20230824\_Metagenomics data processing using the CHOICE dataset

2023-08-24

* Hello! This doc will go over the code I used to process the shotgun metagenomics data from the CHOICE study.
* I hesitate to call it a guide since I will be doing no such thing here; rather than guiding, the point is to give direction for tools that can be used, general workflow parameters, and anything else I think of along the way.
  + Wikipedia has a great overview of the pipeline and popular tools used to process this sort of data, so check it out: <https://en.wikipedia.org/wiki/Metagenomics>
* When doing this, I ran into many issues that I *solved*, but were likely solved *badly*. I will talk about these issues when we get to them, just know that my solutions are probably naive and that there are almost certainly better ways to resolve these issues.
  + Example: The OSCER servers in Norman have a fairly large scratch directory. However, it is not large enough to hold all of the sequence files, their outputs, and the intermediate files at the same time (I got scolded by their maintenance team). Thus, my “fix” was to only do 4-5 samples at a time (in this case, a “sample” refers to all the sequences associated with a signle participant) and pull out specific files as “checkpoint” files–we will see that I designated these checkpoints poorly, and that there are more automated ways to do these steps.
  + There are other instances of inefficiencies, which I will highlight by surrounding the offending code with a long string of #’s
* More info on the pipeline I use can be found here (they also provide a list of programs you’ll need to have installed or otherwise in your programming environment): <https://github.com/thepanlab/MetagenomicsTree>
  + Almost all of the content presented here comes from that link, though I will try to add any insight, advice, or other knowledge I can think of along the way
  + Dr. Chongle Pan’s lab in Norman helped get me set up–specifically, Li Zhang and Dongyu Wang assembled these pipelines and answered my queries. I’d leave their contact info here, but academia is (sometimes) a revolving door and thus they may not be around to answer future questions. Contact me instead, I guess ([kameron.sugino@gmail.com](mailto:kameron.sugino@gmail.com))? Hopefully by the time someone does I’ll have a firmer grasp on
* Note that I will not be going over how to submit a job here on the servers (which is a whole other thing I’m not sure comfy with). That info can be found in another workshop doc: “20230816\_Workshop\_code\_QIIME\_job”
* The data: we have 192 different participant-timepoint sequences (up to 2 samples per mom, and 3 per infant)
  + Each of the samples was sequenced in triplicate, with each file taking up ~12Gb while compressed. In total, the compressed sequence data is ~1.2Tb–makes sense why OSCER folks weren’t happy with me hen I was taking up most of their scratch space.
  + The data itself is already demultiplexed (i.e., separated the sequences per sample, rather than have them all in one big doc), so I also won’t go over how to do that.
  + I’ve not had to demultiplex for this much data, but here’s a short primer on what it is and sorta how to do it: <https://bioinformatics.cvr.ac.uk/how-to-demultiplex-illumina-data-and-generate-fastq-files-using-bcl2fastq/>
* So, here’s a quick overview of what we’ll be doing here:
  + QC of the raw sequences
  + Taxonomic ID of the cleaned reads
  + Assembly of reads, mapping rate, and count of mapped reads
  + Gene prediction and annotation
  + MAG assembly (?)
* Another quick note before we get started. All the proceeding code will be written to run so that it loops over every file (or sets of files) before moving on to the next one.
  + There is almost certainly a better way to do it, but a while loop or other methods might be better/faster/more appropriate, but this is how I did it (see line comments for why it’s structured like this).
* So, here’s how to trim the sequence adapters:
  + Besides trim adaptors, optionally, reads with Ns can be disregarded by adding “maxns=0” ; and reads with really low average quality can be disregarded with “maq=8”
  + Note that the option “ref=/opt/oscer/software/BBMap/38.36-intel-2016a/resources/adapters.fa” points the program to a database it then uses to identify (then remove) adapter sequences; this is the location that the OSCER folks had for the install, so this (and other options downstream) will need to be changed to reflect the location of your files

for n in CHO\* #find all files/folders starting with "CHO..."  
do  
 cd ./$n #go into the CHO folder (which contains the paired reads)  
for i in \*1.fq.gz #find R1  
do  
for j in \*2.fq.gz #find R2  
do  
 #trim adapters on R1 and R2 (note the variable notation used here in $i and $j; $n pulls the name found for CHO and writes the output to a new file with the same naming convention)  
 bbduk.sh in1=$i in2=$j out=$n"\_output.fastq" ref=/opt/oscer/software/BBMap/38.36-intel-2016a/resources/adapters.fa ktrim=r k=23 mink=11 hdist=1 tpe tbo ftm=5 qtrim=rl trimq=10  
done  
done  
 cd .. #return to the original folder to begin the search again  
done

