Chaper 2: ITS Sample Processing Raw Reads

Emily Giroux

09/08/2020

Using package `BiocManager` to install required packages:

```
r <- getOption("repos")
r["CRAN"] <- "http://cran.us.r-project.org"
options(repos = r)

if (!requireNamespace("BiocManager"))
    install.packages("BiocManager")
BiocManager::install()

library("BiocManager")
.cran_packages <- c("cowplot", "data.table", "ggplot2", "knitr", "rprojroot")
.bioc_packages <- c("BiocStyle", "Biostrings", "dada2",
                    "ShortRead")
.inst <- .cran_packages %in% installed.packages()
if(any(!.inst)) {
   install.packages(.cran_packages[!.inst])
}
.inst <- .bioc_packages %in% installed.packages()
if(any(!.inst)) {
  BiocManager::install(.bioc_packages[!.inst], ask = FALSE)
}
```

Load packages into session, and print package versions:

```
sapply(c(.cran_packages, .bioc_packages), require, character.only = TRUE)
```

```
##    cowplot data.table    ggplot2      knitr  rprojroot  BiocStyle Biostrings
##       TRUE       TRUE       TRUE       TRUE       TRUE       TRUE       TRUE
##      dada2  ShortRead
##       TRUE       TRUE
```

Load the saved image from Chapter 1, then save it as a separate image to retain environment data specific to the ITS processing and analysis workflow.

```
load(paste(imageDirPath, startUpImage, sep = ""))
chptImageA <- "ecobiomics_ITS.RData"
save.image(paste(imageDirPath, chptImageA, sep = ""))
```

When re-starting a session, you can quickly load up the image by running the chunk below:

```
sharedPath <- "/isilon/cfia-ottawa-fallowfield/users/girouxeml/PIRL_working_directory/"
analysis <- "ecobiomics/"
sharedPathAn <- paste(sharedPath, analysis, sep = "/")
imageDirPath <- "/home/CFIA-ACIA/girouxeml/GitHub_Repos/r_environments/ecobiomics/"
chptImageA   <- "ecobiomics_ITS.RData"
load(paste(imageDirPath, chptImageA, sep = ""))
```

Create a variable shortcut to the region-specific analysis directory:
In this chapter, we are working with the ITS amplicon samples, so the ITS directory within our main project directory will contain all the region-specific output.

```
region <- "ITS"
sharedPathReg <- paste(sharedPathAn, "/", region, "_analysis/", sep = "")
if(!dir.exists(sharedPathReg)) dir.create(sharedPathReg)

# Make the region-specific metadatable for this chapter the main metadata table:
metadataRegion <- metadataITS
```

