19\_transcripts.gtf   stringtie19     True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/Oxford\_library\_20\_transcripts.gtf   stringtie20     True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/Oxford\_library\_21\_transcripts.gtf   stringtie21     True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/Oxford\_library\_22\_transcripts.gtf   stringtie22     True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/Oxford\_library\_23\_transcripts.gtf   stringtie23     True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/Oxford\_library\_24\_transcripts.gtf   stringtie24     True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/Oxford\_library\_25\_transcripts.gtf   stringtie25     True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/Oxford\_library\_26\_transcripts.gtf   stringtie26     True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/Oxford\_library\_27\_transcripts.gtf   stringtie27     True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/Oxford\_library\_28\_transcripts.gtf   stringtie28     True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/Oxford\_library\_29\_transcripts.gtf   stringtie29     True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/Oxford\_library\_30\_transcripts.gtf   stringtie30     True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/Oxford\_library\_31\_transcripts.gtf   stringtie31     True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/Oxford\_library\_32\_transcripts.gtf   stringtie32     True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/Oxford\_library\_33\_transcripts.gtf   stringtie33     True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/SRA\_library\_34\_transcripts.gtf      stringtie34     True    0.5

/data/scratch/btx333/10-Mikado/mikado\_input/BGItissuesVsGenome\_gmap.gff3        gmap1   False   -0.5

/data/scratch/btx333/10-Mikado/mikado\_input/HaraldRefTransVsGenome\_gmap.gff3    gmap2   False   -0.5

/data/scratch/btx333/10-Mikado/mikado\_input/OxfordtissuesR1VsGenome\_gmap.gff3   gmap3   False   -0.5

To generate protein homology, one also needs to generate a BLAST database. Download the curated [Swissprot](https://www.uniprot.org/downloads) database and upload it to Apocrita (to /data/SBCS-MartinDuranLab/00-BlastDBs/. Rename it with the date of download (these databases change regularly, and it is important to re-download it regularly). Use DIAMOND Blast to generate a BLAST database.

module load diamond/0.9.22

mv uniprot\_sprot.fasta Uniprot\_SwissProt\_160719.fasta

diamond makedb --in Uniprot\_SwissProt\_160719.fasta --db Uniprot\_SwissProt\_160719

Then, run Mikado to generate a configuration file that tells the program how genes should look like (for that we initially use the score parameters based on human/vertebrate genes, but for future annotation versions and other annelid genomes, we should consider adjusting these scores to annelid genomes, so that Mikado is as accurate as possible). Once the configuration file is done, Mikado will collect all evidences and merge them in a single annotation (that is mikado prepare)

#!/bin/bash

#$ -cwd

#$ -j y

#$ -pe smp 6

#$ -l h\_vmem=3G

#$ -l h\_rt=24:0:0

# Activate the genome annotation pipeline environment installed by James and add all modules into the $PATH

module load anaconda3

conda activate mikado2.1

# Bring genome into the working directory

cp /data/SBCS-MartinDuranLab/02-Chema/00-OweniaGenome/00-DATA/Owenia\_softmasked\_v082020.fa ./

# All required files (gtfs and filtered junctions) are in the folder ./mikado\_input and the file Owenia\_mikadoInput\_list.txt points to them. First step is generate configuration file

mikado configure -t 6 --list Owenia\_mikadoInput\_list.txt --reference Owenia\_softmasked\_v082020.fa --mode permissive --scoring mammalian.yaml --copy-scoring mammalian.yaml --junctions ../00-Mikado\_input/Owenia\_junctions\_consensus.bed -bt /data/SBCS-MartinDuranLab/00-BlastDBs/Uniprot\_SwissProt\_160719.fasta configuration.yaml

Before running the next step, open the configuration.yaml file and modify the parameter max\_intron\_length for the prepare section. By default, it is set to 1,000,000 bp (1Mb), but that generates problems at the last pick stage, because Mikado generates super locus that are too complex to resolve. Considering the *Capitella* and *Dimorphilus* genome, a reasonable maximum intron length is 50kb (50,000). Once that is changed, we can run mikado prepare:

#!/bin/bash

#$ -cwd

#$ -j y

#$ -l highmem

#$ -pe smp 6

#$ -l h\_vmem=3G

#$ -l h\_rt=12:0:0

module load anaconda3

conda activate mikado2.1

mikado prepare --json-conf configuration.yaml

The output of this script is two files: mikado\_prepared.fasta  and mikado\_prepared.gtf.

## 3.3 Generate homology information and potential ORFs

Before generating a consensus set of genes/proteins, it is recommended to generate homology evidence for each of the predicted transcripts provided to Mikado, so that the program can use that information to improve annotation. To speed up the process, one can use [DIAMOND Blast](https://github.com/bbuchfink/diamond). Apocrita has an old version, better install the newest stable via conda:

#!/bin/bash

#$ -cwd

#$ -j y

#$ -l highmem

#$ -pe smp 10

#$ -l h\_vmem=4G

#$ -l h\_rt=1:0:0

module load anaconda3

conda activate diamond

diamond blastx --query ../mikado\_prepared.fasta --max-target-seqs 5 --sensitive --index-chunks 1 --threads 10 --db /data/SBCS-MartinDuranLab/00-BlastDBs/Uniprot\_SwissProt\_160719.dmnd --evalue 1e-6 --outfmt 5 --out mikado.diamond.xml

The output is mikado.diamond.xml, which we can use in the final steps of Mikado.