* Removal of artifacts and spike-in calibrators
  + Some of this code is written weird, so I’ll try to comment on where I think inefficiencies are
  + This will remove all reads that have a 31-mer match to PhiX, allowing one mismatch. The “outm” stream will catch reads that matched a reference kmers; “stats” will produce a report of which contaminant sequences were seen, and how many reads had them.
  + Note again the reference databases queried here

for i in CHO\* #looks like I put the output file from the trimming step into the CHO\* folder, so we enter each folder again to find the output file  
do  
cd ./$i  
for n in \*output.fastq #here's the file we want to process, note the $n notation below is a stand-in name for the \*output.fastq file found here  
do  
bbduk.sh in=$n out=$n"\_u.fastq" outm=$n"\_m.fastq" ref=/opt/oscer/software/BBMap/38.36-intel-2016a/resources/sequencing\_artifacts.fa.gz ref=/opt/oscer/software/BBMap/38.36-intel-2016a/resources/phix174\_ill.ref.fa.gz k=31 hdist=1 stats=$n"\_stats.txt"  
done  
cd ..  
done

* Error correction
  + Low-depth reads can be discarded here with the “tossdepth”, or “tossuncorrectable” flags. For very large datasets, “prefilter=1” or “prefilter=2” can be added to conserve memory

for i in CHO\*   
do  
cd ./$i  
for n in \*u.fastq  
do  
tadpole.sh in=$n out=$n"\_ec.fastq" ecc=t passes=1 prefilter  
done  
cd ..  
done

* Since the data were sequenced in triplicate, we will have three separate error corrected files. Here, we concatenate the files together for downstream processing, but I suppose you could process them separately and concatenate later if memory is an issue
  + I have no idea if this would work, or when you would have to concatenate, but it may be necessary in some cases

####reminder: the #’s designate bad code. Here, that means the names need to be manually replaced rather than automatically found and collected by the program \* So, I don’t have generalized code for this (sorry), but here’s what I did for the 3 infant timepoints:

cat /scratch/suginoka/CHO57\_inf/CHO57B2Week\_1/CHO57B2Week\_1\_output.fastq\_u.fastq\_ec.fastq /scratch/suginoka/CHO57\_inf/CHO57B2Week\_2/CHO57B2Week\_2\_output.fastq\_u.fastq\_ec.fastq /scratch/suginoka/CHO57\_inf/CHO57B2Week\_3/CHO57B2Week\_3\_output.fastq\_u.fastq\_ec.fastq > CHO57B2Week.merged.ec.fastq  
  
cat /scratch/suginoka/CHO57\_inf/CHO57B2Month\_1/CHO57B2Month\_1\_output.fastq\_u.fastq\_ec.fastq /scratch/suginoka/CHO57\_inf/CHO57B2Month\_2/CHO57B2Month\_2\_output.fastq\_u.fastq\_ec.fastq /scratch/suginoka/CHO57\_inf/CHO57B2Month\_3/CHO57B2Month\_3\_output.fastq\_u.fastq\_ec.fastq > CHO57B2Month.merged.ec.fastq  
  
cat /scratch/suginoka/CHO57\_inf/CHO57B4to5Month\_1/CHO57B4to5Month\_1\_output.fastq\_u.fastq\_ec.fastq /scratch/suginoka/CHO57\_inf/CHO57B4to5Month\_2/CHO57B4to5Month\_2\_output.fastq\_u.fastq\_ec.fastq /scratch/suginoka/CHO57\_inf/CHO57B4to5Month\_3/CHO57B4to5Month\_3\_output.fastq\_u.fastq\_ec.fastq > CHO57B4to5Month.merged.ec.fastq

* I’m sure this is easily generalizable, but I didn’t figure it out then, and I don’t really intend to now. However, you could match an ID pattern in the filename or make a .txt file with the names/locations of the files to be concatenated and run it in a loop. You’d need to double check that all the files were found and merged, which I think you can do by checking the file sizes. If you need to do this (or choose to?) then best of luck!

