Illumina data, Fungi: Make ASV matrix

Marissa Lee

12/16/2019

Important note: The purpose of this Rmd is to process the fungal sequencing data into an ASV table. Since the sequence data and intermediates can be large files, I have commented-out a large chunk of this code so that it can be referenced but doesn't run by default.

The *input data* are stored on the Hawkes Lab GDrive at the paths below. Also, these data have been deposited in the NCBI Short Read Archive under BioProject PRJNA648664 (https://www.ncbi.nlm.nih.gov/bioproject/PRJNA648664/).

- path1 <- "/Volumes/GoogleDrive/Shared drives/HawkesLab/Projects/DOE LLNL/Data/NC Data/Sequencing - Illumina/GSL_data/fastq_DOENCFIELD-FUN-HAWKES_2020_01_13"
- path2 <- "/Volumes/GoogleDrive/Shared drives/HawkesLab/Projects/DOE LLNL/Data/NC Data/Sequencing - Illumina/GSL_data/fastq_redo_DOENCFIELD-FUN-HAWKES_2020_02_05"

The *output* ASV table can be found in "data/ASVmatrix.csv" with metadata "data/Metadata.xlsx".

Table of contents

A. Process sequence data with dada2 [commented out]

1. Download and store sequence data
2. Preprocess to remove ambiguous bases, adapters, and primers
3. Trim and filter
4. Infer sequence variants: Determine sequencing error rates and then loop through each sample to de-replicate, infer variants, and merge paired reads. Then, construct the sequence table
5. Remove chimeras
6. Track the number of reads lost through the dada2 pipeline
7. Merge sequence tables from run1 and run2 and save reference ASVs

NOTE – This project had 2 MiSeq runs, so dada2 processing happens 1x for each run and then the resulting ASV tables get merged

B. ASV curation and classification

1. Summarize the number of synthetic mock reads in biological samples
2. Get ASV taxonomic ids with dada2 and unite database
3. Curate ASVs (lulu) to combine nested mother/daughter matches
4. Identity low confidence IDs, remove plant ASVs
5. Contaminants in negative controls? If so, need to account for this in the samples.
6. Remove failed samples and mocks

C. Summarize sequencing results per sample type (leaf, root, soil)

1. Number of reads and ASVs per sample type
2. Sample-effort curves
3. ASV occupancy-abundance plots

Load packages and custom functions

A. Process sequence data [commented out]

Comments about using different computers: there are are couple of key things to keep in mind when running this script on a different operating systems (PC versus Mac/Linux) and RAM.

1. Dada2 functions cannot take advanage of multithreading if on a PC. It's best to just set these to FALSE. It will take significantly longer for the process to complete without multithreading
2. The software that dada2 calls to do primer removal does not work on a PC. ML's current approach is to do that step on her Mac laptop and then move the downstream files to the Lab PC if need be.
3. De-replication can be very memory intensive. For reference, it took >8G of physical RAM to de-replicate a 12M read MiSeq run.

1. Download and store sequence data

Downloaded sequence data from GSL on 1/14/20: fastq_Hawkes_2019_12_10.tar. ML updated filename to fastq_DOENCFIELD-FUN-HAWKES_2020_01_13.tar; see data/illumina/readme_GSL_data.xlsx

Downloaded sequence data from GSL on 2/5/20: MiSeq2_140_Hawkes_redo.tar. ML updated filename to fastq_redo_DOENCFIELD-FUN-HAWKES_2020_02_05.tar; see data/illumina/readme_GSL_data.xlsx

Zipped files stored here: - `/Volumes/GoogleDrive/Shared drives/HawkesLab/Projects/DOE LLNL/Data/NC Data/Sequencing - Illumina/GSL_data/GSL_data`

To unzip the .tar file, type this in the Terminal: ex `tar -xvf [filename]`

2. Preprocess to remove ambiguous bases, adapters, and primers [commented-out]

Set up paths

```
# # set path to fastq files from each MiSeq run
#
# path1 <- "/Volumes/GoogleDrive/Shared drives/HawkesLab/Projects/DOE LLNL/Data/NC Data/Sequencing - Il
#
# path2 <- "/Volumes/GoogleDrive/Shared drives/HawkesLab/Projects/DOE LLNL/Data/NC Data/Sequencing - Il
#
# # set up folders to hold data intermediates for each run
# base_intermed_path <- "data_intermediates/Illum_analyses"
# run1_path<- file.path(base_intermed_path, "FUN-2020_01_13")
# run2_path<- file.path(base_intermed_path, "FUN-2020_02_05")
# add.dir(run1_path)
# add.dir(run2_path)
#
# # function to set up subdirectories within each run folder
# estab_dada2_folders <- function(runPath){
#
#   # fastq file versions
#   prefilt_path <- file.path(runPath, "prefiltered") # after ambigious bases have been removed
#   cutAdapt_path <- file.path(runPath, "cutAdapt") # after Illumina adapters have been removed
#   filt_path <- file.path(runPath, "filtered") # after filtering
```

```
#
#   # Rdata intermediates
#   misc_path <- file.path(runPath, "misc")
#
#   # ASV table and associated data
#   seqtab_path <- file.path(runPath, "seqtab") # location of seqtabs
#
#   # pipeline checks (fastq quality, track read losses, etc)
#   track_path <- file.path(runPath, "track")
#
#   # make these new folders if they do not already exist
#   subdirectories <- c(prefilt_path, cutAdapt_path, filt_path,
#                       misc_path,
#                       seqtab_path,
#                       track_path)
#
#   return(subdirectories)
#
# }
#
# # set up subdirectories
# run1_subdirs <- estab_dada2_folders(runPath = run1_path)
# run2_subdirs <- estab_dada2_folders(run2_path)
```

Parse the file sample names ([sampleID]_S###_L###_R#_001.fastq.gz) and populate the sample dataframe with intermediate filepaths

```
# Set up function for both runs
generate_sample_matrix <- function(curr.path, curr.subdirs, curr.samplemat, curr.sampledf){

  ### 1. Parse the sample names in the fastq files ###
  fnFs <- sort(list.files(curr.path, pattern="_R1_001.fastq.gz")) # Sort ensures forward/reverse reads
  fnRs <- sort(list.files(curr.path, pattern="_R2_001.fastq.gz"))
  fnFs <- file.path(curr.path, fnFs) # Specify the full path to the fnFs and fnRs
  fnRs <- file.path(curr.path, fnRs)
  basename(fnFs) %>%
    strsplit("_") %>%
    list_to_df()  -> namestr.df # Extract sample names
  colnames(namestr.df) <- c("sample.name.match","S","L","R","numExt")
  samples <- namestr.df$sample.name.match
  #samples


  ### 2. Read in the sample matrix ###
  curr.samplemat %>%
    mutate(sample.name.match = `Sequencing sample id`) %>% # match sample name in files
    mutate(sample.type = ifelse(grepl("Mock", sample.name.match), "Mock", "sample")) %>% # is it a cont
    mutate(sample.type = ifelse(grepl("NEG", sample.name.match), "NEG", sample.type)) %>%
    select(`Sequencing sample id`,`i7 Index`,`i5 Index`,`PCR Tube id`,
           sample.name.match, sample.type,
           `PCR2.AccuClear__ng/ul`) -> samplemat
  curr.sampledf %>%
    select(`PCR Tube id`,Site, SiteSamp, Tissue,
           `Tube num`, `DNA Tube id`,
```

```
          `DNA.AccuClear__ng/ul`,
          `PCR1 redo notes`,`PCR2 redo notes`) -> sampledf
  # check that this is 0... if not, then there are missing samples
  sum(!samplemat$`PCR Tube id` %in% sampledf$`PCR Tube id`)
  # merge the dataframes
  samplemat %>%
    left_join(sampledf) %>%
    select(sample.name.match, sample.type,
          `Sequencing sample id`, `i7 Index`, `i5 Index`, `PCR Tube id`,
          Site, SiteSamp, Tissue,
          `Tube num`, `DNA Tube id`,
          `DNA.AccuClear__ng/ul`, `PCR2.AccuClear__ng/ul`,
          `PCR1 redo notes`, `PCR2 redo notes`) -> sample.df


  ### 3. Annotate the sample matrix with dada2 output filepaths ###
  sample.df %>%
    left_join(namestr.df) %>% # add namestr.df
    dplyr::rename('R1'='R') %>% # add column to id forward reads
    mutate(R2 = "R2") %>% # add column to id reverse reads
    # raw sequence data
    mutate(fnFs = file.path(curr.path, paste(sample.name.match, S, L, R1, numExt, sep = "_"))) %>%
    mutate(fnRs = file.path(curr.path, paste(sample.name.match, S, L, R2, numExt, sep = "_"))) %>%

    # ambiguous bases removed
    mutate(prefiltFs = file.path(curr.subdirs[1],
                                 paste(sample.name.match, S, L, R1, "prefilt.fastq.gz", sep = "_"))) %>%
    mutate(prefiltRs = file.path(curr.subdirs[1],
                                 paste(sample.name.match, S, L, R2, "prefilt.fastq.gz", sep = "_"))) %>%
    # adapters removed
    mutate(Fs.cutAdapt = file.path(curr.subdirs[2],
                                   paste(sample.name.match, S, L, R1, "cutAdapt.fastq.gz", sep = "_"))) %>%
    mutate(Rs.cutAdapt = file.path(curr.subdirs[2],
                                   paste(sample.name.match, S, L, R2, "cutAdapt.fastq.gz", sep = "_"))) %>%
    # filtered
    mutate(filtFs = file.path(curr.subdirs[3],
                              paste(sample.name.match, S, L, R1, "filt.fastq.gz", sep = "_"))) %>%
    mutate(filtRs = file.path(curr.subdirs[3],
                              paste(sample.name.match, S, L, R2, "filt.fastq.gz", sep = "_"))) -> df

  return(df)
}

# # # Read in sample matrix data
# samplemat <- read_excel(path = "data/illumina/DOE-NC-FIELD_DNAsamples_2019_11_06.xlsx", sheet = "PCR s
# sampledf <- read_excel(path = "data/illumina/DOE-NC-FIELD_DNAsamples_2019_11_06.xlsx", sheet = "PCR s
#
# # Run 1
# # run1.df <- generate_sample_matrix(curr.path = path1,
# #                                   curr.subdirs = run1_subdirs,
# #                                   curr.samplemat = samplemat,
# #                                   curr.sampledf = sampledf)
# # write.csv(run1.df, file = file.path(run1_subdirs[6], "sample_df.csv"))
```

```
# run1.df <- read.csv(file = file.path(run1_subdirs[6], "sample_df.csv"), row.names = 1, stringsAsFacto
#
# # Run 2
# # run2.df <- generate_sample_matrix(curr.path = path2,
# #                                   curr.subdirs = run2_subdirs,
# #                                   curr.samplemat = samplemat,
# #                                   curr.sampledf = sampledf)
# # write.csv(run2.df, file = file.path(run2_subdirs[6], "sample_df.csv"))
# run2.df <- read.csv(file = file.path(run2_subdirs[6], "sample_df.csv"), row.names = 1, stringsAsFacto
```

Visualize samples with sequencing success in study design matrix.

```
# remove negatives and positives, filter to just include samples in the final pooled library

# run1.df %>%
#   filter(sample.type == "sample") %>%
#   group_by(Tissue, Site) %>%
#   summarize(n = length(sample.name.match))-> tmp.df
# ggplot(tmp.df, aes(x = Tissue, y = Site, label = n, fill = n)) +
#   geom_tile(color = 1) +
#   geom_text() +
#   ggtitle("Fungal Illumina samples")
# ggsave(file.path(run1_subdirs[7], "fungalSamples.png"), width = 4, height = 6)
#
#
# run2.df %>%
#   filter(sample.type == "sample") %>%
#   group_by(Tissue, Site) %>%
#   summarize(n = length(sample.name.match))-> tmp.df
# ggplot(tmp.df, aes(x = Tissue, y = Site, label = n, fill = n)) +
#   geom_tile(color = 1) +
#   geom_text() +
#   ggtitle("Fungal Illumina samples")
# ggsave(file.path(run2_subdirs[7], "fungalSamples.png"), width = 4, height = 6)
```