In parallel, generate ORF predictions of the mikado\_prepared.fasta transcripts employing [Transdecoder](https://github.com/TransDecoder/TransDecoder/wiki) (installed in my $PATH):

#!/bin/bash

#$ -cwd

#$ -j y

#$ -l highmem

#$ -pe smp 4

#$ -l h\_vmem=3G

#$ -l h\_rt=8:0:0

# Load the application module

module load perl

TransDecoder.LongOrfs -t mikado\_prepared.fasta

TransDecoder.Predict -t mikado\_prepared.fasta

Transdecoder generates a series of files, among them mikado\_prepared.fasta.transdecoder.bed, which contains the predicted ORFs from the (redundant) genes initially generated by Mikado.

## 3.4 Merge all info and generate final output consensus gene set

As a final step, Mikado will take all evidences (input alignments, BLAST homologies, ORF predictions, etc) to generate best-fit gene models. To do so, one needs to run the Mikado steps serialise and pick:

#!/bin/bash

#$ -cwd

#$ -j y

#$ -pe smp 1

#$ -l highmem

#$ -l h\_vmem=10G

#$ -l h\_rt=2:0:0

# Activate the genome annotation pipeline environment installed by James and add all modules into the $PATH

module load anaconda3

conda activate mikado2.1

# serialise and pick

mikado serialise --procs 1 --json-conf configuration.yaml --xml ./00-Diamond/mikado.diamond.xml --orfs ./01-Transdecoder/mikado\_prepared.fasta.transdecoder.bed --blast\_targets /data/SBCS-MartinDuranLab/00-BlastDBs/Uniprot\_SwissProt\_160719.fasta --transcripts mikado\_prepared.fasta --junctions ../00-Mikado\_input/Owenia\_junctions\_consensus.bed

mikado pick --procs 40 --json-conf configuration.yaml

The final output of this step is mikado.loci.gff3 file, with all the transfrags (~ genes, isoforms, non coding RNAs, super loci, etc) considered by Mikado. This is a curated RNA-seq dataset that can be used as hints for Augustus ([Step4](https://paper.dropbox.com/doc/Genome-annotation-pipeline--AhQXhh_0sETRQlkWUPnaSutNAg-8u4ZjaBxKMwaaUFRG0F8m#:uid=331663410328713962126515&h2=Step-4:-Generating-ab-initio-g)) and to update the annotation with PASA ([Step5](https://paper.dropbox.com/doc/Genome-annotation-pipeline-8u4ZjaBxKMwaaUFRG0F8m#:uid=939399985718456148312484&h2=Step-5:-Merging-Augustus-and-M)).

**Note1:** With the update of Diamond, Mikado now gives an error for the .xml format of Diamond. This is probably a bug, since Mikado seems to load queries, targets and alignments properly, and the BLAST metrics and scores are incorporated during the pick stage.

**Note2:** My experience is that it is better to use just 1 processor for mikado serialise.

# Step 4: Generating gene predictions with Augustus

[Augustus](https://github.com/Gaius-Augustus/Augustus) is a pipeline that can either generate *ab initio* gene predictions (without any sort of guidance) or incorporate curated gene/protein/intron hints to generate high confident gene models. We will take advantage of these functionalities to train and run Augustus using our Mikado gene set ([Step3](https://paper.dropbox.com/doc/Genome-annotation-pipeline--AhQXhh_0sETRQlkWUPnaSutNAg-8u4ZjaBxKMwaaUFRG0F8m#:uid=071853820811796006522229&h2=Step-3:-Merging-all-gene-evide)) and Portcullis exon junctions ([Step2](https://paper.dropbox.com/doc/Genome-annotation-pipeline--AhQXhh_0sETRQlkWUPnaSutNAg-8u4ZjaBxKMwaaUFRG0F8m#:uid=834308701252110318333419&h2=2.3-Generate-curated-intron-ju)). In addition, we will provide spliced-aligned proteins from a related species. A complete pipeline of how to use Augustus can be found [here](https://currentprotocols.onlinelibrary.wiley.com/doi/pdf/10.1002/cpbi.57).

## 4.1 Training Augustus

**4.1.1 First round of training**

Use the script select\_mik\_train.py ([link](https://www.dropbox.com/s/kgtty04uwwfig67/select_mik_train.py?dl=0,)) to extract (i) full-length (start and stop codon) (ii) non-redudant transcripts with (iii) a blast hit (on a given fraction of their length, flag -f) and (iv) at least *n* exons (at least 2, flag -e):

select\_mik\_train.py -f 0.5 -e 2 mikado.loci.metrics.tsv mikado.loci.gff3

This will generate a file called training.gff3 containing the coding transcripts selected for training Augustus.

For training, Augustus needs to have full access to the /path/to/Augustus/config directory. Since the HPC installation does not give local access, I recommend downloading Augustus to your local src folder (which should be in your PATH) and pointing Augustus to that folder, but still running the HPC installation.

