**Overview of analysis**

This protocol is written for users who wish to analyze sequence data with the *dbcAmplicons* analysis pipeline. Raw sequence files from the MiSeq will need to be configured by the miseq\_pipeline.py script to fit the expectations of *dbcAmplicons* before the analysis can begin. The sample sheet from the run is reconfigured to support the output of four individual reads (Read 1, Index 1, Index 2, Read 2) and CASAVA (the Illumina-specific post-run processing software) is then run using the reconfigured sample sheet. The output from this step is four individual reads ready for analysis using *dbcAmplicons*.

**Configuring raw MiSeq data for input into the pipeline**

The following items are necessary for the proper execution of the pipeline:

* Python version 2.X (this protocol written for 2.7.5)
* A properly formatted sample sheet named SampleSheet.csv and the RunInfo.xml sheet
* Raw data generated from sequencing, before CASAVA is run
* Access and read permissions to the raw data
* Installed and executable CASAVA (this protocol written for 1.8)

Navigate to the desired directory and execute the miseq\_pipeline.py script, giving the first parameter as the raw data from the run and the second parameter as the name of the desired output directory. The sample sheet from the run must be named SampleSheet.csv and it must be in the run output folder, along with the RunInfo.xml sheet.

An example command (written in the shell terminal) run in the desired directory is as follows:

**python miseq\_pipeline.py 150325\_M01380\_0037\_00000-AEMV5 Output\_directory**

where 'python' is the call to execute the script, '150325\_M01380\_0037\_000000000-AEMV5' is the name of the directory that holds the MiSeq run information, and 'Output\_directory' is the prefix for the desired output. The script will generate and run CASAVA within the output directory until completion. Once the process is complete, the reads are ready for analysis.

**Analyzing dual-barcoded amplicon sequence data using *dbcAmplicons***

This package was developed for command line usage to analyze high volumes of microbial 16S, LSU, and ITS amplicons from high-throughput sequencing platforms. All components of the package are designed to flow directly into the next with no intermediate formatting steps required. The following protocol is a detailed vignette that uses sample data provided within the *dbcAmplicons* package. A full executable test script (test\_dbAmplicons.sh) is located within the **tests** directory if the user want to ensure all the major components are installed without going through the tutorial.

*Installation*

Minimum requirements for the *dbcAmplicons* package: Python 2.X is required to run any and all of the

elements in the package. Some applications within the package rely on **flash2** or the **RDP classifier** to complete, so the user must download and install the external tools. Root access is recommended for all of the installations.

All executable scripts are located in the github repository <https://github.com/msettles/dbcAmplicons.git>

The **flash2** executable and installation instructions can be found at <https://github.com/dstreett/FLASH2.git>. The **RDP** **classifier** may be found at <http://sourceforge.net/projects/rdp-classifier/>. The user must install the classifier.jar and its dependencies in the RDPtools package.

*Mac OSX*

Mac users should first download and install **Xcode** and then download the app via a terminal by:

git clone https://github.com/msettles/dbcAmplicons.git

cd dbcAmplicons

sudo python setup.py install

dbcAmplicons

If installation was successful, this will bring up the usage arguments.

*Linux*

For Linux users, some python dependencies may be required before the installation is successful. Root privileges are also required to install these dependencies. To install, open a terminal and type:

sudo apt-get install python-dev

Type the sudo password at the prompt. Then, download the app via a terminal by:

git clone https://github.com/msettles/dbcAmplicons.git

cd dbcAmplicons

sudo python setup.py install

dbcAmplicons

If installation was successful, this will bring up the usage arguments.

***dbcAmplicons******preprocess***

*dbcAmplicons* *preprocess* takes four fastq files (Read 1, Index 1, Read 2, and Index 2), the assigned barcode indices for each sample, the sequences of the template-specific primers, and the individual sample names along with an assigned project folder. *dbcAmplicons preprocess* will then assign each read in the fastq file to a specific project according to both barcode and primer sequences, and return 2 fastq files with all of the identifying information in the read header. If a project directory is specified, *dbcAmplicons preprocess* will create a directory with the indicated output prefix.