Remember that leaves from 2 plant samples were missing from WBI-NRT because the leaves were too brown. This is recorded in DOE-NC-FIELD_DNAsamples_2019_11_06.xlsx, sheet = Metadata

Remove ambigious bases

```
# only need to do this once; re-run if update data

# # run1
# start.time <- Sys.time()
# filterAndTrim(fwd = run1.df$fnFs,
#               filt = run1.df$prefiltFs,
#               rev = run1.df$fnRs,
#               filt.rev = run1.df$prefiltRs,
#               maxN = 0, multithread = TRUE, verbose = FALSE)
# end.time <- Sys.time()
# time.taken <- end.time - start.time
# time.taken # 22 mins

# run2
# start.time <- Sys.time()
```

```
# filterAndTrim(fwd = run2.df$fnFs,
#               filt = run2.df$prefiltFs,
#               rev = run2.df$fnRs,
#               filt.rev = run2.df$prefiltRs,
#               maxN = 0, multithread = TRUE, verbose = FALSE)
# end.time <- Sys.time()
# time.taken <- end.time - start.time
# time.taken # 16 mins
```

Remove primers as "linked" adapters in cutadapt. Also, include the argument "–discard-untrimmed". This makes it so that only the regions padded by the primers are retained. https://github.com/marcelm/cutadapt/issues/237 AND https://cutadapt.readthedocs.io/en/latest/recipes.html#trimming-amplicon-primers-from-both-ends-of-paired-end-reads

- Note that this can only be excuted on a Mac/Linux right now since cutadapt does not work on Windows. Cutadapt software is called in the function removePrimers(). If you wish to switch back to a Windows machine, just move the files created by RemovePrimers() in `df$Fs.cutAdapt` and `df$Rs.cutAdapt`. If you wish to avoid using a Mac/Linux, look into using BBDuk instead of cutadapt.
- You can find primers in unusual orientations in the ITS region if you end up sequencing a particularly short region. From Illumina: "sequencing extends beyond the length of the DNA insert, and into the adapter on the opposite end of the library fragment, that adapter sequence will be found on the 3' end of the read" https://support.illumina.com/bulletins/2016/04/adapter-trimming-why-are-adapter-sequences-trimmed-from-only-the--ends-of-reads.html

Amplicon map

```
# ITS2 region
#- '5.8S_Fun Forward primer = 20bp'
FWD.its2 <- "AACTTYRRCAAYGGATCWCT"


#- 'ITS4_Fun Reverse primer = 27bp'
REV.its2 <- "AGCCTCCGCTTATTGATATGCTTAART"
## 606bp region
# ** map **
# forward adapter + linker + index = 51 bps
# forward primer + overhang = 20 + 33 bps
# target region = 394 bps (Taylor et al 2016)
# reverse primer + overhang = 27 + 34 bps
# reverse adapter + linker + index = 47 bps
# total = 606 bps

#Nextera XT adapter: https://support.illumina.com/bulletins/2016/12/what-sequences-do-i-use-for-adapter
adapter <- c("CTGTCTCTTATACACATCT")
```

Linked adapters function

```
removePrimers_linked_sample1 <- function(FWD, REV, fnFs, Fs.out, Rs.out, Fs.in, Rs.in, sample.num){

  # Find the reverse complement of each primer
  FWD.RC <- dada2:::rc(FWD)
  REV.RC <- dada2:::rc(REV)

  # Trim FWD and the reverse-complement of REV off of R1 (forward reads)
  R1.flags <- paste("-a", paste0(FWD, "...", REV.RC))
  # Trim REV and the reverse-complement of FWD off of R2 (reverse reads)
```

```
  R2.flags <- paste("-A", paste0(REV, "...", FWD.RC))

  # Run Cutadapt
  system2(cutadapt, args = c(R1.flags, R2.flags,
                             "-m", 1, # -m 1 Discard processed reads that are shorter than 1; keeps emp
                             "--discard-untrimmed",
                             "-o", Fs.out[sample.num],
                             "-p", Rs.out[sample.num], # output files
                             Fs.in[sample.num],
                             Rs.in[sample.num])) # input files
}

removePrimers_linked <- function(FWD, REV, fnFs, Fs.out, Rs.out, Fs.in, Rs.in){

  # Find the reverse complement of each primer
  FWD.RC <- dada2:::rc(FWD)
  REV.RC <- dada2:::rc(REV)

  # Trim FWD and the reverse-complement of REV off of R1 (forward reads)
  R1.flags <- paste("-a", paste0(FWD, "...", REV.RC))
  # Trim REV and the reverse-complement of FWD off of R2 (reverse reads)
  R2.flags <- paste("-A", paste0(REV, "...", FWD.RC))

  # Run Cutadapt
  for(i in 1:length(fnFs)){
    system2(cutadapt, args = c(R1.flags, R2.flags,
                             "-m", 1, # -m 1 Discard processed reads that are shorter than 1; keeps emp
                             "--discard-untrimmed",
                             "-o", Fs.out[i],
                             "-p", Rs.out[i], # output files
                             Fs.in[i],
                             Rs.in[i])) # input files
  }

}
```

Remove primers

```
# remove the primers

# # run1
# removePrimers_linked(FWD = FWD.its2, REV = REV.its2,
#                              fnFs = run1.df$fnFs,
#                              Fs.out = run1.df$Fs.cutAdapt, Rs.out = run1.df$Rs.cutAdapt,
#                              Fs.in = run1.df$prefiltFs, Rs.in = run1.df$prefiltRs)
# # run2
# removePrimers_linked(FWD = FWD.its2, REV = REV.its2,
#                              fnFs = run2.df$fnFs,
#                              Fs.out = run2.df$Fs.cutAdapt, Rs.out = run2.df$Rs.cutAdapt,
#                              Fs.in = run2.df$prefiltFs, Rs.in = run2.df$prefiltRs)


# ran on 2/14/20 from 5-6pm
```

Visualize quality of the reads after primers have been removed, but before trimAndFilter. Note that if the

sequences vary in length, a red line will be plotted showing the percentage of reads that extend to at least that position.

```
# # # randomly select 20 samples to visualize read quality
# Fs.run1.sample <- sample(run1.df$Fs.cutAdapt, size =  20)
# Fs.run2.sample <- sample(run2.df$Fs.cutAdapt, size =  20)
# Rs.run1.sample <- sample(run1.df$Rs.cutAdapt, size =  20)
# Rs.run2.sample <- sample(run2.df$Rs.cutAdapt, size =  20)
#
#
# p.qual.Fs.run1 <- plotQualityProfile(Fs.run1.sample, aggregate = T) +
#   geom_hline(yintercept = 30, linetype = 2) +
#   ggtitle("Forward, primers removed (20 samples), run1")
# ggsave(plot = p.qual.Fs.run1, filename = file.path(run1_subdirs[6],"quality-f.png"))
#
# p.qual.Fs.run2 <- plotQualityProfile(Fs.run2.sample, aggregate = T) +
#   geom_hline(yintercept = 30, linetype = 2) +
#   ggtitle("Forward, primers removed (20 samples), run2")
# ggsave(plot = p.qual.Fs.run2, filename = file.path(run2_subdirs[6],"quality-f.png"))
#
# p.qual.Rs.run1 <- plotQualityProfile(Rs.run1.sample, aggregate = T) +
#   geom_hline(yintercept = 30, linetype = 2) +
#   ggtitle("Reverse, primers removed (20 samples), run1")
# ggsave(plot = p.qual.Rs.run1, filename = file.path(run1_subdirs[6],"quality-r.png"))
#
# p.qual.Rs.run2 <- plotQualityProfile(Rs.run2.sample, aggregate = T) +
#   geom_hline(yintercept = 30, linetype = 2) +
#   ggtitle("Reverse, primers removed (20 samples), run2")
# ggsave(plot = p.qual.Rs.run2, filename = file.path(run2_subdirs[6],"quality-r.png"))
```

3. Trim and filter [commented out]

Notes about read quality and merging based on this post: https://github.com/benjjneb/dada2/issues/232 -> The most important thing is to make sure you don't overtrim (because of quality issues) and lose diversity because the reads are too short to merge -> Trim off no more than half that expected number of overlapping basepairs -> Higher quality data is better, but dada2 handles lower quality data quite well because it incorporates the quality scores into its error model -> Find a set of quality filtering parameters that works on a small set of test data (maybe 5 samples). -> On truncQ specifically, just leave it at its default of truncQ=2 and let the maxEE parameter be the primary quality filter. Callahan et al found expected errors to generally be the best quality filtering method (see also Flyvberg and Edgar 2014 on that). (ML: But relying on maxEE assumes that you are not limited by the number of reads; truncQ will let you salvage crappy reads). Something like maxEE = c(2,6) is not unreasonable. Then see what fraction of my reads come out the other end. If it's too low (>10% losses), I'd try a different set of parameters.

In this case, parameters of the filterAndTrim() function are set to do the following:

- (truncQ = 2, default) truncate reads at the first instance of a quality score less than or equal to 2
- (maxN = 0, default) remove reads with unknown bases (aka N)
- (maxEE = c(2,5)) remove reads with higher than X maxEE "expected errors"; first number is for fowards and second number is for reverses

ALL samples

```
# run 1
# start.time <- Sys.time()
# out.run1 <- filterAndTrim(fwd = run1.df$Fs.cutAdapt,
```

```
#                           filt = run1.df$filtFs,
#                           rev = run1.df$Rs.cutAdapt,
#                           filt.rev = run1.df$filtRs,
#                       maxN = 0, maxEE = c(2, 5),
#                       truncQ = 2, minLen = 50,
#                       rm.phix = TRUE, compress = TRUE,
#                       multithread = TRUE, verbose = TRUE) # cannot do multithreading on Windows
# saveRDS(out.run1, file = file.path(run1_subdirs[5],"out.RData"))
# #out.run1 <- readRDS(file = file.path(run1_subdirs[5],"out.RData"))
# end.time <- Sys.time()
# time.taken <- end.time - start.time
# time.taken # 12.98 mins
#error message: Some input samples had no reads pass the filter.


# run 2
# start.time <- Sys.time()
# out.run2 <- filterAndTrim(fwd = run2.df$Fs.cutAdapt,
#                           filt = run2.df$filtFs,
#                           rev = run2.df$Rs.cutAdapt,
#                           filt.rev = run2.df$filtRs,
#                       maxN = 0, maxEE = c(2, 5),
#                       truncQ = 2, minLen = 50,
#                       rm.phix = TRUE, compress = TRUE,
#                       multithread = TRUE, verbose = TRUE) # cannot do multithreading on Windows
#
# saveRDS(out.run2, file = file.path(run2_subdirs[5],"out.RData"))
# #out.its2 <- readRDS(file = file.path(run2_subdirs[5],"out.RData"))
# end.time <- Sys.time()
# time.taken <- end.time - start.time
# time.taken #12.14 mins
```