#!/bin/bash

#$ -cwd

#$ -j y

#$ -pe smp 4

#$ -l h\_vmem=1G

#$ -l h\_rt=48:0:0

# Load the application module

module load augustus/3.2.3

# Copy genome into working directory

cp /data/SBCS-MartinDuranLab/03-OweniaGenome/00-DATA/Owenia\_softmasked\_v072019.fa ./

# Set environmental variables and run training employing an Augustus script that does the entire training pipeline

export AUGUSTUS\_CONFIG\_PATH=/data/SBCS-MartinDuranLab/02-Chema/src/Augustus/config

mkdir aug\_training

autoAugTrain.pl --trainingset=training.gff3  --genome=Owenia\_softmasked\_v072019.fa --species=Owenia --workingdir=aug\_training --optrounds=1 --verbose

For the first time, it is better to apply one round of optimisation and check the sensitivity and specificity at the nucleotide and exon level at the end of the training report, which is in aug\_training/autoAugTrain/training/test/augustus.2.withoutCRF.out. If these parameters are not satisfactory, you can consider going for 2 rounds of optimisation and using CRF training instead of HMM (but check discussion regarding CRF training [here](https://currentprotocols.onlinelibrary.wiley.com/doi/pdf/10.1002/cpbi.57)).

The results after one round of optimisation are at the nucleotide level 94.5% of sensitivity and 26.9% of specificity, and at the exon level 83.2% of sensitivity and 38.3% of specificity. However, gene level prediction is very low: 18.5% sensitivity and 4.28% specificity. This might have something to do with the STOP codon, which might not be part of the last exon (see [here](https://currentprotocols.onlinelibrary.wiley.com/doi/pdf/10.1002/cpbi.57) to an explanation)

## 4.2 Generate gene hints

**4.2.1 Generate species-specific exon hints**

We can use the script gtfToHintsMik.py ([link](https://www.dropbox.com/s/wqmwlj6c9dn440l/gtfToHintsMik.py?dl=0)) to convert mikado.loci.gff3 into hints.

cp /data/SBCS-MartinDuranLab/03-OweniaGenome/10-Mikado/mikado.loci.gff3 ./

# Convert gff3 to gtf (-T flag) and keep only coding transfrag (-C flag), highlighting any error (-E flag)

gffread -T -E -C -o mikado.loci.gtf mikado.loci.gff3

./gtfToHintsMik.py mikado.loci.gtf

The output is mikado.loci.exh.gff (exh = exon hints)

**4.2.2 Generate species-specific intron hints**

We can use the information generated by Portcullis, to filter the original BAM file with all the RNA-seq data and generate a filtered BAM with reliable junctions. Then, we call an Augustus script to generate intron hints from the filtered BAM file of each library.

#!/bin/bash

#$ -cwd

#$ -j y

#$ -pe smp 4

#$ -l highmem

#$ -l h\_vmem=1G

#$ -l h\_rt=6:0:0

#$ -t 1-34

# Load the application module

module load augustus/3.2.3

# Get intron hints

bam2hints --intronsonly --minintronlen=15 --in=/data/scratch/btx333/01-Annotation/08-STAR/03-Portcullis/portcullis\_library\_${SGE\_TASK\_ID}/portcullis.filtered.bam --out=/data/scratch/btx333/01-Annotation/11-Augustus/00-Hints/Owenia.intronhints\_${SGE\_TASK\_ID}.gff

This generates 34 Owenia.intronhints.gff

**4.2.3 Generate spliced protein alignments with Exonerate**

*Capitella* has a well-conserved gene repertoire, and as such, aligning its proteome (without TEs, filtered in [Step 1 after running RepeatModeler](https://paper.dropbox.com/doc/Genome-annotation-pipeline--AgcWjA2ABQCbY7J2EML8x~lpAg-8u4ZjaBxKMwaaUFRG0F8m#:uid=284257211074393602529197&h2=1.2-Filtering-bona-fide-genes-)) with [Exonerate](https://www.ebi.ac.uk/about/vertebrate-genomics/software/exonerate) can provide evidences of genes not present in the RNA-seq data and/or support those obtained from species-specific RNA-seq/transcriptome alignments. Exonerate is slow. Before running it, split the fasta in multiple files with fastasplit

module load exonerate/2.4.0

fastasplit -f cte\_prot\_NR\_noTE.fasta -o ./ -c 100

Now we can create an array job that takes it chunk and runs Exonerate on it. This involves renaming the chunks to something like this cte\_prot\_NR\_noTE.fasta\_chunk\_1 with the last number being the number of the chunk, from 1 to 100. That will help build the array of jobs. Dividing the proteome in 100 chunks generates files of <400 proteins, which is quite efficient.

#!/bin/bash

#$ -cwd

#$ -pe smp 4

#$ -l h\_vmem=4G

#$ -j y

#$ -l h\_rt=6:00:00

#$ -t 1-100

# Load the application module

module load exonerate/2.4.0

# Run exonerate

exonerate --model protein2genome --cores 4 --bestn 3 --showtargetgff T --showvulgar F --maxintron 500000 --fsmmemory 16000 --softmasktarget T cte\_prot\_NR\_noTE.fasta\_chunk\_${SGE\_TASK\_ID} Owenia\_softmasked\_v072019.fa > Owenia\_CapitellaProt\_Exo\_${SGE\_TASK\_ID}.gff

Merge all .gff files generated by Exonerate into one and extract a hint file employing Ferdi’s script [exoToHints.py](https://www.dropbox.com/s/bw16z5l1mbtn1w0/exoToHints.py?dl=0).

cat Owenia\_CapitellaProt\_Exo\_\*.gff > Owenia\_CapitellaProt\_Exo.gff

grep "^Scaffold" Owenia\_CapitellaProt\_Exo.gff > Owenia\_CapitellaProt\_Exo\_parsed.gff

./exoToHints.py Owenia\_CapitellaProt\_Exo\_parsed.gff

This generates Owenia\_CapitellaProt\_Exo\_parsed.exh.gff (exh = exonerate hints)