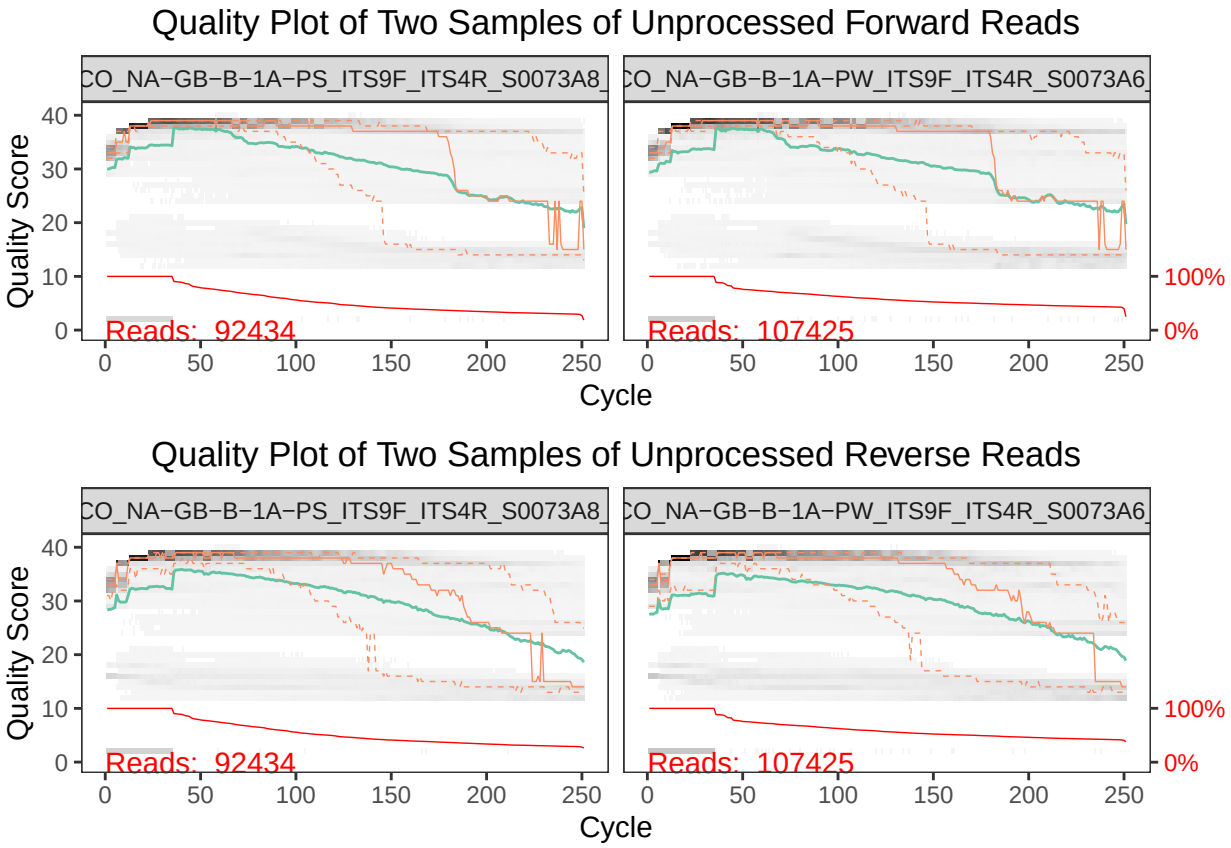
Sample processing follows the steps outlined in Chapter 1

1. ggPlotRaw ITS:

```
library("dada2")
library("ggplot2")
plotRawQFwd <- dada2::plotQualityProfile(paste(metadataRegion$rawFastqPath[1:2],
                                               metadataRegion$rawR1[1:2], sep = "")) +
  ggplot2::ggtitle("Quality Plot of Two Samples of Unprocessed Forward Reads") +
  ggplot2::theme(plot.title = element_text(hjust = 0.5))

plotRawQRev <- dada2::plotQualityProfile(paste(metadataRegion$rawFastqPath[1:2],
                                               metadataRegion$rawR2[1:2], sep = "")) +
  ggplot2::ggtitle("Quality Plot of Two Samples of Unprocessed Reverse Reads") +
  ggplot2::theme(plot.title = element_text(hjust = 0.5))

library("cowplot")
cowplot::plot_grid(plotRawQFwd, plotRawQRev, nrow = 2)
```



Checking for Adapter Sequences
We're going to use Cutadapt to detect and trim off ITS-specific adapter sequences. Cutadapt uses Biostrings, which only recognises the main IUPAC codes. We need to replace any extended IUPAC nucleic acid ambiguity codes to the primary codes. In our ITS primers we see the following ambiguiity codes:
R is for A or G
Y is for C or T
I is from the extended IUPAC codes, and gives a probability order for C, T or G. The closest resemblance to the main IUPAC for I is B.

ITS9F:

5' GAACGCAGCRAAIIGYGA3'

We'll replace the Is with Bs to get:

5' GAACGCAGCRAABBGYGA 3'

No changes are required for the reverse primer as there are no extended ambiguity codes in the primer sequence:
ITS4R:

5' TCCTCCGCTTATTGATATGC 3'

2. Cutadapt. We are replacing the code for SeqPrep with the use of cutadapt and workflow developed by Benjamin Callahan. See page: https://benjjneb.github.io/dada2/ITS_workflow.html

Capture your forward and reverse ITS primers:

```
fwdPrimer <- "GAACGCAGCRAABBGYGA"
revPrimer <- "TCCTCCGCTTATTGATATGC"
```

Create the custom 'AllOrients' function for primer sequences for all possible orientations. This function was created by Benjamin Callahan for the ITS-specific workflow. See: https://benjjneb.github.io/dada2/ITS_workflow.html

```
library("Biostrings")
AllOrients   <- function(primer) {
     require(Biostrings)
     dna     <- Biostrings::DNAString(primer)
     orients <- c(Forward    = dna,
                  Complement = Biostrings::complement(dna),
                  Reverse    = Biostrings::reverse(dna),
                  RevComp    = Biostrings::reverseComplement(dna))
    return(sapply(orients, toString))
}
```