* So, now that our sequences are cleaned, we can get taxonomy IDs
* Here, we will be using Metaphlan2, which was on the OSCER servers. Note two things (maybe more):
  + There are many other programs that can return the taxonomic IDs (see the wiki page); what you use is up to you and your needs. Unfortunately, I am not well-versed enough in bioinformatics to know all the ins and outs of these programs. BUT the best way to learn this info is to read the official documentation/manuals, go through tutorials written by the community, and read papers and see how they used the program (which is nice because they published with the method, but the methods aren’t always written with enough detail)
  + There are varying inputs for the annotation programs; some use the cleaned sequences with no further processing, others use assembled contigs or scaffolds or other gene markers. Here, we use the cleaned sequences in Metaphlan2 (Metaphlan4 is the newest release and can be used as well–just need to update the databases and such to work).

module load MetaPhlAn2/2.7.8-foss-2018b-Python-3.6.6 #load in the program  
  
#below we specify "--nproc 10", which is the number of processors available on the servers (which you need to designate); change this number to reflect how many processors you have access to or want to use (see the QIIME doc for a little more info)  
for n in \*ec.fastq #find the concatenated, error corrected sequences  
do  
metaphlan2.py $n --bowtie2out $n"\_metagenome.bowtie2.bz2" --nproc 10 --input\_type fastq  
done  
  
  
#this is an intermediate file that is meant to be saved as a checkpoint; the above code chunk takes a while to compute, while this one is almost instant  
for n in \*.bz2  
do  
metaphlan2.py $n --input\_type bowtie2out -o $n"\_rel\_abundance\_table.txt" -t rel\_ab\_w\_read\_stats  
done  
  
#This is a weird custom program in metaphlan that take the individual abundance tables and merges them by row  
module load python  
python /opt/oscer/software/MetaPhlAn2/2.7.8-foss-2018b-Python-3.6.6/utils/merge\_metaphlan\_tables.py \*rel\_abundance\_table.txt > merged\_abundance\_table.txt

* The file “merged\_abundance\_table.txt” is your OTU table
* Before we get to the assembly, we need to re-pair the data (as in organize our paired reads)

#See cat function below

module purge #I needed to include this to remove all other modules loaded. I think I was running into version incompatability issues with various python packages, but removing/reloading modules seemed to take care of that  
  
#we concatenate all of one participant's samples into one file to map their contigs (or scaffolds) back onto their own assembly; the idea is that some assemblies will be better than others, so including all of one persons data (with all the replicates and longitudinal collection) will return a better assembly than if we used less data  
cat CHO57B2Week.merged.ec.fastq CHO57B2Month.merged.ec.fastq CHO57B4to5Month.merged.ec.fastq > CHO57\_inf.merged  
  
module load BBMap  
for n in \*.merged  
do  
repair.sh in=$n out=$n"\_cat\_output.fastq\_u.fastq\_ec.fastq.fixed" outs=singletons.fq repair  
done

* Assembly of the data is tricky. There are many options, but we will loosely go over 2:
  + Megahit
    - Lower quality contig assembly, but much faster (in my tests it was ~12x faster, or 1hr run time vs metaspades’ 12hr run time *per sample*)
  + MetaSpades
    - Higher quality scaffold assembly (scaffolds are large gene regions consiting of contigs+gaps), much slower

module load MEGAHIT  
for n in \*.fixed  
do  
#--12 tells Megahit that the fastq file is interleaved (i.e., paired as so: R1-1, R2-1, R1-2, R2-2, etc. )  
megahit --12 $n -o $n"\_assembly.fastq"  
done

* Here, I just used the default options for Megahit. You can go into their documentation and really specify what you want the program to do, but be *very* sure that what comes out is good and expected.
* And metaspades
  + Note that the default memory is 250G, but you can designate -m 500 if you need more; you can designate even more than that, which can help even if your job has less memory (i.e., in cases where metapspades complains about memory issues, you can sometimes designate *more* memory than you have access to and, occasionally, the program will use *less* memory than it was previously complaining about).