**4.2.4 Merge all hints together and build a configuration file**

All the steps to generate hints include in the final outputs the type of hint and the source, meaning that one can concatenate all of them in one single file.

cat mikado.loci.exh.gff Owenia\_CapitellaProt\_Exo\_parsed.exh.gff Owenia.intronhints.gff > Owenia\_merged\_hints.gff

After merging, one needs to modify the configuration file accordingly:

[GENERAL]

   exonpart      100   .7       E 1  1e+3    P  1       1

     intron      100   .7       E 1  1e+3    P  1       1

    CDSpart      10     1       E 1     1    P  1      10

## 4.3 Augustus run

Run Augustus:

#!/bin/bash

#$ -cwd

#$ -j y

#$ -l highmem

#$ -pe smp 30

#$ -l h\_vmem=3G

#$ -l h\_rt=72:0:0

# Load the application module

module load augustus/3.2.3

# Run augustus

export AUGUSTUS\_CONFIG\_PATH=/data/SBCS-MartinDuranLab/02-Chema/src/Augustus/config

augustus --uniqueGeneId=true --gff3=on --species=Owenia --hintsfile=Owenia\_merged\_hints.gff --extrinsicCfgFile=extrinsic.Ferdi.E.W.P.cfg --allow\_hinted\_splicesites=atac --alternatives-from-evidence=false Owenia\_softmasked\_v082020.fa > Owenia.aug.out

Augustus predicted 28,141 protein coding genes.

# Step 5: Merging Augustus and Mikado into a single gene set

To do so, we load the Augustus predicted proteins into [PASA](https://github.com/PASApipeline/PASApipeline/wiki) (which are based on curated hints and are solid predictions), and thereafter update them with the Mikado output, which is based on RNA-seq and thus contains UTR information, isoforms, etc.

Follow the instructions in the website to install PASA. In order to install the required Perl modules locally, use [cpanm](https://metacpan.org/pod/distribution/App-cpanminus/bin/cpanm) and [local::lib](https://metacpan.org/pod/local::lib). To use PASA with a SQLite database (easier to install and manage in Apocrita), you need to use the configuration file specifying the entire path to the database to be created. For instance, a standard alignment configuration file would look like this:

## templated variables to be replaced exist as <\_\_var\_name\_\_>

# database settings

DATABASE=/data/scratch/btx333/12-PASA/OweniaPASA\_Augustus1st

#######################################################

# Parameters to specify to specific scripts in pipeline

# create a key = "script\_name" + ":" + "parameter"

# assign a value as done above.

#script validate\_alignments\_in\_db.dbi

validate\_alignments\_in\_db.dbi:--MIN\_PERCENT\_ALIGNED=<\_\_MIN\_PERCENT\_ALIGNED\_\_>

validate\_alignments\_in\_db.dbi:--MIN\_AVG\_PER\_ID=<\_\_MIN\_AVG\_PER\_ID\_\_>

#script subcluster\_builder.dbi

subcluster\_builder.dbi:-m=50

## 5.1 Extract protein coding genes from Augustus loci

One can extract the Augustus-based protein coding transcripts (CDS) using gffread:

gffread Owenia.aug.out -V -w Owenia.Augustus.genes.fa -g Owenia\_softmasked\_v072019.fa

## 5.2 Run SeqClean on the Augustus coding genes

First, run seqclean. There is no need to include vector sequences, because the transcripts do not derive from a *de novo* assembly. seqclean will treat polyA regions, if included in the Augustus sets, and that will aid to better annotate terminating sites.

#!/bin/bash

#$ -cwd

#$ -j y

#$ -l highmem

#$ -pe smp 4

#$ -l h\_vmem=2G

#$ -l h\_rt=4:0:0

module load perl

module load samtools/1.9

/data/SBCS-MartinDuranLab/02-Chema/src/PASApipeline-v2.3.3/bin/seqclean Owenia.Augustus.genes.fa

The output is Owenia.Augustus.genes.fa.clean

## 5.3 Run PASA pipeline with the clean transcripts

We call the main PASA pipeline to generate the initial database with the transcripts, including the following flags: -C creates the MYSQL database, -R runs the alignment/assembly pipeline, -g inputs the genome, -t  inputs the cleaned transcripts, -c inputs the configuration file, -u the original transcripts and -T to indicate that the transcripts were trimmed using the TGI seqclean tool. You can use a standard alignAssembly.config file (as [above](https://paper.dropbox.com/doc/Genome-annotation-pipeline-8u4ZjaBxKMwaaUFRG0F8m#:uid=939399985718456148312484&h2=Step-5:-Merging-Augustus-and-M))

#!/bin/bash

#$ -cwd

#$ -j y

#$ -l highmem

#$ -pe smp 40

#$ -l h\_vmem=1G

#$ -l h\_rt=18:0:0

module load perl

module load samtools/1.9

/data/SBCS-MartinDuranLab/02-Chema/src/PASApipeline-v2.3.3/Launch\_PASA\_pipeline.pl -c alignAssembly.config -C -R -g Owenia\_softmasked\_v082020.fa -t Owenia.Augustus.genes.fa.clean -T -u Owenia.Augustus.genes.fa --ALIGNERS blat --CPU 40

