Mitogenomes assembly pipeline (DRAFT):

Flowchart:
https://drive.google.com/file/d/12mxYNxJ9L7IBR3anHJm9JUyOzcglMZUL/view?usp=sharing

PART 1. Assembly.

On cluster, working directory structure:
"MetaInvert batch directory" (b# = batch number) is as:

```
b###/
├──lib_01/lib_01.autotrim.filter1.paired_1.fq.gz
├──lib_01/lib_01.autotrim.filter1.paired_2.fq.gz
├──lib_02/lib_02.autotrim.filter1.paired_1.fq.gz
├──lib_02/lib_02.autotrim.filter1.paired_2.fq.gz
.
.
├──lib_n/lib_n.autotrim.filter1.paired_1.fq.gz
├──lib_n/lib_n.autotrim.filter1.paired_2.fq.gz
└──lib_taxa.tsv
```

Starting the assembly pipeline.
The mitogenome assembly pipeline is now part of our general genome assembly pipeline.

1 - The working directory must be a MetaInvert batch directory as described above. Then execute the custom script:

```
metainvert-pipelines/genome_assembly_pipeline/individual_scripts/Novoplasty -p 24
```

(-p is the number of parallel jobs. The bottleneck here is RAM. Spawning to much may overload it quickly)

2- Once all the Novoplasty jobs are over, run GetOrganelle with the custom script:

```
metainvert-pipelines/genome_assembly_pipeline/individual_scripts/GetOrganelle -j 12 -t 4
```

(-j is the number of parallel jobs, t the number of threads GetOrganelle can use; j * p must not be more than the number of CPUs).

3- Once the jobs are over, the assemblies are gathered for the validation part. Again, the working directory must be the MetaInvert batch directory, then execute the script

```
gather_mitogenomes.sh
```

This will creates a new mitogenomes folder, with Novoplasty contig files (*libid*_Novo.fasta); Get_Organelle contig files (*libid*_GO.fasta) and Get_Organelle assembly graph (*libid*_GO.gfa). Sometimes, an alternate assembly file and graph for GO can be reported (*libid*_GO2.fasta/gfa).

(Optional) For selectings assemblies only from a specific taxa (since our batches are mixing several taxa), go into the new "`mitogenomes/01_Assemblers_output`" folder in the MetaInvert batch directory. Then use **`filter_countgroup.sh` *taxa_name* ../lib_taxa.tsv**.

The corresponding files will be copied in a TMP directory. One will have to also subselect the lib_taxa.tsv with `grep taxa_name ../lib_taxa.tsv > subselected_lib_taxa.tsv` and then move things manually to get the correct directory structure for PART 2.

PART 2. Assemblies validation.

Part 2 involves some validation and checking from the user. It can run on the server, but may be simply more practical to run locally. One needs the content of the `mitogenomes/` folder created by `gather_mitogenomes.sh` from part 1.

Working directory structure:

```
bin/
└──muscle
preprocess_assemblies.py
mitogenomes/ (can be renamed so that several projects can be put in the same dir)
├──01_Assemblers_output
└──lib_taxa.tsv
```

/!\ Currently, the following script does not deal with multiple assemblies reports from Get_Organelle (=xxx__GO.fasta and xxx__GO2.fasta; this is something new from my past experiment). So far it seems like one can rather easily pick the 'right one' by checking the sequence size. If one is really small, it is probably not the good one anyway. The script below will always use the GO.fasta and ignore the GO2.fasta. So user has to manually rename the files if necessary /!

Perform an automatic sanity check by comparing the assemblies resulting from Get_Organelle and Novoplasty (library-wise of course); by running:

```
python preprocess_assemblies.py --dirpath path/to/mitogenomes
```

Inconsistencies and issues will be reported in Preprocessing.log file. Inconsistencies can consist in:

a. ambiguous bases in Novoplasty → accept Get_Organelle better resolution
b. several contigs in one or the other file → manually check the GFA with Bandage, blast the contigs to check if they really belong to the target organism...
c. Missing Get_Organelle file → check Novoplasty, redo assembly, or accept failure…
d. Both output are very different but plausible: keep them both, make a note and check later on annotated sequences. Or blast to check sequences identity.
e. Maybe stuff might happen ??