At a minimum, *dbcAmplicons* *preprocess* requires a properly formatted reference barcode sequence file to demultiplex raw sequences. For maximum separation and identification of the reads, the following three inputs should be used, all in a tab-separated text format:

1. Reference barcode sequences file. This file should have three columns: the first column is the user-defined barcode name; the second column is the Read 2 sequence, also referred to as Index 1 on the P7 end of the sequence; and the third column is the Read 3 sequence, also known as Index 2 on the P5 end of the sequence. Users should note that Index 2 produces a sequence that is in reverse complement orientation relative to the p7 adapter end, so the user must be aware of the orientation of their barcodes before they attempt to use *dbcAmplicons.* Example format is below:

#BarcodeID Read2RC Read3

Alpha1 TAAGGCGA TAGATCGC

Alpha2 TAAGGCGA CTCTCTAT

Alpha3 TAAGGCGA TATCCTCT

Alpha4 TAAGGCGA AGAGTAGA

Alpha5 TAAGGCGA GTAAGGAG

Alpha6 TAAGGCGA ACTGCATA

2. Reference primer sequences file. Similar to the barcode reference file, it should also be a tab-separated text file. There are 4 necessary columns in the primer reference sheet. *dbcAmplicons* will ignore any columns that are outside of the 4 required columns, so the user may include additional information to each sample if necessary. Column 1 (Read) indicates the orientation of the primer. The user can denote this by the Illumina-specific adapter (P7 or P5); alternatively, the user may use R1, R2, Read1, Read2, F, R, Forward, and Reverse. The “Pair\_ID” column is the user-specified name for the primer pair that was used and must have at least one P5 and P7 combination assigned to it. The “Primer\_ID” column denotes the unique name of each individual primer, and should correspond in orientation with the “Read” column. The “Sequence” column contains the full primer sequence without the CS-tag sequences, but should include spacing nucleotides inserted for sequence diversity (Refer to *dbcAmplicons* Lab Protocol, Section X). The “Sequence” column accepts IUPAC nucleotide codes and treats each variation as a separate primer for searching. Example format is below:

#Read Pair\_ID Primer\_ID Sequence

P5 16S 27F\_1 CGTACGTAGAGTTTGATCCTGGCTTAG

P5 16S 27F\_2 ACGTACGTAGAATTTGATCTTGGTTCAG

P7 16S 534R\_1 CCATTACCGCGGCTGCTGG

P7 16S 534R\_2 GCCATTACCGCGGCTGCTGG

3. Sample metadata file with sample IDs, primer IDs, and barcode IDs. The sample metadata file contains information to separate samples into different projects along with the sample, primer, and barcode identifications. Additional columns outside of the four required columns will be ignored.

SampleID BarcodeID PrimerPairID ProjectID Metadata

Sample1 Alpha1 16S subfolder/match\_16S A

Sample2 Alpha2 16S subfolder/match\_16S B

Sample3 Alpha3 16S subfolder/match\_16S C

Sample4 Alpha4 16S subfolder/match\_16S D

Sample5 Alpha5 16S subfolder/match\_16S A

Sample6 Alpha6 16S subfolder/match\_16S B

Full examples of each of these sheets may be found in the **tests** directory. *dbcAmplicons preprocess* will automatically run a validation of the inputs before any reads are separated. The barcode and primer sheets will be checked against the sample metadata file to make sure that any assigned barcodes and primers exist within the reference files.

The syntax for usage is as follows:

dbcAmplicons preprocess [-h] -B barcodesFile [-d BARCODEDIFF]

[-P primerFile] [-D PRIMERDIFF] [-e PRIMEREND]

[-S sampleFile] [-q MINQ] [-l MINL]

[-b BATCHSIZE] [-O PREFIX] [-U] [-u] [-v]

[-1 read1 [read1 ...]] [-2 read2 [read2 ...]]