## 5.4 Load the Mikado predictions to the PASA database

Once the database is created, we can incorporate the loci of Mikado. However, Mikado predicts both coding and non-coding genes, and the latter cannot be loaded into PASA. First, we need to “clean” the Mikado GFF3 file, to remove some non-canonical terms (e.g. super loci). We can do that with [AGAT](https://github.com/NBISweden/AGAT)pipeline:

module load anaconda3

conda activate AGAT

agat\_convert\_sp\_gxf2gxf.pl -g mikado.loci.gff3 -o mikado.loci.AGAT.gff3

Then, we need to remove the non-protein coding genes, which appear as ncRNA\_gene feature. We can grep the IDs of this genes, and then remove them from the GFF3 file with fgrep (much much faster than grep when working with large files).

fgrep -v -w -f mikado.ncRNA.IDs mikado.loci.AGAT.gff3 > mikado.loci.AGAT.NOncRNA.gff3

Once this is done, we can load the Mikado coding loci:

#!/bin/bash

#$ -cwd

#$ -j y

#$ -pe smp 2

#$ -l highmem

#$ -l h\_vmem=1G

#$ -l h\_rt=18:0:0

module load perl

module load samtools/1.9

/data/SBCS-MartinDuranLab/02-Chema/src/PASApipeline-v2.3.3/scripts/Load\_Current\_Gene\_Annotations.dbi -c alignAssembly.config -g Owenia\_softmasked\_v082020.fa -P mikado.loci.AGAT.NOncRNA.gff3

## 5.5 Update the PASA database with Mikado prediction

Next, compare the two annotations (the initially loaded based on Mikado and the *ab initio* of Augustus) and generate an updated, consensus one.

#!/bin/bash

#$ -cwd

#$ -j y

#$ -l highmem

#$ -pe smp 30

#$ -l h\_vmem=1G

#$ -l h\_rt=36:0:0

module load perl

module load samtools/1.9

/data/SBCS-MartinDuranLab/02-Chema/src/PASApipeline-v2.3.3/Launch\_PASA\_pipeline.pl -c annotCompare.config -A -g Owenia\_softmasked\_v082020.fa -t Owenia.Augustus.genes.fa.clean --CPU 30

The output will have the format . Repeat the steps [5.4](https://paper.dropbox.com/doc/Genome-annotation-pipeline-8u4ZjaBxKMwaaUFRG0F8m#:uid=504609117209254916873632&h2=5.4-Load-the-Augustus-predicti) and [5.5](https://paper.dropbox.com/doc/Genome-annotation-pipeline-8u4ZjaBxKMwaaUFRG0F8m#:uid=559789846707187497974684&h2=5.5-Update-the-PASA-database-w) again, starting with the output of the first update, to incorporate and update as many possible genes (recommended by PASA). The final output of these two rounds of updating the annotation is OweniaPASA\_Augustus1st.gene\_structures\_post\_PASA\_updates.5127.gff3. We can use AGAT to calculate some statistics:

|  |  |
| --- | --- |
| **Type** | **Number** |
| **Gene** | **35,279** |
| **Transcript** | **40,268** |
| **5’ UTR** | 36,222 |
| **Exon** | 265,085 |
| **CDS** | 250,227 |
| **3’ UTR** | 33,349 |

# Step 6: Filtering Gene Models

Now that we have a merged all gene annotations, and before we start analysing this gene repertoire, we need to remove **(i)** spurious gene models; and **(ii)** genes that correspond to repeats and transposable elements (TEs) (annotated in [Step1](https://paper.dropbox.com/doc/Genome-annotation-pipeline-8u4ZjaBxKMwaaUFRG0F8m#:uid=740961193939889955102401&h2=Step-1:-Mask-repeat-elements)).

## 6.1 Verify and remove predictions with in-frame STOP codons

We can use again AGAT to clean that GFF3 file, removing identical isoforms (it finds 4):

agat\_convert\_sp\_gxf2gxf.pl -g OweniaPASA\_Augustus1st.gene\_structures\_post\_PASA\_updates.5127.gff3 -o OweniaPASA.AGAT.gff3

The gffread can be used to identify any potential gene model with in frame STOP codons (-V flag and -H to try to correct them):

gffread -E OweniaPASA.AGAT.gff3 -g Owenia\_softmasked\_v082020.fa -V -H -o Owenia.PASA.simp.noSTOP.gff3

In this case, gffread finds 1 gene model (mikado.Scaffold06G1968) with in frame STOP codon. I remove it by hand and generate the file OweniaPASA.AGAT.noSTOP.gff3.

## 6.2 Remove gene predictions that overlap with repeats

First, we need to convert the repeat masker output file to a BED file, using Ferdi’s [rep2bed.py](https://www.dropbox.com/s/zy6lteysc826nww/rep2bed.py?dl=0) script:

module load python/2.7.15

./rep2bed.py owenia\_unmasked.fa.out > Ofus\_RM.bed

Then, we can use Ferdi’s [filt-rep-gtf.py](https://www.dropbox.com/s/qdj243d391wozcl/filt-rep-gtf.py?dl=0) script, which requires the dependency pybedtools (easily installed with conda) to remove those gene models that overlap with repeats. Ferdi’s script only works with gtf file formats, thus first thing is to convert the gff3 to gtf (e.g. with gffread)

module load python/2.7.15

module load bedtools

pip install pybedtools --user #NOTE! This you only need to run once, on the first time