The script will create the folder 02_Assemblies_Preprocessed/, and report here two class of assemblies:

a. The “*libid__circular/linear*_mtGenome.fasta”. This name means that no inconsistency was detected; therefore no manual validation is really needed here.
b. The 'to_check__xxxx' assemblies. This means that Novoplasty and Get_Organelle reported both overall matching sequences, but with some nucleotide bases differences in those sequences. The differences are reported in Preprocessing.log

The user must select one of those sequence then approve the file by renaming it (I recommend to stick to the following name template: *libid__circular/linear*_mtGenome.fasta)

Copy the validated assemblies fasta in a folder named ***03_Assemblies_Validated.*** Next part is Annotation.

/!\ The preprocessing script also automatically format the sequence name, with important keyword "circular/linear". For any sequence that would be put in the validated set manually, it is very important that the name follow the same format, since the follow up script makes use of this /!
Example:
>**a_23** organism=species assembler=Get_Organelle len=14975 topology=**circular**
"a_23" is the sequence name. It must be unique. If two different sequences of the same library are annotated, it is important to give them a unique name. And the sequence name should terminate with a number because of the script later expects it.

PART 3. Annotation.

Working directory structure:

```
bin/
└──muscle
preprocess_assemblies.py
mitogenomes/ (can be renamed so that several projects can be put in the same dir)
├──01_Assemblers_output
├──02_Assemblies_Preprocessed
├──03_Assemblies_Validated
├──04_Annotations Create this folder manually
└──lib_taxa.tsv
```

Creates the folder 04_Annotations/. Make sure that no gaps or the special character used by Novoplasty remains in nucleotides sequences and concatenate all validated fasta into the "04_Annotations/all.fasta". This can be done with the unix shell one-liner:

```
sed -i 's/[-*]//g' ./03_Assemblies_Validated/*.fasta && cat
./03_Assemblies_Validated/*.fasta > ./04_Annotations/all.fasta
```

1 - Submit all.fasta to Mitos2 webserver http://mitos2.bioinf.uni-leipzig.de/index.py:
Submit twice: using RefSeq63 Metazoan; and using RefSeq89 Metazoan. Do not forget to set the genetic code to 5 = invertebrate mitochondrial."

2- When annotation is complete, download Mitos2 results (raw_data archive) into ***04_Annotations***, extract in Mitos2, rename the new directories to RefSeq63 and RefSeq89 respectively and concatenate all the bed files in "RefSeq63.bed" and "RefSeq89.bed".

sed -i '' -e 's/[-*]//g' /Users/jamilshuvo/Documents/MetaInvert/mitogenomes
/03_Assemblies_Validated/*.fasta && cat /Users/jamilshuvo/Documents/MetaInvert
/mitogenomes/03_Assemblies_Validated/*.fasta > all.fasta

sed 's/>/\n>/g' /Users/jamilshuvo/Documents/MetaInvert/mitogenomes/03_Assemblies_Validated/*.fasta && cat /Users/jamilshuvo/Documents/MetaInvert/mitogenomes/03_Assemblies_Validated/*.fasta > all.fasta

```
cat ./04_Annotations/Mitos2/RefSeq63/**/*.bed >
./04_Annotations/Mitos2/RefSeq63.bed
cat ./04_Annotations/Mitos2/RefSeq89/**/*.bed >
./04_Annotations/Mitos2/RefSeq89.bed
```

3- Submit the all.fasta file to Arwen http://130.235.244.92/ARWEN/ in batch mode. Save the results in `./04_Annotations/Mitos2/RefSeq89/**/*.bed > ./04_Annotations/Arwen/arwen.cgi`

4- Combine the annotations by running

```
python harmonize_annotation.py --cmd combine_annotations --fasta
./04_Annotations/all.fasta
```