Look at the read qualities after filtering

```
# # # randomly select 20 samples to visualize read quality
# Fs.run1.sample <- sample(run1.df$filtFs, size =  20)
# Fs.run2.sample <- sample(run2.df$filtFs, size =  20)
# Rs.run1.sample <- sample(run1.df$filtRs, size =  20)
# Rs.run2.sample <- sample(run2.df$filtRs, size =  20)
#
#
# p.qual.Fs.run1 <- plotQualityProfile(Fs.run1.sample, aggregate = T) +
#   geom_hline(yintercept = 30, linetype = 2) +
#   ggtitle("Forward, filtered (20 samples), run1")
# ggsave(plot = p.qual.Fs.run1, filename = file.path(run1_subdirs[6],"quality-f-filt.png"))
#
# p.qual.Fs.run2 <- plotQualityProfile(Fs.run2.sample, aggregate = T) +
#   geom_hline(yintercept = 30, linetype = 2) +
#   ggtitle("Forward, filtered (20 samples), run2")
# ggsave(plot = p.qual.Fs.run2, filename = file.path(run2_subdirs[6],"quality-f-filt.png"))
#
# p.qual.Rs.run1 <- plotQualityProfile(Rs.run1.sample, aggregate = T) +
#   geom_hline(yintercept = 30, linetype = 2) +
#   ggtitle("Reverse, filtered (20 samples), run1")
```

```
# ggsave(plot = p.qual.Rs.run1, filename = file.path(run1_subdirs[6],"quality-r-filt.png"))
#
# p.qual.Rs.run2 <- plotQualityProfile(Rs.run2.sample, aggregate = T) +
#   geom_hline(yintercept = 30, linetype = 2) +
#   ggtitle("Reverse, filtered (20 samples), run2")
# ggsave(plot = p.qual.Rs.run2, filename = file.path(run2_subdirs[6],"quality-r-filt.png"))
#
```

4. Infer sequence variants – Determine sequencing error rates and then loop through each sample to de-replicate, infer variants, and merge paired reads [commented out]

Learn errors for each run

```
# set.seed(100)
#
# # RUN 1
# errF.run1 <- learnErrors(run1.df$filtFs, nbases=1e8, multithread=TRUE) # Learn forward error rates
# saveRDS(errF.run1, file = file.path(run1_subdirs[4], "errF.RData"))
# plotErrors(errF.run1, nominalQ=TRUE) +
#   ggtitle("Forward errors, run1")
# ggsave(filename = file.path(run1_subdirs[6], "errF.png"))
#
# errR.run1 <- learnErrors(run1.df$filtRs, nbases=1e8, multithread=TRUE) # Learn reverse error rates
# saveRDS(errR.run1, file = file.path(run1_subdirs[4], "errR.RData"))
# plotErrors(errR.run1, nominalQ=TRUE) +
#   ggtitle("Reverse errors, run1")
# ggsave(filename = file.path(run1_subdirs[6], "errR.png"))
#
#
# # RUN 2
# errF.run2 <- learnErrors(run2.df$filtFs, nbases=1e8, multithread=TRUE) # Learn forward error rates
# saveRDS(errF.run2, file = file.path(run2_subdirs[4], "errF.RData"))
# plotErrors(errF.run2, nominalQ=TRUE) +
#   ggtitle("Forward errors, run2")
# ggsave(filename = file.path(run2_subdirs[6], "errF.png"))
#
# errR.run2 <- learnErrors(run2.df$filtRs, nbases=1e8, multithread=TRUE) # Learn reverse error rates
# saveRDS(errR.run2, file = file.path(run2_subdirs[4], "errR.RData"))
# plotErrors(errR.run2, nominalQ=TRUE) +
#   ggtitle("Reverse errors, run2")
# ggsave(filename = file.path(run2_subdirs[6], "errR.png"))

# Error plots
# Title of each plot indicates character change error (e.g., A2C); P(error)=10^(-q/10)
# Red line = quality scores, black dots = obs, black line = dada2 modeled error
# If perfect, black line and red lines will match
# Check that error rates mostly decrease with quality score (although little uplifts at the end are not
# The goal here is good, not perfect, so don't sweat the small stuff (or non-convergence). No convergen
```

The title of each error plot indicates character change error (e.g., A2C); P(error)=10^(-q/10). Red line = quality scores, black dots = obs, black line = dada2 modeled error. If perfect, black line and red lines will match. Check that error rates mostly decrease with quality score (although little uplifts at the end are not a worry). The argument "nbases = 1e8" uses a subset of the bases to learn the error rates. This should reduce

the memory demands for this processes by learning errors from a subset of the data at time.

Sample inference and merger of paired-end reads

```
# # one sample S9 did not have any reads pass through trimAndFilter
# length(list.files(run1_subdirs[2]))
# length(list.files(run1_subdirs[3]))
# # exclude this sample from the loop

# RUN 1
# sample.df = run1.df
# errF = errF.run1
# errR = errR.run1
#### START dada2 "big data" pipeline #######
# sample.names <- sample.df$sample.name.match
# filtFs <- sample.df$filtFs
# filtRs <- sample.df$filtRs
# names(filtFs) <- sample.names
# names(filtRs) <- sample.names
# mergers <- vector("list", length(sample.names))
# names(mergers) <- sample.names
# #for(sam in sample.names) {
# for(sam in 265:length(sample.names)) {
#   if(sam != 265){
#     cat("Processing:", sam, "\n")
#     derepF <- derepFastq(filtFs[[sam]])
#     ddF <- dada(derepF, err=errF, multithread=TRUE)
#     derepR <- derepFastq(filtRs[[sam]])
#     ddR <- dada(derepR, err=errR, multithread=TRUE)
#     merger <- mergePairs(ddF, derepF, ddR, derepR, minOverlap = 10, justConcatenate=FALSE, trimOverha
#     mergers[[sam]] <- merger
#   }else{
#     mergers[[sam]] <- "no reads"
#   }
# }
# rm(derepF); rm(derepR)
#### END dada2 "big data" pipeline #######
# mergers.run1 <- mergers
# saveRDS(mergers.run1, file.path(run1_subdirs[5], "mergers.rds"))
# mergers.run1.n <- mergers.run1[-265] # remember this one has no reads
# seqtab.run1 <- makeSequenceTable(mergers.run1.n)
# dim(seqtab.run1)
# saveRDS(seqtab.run1, file.path(run1_subdirs[5], "seqtab.rds"))

# RUN 2
# sample.df = run2.df
# errF = errF.run2
# errR = errR.run2
#### START dada2 "big data" pipeline #######
# sample.names <- sample.df$sample.name.match
# filtFs <- sample.df$filtFs
# filtRs <- sample.df$filtRs
# names(filtFs) <- sample.names
# names(filtRs) <- sample.names
# mergers <- vector("list", length(sample.names))
```

```
# names(mergers) <- sample.names
# for(sam in sample.names) {
#   cat("Processing:", sam, "\n")
#   derepF <- derepFastq(filtFs[[sam]])
#   ddF <- dada(derepF, err=errF, multithread=TRUE)
#   derepR <- derepFastq(filtRs[[sam]])
#   ddR <- dada(derepR, err=errR, multithread=TRUE)
#   merger <- mergePairs(ddF, derepF, ddR, derepR, minOverlap = 10, justConcatenate=FALSE, trimOverhang
#   mergers[[sam]] <- merger
# }
# rm(derepF); rm(derepR)

#### END dada2 "big data" pipeline #######
# mergers.run2 <- mergers
# saveRDS(mergers.run2, file.path(run2_subdirs[5], "mergers.rds"))
# seqtab.run2 <- makeSequenceTable(mergers.run2)
# saveRDS(seqtab.run2, file.path(run2_subdirs[5], "seqtab.rds"))
```

Steps included here:

- De-replication – combines identical reads into "unique sequences" with a corresponding abundance of reads per unique sequence. **Note** Dereplication of the reverse reads (all samples at once) maxed out the RAM on ML's laptop, so needed to move to using the Lab Windows computer. Changing this so that each sample is processed individual should fix memory issues.
- Merge paired reads – Merge paired reads with minOverlap = 10 (12 is the default). Save table of accepted and rejected sequence pairs in data_intermediates/dada2 folder. Re-run mergePairs() with returnRejects = FALSE for downstream use.
- Construct the sequence table – Create the ASV table. Matrix with rows corresponding to (and named by) the samples; columns corresponding to (and named by) the sequence variants.

Inspect the distribution of sequence lengths and check to make sure they are in the correct range.

```
# RUN 1
# png(file = file.path(run1_subdirs[6], "seqLength_distribution.png"), width = 400, height = 400)
# hist(nchar(getSequences(seqtab.run1)), xlab = "Sequence length (bps)", main = NULL)
# dev.off()
#
# # RUN 1
# png(file = file.path(run2_subdirs[6], "seqLength_distribution.png"), width = 400, height = 400)
# hist(nchar(getSequences(seqtab.run2)), xlab = "Sequence length (bps)", main = NULL)
# dev.off()


# Do the lengths of the merged sequences all fall in the expected range for this amplicon?
# Some seqs may be far away from mode - check them to see if real (might be or if not, discard)
# Sequences that are much longer or shorter than expected may be the result of non-specific priming, an
## You can remove non-target-length sequences with base R manipulation of the sequence table, e.g.,
# seqtab1 <- seqtab[,nchar(colnames(seqtab1)) %in% seq(250,256)]
## This is analogous to "cutting a band" in-silico to get amplicons of the targeted length
```

5. Remove chimeras [commented out]

```
## REMOVE CHIMERAS
# Default is "consensus". Only has an effect if a sequence table is provided.
# If "consensus": The samples in a sequence table are independently checked for bimeras, and a consensu
```

```
# seqtab.nochim.run1 <- removeBimeraDenovo(seqtab.run1,
#                                          method="consensus",
#                                          multithread=TRUE, verbose=TRUE)
# saveRDS(seqtab.nochim.run1, file.path(run1_subdirs[5],"seqtab_nochim.rds"))
#
#
# seqtab.nochim.run2 <- removeBimeraDenovo(seqtab.run2,
#                                          method="consensus",
#                                          multithread=TRUE, verbose=TRUE)
# saveRDS(seqtab.nochim.run2, file.path(run2_subdirs[5],"seqtab_nochim.rds"))
```

6. Track the number of reads lost [commented out, but see output/illumina]

Percent of reads lost at each step

```
# # RUN 1
# out.run1 <- readRDS(file.path(run1_subdirs[5],"out.RData")) # filtered in and out
# mergers.run1 <- readRDS(file.path(run1_subdirs[5],"mergers.rds")) # filtered in and out
# seqtab.nochim.run1 <- readRDS(file.path(run1_subdirs[5],"seqtab_nochim.rds")) # after chimera check
# tracked.run1 <- trackReads(out = out.run1[-265,],
#             mergers = mergers.run1[-265],
#             seqtab.nochim = seqtab.nochim.run1)
# plot_trackReads(tracked.run1)
# #ggsave(filename = file.path(run1_subdirs[6], "percReadsLost.png"))
#
# # RUN 2
# out.run2 <- readRDS(file.path(run2_subdirs[5],"out.RData")) # filtered in and out
# mergers.run2 <- readRDS(file.path(run2_subdirs[5],"mergers.rds")) # filtered in and out
# seqtab.nochim.run2 <- readRDS(file.path(run2_subdirs[5],"seqtab_nochim.rds")) # after chimera check
# tracked.run2 <- trackReads(out = out.run2,
#             mergers = mergers.run2,
#             seqtab.nochim = seqtab.nochim.run2)
# plot_trackReads(tracked.run2)
# #ggsave(filename = file.path(run2_subdirs[6], "percReadsLost.png"))
```

Initial and final read numbers

```
#
# plot_initFinalReads(tracked.run1)
# ggsave(filename = file.path(run1_subdirs[6], "reads_per_sample.png"))
#
# plot_initFinalReads(tracked.run2)
# ggsave(filename = file.path(run2_subdirs[6], "reads_per_sample.png"))
```

Most samples started with ~30K reads and ended with ~10-15K.