./filt-rep-gtf.py OweniaPASA\_Augustus1st.AGAT.noSTOP.2.1.gtf Ofus\_RM.bed

## 250183 exons...

## initial number of genes: 35278

## number of genes after filtering 29385

We remove 5,893 gene models that overlapp significantly with masked regions. The script generates two outputs, a “clean” GTF file and a list of retained genes. You can use AGAT to obtain basic statistics of the final output:

agat\_sq\_stat\_basic.pl -i OweniaPASA.AGAT.noSTOP.filt.AGAT.gff3 -g Owenia\_softmasked\_v082020.fa

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Number** | **Size mean (kb)** | **% genome** |
| **Gene** | **29,385** | 12,009.98 | 70.56 |
| **Transcript** | **34,353** | 13,547.80 | - |
| **5’ UTR** | 39,026 | 137.88 | 1.08 |
| **Exon** | 257,455 | 278.37 | 14.33 |
| **CDS** | 242,753 | 223.50 | 10.85 |
| **3’ UTR** | 32,941 | 365.24 | 2.41 |

## 6.3 Remove gene predictions that hit TEs

Finally, we could have predicted TEs expressed in our RNA-seq datasets. We can implement a similar strategy than the one followed to de-contaminate *Capitella*’s proteome in [Step 1](https://paper.dropbox.com/doc/Genome-annotation-pipeline-8u4ZjaBxKMwaaUFRG0F8m#:uid=284257211074393602529197&h2=1.2-Filtering-bona-fide-genes-). First, we extract the protein coding genes and then we BLAST them against a library of TE and repetitive elements. Finally, we remove those gene models that give hit against TEs.

gffread -E OweniaPASA.AGAT.noSTOP.filt.gtf -S -g Owenia\_softmasked\_v082020.fa -y OweniaPASA.AGAT.noSTOP.filt.prot.fasta

cp /data/SBCS-MartinDuranLab/02-Chema/00-OweniaGenome/02-FilteringRB/RepeatPeps.dmnd ./

conda activate diamond

diamond makedb --in RepeatPeps.lib -d RepeatPeps

diamond blastp -d RepeatPeps -q OweniaPASA.AGAT.noSTOP.filt.prot.fasta -o Owenia.VS.TEs.1e5.blastp -f 6 qseqid bitscore evalue stitle -k 25 -e 1e-5 -p 8

## 2450 queries aligned.

We identify 2,450 gene models as potential TEs. Most of them have very low e-values, meaning that are true TEs. Those with higher e-values are small predictions. I manually checked a couple of them, and they give hit against TEs. To remove them, we first get the gene ID and then we fgrep them out (with the -v flag):

cat Owenia.VS.TEs.1e5.blastp | cut -f 1 | sort | uniq > TEsIDs

fgrep -w -v -f TEsIDs OweniaPASA.AGAT.noSTOP.filt.gtf > OweniaPASA.AGAT.noSTOP.filt.noTEs.gtf

I re-run the analysis to make sure that all TEs are gone:

gffread -E Owenia.PASA.simp.noSTOP.noRepeat.noTE.gtf -S -g Owenia\_softmasked\_v072019.fa -y Owenia.PASA.simp.noSTOP.noRepeat.noTe.prot.fasta

diamond blastp -d RepeatPeps -q Owenia.PASA.simp.noSTOP.noRepeat.noTe.prot.fasta -o OweniaPASAnoSTOPnoRepeatnoTE.vs.RepeatPeps.1e5.blastp -f 6 qseqid bitscore evalue stitle -k 25 -e 1e-5 -p 8

## 0 queries aligned.

Because the GTF format (in version 2) does not retain UTR information etc, we can add that information with AGAT, transforming it to GFF3:

agat\_convert\_sp\_gxf2gxf.pl -g OweniaPASA.AGAT.noSTOP.filt.gtf -o OweniaPASA.AGAT.noSTOP.filt.AGAT.gff3

You can use AGAT to obtain basic statistics of the final output:

agat\_sq\_stat\_basic.pl -i OweniaPASA.AGAT.noSTOP.filt.noTE.AGAT.gff3 -g Owenia\_softmasked\_v082020.fa

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Number** | **Size mean (kb)** | **% genome** |
| **Gene** | **26,966** | 12,763.72 | 68.82 |
| **Transcript** | **31,903** | 14,305.44 | 91.25 |
| **5’ UTR** | 37,141 | 134.57 | 1.00 |
| **Exon** | 252,326 | 262.47 | 13.24 |
| **CDS** | 237,854 | 209.32 | 9.95 |
| **3’ UTR** | 31,076 | 368.17 | 2.29 |

## 6.4 Rename annotation file, gene predictions and produce a Non-Redundant version

At this point, safe the original PASA file (OweniaPASA\_Augustus1st.gene\_structures\_post\_PASA\_updates.5127.gff3) and the filtered annotation () to the lab folder.