The script will create all.bed by merging the beds and the cgi file:
`./04_Annotations/all.bed`

python harmonize_annotations.py --cmd combine_annotations --fasta

04_Annotations is organized as follow now

```
04_Annotations/
├──Arwen/
│   └──arwen.cgi
├──Mitos2/
│   ├──RefSeq63/
│   ├──RefSeq89/
│   ├──RefSeq63.bed
│   └──RefSeq89.bed
└──all.bed
└──all.fasta
```

2. Run `python harmonize_annotations.py --cmd set_origin --fasta ./04_Annotations/all.fasta`.
   The script will re-align all genomes on the end of Mitos2 annotated genes, by order of priority= ['nad4l', 'nad4','cox1']: then will move all.bed and all.fasta into a backup folder, and new “realigned_all.fasta” and “realigned_all.bed” will be created.
   python harmonize_annotations.py --cmd blast_tRNAs --fasta ./04_Annotations/realigned_all.fasta
3. Run `python harmonize_annotations.py --cmd extract_CDS --fasta ./04_Annotations/realigned_all.fasta`
   The script will extract all genes and align them. Alignments can then be used for comparative annotation.

Workbench
The so-called “Workbench” is a mass gene sequence approach I suggest to quickly validate the gene annotation or to notes or correct potential problems and solutions before doing the visual annotation on Geneious. Correctly used, it may save a lot of time.
So far, I found it very useful essentially for PCG. For rRNA and tRNA, not quite really.

Do manual adjustment in **./04_Annotations/workbench** on the *_prefixed.tsv files and save each gene in a separate tsv file “Seq_id sequence”, the tsv file is named Gene_Name.tsv and deposited in folder **./04_Annotations/workbench/CDS_OK.**

Workbench text editors recipes.

1- Distinguish codons. Search and replace “**([A-Z\-]{3})**” by “**\1** “ in regex mode.
2- Light up STOP codons: Search **(TAG|TAA)** in regex mode (keep in mind the “incomplete stop exception”)
3- Light up valid START codons: Search **(ATG|ATA|ATT|ATC|GTG|TTG)** in regex mode. (those are the known start codon in invertebrate mitochondrial genomes, by order of commonness).
4- Collapse gap at the beginning of a sequence..
Replace “**(([A-Z]{3})+)((-{3})+)**” with “**\3\1**”

How to work.
First look for clear homologies. If all or most the sequences are perfectly aligned, it should be easy to detect the homologous sequence pattern at the end and start of a sequence. Sequences with unusual length should be investigated (maybe incomplete stop codon ?). If the initiation of the gene is not perfectly conserved between taxon, choose a rule to select it a note it down. Options are: minimizing transformations or maximizing theoretically valid sequences without creating overlap with previous gene (this need to be checked with Geneious in the next step). Its always worth to have a look at some more distant references see if genes are highly preserved (if yes, then the option 1 may be the best).

4. Update annotation file by running

```
python ../../harmonize_annotations.py --cmd update_annot -
-fasta ./realigned_all.fasta --bed ./realigned_all.bed
--gene_dir ./workbench/aligned/CDS_OK/
```

Import realigned genomes + bed file in Geneious. Check for weird stuff in annotation (strong overlap of genes, inconsistencies with closely related species annotation, missing genes) and do manual adjustment if necessary. Deduplicate tRNA annotated by Mitos2 and Arwen. RefSeq89 is fully re-upload and can act as a cross-check with manual annotations.

```
python3 /Users/jamilshuvo/Documents/Thesis_Project/harmonize_annotations.py --cmd update_annot --fasta /Users/jamilshuvo/Documents/Thesis_Project/mitogenomes
/04_Annotations/realigned_all.fasta --bed /Users/jamilshuvo/Documents/Thesis_Project/mitogenomes/04_Annotations/realigned_all.bed --gene_dir /Users/jamilshuvo/Documents
/Thesis_Project/mitogenomes/04_Annotations/workbench/aligned/CDS_OK
```