Which samples have very few reads? (run1)

```
# tracked.run1 %>%
#     select(sample.name.match, nonchim) %>%
#     separate(sample.name.match, into = c("tissue","drop"), sep = 1, remove = F) %>%
#     select(-drop) -> tmp
# ggplot(tmp, aes(x = tissue, y = log(nonchim), label = sample.name.match)) +
#   geom_text() +
#   ylab("Log-transformed reads per sample")
#   #ylim(c(-10,6))
```

```
# ggsave(file.path(run1_subdirs[6], "LowReadSamples.png"), width = 4, height = 4)
# tmp %>%
#   filter(log(nonchim) < 5)
```

The negatives look relatively clean. There are three samples that failed: S9 (no reads passed cutAdapt), L92 and R115 (less than 10 reads post-chimera check)

Which samples have very few reads? (run2)

```
# tracked.run2 %>%
#     select(sample.name.match, nonchim) %>%
#     separate(sample.name.match, into = c("tissue","drop"), sep = 1, remove = F) %>%
#     select(-drop) -> tmp
# ggplot(tmp, aes(x = tissue, y = log(nonchim), label = sample.name.match)) +
#   geom_text() +
#   ylab("Log-transformed reads per sample")
# #ylim(c(-10,6))
# ggsave(file.path(run2_subdirs[6], "LowReadSamples.png"), width = 4, height = 4)
# tmp %>%
#   filter(log(nonchim) < 5)
```

The negatives look relatively clean, but possible exception of NEGpcr1. Need to check out what those reads match to and potentially subtract from other samples. There are two samples that failed: L92 and R115 (less than 10 reads post-chimera check). Why was S9 OK during this run?

Examine sample-effort curves

```
# RUN1
# seqtab.nochim <- seqtab.nochim.run1
# curr.subdir <- run1_subdirs[5]
#
# # start generalized script _____________________________#
# leaf.samps <- grepl("R",row.names(seqtab.nochim))
# root.samps <- grepl("L",row.names(seqtab.nochim))
# soil.samps <- grepl("S",row.names(seqtab.nochim))
# png(file = file.path(curr.subdir, "rarecurve_leaf.png"), width=500, height=500)
# rarecurve(seqtab.nochim[leaf.samps,],
#           step=100,
#           xlab="Number of reads per sample",
#           ylab="Cumulative number of ASVs", label=TRUE,
#           main = "Leaf samples")
# dev.off()
#
# png(file = file.path(curr.subdir, "rarecurve_root.png"), width=500, height=500)
# rarecurve(seqtab.nochim[root.samps,],
#           step=100,
#           xlab="Number of reads per sample",
#           ylab="Cumulative number of ASVs", label=TRUE,
#           main = "Root samples")
# dev.off()
#
# png(file = file.path(curr.subdir, "rarecurve_soil.png"), width=500, height=500)
# rarecurve(seqtab.nochim[soil.samps,],
#           step=100,
#           xlab="Number of reads per sample",
#           ylab="Cumulative number of ASVs", label=TRUE,
```

```
#           main = "Soil samples")
# dev.off()
# # end generalized script _____________________________#
#
#

# RUN2
# seqtab.nochim <- seqtab.nochim.run2
# curr.subdir <- run2_subdirs[5]
#
# # start generalized script _____________________________#
# leaf.samps <- grepl("R",row.names(seqtab.nochim))
# root.samps <- grepl("L",row.names(seqtab.nochim))
# soil.samps <- grepl("S",row.names(seqtab.nochim))
# png(file = file.path(curr.subdir, "rarecurve_leaf.png"), width=500, height=500)
# rarecurve(seqtab.nochim[leaf.samps,],
#           step=100,
#           xlab="Number of reads per sample",
#           ylab="Cumulative number of ASVs", label=TRUE,
#           main = "Leaf samples")
# dev.off()
#
# png(file = file.path(curr.subdir, "rarecurve_root.png"), width=500, height=500)
# rarecurve(seqtab.nochim[root.samps,],
#           step=100,
#           xlab="Number of reads per sample",
#           ylab="Cumulative number of ASVs", label=TRUE,
#           main = "Root samples")
# dev.off()
#
# png(file = file.path(curr.subdir, "rarecurve_soil.png"), width=500, height=500)
# rarecurve(seqtab.nochim[soil.samps,],
#           step=100,
#           xlab="Number of reads per sample",
#           ylab="Cumulative number of ASVs", label=TRUE,
#           main = "Soil samples")
# dev.off()
# end generalized script _____________________________#
```

7. Merge sequence tables from run1 and run2 and save reference ASVs [commented out, but see data_intermediates/Illum_analyses/FUN-merged]

```
# seqtab.nochim <- mergeSequenceTables(seqtab.nochim.run1, seqtab.nochim.run2, repeats = "sum")
# dim(seqtab.nochim)
# argument repeats = "sum" will sum reads from the same sample across merged runs


# # save fasta file of reference sequences
# asv.refs <- colnames(seqtab.nochim)
# asv.names <- paste("ASV", seq(1,dim(seqtab.nochim)[2]), sep = "_")
# asv.stringset <- DNAStringSet(asv.refs)
# names(asv.stringset) <- asv.names
# writeXStringSet(asv.stringset,
```

```
#                 filepath = file.path("data_intermediates/Illum_analyses/FUN-merged", "seqtab_nochim_r

# save the ASV table with nicer colnames
# colnames(seqtab.nochim) <- asv.names
# write.csv(seqtab.nochim, file = file.path("data_intermediates/Illum_analyses/FUN-merged","seqtab_noch
```

ASV table has been written to FUN-merged,"seqtab_nochim.csv". ASV reference sequences have been written to "seqtab_nochim_refs.fasta"

B. ASV curation and classification [commented out]

1. Summarize the number of synthetic mock reads in biological samples

Calculate the number of synthetic mocks in the biological samples

```
# merged_path <- "data_intermediates/Illum_analyses/FUN-merged"
# # #
# # # #load ASV file - then check that ASVs are columns and samples are rows
# asv <- read.csv(file.path(merged_path,"seqtab_nochim.csv"), row.names = 1)
# dim(asv) # rows = samples, cols = ASVs
#
# #
# libsize <- data.frame(sample.name.match = names(rowSums(asv)), lib = rowSums(asv))
# libsize %>%
#   arrange(lib)
#
#
# # #
# # # load ASV reference sequences
# asv.refs <- Biostrings::readDNAStringSet(file.path(merged_path, "seqtab_nochim_refs.fasta"))
# length(asv.refs)
#
# # load mock reference sequences
# mock.refs <- Biostrings::readDNAStringSet("data/illumina/amptk_synmock.fasta")
# mock.refs
#
# # identify mock samples
# x <- row.names(asv) %in% c("Mock1","Mock2","Mock3","Mock4")
# mock.rows <- asv[x,]
# rowSums(mock.rows) # mock1 failed
# put.mock.asvs <- colSums(mock.rows)[colSums(mock.rows) != 0]
# # putative mock ASVs
# put.mock.seq <- asv.refs[names(asv.refs) %in% names(put.mock.asvs)]
# # do these all appear to come from the mock.refs?
# #library(Biostrings)
# counts <- list()
# for(i in 1:length(put.mock.seq)){
#   counts[[i]]<- vcountPattern(put.mock.seq[[i]], mock.refs)
# }
# names(counts)<- names(put.mock.seq)
# df.mock <- data.frame(mock.name = names(mock.refs), counts)
# colSums(df.mock[,-1])
```

```
# # exclude ASV_9599.. this doesn't appear to come from a synthetic mock
# mock.seq <- put.mock.seq[names(put.mock.seq) != "ASV_9599"]
# mock.seq
# df.mock %>%
#   select(-"ASV_9599") %>%
#   gather(key = "ASV", value = "n.matches", -mock.name) %>%
#   filter(n.matches > 0) -> df.mock.indx
# df.mock.indx
#
# # look for mock seqs in biological samples
# cols <- colnames(asv) %in% names(mock.seq)
# rows <- !row.names(asv) %in% c("Mock1","Mock2","Mock3","Mock4")
# df <- data.frame(samp = row.names(asv[rows,cols]), asv[rows,cols])
# df %>%
#   gather(key = "ASV", value = "reads", -samp) -> df.l
# mock.seq
# df.l %>%
#   filter(reads != 0) %>%
#   left_join(df.mock.indx) %>%
#   dplyr::rename('sample.name.match'='samp') %>%
#   select(sample.name.match, ASV, mock.name, reads) -> df.mock.out
# df.mock.out
# #write.csv(df.mock.out, file = file.path(out_path,"mock_reads.csv"))

# nbiosamps <- dim(asv[rows,])[1]
# nbiosamps
# nmocks <- length(mock.seq)
# possible.hops <- nbiosamps * 4
# possible.hops
# obs.hops <- sum(df.l$reads)
# (obs.hops / possible.hops)*100
```

2. Get ASV taxonomic ids with dada2 and unite database

Reference database: sh_general_release_dynamic_04.02.2020.tar.gz (DOI: 10.15156/BIO/786368) NOTE: needed to do this on the HPC because only 8GB of memory will make this extremely slow

```
# merged_path <- "data_intermediates/Illum_analyses/FUN-merged"
# #load ASV file - then check that ASVs are columns and samples are rows
# asv <- read.csv(file.path(merged_path,"seqtab_nochim.csv"), row.names = 1)
# dim(asv) # rows = samples, cols = ASVs
#
# # load ASV reference sequences
# asv.refs <- Biostrings::readDNAStringSet(file.path(merged_path, "seqtab_nochim_refs.fasta"))
# length(asv.refs)
# #total.asvs <- length(asv.refs) #14365

# # calculate the estimated HPC time based on test with 100 ASVs
# secs.100asvs <- 198
# all.sec <- (total.asvs / 100) * secs.100asvs
# all.min <- all.sec/60
# all.min/60 # 8 hrs
# trim.sec <- (trim.asvs / 100) * secs.100asvs
# trim.sec/60 # 40 mins
```

```
# ID taxa
# NOTE: need to do this on the HPC because only 8GB of memory will make this extremely slow
#set.seed(100) # Initialize random number generator for reproducibility
#taxa <- assignTaxonomy(asv.refs, "data/phylogeneticTree/sh_general_release_dynamic_04.02.2020.fasta",
# read in taxonomic assignments for all ASVs (14,361)
# taxa <- readRDS(file.path(merged_path, "assignTax_allasvs.RData"))
# tax.mat <- unname(taxa)
# colnames(tax.mat) <- c("kingdom","phylum","class","order","family","genus","species")
# dim(tax.mat)
# row.names(tax.mat) <- names(asv.refs)
```

Create the phyloseq object

```
# # load sample matrix
# sam <- read.csv(file = "output/illumina/track_FUN-2020_02_05/sample_df.csv",
#                 row.names = 1, stringsAsFactors = F)
# row.names(sam) <- sam$sample.name.match
# dim(sam)
#
# # put all into the correct format
# library(phyloseq)
# library(speedyseq)
# OTU=otu_table(asv, taxa_are_rows = FALSE)
# SAM=sample_data(sam)
# TAX=tax_table(tax.mat)
# REF=refseq(asv.refs)
# ps_obj <- phyloseq(OTU, SAM, TAX, REF)
# ps_obj # 14361 ASVs
```

3. Curate ASVs (lulu) to combine nested mother/daughter matches

Make a blast db using the fasta of ASV reference sequences

Blast the ASVs against the database - Open Terminal, cd to seqtab folder where the data are located... - makeblastdb -in seqtab_nochim_refs.fasta -parse_seqids -dbtype nucl - blastn -db seqtab_nochim_refs.fasta -outfmt "6 qseqid sseqid pident" -out match_list.txt -qcov_hsp_perc 80 -perc_identity 84 -query seqtab_nochim_refs.fasta