For the sake of simplicity with downstream analyses, we can rename gene names. We can do that with AGAT, adopting an Ensembl like nomenclature:

agat\_sp\_manage\_IDs.pl -f OweniaPASA\_Augustus1st.AGAT.noSTOP.noRepeats.gff3 --ensembl --prefix OFUS --type\_dependent --tair -o Owenia\_annotation\_v250920.gff3

This command generates a final output named Owenia\_annotation\_v250920.gff3 that will be used in all downstream analyses. We can additionally generate an annotation file without isoforms (picking just the longest one), with AGAT, which will be used in gene family evolution analyses (see [+Gene family evolution: Step-2:-Generate-Non-Redundant](https://paper.dropbox.com/doc/GQbPbnCUkEqDjaEBCJGx9#:uid=323367542266560986210377&h2=Step-2:-Generate-Non-Redundant))

agat\_convert\_sp\_gxf2gxf.pl --gff Owenia\_annotation\_v250920.gff3 --merge\_loci -o Owenia\_lociMerged.gff

agat\_sp\_keep\_longest\_isoform.pl --gff Owenia\_lociMerged.gff -o Owenia\_lociMerged\_longestIsoform.gff

The Ensembl nomenclature has potential for 11 digits numbers, so I just adjust that with a text editor so that it is 5-digit based (i.e. OFUSG00001 and OFUSG13589). The transcript/isoform is indicated after the dot, i.e. OFUSG00001.1 and OFUSG00001.2 are isoforms 1 and 2 of the gene OFUSG00001 respectively.

# Step 7: Validating the annotation

## 7.1 Run BUSCO

We can use BUSCO to assess that the final annotated gene set has similar completeness than the entire genome (see [Step 0](https://paper.dropbox.com/doc/Genome-annotation-pipeline-8u4ZjaBxKMwaaUFRG0F8m#:uid=639185192046548778440966&h2=Step-0:-Run-BUSCO)).

gffread -E Owenia\_annotation\_v250920.gff3 -g Owenia\_softmasked\_v082020.fa -y Owenia\_annotation\_v250920\_filteredPeps.fasta

#!/bin/bash

#$ -pe smp 20

#$ -l highmem

#$ -l h\_vmem=1G

#$ -l h\_rt=0:30:0

#$ -cwd

#$ -j y

module load anaconda3

conda activate BUSCO

busco -m proteins -c 20 -i Owenia\_annotation\_v250920\_filteredPeps.fasta -o Owenia\_final -l metazoa\_odb10

conda deactivate

The results are very comparable to what was reported for the BUSCO genome (97.8% vs 99%)

        --------------------------------------------------

        |Results from dataset metazoa\_odb10               |

        --------------------------------------------------

        |C:96.2%[S:86.9%,D:9.3%],F:1.5%,M:2.3%,n:954      |

        |918    Complete BUSCOs (C)                       |

        |829    Complete and single-copy BUSCOs (S)       |

        |89     Complete and duplicated BUSCOs (D)        |

        |14     Fragmented BUSCOs (F)                     |

        |22     Missing BUSCOs (M)                        |

        |954    Total BUSCO groups searched               |

        --------------------------------------------------

# Step 8: Generate a functionally annotated database

The final step of the annotation is to functionally annotate the gene set, i.e. assign gene orthology, PFAM domains, transmembrane regions, GO terms, etc to each gene model. To do so, we use [Trinotate](https://github.com/Trinotate/Trinotate.github.io/wiki).

## 8.1 Set up Trinotate

To set up Trinotate, you need to have two Perl5 modules installed locally DBI and DBD::SQLite. To install a perl module locally, you need to add a PREFIX flag when running Makefile.PL:

module load perl

cd <DIRECTORY WHERE THE PERL MODULE IS>

perl Makefile.PL PREFIX=~/lib/perl5

make

make test

make install

Then, add to your .bash\_profile the following line:

export PERL5LIB=~/lib/perl5/lib/site\_perl

Follow guidelines in the [Trinotate wiki](https://github.com/Trinotate/Trinotate.github.io/wiki/Software-installation-and-data-required). Trinotate still uses some old versions (e.g. of signalP) and it is important to employ the proper versions of the different modules. **Note:** I don’t install RNAMMER, as it requires a lot of tinkering. Just install signalP and tmhmm in your user, and use the NCBI BLAST+ and HMMER already installed in Apocrita.

#Transcript ID

>Ofus.G0115.1

#Peptide ID

>Ofus.G0115.1 len:142 Ofus.G0115.1:1-429(+)

#GeneTransMap\* \*\*

Ofus.G0115.1 len:142 Ofus.G0115.1:1-429(+)        Ofus.G0115.1

peptideID<TAB>transcriptID

## 8.2 Generate input files

Trinotate will consider five functional evidences:

* **BLASTp**

#!/bin/bash

#$ -cwd

#$ -j y

#$ -pe smp 8

#$ -l h\_vmem=1G

#$ -l h\_rt=36:0:0

module load blast+

blastp -query Owenia\_filtered\_proteins.fa -db /data/SBCS-MartinDuranLab/02-Chema/src/Trinotate-Trinotate-v3.2.0/TrinotateDBs/uniprot\_sprot.pep -num\_threads 8 -max\_target\_seqs 1 -outfmt 6 -evalue 1e-3 > blastp.outfmt6

* **BLASTx**

#!/bin/bash

#$ -cwd

#$ -j y

#$ -pe smp 8

#$ -l h\_vmem=1G

#$ -l h\_rt=36:0:0

module load blast+

blastx -query Owenia\_filtered\_CDS.fa -db /data/SBCS-MartinDuranLab/02-Chema/src/Trinotate-Trinotate-v3.2.0/TrinotateDBs/uniprot\_sprot.pep -num\_threads 8 -max\_target\_seqs 1 -outfmt 6 -evalue 1e-3 > blastx.outfmt6

* **HMMER**

#!/bin/bash

#$ -cwd

#$ -j y

#$ -l highmem

#$ -pe smp 12

#$ -l h\_vmem=1G

#$ -l h\_rt=36:0:0

module load hmmer/

hmmscan --cpu 12 --domtblout PFAM.out /data/SBCS-MartinDuranLab/00-BlastDBs/Pfam-A.hmm ./00-DATA/Ofus\_peps.fasta > pfam.log