We can now use the custom AllOrients function to generate the primer sequences in all possible orientations in which they may be found:

```
fwdOrients <- AllOrients(fwdPrimer)
revOrients <- AllOrients(revPrimer)
fwdOrients
```

```
             Forward           Complement              Reverse
"GAACGCAGCRAABBGYGA" "CTTGCGTCGYTTVVCRCT" "AGYGBBAARCGACGCAAG"
             RevComp
"TCRCVVTTYGCTGCGTTC"
```

Create the custom 'PrimerHits' function for checking our sequences for all orientations of primer sequences as generated above using the AllOrients function. This function generates a table of counts of the number of reads in which the primer is found. This function was created by Benjamin Callahan for the ITS-specific workflow. See: https://benjjneb.github.io/dada2/ITS_workflow.html

```
library("ShortRead")
PrimerHits <- function(primer, fn) {
    nhits  <- Biostrings::vcountPattern(primer, ShortRead::sread(readFastq(fn)),
                                        fixed = FALSE)
    return(sum(nhits > 0))
}
```

Before checking our ITS sequences for primers with cutadapt, we need to remove those sequences with ambiguous bases. Ambiguous bases (Ns) in the sequencing reads makes accurate mapping of short primer sequences difficult, so we "pre-filter" these sequences so that we only remove those with Ns and perform no other fitlering.

```
library("dada2")
# Path to the processed fastq file directory:
processedFastq <- paste(sharedPathReg, "processedFQ", sep = "")
if(!dir.exists(processedFastq)) dir.create(processedFastq)

# Path to fastq filtered for ambiguous bases (Ns):
filtNsPath <- file.path(processedFastq, "A_removedAmbiguous")
if(!dir.exists(filtNsPath)) dir.create(filtNsPath)

# Define path and file names:
fwd   <- paste(metadataRegion$rawFastqPath, metadataRegion$rawR1, sep = "")
filtF <- file.path(filtNsPath,
                   paste(metadataRegion$LibraryName, "_F.fastq.gz", sep = ""))
rev   <- paste(metadataRegion$rawFastqPath, metadataRegion$rawR2, sep = "")
filtR <- file.path(filtNsPath,
                   paste(metadataRegion$LibraryName, "_R.fastq.gz", sep = ""))

# Run DADA2's filterAndTrim function to remove sequences with ambiguous bases:
```

```
dada2::filterAndTrim(fwd, filtF, rev, filtR, maxN = 0,
                     multithread = TRUE, verbose = TRUE)
```

We can now check the N-filtered sequences of just one sample set for primer hits:

```
library("ShortRead")
rbind(FWD.ForwardReads = sapply(fwdOrients, PrimerHits, fn = filtF[1]),
      FWD.ReverseReads = sapply(fwdOrients, PrimerHits, fn = filtR[1]),
      REV.ForwardReads = sapply(revOrients, PrimerHits, fn = filtF[1]),
      REV.ReverseReads = sapply(revOrients, PrimerHits, fn = filtR[1]))
```

```
                  Forward Complement Reverse RevComp
FWD.ForwardReads   26352          0       0     810
FWD.ReverseReads     889          0       0   13325
REV.ForwardReads    2812          0       0   22144
REV.ReverseReads   36808          0       0    2230
```

As Expected, the vast majority of forward primer is found in its forward orientation, and in some of the reverse reads in its reverse-complement orientation (due to read-through when the ITS region is short). Similarly, the reverse primer is found with its expected orientations. There are some where the primers appear in incorrect orientations and I need to better understand this.