[-3 read3 [read3 ...]][-4 read4 [read4 ...]] [--test]

[--keepPrimers] [--debug]

The user will give the paths to each of the input files. Default settings for primer and barcode distances may be changed, but it is not recommended to set them to be less stringent unless the user has good reason to do so. The complete listing of default settings may be found by entering:

dbcAmplicons preprocess -h

The first step within *preprocess* is to validate the user inputs to ensure that there are no repeats of primer/sample combinations, no illegal characters, and no missing entries. The first line in the test\_dbAmplicons.sh script gives the user an example of entries that will cause the process to fail:

Testing dbcAmplicons validate

barcode table length: 864

primer table length P5 Primer (expanded) Sequences:12, P7 Primer (expanded) Sequences:12

ERROR:[validate] TEST pair ID(s) are missing from P5

sample table length: 208, and 5 projects.

ERROR:[validate] barcode Delta 162 not found in barcode table

ERROR:[validate] barcode echo198 not found in barcode table

ERROR:[validate] primer pair BOB not found associated with barcode Echo217, sample 56 in project nomatch

ERROR:[validate] primer pair ITS BOB not found associated with barcode Foxtrot264, sample 88 in project match\_twoprimer

ERROR:[validate] barcode Foxtrot29 not found in barcode table

Failed validation

The validation will point out the errors that failed and will not complete until they have been fixed.

By default, *preprocess* runs validation as the first step in the analysis. If the validation succeeds the user will see *preprocess* writing to standard out. The second step in the test script passes a correctly formatted sample sheet and primer table to the program, and the entire command is below:

dbcAmplicons preprocess -b 15001 -q 10 -l 200 -S sampleLookupTable.txt -B barcodeLookupTable.txt -P primerLookupTable.txt -1 Amplicon\_Raw\_fastq/Test100K\_16S\_R1\_001.fastq.gz Amplicon\_Raw\_fastq/test40k\_R1\_001.fastq.gz -O preprocess/trimL --debug

The above command tells the *preprocess* application that the reads should be processed in batches of 15,001 (-b), the read ends should be trimmed to a minimum quality score of 10 (-q), and that the minimum length of kept sequences should be 200 base pairs or greater (-l). -B, -P, and -S all point to the tab-separated input sheets detailed above that are located within the same directory. The -O option creates a directory named **preprocess** and a sub-directory within that directory with a prefix of **trimL**. The --debug option prints the last command sent to the process to standard out if there is an error that causes the program to fail.

If the command is executed correctly, the program will find the specified files and send updates to standard out after every batchsize of sequences have been processed. If the user is re-running the step without removing any prior outputs or renaming the output paths, the program will give a warning while it overwrites the existing file. The entire process should appear as follows:

dbcAmplicons preprocess -b 15001 -q 10 -l 200 -S sampleLookupTable.txt -B barcodeLookupTable.txt -P primerLookupTable.txt -1 Amplicon\_Raw\_fastq/Test100K\_16S\_R1\_001.fastq.gz Amplicon\_Raw\_fastq/test40k\_R1\_001.fastq.gz -O preprocess/trimL --debug

barcode table length: 864

primer table length P5 Primer Sequences:12, P7 Primer Sequences:12

sample table length: 208, and 5 projects.