My feeling about manual adjustment: accept all tRNA, just deduplicate by keeping the MITOS2 over Arwen. 16S and 12S starting point limit seems to be never clear: be conservative, extend the region up to the next gene. Maybe it will be possible to refine it in a second iteration ? CDS: trust manual over automatic, but if automatic suggested a larger open frame not overlapping with previous gene, then accept it over manual! Minimize overlapping but maximize compaction (less intergenic space).
rrnS - 12SrDNA: 3' extremities seems better conserved. 5' is elongated until expected trnN.
rrnL - 16SrDNA: 5' is elongated until expected trnV. trnV is sometimes not detected but seems to be often there. So check. Stop 3' when meeting **trnP** if present. After the first batch is annotated, check for missing trnP “TT**TGG**G” reading head, and manually annotated

trnP based on similar sequences. Adjust 16S 3' accordingly. If trnP is annotated but not well aligned with others, keep the annotation as it is based on a modelisation we cannot repeat by eye. Never mind if there is a poor delineation in those very modified area.

Rule 1: alignment is checked to try to find homologous start and end point.
Rule 2: minimize overlap with preceding / following gene
Rule 3: maximize ORF if rule 1 seems not satisfactory, or gene too short.

5. Import realigned genomes + bed file in Geneious. Check for weird stuff in annotation (strong overlap of genes, inconsistencies with closely related species annotation, missing genes) and do manual adjustment if necessary. Deduplicate tRNA annotated by Mitos2 and Arwen. RefSeq89 is fully re-upload and can act as a cross-check with manual annotations.
6. Freeze annotation, export and be done with it.

16Srna 3': CCTAAA 12S 5' HTRHRVTTRAMY
tRNA manual annotation. Ambiguous regions were aligned to each other's. When a tRNA was not tagged by an automatic tool but that the region seemed to exists, presence of the codon was checked in the alignement. If found, then we decided to recognize the presence of the homologous tRNA, and annotate it partially.

Alignments:
Best is to limit manual intervention to the maximum. Each genes sequence are checked using a nucleotides and a AA alignement for completness / aberrration.

cox2:
Histiostomatidae__MH921997 seems very divergent ??

atp8
Quadrioppia maritalis: very short observed in both genome..

Cob
Dissorhina_ornata P1_12: added N to match Dissorhina_ornata P4_7
Huge gape opening for Steganacarus and Leptotromb in 5', maby close it with “?”. Not decided.
All stop codon are aligning with seldom gap opening in 3'. Close with “???” maybe ?

Nad4
Paraleius_leontonychus, large missing part in 5'

<u>Quick extraction of the 658 bp barcode locus of the COI:</u>

Copy fasta with validated genomes assembly in 05_Quick_Barcode_Extraction/DB and create a blast database.

```
~/bin/ncbi-blast-2.9.0+/bin/makeblastdb -dbtype nucl -in
./DB/b4_validated_genomes.fasta -title mitogenomes -out mitogenomes
```

Bed fil definition:
Start = position before the first base.
End = position of the last base.
Example:

```
Bed         ↓                       ↓
orf         ↓ |-------- ORF --------|
num    1 - 3 4 - - - - - - - - - - 15
Seq    T A A A T G T T T C T A T A A
Pyn    0 - 2 3 - - - - - - - - - - 14
```

Fetch COI with the extended COI query (6 extra bases in 5' and in 3' to represent primer region).

ORF: Open Reading Frame:
ATG *** *** *** *** *** *** *** TAG
Start Stop codon.

Finding an ORF in a DNA sequence:

TTGCTATGCATGGCTTATCCAAATAGATTCATCATTTAGGTCAAT

Genetic code: 5 - invertebrate mitochondrial.
Possible stop codon: **TAG, TAA, T****
Regex to highlight possible stop codons (**TAG|TAA**)
Start codon:

Known start codon in invertebrate mitochondrial **ATG, ATA, ATT, ATC, GTG, TTG**

Regex to **highlight possible start codons**: (ATG|ATA|ATT|ATC|GTG|TTG)

CDS: Coding DNA sequence

*Code IUPAC
*[A-Z]{3}

Collapse gap at the beginning of a sequence..
(([A-Z]{3})+)((-{3})+)

Heavy strand: more A+G
Light strand: less A+G

Notes:
P4_10: large intergenic region between mrRNA-S and nad1: seems to include a fragment of CYTB/cob