metaspades --12 output\_tecc.fastq -o assembly\_data.fasta

* Let’s check the N50 and N90. For more info, check the wiki or google (<https://en.wikipedia.org/wiki/N50,_L50,_and_related_statistics>)
  + Briefly, N50 is the minimum contig length required to cover 50 percent of the assembled genome sequence, while N90 is the minimum contig length to cover 90 percent of the genome. It gives you an idea of how long your contigs/scaffolds are and the number of long vs short scaffolds. So, you can kind of think of the N50 as the mean or length of your assembled reads. I do not know what a good N50 or N90 are and I haven’t really been able to find an answer.

for n in \*assembly.fastq  
do  
/home/suginoka/Metagenomics\_Programs/bbmap/stats.sh in=$n/final.contigs.fa out=$n"\_assembly\_data\_stats"  
done

* Let’s calculate the mapping rate of the contigs (I’m just gonna call them contigs even though these processes will also work with scaffolds from metaspade)

#see last step; it directly calls a folder and file within the folder, so that needs to be changed in order for this chunk to work correctly; I think the reason I had it hard coded is because I’m asking the loop to find .ec files, and it also needs to find another companion file, which I did not known how to do at the time, and don’t really want to figure out right now–I’ll try to get to it, but no promises (remember, you can always contact me if these issues arise and need to be fixed–or to just yell at me, if needed).

#We use pullseq and bowtie2 to scaffold, align and map the number of aligned reads  
#Here, we're selecting scaffolds that are >=1000 bp   
#Note: Need to change "$n/final.contigs.fa" if metaspades was used. I think the file you want is called "contigs.fasta", though this may need to be changed to the "scaffold.fasta" file if you expect to have scaffolds--if your input are paired-end reads and used Megahit, you should have almost identical scaffolds and contigs.  
  
for n in \*assembly.fastq  
do  
/home/suginoka/Metagenomics\_Programs/pullseq/src/pullseq -i $n/final.contigs.fa -m 1000 > $n"\_sequence\_min1000.fastq"  
done   
  
##Create bowtie2 index  
##NOTE: May need to go into bowtie2-build file and change the first line from calling python3 to calling python  
##Note that I have a personal download of bowtie2, so the above advice may not always be needed, just if you're running into issues  
module load python  
for n in \*min1000.fastq  
do  
mkdir $n".min"  
/home/suginoka/Metagenomics\_Programs/bowtie2-2.4.1-linux-x86\_64/bowtie2-build $n $n".min"/sequence\_min1000  
done  
  
##Reformat paired reads; i.e., split out R1 and R2  
for n in \*ec.fastq  
do  
/home/suginoka/Metagenomics\_Programs/bbmap/reformat.sh in=$n out1=$n"\_read1.fastq" out2=$n"\_read2.fastq"  
done  
  
#Performs the alignment of R1 and R2 onto the assembly made by megahit  
# You can obtain the number of aligned reads in the output file and the number of total reads can be obtained in the log file of the assembly  
# Mapping rate = the number of aligned reads divided by the number of total reads from the RPKM Calculation (shrinksam)  
for n in \*ec.fastq  
do  
/home/suginoka/Metagenomics\_Programs/bowtie2-2.4.1-linux-x86\_64/bowtie2 -x CHO57\_infM.merged\_cat\_output.fastq\_u.fastq\_ec.fastq.fixed\_assembly.fastq\_sequence\_min1000.fastq.min/sequence\_min1000 -1 $n"\_read1.fastq" -2 $n"\_read2.fastq" -S $n"\_alignment.sam" -p 19  
done

* Note that the “\*\_alignment.sam” file is also the file used for genome binning (MAGs–I’ll try to go over this later, but I don’t really know what I’m doing)
* Moving on! Let’s count the mapped reads
  + We use shrinksam to remove unmapped reads from bowtie2-created SAM files (i.e., the error corrected reads that did not map to the megahit/metaspades assembly), which will generate a SAM file with only the reads that mapped to an assembled contigs.