processed 15001 total reads, 16806.0 Reads/second, 396 identified reads(2.6%), 14605 unidentified reads

processed 30002 total reads, 16394.0 Reads/second, 859 identified reads(2.9%), 29143 unidentified reads

processed 45003 total reads, 13762.0 Reads/second, 3269 identified reads(7.3%), 41734 unidentified reads

processed 60004 total reads, 10670.0 Reads/second, 10327 identified reads(17.2%), 49677 unidentified reads

processed 75005 total reads, 9309.0 Reads/second, 17636 identified reads(23.5%), 57369 unidentified reads

processed 90006 total reads, 8539.0 Reads/second, 25148 identified reads(27.9%), 64858 unidentified reads

processed 105007 total reads, 8048.0 Reads/second, 32937 identified reads(31.4%), 72070 unidentified reads

processed 120008 total reads, 7686.0 Reads/second, 41002 identified reads(34.2%), 79006 unidentified reads

processed 135009 total reads, 7426.0 Reads/second, 49002 identified reads(36.3%), 86007 unidentified reads

processed 140000 total reads, 7339.0 Reads/second, 51744 identified reads(37.0%), 88256 unidentified reads

140000 reads processed in 0.32 minutes, 51744 (37.0%) identified

17427 reads (12.4% of total run) found for project match\_twoprimersecond

21570 reads (15.4% of total run) found for project match\_twoprimer

1436 reads (1.0% of total run) found for project subfolder/match\_16S

11311 reads (8.1% of total run) found for project match\_wildcard

0 reads (0.0% of total run) found for project nomatch

Cleaning up.

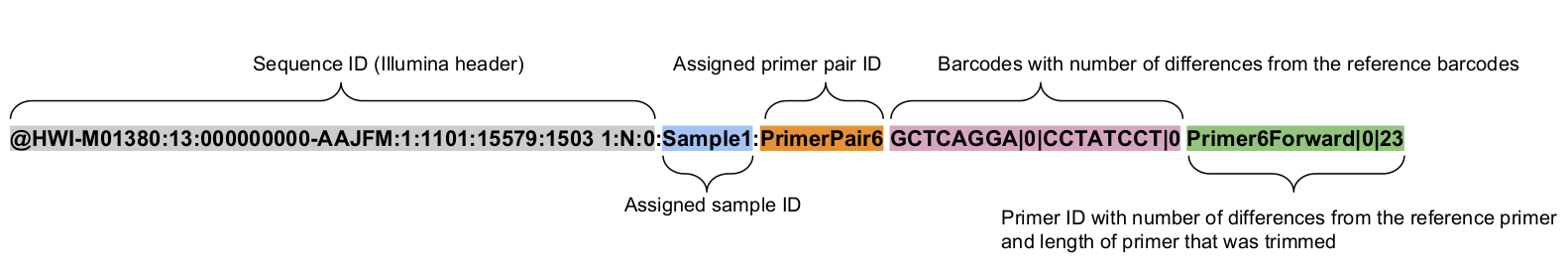
A summary of the amount of reads and the proportion of the full run found for each specified project

will be generated. The user should check the reported proportions carefully to ensure they conform to expectations. In this test case, there were five specified projects, and reads were assigned to only four projects.

The standard final outputs of *preprocess* include a text file listing the barcodes that were identified and two fastq files for each project that was specified. Each project specified within the sample sheet will have an output prefix or directory created. For the above command, a new **trimL** directory will appear within the **preprocess** directory. The layout of the **trimL** directory will appear as follows:

Identified\_Barcodes.txt match\_twoprimer\_R2.fastq.gz match\_twoprimersecond\_R2.fastq.gz match\_wildcard\_R2.fastq.gz match\_twoprimer\_R1.fastq.gz match\_twoprimersecond\_R1.fastq.gz match\_wildcard\_R1.fastq.gz subfolder

The sequence identifications will have been added to the header as follows:



The first part of the sequence read will remain the same, with the Illumina-specific header denoting the read ID, the machine, and the cluster location from which it originated. d*bcAmplicons* will add the assigned sample name, the primer pair name, and the two identified barcodes with the number of sequence differences from the reference barcodes. Lastly, the identified primer name will also be appended to the sequence header, with two numbers that indicate the number of mismatches from the identification and also the length of the primer that was trimmed. In our test example, the first lines of the match\_twoprimer\_R1.fastq.gz will appear as below:

@HWI-M01380:41:000000000-A5H69:1:1101:20694:1535 1:N:0:109:16S TCCTCTAC|0|AGGCGAAG|0 27F\_YM2|0|23

ATTGAACGCTGGCGGCATGCCTTACACATGCAAGTCGAACGGTAACGCGGGGCAACCTGGCGACAAGTGGCGAACGGGTGAGTAATGCATCGGAACGTGCCCAGTAGTGGGGGATAGCCCGGCGAAAGCCGGATTAATACCGCATACGACCTAAGGGTGAAAGCGGGGGATCGCAAGACCTCGCGCTATTGGAGCGGCCGATGTCAGATTAGGTAGTTGGTGGGGTAAAGGCCTACCAAGCCTACGATCTGTAGCTGGTCTGAGAGGGCGACCAGCCC

+

CGGFGGGGGGCFGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGDGGGGGGGGGGGGGGGGGGFGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGDGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGFGFGEG\*8CGGGFGFF=CGGGGGGDGC?FGFFGFDGFFFFFFD3>DFFGFFF?GFFFFFFF@FFFFF?FBBFADF?<FA?FAF?265:?0:99B>9?BF3

@HWI-M01380:41:000000000-A5H69:1:1101:18536:1551 1:N:0:106:16S TCCTCTAC|0|CTAGAGGC|1 27F\_YM2|0|23

GATGAACGCCGGCGGTGTGCCTAATACATGCAAGTCGAACGCGTTGACCCAACTGATTGAACGTGCTTGCACGGACTTGACGTTGGTTTACCAACGAGTGGCGGACGGGTGAGTAACACGTAGGTAACCTGCCCCAAAGCGGGGGATAACATTTGGAAACAGATGCTAATACCGCATAACAGTTTGGATCGCATGATTCAAACTTAAAAGATGGTTTCGGCTATCACTTTGGGATGGACCTGCGGCGCATTAGCTTGTTGGT

+

GGFGGGGFEGGGGGGGGFCGGGFFGDECGGGF9,EFGGDFGEGFFGFCFG8<FCEAFCCEFGGGCFGGFG<FGG7FFGF,E?CGGDGGDFACFFGG7CCEFGEFCFG7:CCG7FCFGFFGGF<<FCEBCFFEGGGE7FGCFEEGG\*<+9CFDEC9C+?+:CC6@CFFG9CFGGGGGGGFGGFCGGGECEG5EEGD+:EGGFFGGC?FC7C57\*\*9DFFFGDGDFF@FF>?@D@>687<6)>)7)86:>6:?B4?A?:793F?

@HWI-M01380:41:000000000-A5H69:1:1101:17169:1564 1:N:0:109:16S TCCTCTAC|0|AGGCGAAG|0 27F\_YM4|0|25

AACGAACGCTGGCGGCAGGCTTAACACATGCAAGTCGAACGGTTGTAGCAATACAGCAGTGGCGCACGGGTGAGTAACGCGTGGATATCTGCCTTTTGGTTCGGAATAACTCCGGGAAACTGGAGCTAATACCGGATGGTTCCTACGGGATAAAGATTTATCGCCAAAAGATGAGTCCGCGTACGATTAGCTAGTTGGTGGGGTAATGGCCCACCAAGGCGACGATCGTTAGCTGGTCTGAGAGGACGATCAGCCACACTGGGACTGAGACAC

Another output from *dbcAmplicons preprocess* is the Identified\_Barcodes.txt file. This file is a summary table listing the number of reads found for each barcode and primer combination. From our test example, the first 10 lines of the file show the first 10 barcodes identified and two primers:

Barcode 16S ITS None

Delta146 30 0 0

Delta147 37 0 0

Delta148 30 0 0

Delta149 32 0 0

Delta150 28 0 0

Delta151 17 0 0

Delta152 29 0 0

Delta153 39 0 0

Delta154 22 0 0

Delta155 77 0 0

Only the 16S primers were matched to corresponding reads. Because this test set was only amplified using 16S primers, we do not expect for the ITS primers to match and the results confirm our expectations.