Set up the paths and fastq file input and output names in preparation for running cutadapt. **Note:** You may need to install cutadapt locally if it is not installed system wide. I chose to use conda to install it locally:

```
cutadapt <- "/home/CFIA-ACIA/girouxeml/prog/anaconda3/envs/bio/bin/cutadapt"
```

We can use the `system2` call to run shell commands directly from R

```
system2(cutadapt, args = "--help")
```

Create a directory for the cutadapted fastq files, only if it doesn't already exist. Also create the output filenames for the cutadapt-ed fastq files.

```
cutAdaptDir <- file.path(processedFastq, "B_cutadapt", sep = "")
if(!dir.exists(cutAdaptDir)) dir.create(cutAdaptDir)

cutFqs <- paste(cutAdaptDir,
                metadataRegion$LibraryName, "_Fcut.fastq.gz", sep = "")
cutRqs <- paste(cutAdaptDir,
                metadataRegion$LibraryName, "_Rcut.fastq.gz", sep = "")
```

Now we can proceed to using cutadapt to remove primer sequences at the ends of our reads. We'll use dada2:::rc()

An aside: The ::: is one of 2 possible namespace operators in R. This triple-colon operator acts like a double-colon operator (which selects definitions from a particular namespace), AND allows access to hidden objects. In this command, we are specifying that we want to use the `rc` function that is from the dada2 package, not the `rc` function that may also exist in base R or other packages we possibly loaded. See this page for more on namespace operators in R: http://r-pkgs.had.co.nz/namespace.html

In the following, we use dada2:::rc to get the reverse complement of the primer sequences. The idea is that it can be used to compare each sequence and its reverse complement to a reference sequence(s) in the right orientation, and then choose the orientation that minimizes that distance. See page: https://github.com/benjjneb/dada2/issues/451

Cutadapt flag parameter explanations:

Parameter	Definition
-g:	Regular 5' adapter/primer sequence on forward reads
-a:	Regular 3' adapter/primer sequence on forward reads
-G:	Regular 5' adapter/primer sequence on reverse reads
-A:	Regular 3' adapter/primer sequence on reverse reads
-p:	Specify paired-end sequencing reads
-o:	Specify output files
-n:	Number of times to trim when more than one adapter is present in a read

Documentation for fine-tuning use of cutadapt is available at: https://cutadapt.readthedocs.io/en/v1.7.1/guide.html

```
library("dada2")
fwdPrimerRC <- dada2:::rc(fwdPrimer)
revPrimerRC <- dada2:::rc(revPrimer)
# Trim fwdPrimer and the reverse-complement of the revPrimer off forward reads:
fwdFlags <- paste("-g", fwdPrimer, "-a", revPrimerRC)
# Trim revPrimer and the reverse-complement of the fwdPrimer off reverse reads:
revFlags <- paste("-G", revPrimer, "-A", fwdPrimerRC)
# Run Cutadapt
for(i in seq_along(metadataRegion$LibraryName)) {
  system2(cutadapt, args = c(fwdFlags, revFlags, "-n", 2,
                             "-o", cutFqs[i], "-p", cutRqs[i],
                             filtF[i], filtR[i]))
}
```

Note: There is a lot of output generated to the console when running cutadapt. One warning we may see often is about detection of incomplete adapter sequences:

```
WARNING:
One or more of your adapter sequences may be incomplete.
Please see the detailed output above.
```

Because our DNA fragments are generated from amplicon sequences and are not random, we can ignore this warning.

As a sanity check, we will count the presence of primers in the first cutadapt-ed sample:

```
rbind(FWD.ForwardReads = sapply(fwdOrients, PrimerHits, fn = cutFqs[1]),
      FWD.ReverseReads = sapply(fwdOrients, PrimerHits, fn = cutRqs[1]),
      REV.ForwardReads = sapply(revOrients, PrimerHits, fn = cutFqs[1]),
      REV.ReverseReads = sapply(revOrients, PrimerHits, fn = cutRqs[1]))
```

```
                 Forward Complement Reverse RevComp
FWD.ForwardReads       0          0       0     810
FWD.ReverseReads     889          0       0       0
REV.ForwardReads    2811          0       0       0
REV.ReverseReads       0          0       0    2229
```

3. filterAndTrimming. Raw read processing.
   If we're going to spend anytime optimizing steps during procesing, it should be on filterring and trimming. Likewise, while we may see a big loss of reads in this step, outside of this step we should see no major loss of reads.