* **SignalP**

#!/bin/bash

#$ -cwd

#$ -j y

#$ -pe smp 1

#$ -l h\_vmem=8G

#$ -l h\_rt=36:0:0

module load perl

signalp -f short -n signalp.out Owenia\_filtered\_proteins.fa

## 8.3 Populate the SQLite database and output the report

/data/SBCS-MartinDuranLab/02-Chema/src/Trinotate-Trinotate-v3.2.0/admin/Build\_Trinotate\_Boilerplate\_SQLite\_db.pl Owenia

/data/SBCS-MartinDuranLab/02-Chema/src/Trinotate-Trinotate-v3.2.1/Trinotate Owenia.sqlite init --gene\_trans\_map 00-DATA/Ofus\_geneTransMap --transcript\_fasta 00-DATA/Ofus\_mRNA.fasta --transdecoder\_pep 00-DATA/Ofus\_peps.fasta

#LOAD annotations, below are examples from Trinotate manual, change accordingly to your species ouput from blast+:

/data/SBCS-MartinDuranLab/02-Chema/src/Trinotate-Trinotate-v3.2.1/Trinotate Owenia.sqlite LOAD\_swissprot\_blastp blastp.outfmt6

/data/SBCS-MartinDuranLab/02-Chema/src/Trinotate-Trinotate-v3.2.1/Trinotate Owenia.sqlite LOAD\_swissprot\_blastx blastx.outfmt6

/data/SBCS-MartinDuranLab/02-Chema/src/Trinotate-Trinotate-v3.2.1/Trinotate Owenia.sqlite LOAD\_signalp signalp.out

/data/SBCS-MartinDuranLab/02-Chema/src/Trinotate-Trinotate-v3.2.1/Trinotate Owenia.sqlite LOAD\_pfam TrinotatePFAM.out

/data/SBCS-MartinDuranLab/02-Chema/src/Trinotate-Trinotate-v3.2.1/Trinotate Owenia.sqlite report --incl\_pep --incl\_trans > Owenia\_annotation\_report.xls

# Step 9: Generate PANTHER annotation

[Panther](http://www.pantherdb.org/about.jsp) is a classification system for proteins that combines the specificity of HMMR searches with a curated GO annotation for hundreds of genomes. It allow to classify genes in 15,702 protein families, divided into 123,989 functionally distinct protein subfamilies, all of them with associated GO terms. In order to run PANTHER and annotate each gene of the genome, one can download the Panther HMM Scoring tool from its [website](http://www.pantherdb.org/downloads/index.jsp) and run it like so (add the -s flag so that it uses HMMscan instead of HMMsearch, otherwise it will never finish). Be sure to download the script itself “pantherScore2.2.pl” and the folder “lib” containing the required perl modules. The directory “lib” has to be in the same folder of the script “pantherScore2.2.pl”. Finally this script requires also the PANTHER15.0 directory downloadable from the same link (which it is needed for the -l option).

#!/bin/bash

#$ -pe smp 30

#$ -l highmem

#$ -l h\_vmem=1G

#$ -l h\_rt=120:00:0

#$ -cwd

#$ -j y

module load perl

module load hmmer/

pantherScore2.2.pl -l /data/SBCS-MartinDuranLab/00-BlastDBs/ascii/PANTHER15.0/ -D B -n -o Owenia\_Panther -i Owenia\_annotation\_v250920\_filteredPeps.fasta -c 30 -V -s

**Note:** pantherScore is really really slow, give it >5 days to run for a genome of ~30,000 proteins!

Once it is done, you can combine Trinotate and Panther in a single output. To do so:

cut -f 1 Owenia\_Panther\_sorted > IDs\_panther

cut -f 2 Owenia\_annotation\_v250920\_report.xls | tail -n +2 > IDs\_all

fgrep -v -f IDs\_panther IDs\_all > IDs\_absentPanther ### There are 4923 genes without Panther annotation

awk '{print $0"\t""NO PTHR""\t""NO HIT"}' IDs\_absentPanther > PANTHER\_nohits

cat Owenia\_Panther\_sorted PANTHER\_nohits | sort -k 1,1 > Owenia\_Panther\_sorted\_allgenes

## use vim to add a header in Owenia\_Panther\_sorted\_allgenes so that it matches Trinotate file

paste Owenia\_annotation\_v250920\_report.xls Owenia\_Panther\_sorted\_allgenes > Owenia\_annotation\_v250920\_TrinoPanther.xls

There are some minor formatting issues (e.g. there were a bunch of duplicated Panther terms that missaligned the correspondence between Trinotate and Panther annotations) that one can correct by hand easily directly on Excel.

# Step 10. Lift-over (if needed)

At some point, one might need to transfer the annotation from one assembly version to another (better) one. One can do that with [Liftoff](https://github.com/agshumate/Liftoff) pipeline like so:

#!/bin/bash

#$ -pe smp 5

#$ -l highmem

#$ -l h\_vmem=1G

#$ -l h\_rt=6:00:0

#$ -cwd

#$ -j y

module load anaconda3

conda activate Liftoff

liftoff -p 5 -g Owenia\_annotation\_v300321.1\_filtered\_noGeneID.gff3 -o Owenia\_chrom\_v300321.gff3 Owenia\_v082020\_hic.fa Owenia\_unmasked\_v082020.fa

conda deactivate