**To make Cerberus more modular**

From any NextGen sequencing technology (from Illumina, Pacbio, Oxford Nanopore)

type 1 raw reads (.fastq format)

type 2 nucleotide fasta (.fasta, .fa, .fna, .ffn format), assembled raw reads into contigs

type 3 protein fasta (.faa format), assembled contigs which genes are converted to amino acid sequence

0. setup.py

looks okay. I don’t like the folder on the desktop.

**1. cerberus-master.py (control center of cerberus)**

- calls all the individual scripts etc

Refactor cerberus to call our functions

get\_args.py (put in here)

def args() ()

Add the config file user uses here.

**2. cerberus-setup.py**

- This will replace config file, it will install the program dependances (pulling directly from source) - install in path, and downloading/formatting the databases. Then will work with pip and conda and source install

**3. QC.py (Fastqc script from Rhea)**

→ qc raw fastq

→ qc trim fastq

**5. Trim.py (fastp and porecrop from Rhea)**

→ https://github.com/rrwick/Filtlong

**6. Decon.py (bbduk from Rhea)**

→ We need to do a map based for nanopore and pacbio

→ <https://github.com/wdecoster/nanolyse> (maybe later?)

**5. format.py (convert fq to fna)**

convert fq to fna

def fastq\_processing(fq\_path, path, f\_name):

trim\_path=path+'/'+f\_name+"\_trim.fastq"

trim\_fna=path+'/'+f\_name+"\_trim.fna"

cmd1 = "sed -n '1~4s/^@/>/p;2~4p' "+trim\_path+" > "+trim\_fna

subprocess.call(cmd1,shell=True)

return trim\_fna

**REMOVE N’S here (LIKELY an easier way then this)**

**def Ndeletion(input\_file,output\_file):**

# input\_file='./one.fna'

# output\_file=open('./out.txt','w')

counter=1

mark=0

past=1

with open(input\_file) as lines:

temp=''

for line in lines:

if line[0]=='>':

print('>contig' + str(counter),file=output\_file)

counter+=1

continue

for char in line:

if char not in ['N',' ','\n']:

past=0

temp+=char

mark=0

elif char=='N':

mark=1

if mark==1 and past==0:

past=1

mark=0

print(temp)

print('>contig' + str(counter),file=output\_file)

if len(temp)<=51:

print(temp,file=output\_file)

temp=''

counter+=1

if len(temp)>51:

print(temp[:51],file=output\_file)

temp=temp[51:]

mark=0

output\_file.close()

**6. GeneCall.py (RheaGeneCall.py from Rhea)**

→ prodigal

→ prokka?

→ FGS (FGS++)

**7. hmmer search (uniquely cerberus)**

def faa\_processing(faa\_path, path, f\_name):

output\_path=path+os.sep+f\_name+"\_output"

os.makedirs(output\_path)

output\_path=os.path.join(output\_path + os.sep, f\_name)

script\_dir = os.path.dirname(os.path.realpath(\_\_file\_\_))

hmm\_file = os.path.join(script\_dir, "osf\_Files/FOAM-hmm\_rel1a.hmm.gz")

nCPU = int(os.cpu\_count()/2)

hmm\_cmd = "hmmsearch --cpu %s --domtblout %s.FOAM.out %s %s" %(nCPU, output\_path, hmm\_file, faa\_path)

subprocess.run(hmm\_cmd, shell=True, stdout=open("process\_faa.out", 'w'))

BH\_dict = {}

BS\_dict = {}

minscore = 25

reader = open(output\_path + ".FOAM.out", "r").readlines()

for line in reader:

if line[0] == "#": continue

line = line.split()

score = float(line[13])

if score < minscore: continue

query = line[0]

try:

best\_score = BS\_dict[query]

except KeyError:

BS\_dict[query] = score

BH\_dict[query] = line

continue

if score > best\_score:

BS\_dict[query] = score

BH\_dict[query] = line

KO\_ID\_dict = {}

for BH in BH\_dict:

line = BH\_dict[BH]

KO\_IDs = [KO\_ID.split(":")[1].split("\_")[0] for KO\_ID in line[3].split(",") if "KO:" in KO\_ID]

for KO\_ID in KO\_IDs:

try:

KO\_ID\_dict[KO\_ID] += 1

except KeyError:

KO\_ID\_dict[KO\_ID] = 1

rollup\_file = "%s.FOAM.out.sort.BH.KO.rollup" %(output\_path)

return roll\_up(KO\_ID\_dict, rollup\_file)

**9. Parser.py**

sort XXX.faa.FOAM.out > XXX.faa.FOAM.out.sort **(has to be sorted)**

python bmn-HMMerBestHit\_p3.py XXX.faa.FOAM.out.sort > XXX.faa.FOAM.out.sort.BH

**(has to be sorted to get the top hit)**

awk '{print $4}' XXX.faa.FOAM.out.sort.BH > XXX.faa.FOAM.out.sort.BH.tmp1

**(formats output)**

python bmn-CountEachElement\_p3.py XXX.faa.FOAM.out.sort.BH.tmp1 > XXX.faa.FOAM.out.sort.BH.tmp2

**(counts elements)**

python bmn-KOoneCount\_p3.py XXX.faa.FOAM.out.sort.BH.tmp2 | sed s/KO://g | sort -k 1 > XXX.faa.FOAM.out.sort.BH.KO

**(counts KO elements)**

**ROLL-up of the .ko and .foam function**

def roll\_up(KO\_ID\_dict, rollup\_file):

script\_dir = os.path.dirname(os.path.realpath(\_\_file\_\_))

FOAM\_file = os.path.join(script\_dir, "osf\_Files/FOAM-onto\_rel1.tsv")

FOAM\_dict = {}

reader = csv.reader(open(FOAM\_file, "r"), delimiter="\t")

header = next(reader)

for line in reader:

KO\_ID = line[4]

FOAM\_info = line[0:4]

FOAM\_dict[KO\_ID] = FOAM\_info

KEGG\_file = os.path.join(script\_dir, "osf\_Files/KO\_classification.txt")

KEGG\_dict = {}

reader = csv.reader(open(KEGG\_file, "r"), delimiter="\t")

for line in reader:

if line[0] != "":

tier\_1 = line[0]

continue

if line[1] != "":

tier\_2 = line[1]

continue

if line[2] != "":

pathway = line[3]

continue

KO\_ID = line[3]

KEGG\_info = [tier\_1, tier\_2, pathway] + line[4:]

KEGG\_dict[KO\_ID] = KEGG\_info

KO\_ID\_list = [key for key in KO\_ID\_dict]

KO\_ID\_list.sort()

outfile = open(rollup\_file, "w")

for KO\_ID in KO\_ID\_list:

try:

FOAM\_info = FOAM\_dict[KO\_ID]

except KeyError:

FOAM\_info = ["NA"]

try:

KEGG\_info = KEGG\_dict[KO\_ID]

except KeyError:

KEGG\_info = ["NA"]

outline = "\t".join([str(s) for s in [KO\_ID, KO\_ID\_dict[KO\_ID], FOAM\_info, KEGG\_info]])

outfile.write(outline + "\n")

return rollup\_file

**UNSURE what** **preprocess\_before\_visual.py**

**Formating the rollup prior to plotting?**

**10. visualization.py (merge single and multiple file visual.py)**

**Clean up PCA.py**

***Time\_g.py and mem\_usage.py are for time and memory usage. Should be here. But, we can use them for testing.***