Reads less than 60 bp in length will never overlap for this region and are most likely junk reads, so we will be filtering these reads out.
We can use the quality plots of the raw reads to guide our trimming and filtering approach. In the plots previously generated we saw that the quality of the forwards reads was much higher than the reverse reads, which is to be expected for Illumina data. Forward read quality was relatively adequate throughout read length. Reverse read quality dropped off much sooner, with much lower quality after the 180-200 bp mark. We have the option to truncate our reads in this step, but we need to keep in mind our expected amplicon length for successful merging later on, which depends on a 20-bp overlap between forward and reverse reads.
We combine these trimming parameters with standard filtering parameters, the most important being the enforcement of a maximum of 2 expected errors per-read (Edgar and Flyvbjerg 2015). Trimming and filtering is performed on paired reads jointly, i.e. both reads must pass the filter for the pair to pass.
When filtering - even if I remove the option to discard reads with >2 error rate, I may still lose samples where no reads remain after filtering. Given the importance stressed by the referenced paper by Edgar and Flyvbjerg 2015 for filtering reads with higher error rates, I kept it, but increased the allowed errors to 3 per read, instead of 2. We need to accept that some samples will be lost due to poor sequencing data. We will need to update the metadata table so that the lost sets are removed in a later step.

There is a nice preamble about processing ITS amplicon data at: https://benjjneb.github.io/dada2/ITS_workflow.html
Unlike the 16S rRNA gene, the ITS region is highly variable in length. The commonly amplified ITS1 and ITS2 regions range from 200 - 600 bp in length. This length variation is biological, not technical, and arises from the high rates of insertions and deletions in the evolution of this less conserved gene region.

The length variation of the ITS region has significant consequences for the filtering and trimming steps of the standard DADA2 workflow. First, truncation to a fixed length is no longer appropriate, as that approach remove real ITS variants with lengths shorter than the truncation length. Second, primer removal is complicated by the possibility of some, but not all, reads extending into the opposite primer when the amplified ITS region is shorter than the read length.

Because both scenarios pictured above typically occur within the same ITS dataset, a critical addition to ITS worfklows is the removal of primers on the forward and reverse reads, in a way that accounts for the possibility of read-through into the opposite primer. This is true even if the amplicon sequencing strategy doesn't include the primers, as read-through into the opposite primer can still occur.

In the standard 16S workflow, it is generally possible to remove primers (when included on the reads) via the trimLeft parameter as they only appear at the start of the reads and have a fixed length. However, the more complex read-through scenarios that are encountered when sequencing the highly-length-variable ITS region require the use of external tools. Here we present the cutadapt tool for removal of primers from the ITS amplicon sequencing data.

```
# Path to the final processed fastq file directory:
filtPathFinal <- file.path(processedFastq, "C_finalProcessed")
if(!dir.exists(filtPathFinal)) dir.create(filtPathFinal)

# Create output filenames for the final filtered and trimmed fastq files:
filtFwd <- paste(filtPathFinal, "/",
                 metadataRegion$LibraryName, "_F_filt.fastq.gz", sep = "")
filtRev <- paste(filtPathFinal, "/",
                 metadataRegion$LibraryName, "_R_filt.fastq.gz", sep = "")
```

For this dataset, we will use standard filtering paraments: maxN=0 (DADA2 requires sequences contain no Ns), truncQ = 2, rm.phix = TRUE and maxEE=2. The maxEE parameter sets the maximum number of "expected errors" allowed in a read, which is a better filter than simply averaging quality scores. Note: We enforce a minLen here, to get rid of spurious very low-length sequences.

Run filterAndTrim using cutadapt-ed fastq files as input:

```
library("dada2")
trimOut <- dada2::filterAndTrim(cutFqs, filtFwd, cutRqs, filtRev,
                                maxN = 0, maxEE = c(3,3), truncQ = 2,
                                minLen = 50, rm.phix = TRUE, compress = TRUE,
                                matchIDs = TRUE, multithread = TRUE, verbose = TRUE)
```

Save the workspace image right after this step so it doesn't have to be repeated if R accidentally shuts down:

```
save.image(paste(imageDirPath, chptImageA, sep = ""))
```

Trimming and filtering parameters explanations:

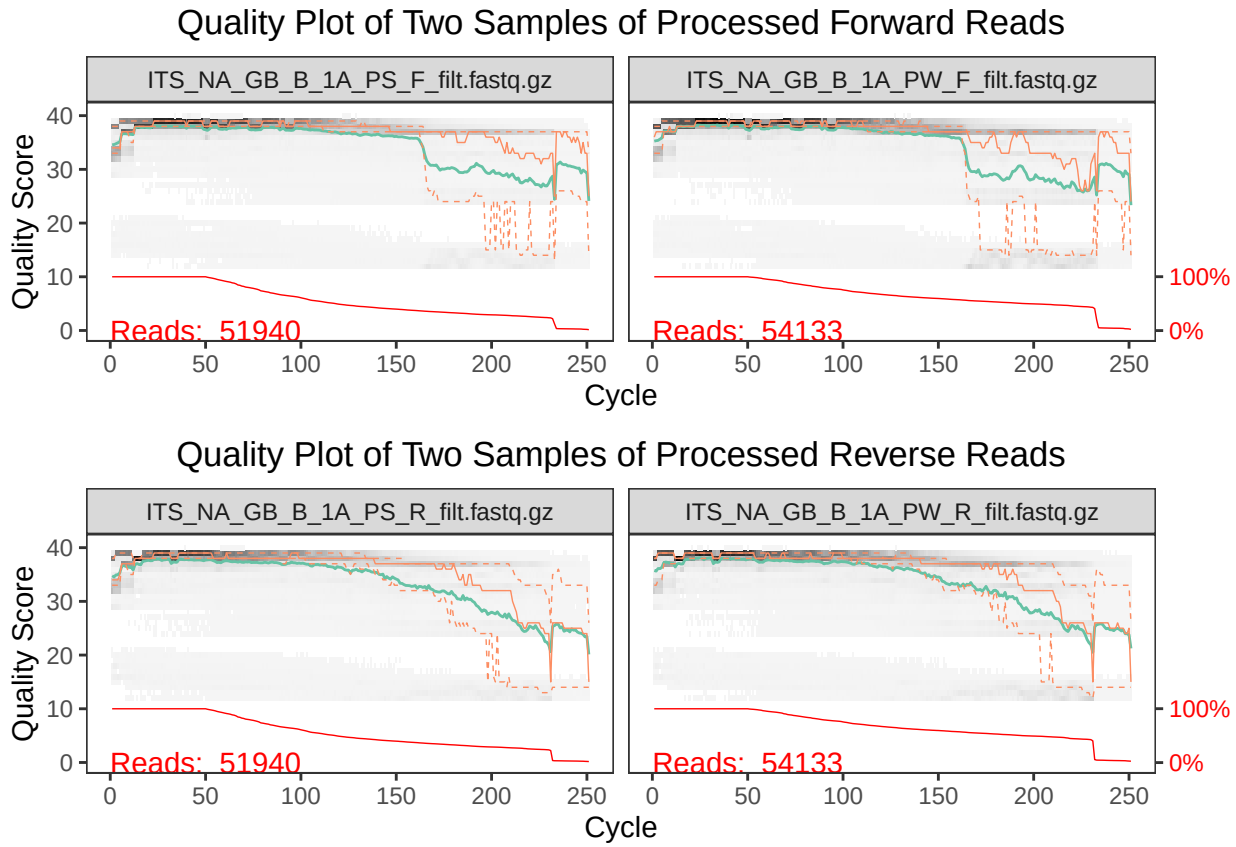
Parameter	Definition
minLen:	Remove reads with length less than minLen. minLen is enforced after trimming and truncation.
minQ:	After truncation, reads that contain a quality score less than minQ will be discarded.
truncQ:	Truncate reads at the first instance of a quality score less than or equal to truncQ.
maxN:	After truncation, sequences with more than maxN Ns will be discarded.
maxEE:	After truncation, reads with higher than maxEE "expected errors" will be discarded. Expected errors are calculated from the nominal definition of the quality score: EE = sum(10^(-Q/10))
rm.phix:	If TRUE, discard reads that match against the phiX genome.
matchIDs:	Whether to enforce matching between the id-line sequence identifiers of the forward and reverse fastq files. If TRUE, only paired reads that share id fields are output.
compress:	Ouput fastq files are gzipped.
multithread:	If TRUE, input files are filtered in parallel via mclapply.
verbose:	Whether to output status messages.