Use the blast match file and the ASV matrix (transformed) to run lulu

```
# matchlist <- read.table(file=file.path(merged_path, "match_list.txt"), sep = "\t", stringsAsFactors =
# str(matchlist) #check that there are 3 cols & first two ASV cols are characters to match ASVtable des
#
# # transpose the ASV matrix so that rows = ASVs and cols = samples
# asv.t <- t(asv)
# otutab <- as.data.frame(asv.t)
# str(otutab) #confirm that the file is a data.frame and is compatible with the matchlist
#
# #run lulu to curate the ASV table
# require(lulu)
# curated_result <- lulu(otutab, matchlist)

#default curated_result <- lulu(otutab, matchlist, minimum_ratio_type = "min", minimum_ratio = 1, minim
#  #originally had error in names(x) <- value : 'names' attribute [3] must be the same length as the ve
#  #str(matchlist) shows only one factor in matchlist file - need to read with 3
```

```
#  ## read in file and specify 3 columns (here, used sep="" to indicate space delimited)
#
#save lulu results
# asvs_lulu <- curated_result$curated_table
#
# write.csv(asvs_lulu, file = file.path(merged_path, "asvs_lulu.csv"))
# asvs_lulu <- read.csv(file = file.path(merged_path, "asvs_lulu.csv"), row.names = 1)
# asvs.lulu.t <- t(asvs_lulu)
# dim(asvs.lulu.t)
# dim(asv)
#
# asv_lulu_discards <- curated_result$discarded_otus
# write.csv(asv_lulu_discards, file = file.path(merged_path, "asv_lulu_discards.csv"))
#
# asv_lulu_retained <- curated_result$curated_otus
# write.csv(asv_lulu_retained, file = file.path(merged_path, "asv_lulu_retained.csv"))
#
# curated_result$curated_count
# # retained 9698 ASVs
#
# curated_result$discarded_count
# # discarded 4663 ASVs
```

Subset the phyloseq object using the lulu-retained ASVs

```
# asv_lulu_retained <- read.csv(file = file.path(merged_path, "asv_lulu_retained.csv"))
# asv_lulu_retained<- as.character(asv_lulu_retained$x)
#
# ps_obj.lulu <- prune_taxa(asv_lulu_retained, ps_obj)
# ps_obj.lulu # 9698 ASVs
```

4. Identity low confidence IDs, remove plant ASVs

How many ASVs are unclassified at the phylum level?

```
# unc80 <- subset_taxa(ps_obj.lulu, is.na(phylum))
# ntaxa(unc80)
# ntaxa(ps_obj.lulu)
# (ntaxa(unc80) / ntaxa(ps_obj.lulu))*100 # 38.68
```

38.68% of the ASVs are unclassified at the phylum level

Use RDP/Warcup to classify that subset of ASVs that are unknown at the phylum level

```
# use this as the input fasta file
#writeXStringSet(refseq(unc80), file.path(merged_path, "unc80.fasta"))

# read in the RDP results
#library(tidyverse)
#source("code/helpers.R") # misc helpful fxns
#sourceDir("code") # loads all the custom functions in this folder

# rdp_taxa <- read.format.rdp_fromWeb(file.path(merged_path, "fixrank_unc80.fasta_classified.txt"))
# rdp_taxa %>%
#   separate(p.perc, into = c("p.perc.num", NA), sep = "%", remove = F) %>%
#   mutate(p.perc.num = as.numeric(p.perc.num)) -> rdp_taxa
```

```
# rdp_taxa %>%
#   filter(p.perc.num < 80) -> tmp
# dim(tmp)
# dim(tmp) / dim(rdp_taxa)
```

95% are still unclassified at the phylum level with confidence of 80%

Add new info with phylum > 80 to the taxonomy

```
# rdp_taxa.unc <- unclassified_at(rdp.df = rdp_taxa, perc.conf.cutoff = 80) # fxn_rdp.R
# # remove lower level assignments with less than 80% confidence
# colnames(rdp_taxa.unc)[1]<- "ASV"
# rdp_taxa.unc[rdp_taxa.unc == "unclassified"] <- NA
# rdp_taxa.unc %>%
#   filter(!is.na(phylum)) -> ok.rdp
#
# # update tax.df with this info
# tax.df <- data.frame(tax_table(ps_obj.lulu), stringsAsFactors = F)
# selection <- is.na(tax.df$phylum)
# asv.order <- row.names(tax.df[selection,])
# df <- data.frame(ASV = asv.order)
# df %>%
#   left_join(ok.rdp) %>%
#   mutate(kingdom = ifelse(!is.na(domain), paste0("k__", domain), NA)) %>%
#   mutate(phylum = ifelse(!is.na(phylum), paste0("p__", phylum), NA)) %>%
#   mutate(class = ifelse(!is.na(class), paste0("c__", class), NA)) %>%
#   mutate(order = ifelse(!is.na(order), paste0("o__", order), NA)) %>%
#   mutate(family = ifelse(!is.na(family), paste0("f__", family), NA)) %>%
#   mutate(genus = ifelse(!is.na(genus), paste0("g__", genus), NA)) %>%
#   separate(species, into = c(NA, "species1"), extra = "merge") %>%
#   mutate(species = ifelse(!is.na(species1), paste0("s__", species1), NA)) %>%
#   select(kingdom,phylum,class,order,family,genus,species, ASV) -> df
#
# tax.df[selection,c("phylum","class","order","family","genus","species")] <-
#   df[,c("phylum","class","order","family","genus","species")]
# tax.mat <- as.matrix(tax.df)
# row.names(tax.mat) <- taxa_names(ps_obj.lulu)
# tax_table(ps_obj.lulu) <- tax.mat
# ps_obj.lulu # 9698 ASVs
#
# # now how many are unknown at the phylum level?
# unk <- subset_taxa(ps_obj.lulu, is.na(phylum))
# ntaxa(unk)
# ntaxa(unk)/ntaxa(ps_obj.lulu)
```

Remove ASVs that are not present in at least 2 samples

```
# # remove asvs that do not show up in at least 2 samples
# ps <- ps_obj.lulu
# ps.pa <- ps
# otu_table(ps.pa)<- (otu_table(ps.pa) > 0)*1
# ps.trim <- subset_taxa(ps, taxa_sums(ps.pa) >= 2)
# ps.trim # 4744 ASVs
```

Use NCBI to identify ASVs that are still unclassified at the phylum level (and present in at least 2 samples). First, Blast against Panicum to remove plant sequences.

```
# # identify the unknowns
# unk <- subset_taxa(ps.trim, is.na(phylum))
# unk # 1168 ASVs
# writeXStringSet(refseq(unk), file.path(merged_path, "blasthits/unk.fasta")) # use this as the input f
# hits <- read.csv(file.path(merged_path, "blasthits/unk-Alignment-HitTable.csv"),
#                  header = F)
# colnames(hits)[1:3] <- c("asv","accession","perc.match")
# colnames(hits)[11] <- "eval"
# asv.hits <- unique(hits$asv)
# length(asv.hits) #738
#
# #filter to just the top 20 hits for each ASV
# ASV <- unique(hits$asv)
# top.hits <- list()
# for(i in 1:length(ASV)){
#   hits %>%
#     filter(asv == ASV[i]) %>%
#     arrange(eval) -> curr.hits
#   if(dim(curr.hits)[1] >= 10){
#     top.hits[[i]] <- curr.hits[1:10,]
#   }else{
#     top.hits[[i]] <- curr.hits
#   }
# }
# names(top.hits) <- ASV
# top.hits <- list_to_df(top.hits)
# length(unique(top.hits$asv)) #738
# # summarize the number of panicum hits per asv
# top.hits %>%
#   group_by(asv) %>%
#   summarize(n.hits = length(asv)) -> top.hits.summ
# top.hits.summ %>%
#   filter(n.hits > 1) -> panicum.asvs
# length(panicum.asvs$asv) #736
# length(panicum.asvs$asv) / ntaxa(unk) # ~60% matched to panicum
#
# # remove panicum asvs
# pavi.asvs <- as.character(panicum.asvs$asv)
# ps <- subset_taxa(ps.trim, !taxa_names(ps.trim) %in% pavi.asvs)
# ps # 4008 ASVs
#
# # identify unknowns again
# unk <- subset_taxa(ps, is.na(phylum))
# unk # 432 ASVs
```

There are still 432 unknown ASVs. Blast these

```
#refseq(unk)
#writeXStringSet(refseq(unk), file.path(merged_path, "blasthits/unk_nopavi.fasta"))
# writeXStringSet(refseq(unk)[1:100], file.path(merged_path, "blasthits/unk_nopavi-1.fasta"))
# writeXStringSet(refseq(unk)[101:200], file.path(merged_path, "blasthits/unk_nopavi-2.fasta"))
# writeXStringSet(refseq(unk)[201:300], file.path(merged_path, "blasthits/unk_nopavi-3.fasta"))
# writeXStringSet(refseq(unk)[301:400], file.path(merged_path, "blasthits/unk_nopavi-4.fasta"))
# writeXStringSet(refseq(unk)[401:432], file.path(merged_path, "blasthits/unk_nopavi-5.fasta"))
```

```
# do a full genbank blast search on the remaining unknowns
# read-in files
# hits1 <- read.csv(file.path(merged_path, "blasthits/nopavi1-Alignment-HitTable.csv"), header = F)
# hits2 <- read.csv(file.path(merged_path, "blasthits/nopavi2-Alignment-HitTable.csv"), header = F)
# hits3 <- read.csv(file.path(merged_path, "blasthits/nopavi3-Alignment-HitTable.csv"), header = F)
# hits4 <- read.csv(file.path(merged_path, "blasthits/nopavi4-Alignment-HitTable.csv"), header = F)
# hits5 <- read.csv(file.path(merged_path, "blasthits/nopavi5-Alignment-HitTable.csv"), header = F)
# hits <- rbind(hits1, hits2, hits3, hits4, hits5)
# colnames(hits)[1:3] <- c("asv","accession","perc.match")
# colnames(hits)[11] <- "eval"
# asv.hits <- as.character(unique(hits$asv))
# sum(!asv.hits %in% taxa_names(unk))
# nohit.asvs <- taxa_names(unk)[!taxa_names(unk) %in% asv.hits]
# length(nohit.asvs)
# length(nohit.asvs) / ntaxa(unk)
```

Remove ASVs that did not return an blast hit against GenBank

```
# ps <- subset_taxa(ps, !taxa_names(ps) %in% nohit.asvs)
# ps # 3927 ASVs
```