##Count mapped reads  
for n in \*alignment.sam  
do  
/home/suginoka/Metagenomics\_Programs/shrinksam/shrinksam -i $n > $n"\_mapped.sam"  
done  
  
# Then we count the number of reads mapped to each contig through `add\_read\_count.rb`  
# This step and the next go through each file and filter the lines containing scaffold name in the output fasta files  
module load Ruby  
for n in \*mapped.sam  
do  
/home/suginoka/Metagenomics\_Programs/misc\_scripts/add\_read\_count/add\_read\_count.rb $n CHO57\_infM.merged\_cat\_output.fastq\_u.fastq\_ec.fastq.fixed\_assembly.fastq\_sequence\_min1000.fastq > $n"\_mapped.fasta.counted"  
grep -e ">" $n"\_mapped.fasta.counted" > $n"\_mapped.counted.result"  
done  
  
##Collect total number of mapped reads  
touch mapped\_reads\_collected.txt  
for n in \*mapped.counted.result  
do  
echo $n | sed 's/.fastq\_output.fastq\_mapped.counted.result//g' | cat >> mapped\_reads\_collected.txt  
sed 's/.\*\_count\_//g' $n | paste -sd+ - | bc >> mapped\_reads\_collected.txt  
done  
  
for n in \*ec.fastq  
do  
/home/suginoka/Metagenomics\_Programs/bbmap/bbduk.sh in=$n stats=$n"\_stats.txt"  
done  
  
##Collect total number of reads  
touch total\_reads.txt  
for n in \*\_stats.txt  
do  
grep -e "Total" $n | cat >> total\_reads.txt  
done  
touch total\_reads\_id.txt  
for n in \*\_stats.txt  
do  
grep -e "File" $n | cat >> total\_reads\_id.txt  
done  
paste total\_reads.txt total\_reads\_id.txt >> total\_reads\_collected\_raw.txt  
sed 's/#Total\|#File//g' total\_reads\_collected\_raw.txt > total\_reads\_collected.txt  
  
#RPKM Calcs  
module load python  
sed 's/#File//g' total\_reads\_id.txt > total\_reads\_names  
while read line  
do  
python /home/suginoka/Metagenomics\_Programs/parse\_scaffolds\_rpkm.py ${line}\_alignment.sam\_mapped.sam\_mapped.counted.result total\_reads\_collected.txt $line  
done < total\_reads\_names

* I honestly don’t really know what’s going on in the above code chunk (a bunch of plain text editing and math), so shout out to Dongyu and Li for helping me through this!
* Let’s get to the gene predictions.
* We use Prodigal to annotate the DNA to predicted protein coding regions (open reading frames–ORFs).

for n in \*min1000.fastq  
do  
/home/suginoka/Metagenomics\_Programs/Prodigal/prodigal -a $n"\_trans\_protein.fasta" -i $n -p meta -o $n"\_predicted.gdk"  
done

* There are two output files:
  + trans\_protein.fasta (protein translations file)
  + predicted.gff (inforamtion for each CDS, we will use the CDS length)
* Let’s annotate the predicted ORFs

ln -s /home/suginoka/Metagenomics\_Programs/kofam\_scan/ruby/bin/ruby  
  
module load Ruby  
module load GNU/4.9.3-2.25  
export OMP\_NUM\_THREADS=10  
for n in \*protein.fasta  
do  
/home/suginoka/Metagenomics\_Programs/kofam\_scan/exec\_annotation -o $n".Coassembly\_KO.txt" $n --tmp-dir=tmp\_KO --cpu=10  
done  
  
#step 1  
# get ko to protein list   
module load python  
for n in \*Coassembly\_KO.txt  
do  
python /home/suginoka/Metagenomics\_Programs/select\_best\_KO.py $n protein2ko\_besthit.txt ko2proteins.txt  
done  
  
# step2  
# sum the protein rpkm abundance in the same ko term.   
for n in \*rpkm.txt  
do  
python /home/suginoka/Metagenomics\_Programs/sum\_ko\_rpkm.py $n ko2proteins.txt $n"\_ko.txt"  
done  
  
# step 3  
# merge the ko rpkm file from all samples into 1 file  
#################################################################NOTE: Need to change directory to current directory  
module load pandas  
echo "sample\_names" > all\_sample.list  
ls \*\_ko.txt >> all\_sample.list  
python /home/suginoka/Metagenomics\_Programs/merge.py all\_sample.list ko\_abundance\_table.txt /scratch/suginoka/mom\_prot/  
# ko\_abundance\_table.txt is the table we want.  
  