For the full list of available parameters and permitted parameter values see the `filterAndTrim` function in the DADA2 manual page:
https://rdrr.io/bioc/dada2/man/filterAndTrim.html

4. ggPlotsProcessed. Let's look at the quality plots of our filtered and processed reads for our first 2 samples to see the effects of our trimming parameters:

```
library("dada2")
library("ggplot2")
plotQfwd <- dada2::plotQualityProfile(paste(filtPathFinal, "/",
                                            metadataRegion$LibraryName[1:2],
                                            "_F_filt.fastq.gz", sep = "")) +
  ggplot2::ggtitle("Quality Plot of Two Samples of Processed Forward Reads") +
  ggplot2::theme(plot.title = element_text(hjust = 0.5))
plotQrev <- dada2::plotQualityProfile(paste(filtPathFinal, "/",
                                            metadataRegion$LibraryName[1:2],
                                            "_R_filt.fastq.gz", sep = "")) +
  ggplot2::ggtitle("Quality Plot of Two Samples of Processed Reverse Reads") +
  ggplot2::theme(plot.title = element_text(hjust = 0.5))
```

```
library("cowplot")
cowplot::plot_grid(plotQfwd, plotQrev, nrow = 2)
```



5. keepTrimmed. Update metadata so that samples that did not have any reads that passed filtering are removed from further analysis, to avoid downstream processing errors.

We can take a brief look at the summary or reads in and out for the first 6 samples:

```
head(trimOut)
```

```
                   reads.in reads.out
ITS_NA_GB_B_1A_PS     83207     51940
ITS_NA_GB_B_1A_PW     94513     54133
ITS_NA_GB_B_1B_NS      9658      7828
ITS_NA_GB_B_1B_PS     15334     10003
ITS_NA_GB_B_1B_PW     44202     30674
ITS_NA_GB_B_2A_NS_B   62190     39752
```

We can order the samples by reads remaining after processing with:

```
head(trimOut[order(-trimOut[,2]),])
```

```
                     reads.in reads.out
ITS_NA_GB_S_5_FD_A     239996    206892
ITS_NA_GB_B_5A_PS      294071    196569
ITS_NA_GB_B_5B_PS      285091    195070
ITS_NA_GB_B_6A_PS      287854    192383
ITS_NA_GB_S_HB1_PW_A   255326    166119
ITS_NA_GB_B_4B_PW      217660    125738
```

The row names we see have retained the fastq file name of the input forward reads, yet the output sums are for both forward and reverse reads. Let's remove the file suffix so that the rownames will apply to both forward and reverse reads.

```
rownames(trimOut) <- gsub("_Fcut.fastq.gz", "", rownames(trimOut))
```

As a precaution, we still check for samples with no remaining reads and update the metadata table for our ITS samples using those results. If you receive an error during the dereplication step later on: `Error in open.connection(con, "rb") : cannot open the connection`, you need to update the files so that those that didn't pass filtering aren't attempted. See issue at: https://github.com/benjjneb/dada2/issues/375

```
keepTrim <- trimOut[,"reads.out"] > 20 # Or other cutoff

filtFs <- file.path(filtPathFinal,
                    paste(metadataRegion$LibraryName,
                          "_F_filt.fastq.gz", sep = ""))[keepTrim]
filtRs <- file.path(filtPathFinal,
                    paste(metadataRegion$LibraryName,
                          "_R_filt.fastq.gz", sep = ""))[keepTrim]

filtNames <- basename(filtFs)[keepTrim]
filtNames <- gsub("_F_filt.fastq.gz", "", filtNames)
```

Run the `keepTrimmed2` chunk. We've updated our list of fastq file names, but we also need to update our metadata table so that rows that had read pairs where a direction no longer had reads after filtering are removed from the metadata table.
Note: With `data.table` and `keys` look-up, `.()` is an alias to `list()`, which is the `filtNames` we set in the previous chunk.

```
library("data.table")
data.table::setkey(metadataRegion, LibraryName)
metadatafilt <- metadataRegion[.(filtNames)]
metadatafilt$filtFwd <- paste(filtPathFinal, "/", metadatafilt$LibraryName,
```

```
                                "_F_filt.fastq.gz", sep = "")
metadatafilt$filtRev <- paste(filtPathFinal, "/", metadatafilt$LibraryName,
                                "_R_filt.fastq.gz", sep = "")
```

6. splitRunsMetadata. No need for this step since all ITS sequencing libraries were sequenced on the same run.

```
library("data.table")
data.table::setkey(metadatafilt, "SeqPlate")
run <- unique(metadatafilt$SeqPlate)
metadatafiltPlate1 <- metadatafilt[run]
```

```
save.image(paste(imageDirPath, chptImageA, sep = ""))
```