Remove initial non-fungal ASVs

```
# #filter to just the top 20 hits for each ASV
# ASV <- unique(hits$asv)
# top.hits <- list()
# for(i in 1:length(ASV)){
#   hits %>%
#     filter(asv == ASV[i]) %>%
#     arrange(eval) -> curr.hits
#   if(dim(curr.hits)[1] >= 10){
#     top.hits[[i]] <- curr.hits[1:10,]
#   }else{
#     top.hits[[i]] <- curr.hits
#   }
# }
# names(top.hits) <- ASV
# top.hits <- list_to_df(top.hits)
# length(unique(top.hits$asv)) # 351 ASVs
#
# # identify accession ids
# accession.names <- as.character(unique(top.hits$accession))
# length(accession.names)
# write.csv(accession.names[1:1643], file = file.path(merged_path, "blasthits/accession_names.csv"), ro
# entered top assession IDs back into NCBI to see what they are
#https://www.ncbi.nlm.nih.gov/portal/utils/batchentrez_p.cgi
# function to read the summaries in
read.accessionfiles <- function(file.name, read2){

  if(read2 == TRUE){
    tmp1 <- read_table2(file = file.path(merged_path, file.name), col_names = F)
    end <- dim(tmp1)[1]
    cur.seq1 <- seq(from = 1, to = end, by = 3)
    text1 <- tmp1[cur.seq1,"X2"]
```

```
    text2 <- tmp1[cur.seq1,"X3"]
    text3 <- tmp1[cur.seq1,"X4"]
    cur.seq2 <- seq(from = 3, to = end, by = 3)
    acc <- tmp1[cur.seq2, "X1"]
    acc.indx <- data.frame(acc, text1, text2, text3, stringsAsFactors = F)
    colnames(acc.indx) <- c("accession","text1","text2","text3")
  }else{
    tmp1 <- read_table(file = file.path(merged_path, file.name), col_names = F, skip_empty_rows = F)
    end <- dim(tmp1)[1]
    cur.seq1 <- seq(from = 2, to = end, by = 4)
    text <- tmp1[cur.seq1,"X1"]
    cur.seq2 <- seq(from = 4, to = end, by = 4)
    acc <- tmp1[cur.seq2, "X1"]
    df <- data.frame(acc, text, stringsAsFactors = F)
    colnames(df) <- c("accession","text")
    df %>%
      separate(text, into = c(NA,"text1","text2","text3"), sep = " ", extra = "drop") %>%
      separate(accession, into = c("accession", NA), sep = " ") %>%
      select(accession, text1, text2, text3) -> acc.indx
  }


  return(acc.indx)
}
# f1 <- read.accessionfiles(file.name = "blasthits/nuccore_result.txt", read2 = TRUE)
# dim(f1)
#
# # id plants
# nonfungal.ids <- c("Acer","Aster","Eupatorium","Gamochaeta","Lucilia",
#                    "Mollugo","Solidago","Linaria","Zea", "Panicum",
#                    "Ipomoea","Echinochloa","Vitis","Rubus","Rhabdodendron","Carex",
#                    "Angiosperm","Digitaria","Nuttallanthus","Setaria","Hordeum",
#                    "Verbesina","Sida","Erigeron","Diptera","Amaranthus","Youngia",
#                    "Senecio","Vahlkampfia","Pardosa","Apocynum","Andropogon","Bambusa",
#                    "Indocalamus","Indosasa","Aristida","Panicum")
# f1 %>%
#   mutate(nonfungal.id = ifelse(text1 %in% nonfungal.ids, TRUE, FALSE)) %>%
#   mutate(nonfungal.id = ifelse(text2 %in% nonfungal.ids, TRUE, nonfungal.id))-> f.all
# dim(f.all)
# # id the remaining unknown hits by merging with accession ids table
# top.hits %>%
#   select(asv, accession, perc.match, eval) %>%
#   left_join(f.all) -> top.hits
#
# # first, id anything that may match to a plant
# top.hits %>%
#   group_by(asv) %>%
#   summarize(n.nonfung.hits = sum(nonfungal.id == TRUE),
#             n.hits = length(nonfungal.id == TRUE),
#             min.eval = min(eval),
#             perc.nonfung = (n.nonfung.hits/n.hits) * 100,
#             top.text1 = paste0(unique(text1[eval == min.eval]), collapse = "_"),
#             top.text2 = paste0(unique(text2[eval == min.eval]), collapse = "_")) -> hits.summ
```

```
# hits.summ %>%
#   filter(n.nonfung.hits > 0) -> hits.summ.nonfung
# init.nonfung.asvs <- as.character(hits.summ.nonfung$asv)
# length(init.nonfung.asvs) #51 additional non-fungal ASVs identified
#
# init.nonfung.asvs
# ps <- subset_taxa(ps, !taxa_names(ps) %in% init.nonfung.asvs)
# ps # 3876 ASVs
```

Parse more of the assession info

```
# unk <- subset_taxa(ps, is.na(phylum))
# unk # 300 ASVs
#
# # now there are ~300 ASVs that are unidentified fungi
# top.hits %>%
#   filter(!asv %in% init.nonfung.asvs) -> top.hits
# length(unique(top.hits$asv)) # 300
#
# # try cleaning up the hit names more for these ASVs
# top.hits %>%
#   mutate(genus = ifelse(!text1 %in% c("Uncultured","PREDICTED:","[Candida]","NA",
#                                       "uncultured"), text1, text2)) -> tmp
# not.genus <- c("root","ectomycorrhiza","ectomycorrhizal","soil","fungus",
#                "basal","Fungal","cf.","basidiomycete","organism","takata")
# tmp %>%
#   mutate(put.taxon = ifelse(!genus %in% not.genus, genus, NA)) %>%
#   mutate(phylum = ifelse(grepl("Asco",put.taxon),
#                          "Ascomycota",
#                          ifelse(grepl("Basidio", put.taxon),
#                                 "Basidiomycota",
#                                 ifelse(grepl("Chytrid", put.taxon),
#                                        "Chytridiomycota",
#                                        ifelse(grepl("chytridiomycete",put.taxon),
#                                               "Chytridiomycota",
#                                               ifelse(grepl("Zygo", put.taxon),
#                                                      "Zygomycota",
#                                                      ifelse(grepl("Glom", put.taxon),
#                                                             "Glomeromycota",
#                                                             NA)))))) %>%
#   mutate(put.taxon = ifelse(put.taxon == "NA", NA, put.taxon)) %>%
#   select(-genus) -> tmp
#
# # assign phylum if possible...
# refgen <- read.table("data_intermediates/phylogeneticTree/rep_genus_tax.txt")
# colnames(refgen) <- c("kingdom","phylum","class","order","family","genus","species","gs")
# refgen %>%
#   select(phylum, genus) -> ref.genus
# refgen %>%
#   select(phylum, family) -> ref.family
# refgen %>%
#   select(phylum, order) -> ref.order
# refgen %>%
#   select(phylum, class) -> ref.class
```

```
# colnames(tmp)
# tmp %>%
#   dplyr::rename('genus'='put.taxon') %>%
#   left_join(ref.genus) %>%
#   dplyr::rename('family'='genus',
#                 'phylum1'='phylum') %>%
#   left_join(ref.family) %>%
#   dplyr::rename('order'='family',
#                 'phylum2'='phylum') %>%
#   left_join(ref.order) %>%
#   dplyr::rename('class'='order',
#                 'phylum3'='phylum') %>%
#   left_join(ref.class) -> tmp.out
#
# tmp.out$phylum1 <- as.character(tmp.out$phylum1)
# tmp.out$phylum2 <- as.character(tmp.out$phylum2)
# tmp.out$phylum3 <- as.character(tmp.out$phylum3)
# tmp.out$phylum <- as.character(tmp.out$phylum)
#
# tmp.out$phylum.new <- NA
# x <- !is.na(tmp.out$phylum1)
# tmp.out[x, "phylum.new"] <- tmp.out[x,"phylum1"]
# x <- !is.na(tmp.out$phylum2)
# tmp.out[x, "phylum.new"] <- tmp.out[x,"phylum2"]
# x <- !is.na(tmp.out$phylum3)
# tmp.out[x, "phylum.new"] <- tmp.out[x,"phylum3"]
# x <- !is.na(tmp.out$phylum)
# tmp.out[x, "phylum.new"] <- tmp.out[x,"phylum"]
#
# tmp.out %>%
#   dplyr::rename('put.taxon'='class') %>%
#   select(-c(phylum, phylum1, phylum2, phylum3)) -> curr.df

#write.csv(unique(tmp.unk$text), file = "lookup.csv")
# https://www.ncbi.nlm.nih.gov/Taxonomy/Browser/wwwtax.cgi
# lookup <- read.csv(file.path(merged_path, "blasthits/lookup.csv"))
# lookup %>%
#   filter(phylum == "not_fungal") %>%
#   select(text, phylum) -> lookup.notfung
# lookup %>%
#   filter(!is.na(phylum)) %>%
#   filter(phylum != "") %>%
#   filter(phylum != "not_fungal") %>%
#   select(text, phylum) -> lookup.fung
#
#
# curr.df %>%
#   dplyr::rename('text'='put.taxon') %>%
#   left_join(lookup.notfung) %>%
#   mutate(phylum.comb = ifelse(phylum == "not_fungal", "not_fungal", phylum.new)) %>%
#   select(-c(phylum.new, phylum)) %>%
#   left_join(lookup.fung) %>%
#   mutate(phylum.comb = ifelse(!is.na(phylum.comb), as.character(phylum.comb), as.character(phylum)))
```

```
#   select(-c(phylum)) %>%
#   mutate(phylum.comb = ifelse(text %in% c("metazoan","Begonia","Aphelenchoides",
#                                           "bacterium","Arthrobacter","Actinomyces","Chloromonas",
#                                           "Chrysophyceae","Chlorococcum","Chlamydomonas",
#                                           "Chaetonotus", "Cercomonas","Cephalozia", "Brevibacillus",
#                                           "Desmodesmus"),
#                               "not_fungal", phylum.comb)) %>%
#   mutate(phylum.comb = ifelse(text == "zygomycete","Zygomycota", phylum.comb)) %>%
#   mutate(phylum.comb = ifelse(text == "Ascomycota", text, phylum.comb)) %>%
#   mutate(phylum.comb = ifelse(grepl("Basidio", text), "Basidiomycota", phylum.comb)) %>%
#   mutate(phylum.comb = ifelse(grepl("Glom", text), "Glomeromycota", phylum.comb)) %>%
#   mutate(phylum.comb = ifelse(grepl("glom", text), "Glomeromycota", phylum.comb)) %>%
#   mutate(phylum.comb = ifelse(text == "Acaulospora", "Glomeromycota", phylum.comb)) %>%
#   mutate(phylum.comb = ifelse(text == "Zygomycota", text, phylum.comb)) %>%
#   mutate(phylum.comb = ifelse(text == "Amanita", "Basidiomycota", phylum.comb)) %>%
#   mutate(phylum.comb = ifelse(text == "arabinofermentans", "Ascomycota", phylum.comb)) %>%
#   mutate(phylum.comb = ifelse(grepl("Chytrid", text), "Chytridioomycota", phylum.comb)) %>%
#   mutate(phylum.comb = ifelse(grepl("Diversisporales", text), "Glomeromycota", phylum.comb)) -> df

# # summarize
# df %>%
#   group_by(asv) %>%
#   summarize(n.hits = length(asv),
#             n.phylum = length(unique(phylum.comb)),
#             u.phylum = paste0(unique(phylum.comb), collapse = "_")) -> df.summ
# df.summ %>%
#   filter(n.phylum ==1) %>%
#   filter(u.phylum != "NA") -> df.summ1
#
# df.summ1 %>%
#   filter(u.phylum != "not_fungal") -> asvs.phylumblast
# asvs.phylumblast # asvs id'd to phylum based on blast results
# dim(asvs.phylumblast) #19
#
# df.summ %>%
#   filter(grepl("not_fungal", u.phylum)) -> more.nonfung.asvs
# more.nonfung.asvs # more non-fungal asvs (in addition to nonfung.asvs)
# dim(more.nonfung.asvs) #64
#
# # what do the remaining asvs look like?
# asv.ex <- c(as.character(more.nonfung.asvs$asv), as.character(asvs.phylumblast$asv))
# df.summ %>%
#   filter(!asv %in% asv.ex) -> tmp.unk.rem
# asvs.nophylum <- tmp.unk.rem
# dim(asvs.nophylum) #217


########

# length(as.character(more.nonfung.asvs$asv)) # 64 additional ASVs that matched to non-fungal organisms
# dim(asvs.phylumblast)[1] #19 ASVs that have been assigned to phylum using consensus blast results to
# dim(asvs.nophylum)[1]  #217 ASVs that are putatively fungi, but could not be assigned to phylum
```

```
#######
# update the phyloseq object

# # remove non-fungal asvs
# ps # 3876 ASVs
# nonfung.asvs <- as.character(more.nonfung.asvs$asv)
# ps <- subset_taxa(ps, !taxa_names(ps) %in% nonfung.asvs)
# ps # 3812 ASVs
#
# # update taxonomy
# tax <- data.frame(tax_table(ps), stringsAsFactors = F)
# tax$ASV <- row.names(tax)
# sum(tax$ASV %in% as.character(asvs.phylumblast$asv))
# asvs.phylumblast %>%
#   select(asv, u.phylum) %>%
#   filter(!is.na(u.phylum)) %>%
#   filter(u.phylum != "NA") %>%
#   mutate(u.phylum = paste0("p__",u.phylum)) -> indx
# colnames(indx) <- c("ASV","u.phylum")
# tax %>%
#   left_join(indx) %>%
#   mutate(phylum = ifelse(!is.na(u.phylum), u.phylum, phylum)) %>%
#   mutate(phylum.fromBlast = ifelse(!is.na(u.phylum), TRUE, FALSE)) %>%
#   select(-u.phylum) -> tax.new
# tax.mat <- as.matrix(tax.new)
# row.names(tax.mat) <- tax.new$ASV
#
# tax_table(ps) <- tax.mat
```

5. Contaminants in negative controls? If so, need to account for this in the samples.

```
# ps_negs <- subset_samples(ps, sample.type == "NEG")
# ps_negs <- prune_taxa(taxa_sums(ps_negs) != 0, ps_negs)
# otu_table(ps_negs)
```