#Minpath pathways  
#edit details file in R; sum rpkm values under each protein family  
cat \*besthit.txt > cat.besthit.txt  
  
for n in cat.besthit.txt  
do  
awk '{print $1,$2}' $n > $n"\_minpath.txt"  
done  
  
module load python  
for n in \*minpath.txt  
do  
python /home/suginoka/Metagenomics\_Programs/MinPath/MinPath.py -ko $n -report $n"\_minpath\_report" -details $n"\_details"  
done  
  
#Format the file for use in R; could be done in shell or python, but I'm bad with both  
sed 's/ /\_/g' cat.besthit.txt\_minpath.txt\_details > cat.besthit.txt\_minpath.txt\_details.edit

* This will give you back a big gene list in the KEGG format (e.g., K19170)
* I don’t know of a better way to do this, but you can search the IDs (<https://www.genome.jp/kegg/tool/map_pathway.html>) and get back a pathyway map. If you combine this with stats to show only significant associations you can see what pathways may be important in the system (you can also combine this with correlation stats to show up or down associations)
* Here’s another version of the pathway maps that can be useful in disentangling the map results: <https://www.genome.jp/kegg/pathway.html#metabolism>
* An alternative to kofamscan is using DIAMOND and Metacyc to annotate the proteins predicted by Prodigal. I don’t know this technique very well and I never used data from it, but I do know the code below runs and returns gene prediction annotations.
  + More info can be found at <https://github.com/thepanlab/MetagenomicsTree/blob/master/metacyc/metacyc_annotation.md>

module load DIAMOND  
for n in \*trans\_protein.fasta  
do  
diamond blastp --query $n \  
 --db uniprot-seq-ids.fasta \  
 --out $n"\_Metacyc\_protein\_top5hit.blst"\  
 --outfmt 6 --evalue 0.001 --max-target-seqs 5 --sensitive  
done  
  
for n in \*Metacyc\_protein\_top5hit.blst  
do  
python creat\_RXN\_dictionary.py $n $n"\_Metacyc\_protein\_RXN\_key\_sen"

* I was also given code for genome binning. Binning is used to create metagenome assembled genomes (MAGs), which can enable species and even strain-level info, as well as better annotations of functional information
  + See this small overview formore info on MAGs:
    - <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8724365/>
  + And see this study on testing MAG assemblers:
    - <https://bmcgenomics.biomedcentral.com/articles/10.1186/s12864-022-08967-x>
* For the code given to me, they use the summarized .bam files as input into MetaBat, which handles the MAG assembly
  + This is followed by CheckM to return stats on the quality of the assembly
  + Note that this methods is hard and the parameters need to be tuned to get a good (i.e., accurate and stable) MAG; I have not successfully done it, and I am working on better understanding the process currently.

##Sort bam for binning  
for n in \*alignment.sam  
do  
samtools view -bS $n -@ 19 > $n"\_alignment.bam"  
done  
  
for n in \*alignment.bam  
do  
samtools sort $n -@ 19 -o $n"\_alignment\_sorted"  
done  
  
for n in \*alignment\_sorted  
do  
samtools index $n   
done  
  
##metabat binning  
module load MetaBAT  
for n in \*\_sorted  
do  
jgi\_summarize\_bam\_contig\_depths --outputDepth $n"\_output\_depth.txt" $n  
done  
  
#I think there should only be the merged min1000 file, not individual sample files  
#need to move all files into sample-specific folders manually -- sorry.  
for n in \*min1000.fastq  
do  
metabat -i $n -o $n"\_bin"/output\_bin -a \*output\_depth.txt -m 1500  
done  
  
##taxonomy annotation  
#################################################################NOTE: Need to change directory to current directory  
module load CheckM  
checkm lineage\_wf -f CheckM.txt -t 10 -x fa --pplacer\_threads 1 /scratch/suginoka/CHO57\_inf/ /scratch/suginoka/CHO57\_inf/pan\_checkm\_wf\_out

* So, I never had to do this next step and I don’t have a great idea as to how to write the code, but you’ll want to get a taxonomy assignment for the MAG.
  + I believe that CheckM sorta does this, but GTDBTK is a better check (I think…)
  + Here’s a link to their quick start guide: <https://github.com/cerebis/GtdbTk#quick-start>
  + You can also use this to method to find protein annotations and link them to the MAG taxonomy assignment–something I’ve been unsuccessfully working on figuring out
* Generally, running GTDBTK will look like this:

export GTDBTK\_DATA\_PATH=/scratch/suginoka/release202/  
   
gtdbtk classify\_wf --genome\_dir $MAG\_bin --out\_dir $n"\_gtdbtk" --extension .fa