Found the following in NEGDNA: - Nigrospora oryzae = 14 reads (ASV_6047) - Sporobolomyces ruber = 12 reads (ASV_8273)

Found the following in NEGpcr1: - Puccinia andropogonis = 43 + 52 reads (ASV_3, ASV_32) - Unclassified Pleosporales = 67 + 7 reads (ASV_7, ASV_393)

Examine how many of these are found in leaf, root, and soil samples

```
# ps_samps <- subset_samples(ps, sample.type == "sample")
# ps_samps <- prune_taxa(taxa_sums(ps_samps) != 0, ps_samps)
#
# selection <- taxa_names(ps_samps) %in% c("ASV_6047","ASV_8273","ASV_3","ASV_32", "ASV_7", "ASV_393")
# ps_tmp <- subset_taxa(ps_samps, selection)
# head(otu_table(ps_tmp))# there is a pretty large range here.
```

Subtract negative reads from samples

```
# asvs.up <- c("ASV_6047","ASV_8273","ASV_3","ASV_32","ASV_7","ASV_393")
# subt <- c(14, 12, 43, 52, 67, 7)
```

```
#
# library(speedyseq)
# otu.tab <- data.frame(otu_table(ps), stringsAsFactors = F)
# otu <- otu.tab
# i<- 1
# for(i in 1:length(asvs.up)){
#   orig <- otu[,asvs.up[i]]
#   up <- orig - subt[i]
#   up[up < 1] <- 0
#   otu[,asvs.up[i]] <- up
# }
#
# otu_table(ps) <- otu_table(otu, taxa_are_rows = F)
# ps_samps <- subset_samples(ps, sample.type == "sample")
# ps_samps <- prune_taxa(taxa_sums(ps_samps) != 0, ps_samps)
# ps_samps
```

6. Remove failed samples and mocks

```
# failed.samps <- subset_samples(ps_samps, sample_sums(ps_samps) < 10)
# sample_names(failed.samps)
#
# selection <- sample_names(ps_samps) %in% c("L92","R115")
# ps_samps.c <- prune_samples(!selection, ps_samps)
# ps_samps
# ps_samps.c
#
# ps_samps.c <- prune_taxa(taxa_sums(ps_samps.c) != 0, ps_samps.c)
# ps_samps.c
# saveRDS(ps_samps.c, file = file.path(merged_path, "phyloseq_samps.RData"))
```

Output phyloseq object from this is "data_intermediates/Illum_analyses/FUN-merged/phyloseq_samps.RData"

C. Summarize sequencing results per sample type (leaf, root, soil)

1. Number of reads and ASVs per sample

```
merged_path <- "data_intermediates/Illum_analyses/FUN-merged"
ps_samps.c <- readRDS(file.path(merged_path, "phyloseq_samps.RData"))

summary(sample_sums(ps_samps.c))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2158    9619   13610   16498   20379   65544
```

```
ntaxa(ps_samps.c)
```

```
## [1] 3811
```

```
# leaves
samps.l <- subset_samples(ps_samps.c, Tissue == "L")
samps.l <- prune_taxa(taxa_sums(samps.l) != 0, samps.l)
```

```
samps.l
```

```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 535 taxa and 109 samples ]
## sample_data() Sample Data:       [ 109 samples by 28 sample variables ]
## tax_table()   Taxonomy Table:    [ 535 taxa by 9 taxonomic ranks ]
## refseq()      DNAStringSet:      [ 535 reference sequences ]
```

```
# roots
samps.r <- subset_samples(ps_samps.c, Tissue == "R")
samps.r <- prune_taxa(taxa_sums(samps.r) != 0, samps.r)
samps.r
```

```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 2372 taxa and 111 samples ]
## sample_data() Sample Data:       [ 111 samples by 28 sample variables ]
## tax_table()   Taxonomy Table:    [ 2372 taxa by 9 taxonomic ranks ]
## refseq()      DNAStringSet:      [ 2372 reference sequences ]
```

```
# soil
samps.s <- subset_samples(ps_samps.c, Tissue == "S")
samps.s <- prune_taxa(taxa_sums(samps.s) != 0, samps.s)
samps.s
```

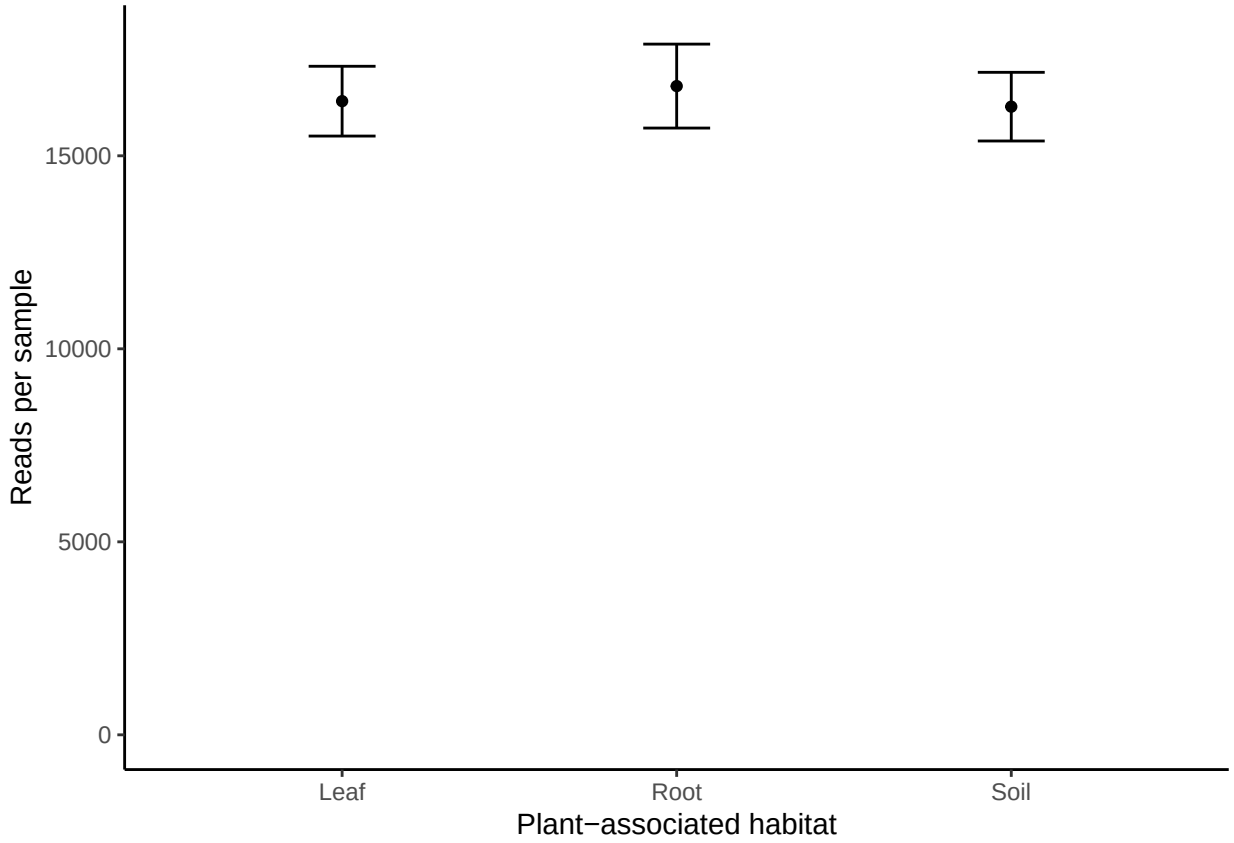
```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 3268 taxa and 112 samples ]
## sample_data() Sample Data:       [ 112 samples by 28 sample variables ]
## tax_table()   Taxonomy Table:    [ 3268 taxa by 9 taxonomic ranks ]
## refseq()      DNAStringSet:      [ 3268 reference sequences ]
```

```
ps.list <- list(l = samps.l, r = samps.r, s= samps.s)

df.summ <- data.frame(n.samples = unlist(lapply(ps.list, nsamples)),
           n.asvs = unlist(lapply(ps.list, ntaxa))
           )
df.summ$source <- row.names(df.summ)

tmp <- lapply(ps.list, function(x)
  t(data.frame(val = c(summary(sample_sums(x)))))
  )
read.summ <- list_to_df(tmp)

df.summ %>%
  full_join(read.summ) -> df.summ
```

```
## Joining, by = "source"
```

```
#write.csv(df.summ, file = file.path(merged_path, "seq_summ.csv"))

#plot
tmp <- lapply(ps.list, function(x)
  t(data.frame(val = sample_sums(x)))
  )
df <- data.frame(mean = lapply(tmp, mean),
           n = lapply(tmp, length),
           sd = lapply(tmp, sd))
```

```r
df %>%
  gather(key = "key", value = "value") %>%
  separate(key, into = c("stat","tissue")) %>%
  mutate(tissue.num = ifelse(tissue == "l",1,
                             ifelse(tissue == "r",2, 3))) %>%
  spread(key = stat, value = value) %>%
  mutate(se = sd/sqrt(n)) -> df.plot

offset <- 0.1
p <- ggplot(df.plot, aes(x = tissue.num, y = mean)) +
  geom_point() +
  geom_errorbar(aes(ymin = mean - se, ymax = mean + se), width = .2) +
  xlab("Plant-associated habitat") +
  ylab("Reads per sample") +
  theme_classic() +
  ylim(c(0, 18000)) +
  scale_x_continuous(breaks = c(1, 2, 3),
                     labels = c("Leaf","Root","Soil"),
                     limits = c(0.5, 3.5))
p
```




```r
out_path <- "output/illumina/summary_FUN-merged"
# ggsave(p, filename = file.path(out_path, "readsPerSamp.png"),
#        width = 3, height = 3)
```

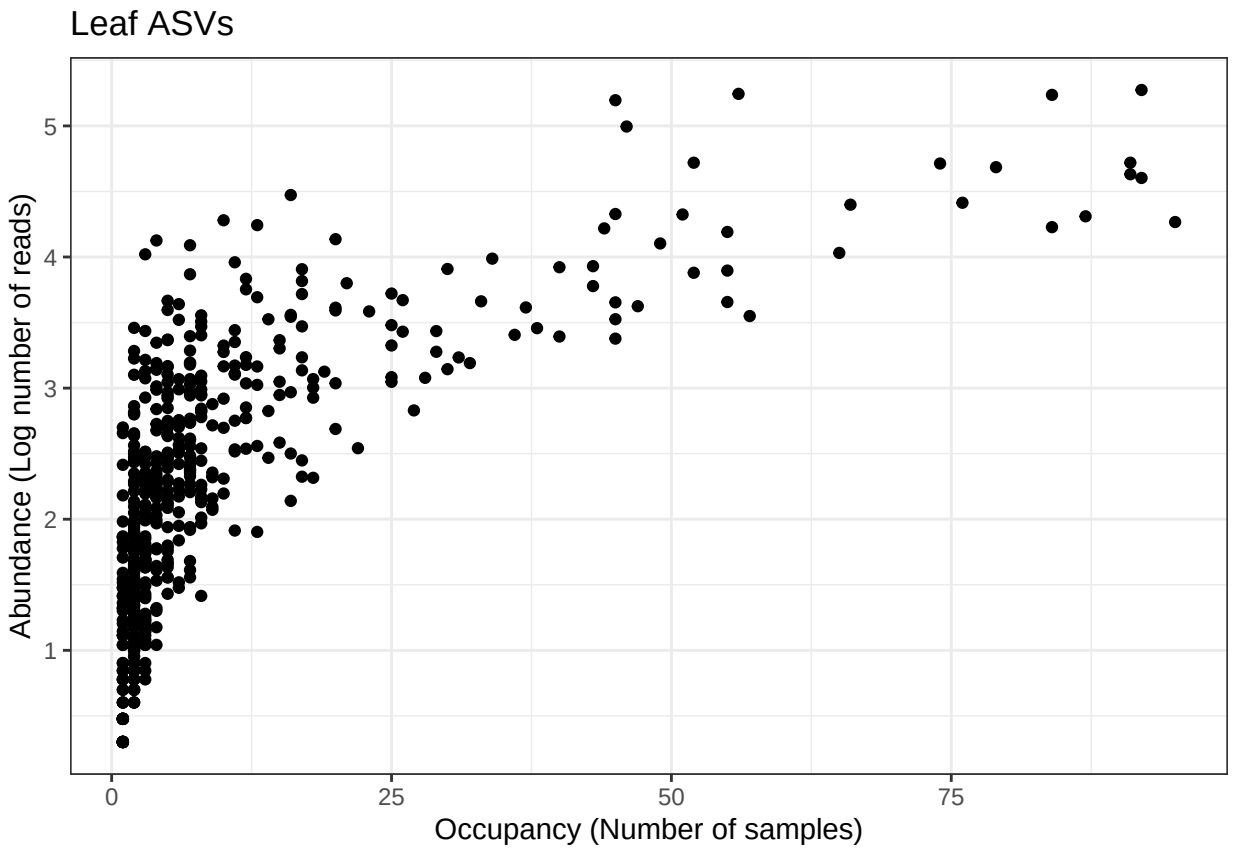
2. Sample-effort curves

```
#require(ggplotify)

# leaf
otumat <- as.matrix(otu_table(samps.l))
# rarecurve(otumat,
#           step=100,
#           xlab="Number of reads per sample",
#           ylab="Cumulative number of ASVs", label=TRUE,
#           main = "Leaf samples")
# p1 <- as.grob(~rarecurve(otumat,
#           step=100,
#           xlab="Number of reads per sample",
#           ylab="Cumulative number of ASVs", label=TRUE,
#           main = "Leaf samples"))
# p1
# ggsave(filename = file.path(merged_path, "rarecurve_leaf_hq.png"),
#        plot = p1,
#        width = 5, height = 5,
#        dpi = 600)

# root
otumat <- as.matrix(otu_table(samps.r))
#png(file = file.path(merged_path, "rarecurve_root.png"), width=500, height=500)
# rarecurve(otumat,
#           step=100,
#           xlab="Number of reads per sample",
#           ylab="Cumulative number of ASVs", label=TRUE,
#           main = "Root samples")
#dev.off()
# p1 <- as.grob(~rarecurve(otumat,
#           step=100,
#           xlab="Number of reads per sample",
#           ylab="Cumulative number of ASVs", label=TRUE,
#           main = "Root samples"))
# p1
# ggsave(filename = file.path(merged_path, "rarecurve_root_hq.png"),
#        plot = p1,
#        width = 5, height = 5,
#        dpi = 600)

# soil
otumat <- as.matrix(otu_table(samps.s))
#png(file = file.path(merged_path, "rarecurve_soil.png"), width=500, height=500)
# rarecurve(otumat,
#           step=100,
#           xlab="Number of reads per sample",
#           ylab="Cumulative number of ASVs", label=TRUE,
#           main = "Soil samples")
#dev.off()
# p1 <- as.grob(~rarecurve(otumat,
#           step=100,
#           xlab="Number of reads per sample",
```

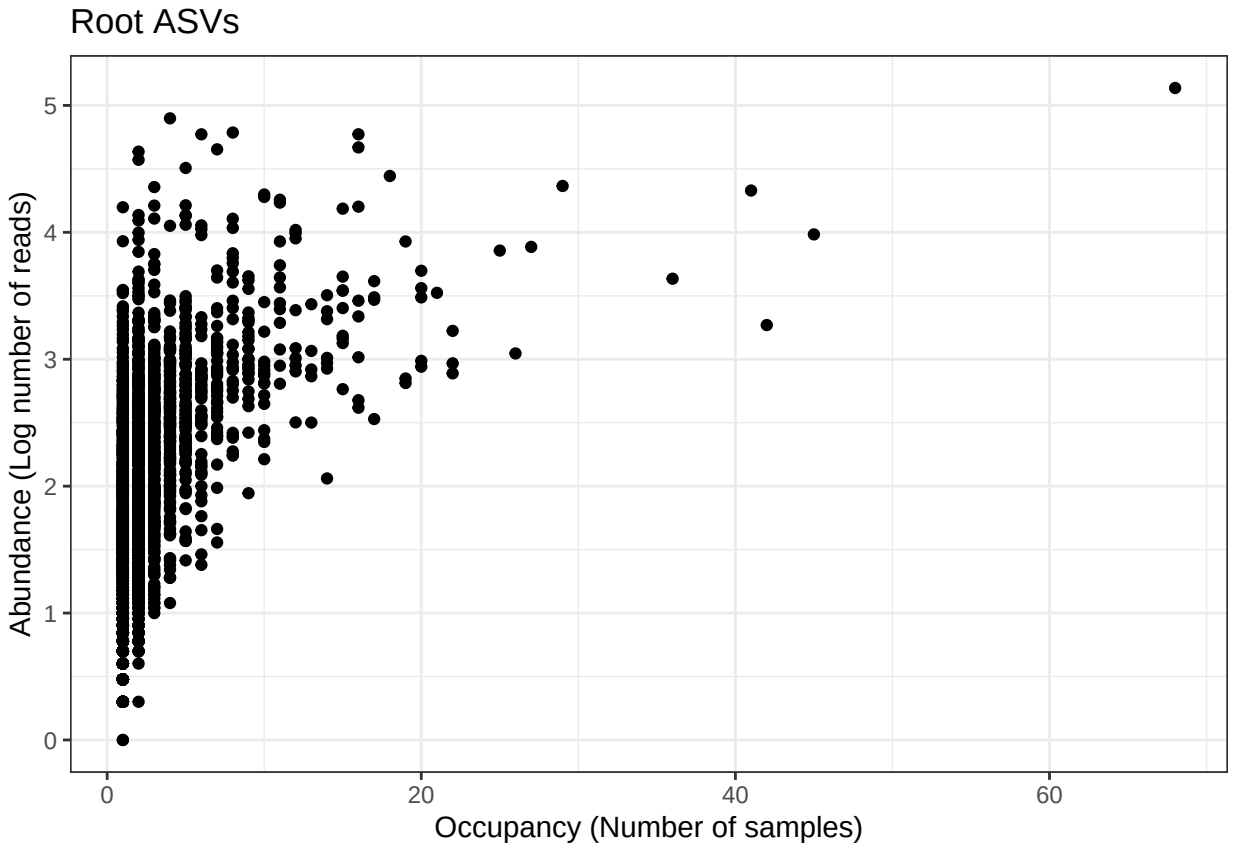
```
#           ylab="Cumulative number of ASVs", label=TRUE,
#           main = "Soil samples"))
# p1
# ggsave(filename = file.path(merged_path, "rarecurve_soil_hq.png"),
#        plot = p1,
#        width = 5, height = 5,
#        dpi = 600)
```

3. ASV occupancy-abundance plots

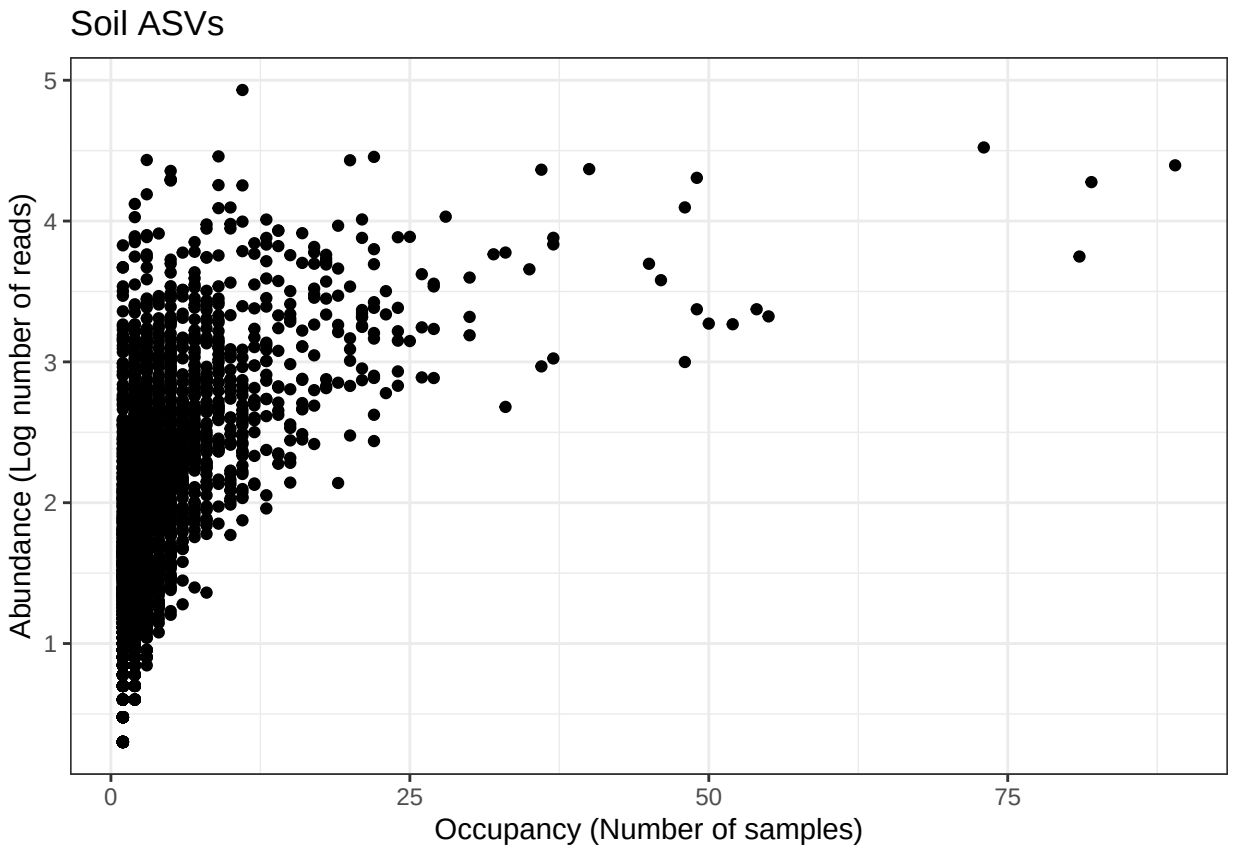
```
create_oa.df <- function(otumat){

  # first, summarize the number of times an OTU was observed in the whole dataset
  abund <- colSums(otumat)

  # second, count the number of samples an OTU appears in
  otumat.pa <- (otumat > 0)*1
  obs <- colSums(otumat.pa)

  df <- data.frame(asv = names(abund), abund, obs)

  p <- ggplot(df, aes(x = obs, y = log(abund, base = 10), label = asv)) +
    geom_point() +
    theme_bw() +
    xlab("Occupancy (Number of samples)") +
    ylab("Abundance (Log number of reads)")

  result<- list(df = df, p = p)
  return(result)
}

# leaf
otumat <- as.matrix(otu_table(samps.l))
oa.l <- create_oa.df(otumat)
oa.l$p +
  ggtitle("Leaf ASVs")
```




```
# ggsave(filename = file.path(merged_path, "oa_leaf.png"),
#        width = 4.5, height = 4.5)

# root
otumat <- as.matrix(otu_table(samps.r))
oa.r <- create_oa.df(otumat)
oa.r$p +
  ggtitle("Root ASVs")
```




```
# ggsave(filename = file.path(merged_path, "oa_root.png"),
#        width = 4.5, height = 4.5)

# soil
otumat <- as.matrix(otu_table(samps.s))
oa.s <- create_oa.df(otumat)
oa.s$p +
  ggtitle("Soil ASVs")
```




```
# ggsave(filename = file.path(merged_path, "oa_soil.png"),
#        width = 4.5, height = 4.5)
```