***dbcAmplicons join***

The next step in analysis is to overlap the reads to create longer sequences, which are more useful than short sequences for classification purposes. *dbcAmplicons join* is the step that requires the **flash2** executable in the user path for overlapping. The process will check for the executable and if it is not found the user will be prompted to download and install from https://github.com/dstreett/FLASH2.git. When the reads are overlapped *dbcAmplicons join* will automatically discard reads that have extremely low quality scores, which indicates the presence of primer dimers that are not biologically relevant. Usage is as follows:

dbcAmplicons join [-h] [-O PREFIX] [-u] [-x NUM] [-t NTHREADS]

[-v] [-1 read1] [-2 read2]

The minimum required inputs are the paths to the preprocessed read 1 and read 2 that the user wants to overlap. Default settings are recommended, although if the user is overlapping very short reads (< 100bp) the maximum expected length of the resulting amplicon many need to be changed. There may be cases where the overlapping criteria are too stringent, especially if there are large amounts of errors on the ends that prevent overlaps. The user must determine which settings would be appropriate in that case. To overlap the reads from the test data set, navigate to **tests** directory and execute the command:

dbcAmplicons join -t 4 -x 0.25

-1 preprocess/trimL/match\_twoprimer\_R1.fastq.gz -O join/match\_twoprimer

Similar to the *preprocess* output, *join* will return information to standard out giving summaries of the parameters and overlapping success.

Using Flash2 version:v2.2.00

Parameters:

Min\_overlap:10

Min\_overlap\_outie:35

Max\_overlap:600

Max\_mismatch\_density:0.250000

Allow\_"outie"\_pairs:true

Cap\_mismatch\_quals:false

Combiner\_threads:4

Input\_format:FASTQ, phred\_offset=33

Output\_format:FASTQ, phred\_offset=33, gzip

Starting 4 combiner threads

Processed 21570 read pairs

Output:

Total\_pairs:21570

Discarded\_pairs:0

Percent\_Discarded:0.00%

Combined\_pairs:16175

Innie\_pairs:16175 (100.00% of combined)

Outie\_pairs:0 (0.00% of combined)

Uncombined\_pairs:5395

Percent\_combined:74.99%

The above output shows that none of the reads were discarded due to poor quality (indicating the presence of primer dimers) and almost 75% of the reads were overlapped.

In the **join** directory, the *join* process outputs consist of 3 fastq files; one of the overlapped sequences (named extendedFrags.fastq) and two of the unpaired sequences (named notCombined\_1/2). The process will also generate histograms showing the size and frequency of the overlapped sequences.

match\_twoprimer.extendedFrags.fastq.gz match\_twoprimer.histogram match\_twoprimer.hist.outie match\_twoprimer.hist match\_twoprimer.histogram.innie match\_twoprimer.hist.innie

match\_twoprimer.notCombined\_2.fastq.gz

match\_twoprimer.notCombined\_1.fastq.gz match\_twoprimer.histogram.outie

The size range we find in the match\_twoprimer.histogram corresponds to the size of the amplicons we generated with the 16S primers with the vast majority within 20 bp of each other.

---

449

450 \*\*\*\*\*\*\*

451

452 \*\*\*\*\*

453

454

455

456 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

457

458

459

460 \*\*\*\*\*\*\*\*\*

461 \*

462 \*

463 \*

464 \*\*

465 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

466 \*\*\*

467 \*\*

468 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

469 \*\*\*\*

470 \*\*\*

---

If the user finds large quantities of unexpected sequence lengths there may be contamination issues within the sample, and classifying the reads will likely give an erroneous result. However, if the overlapped amplicons are within the expected size range the user can proceed to the next step in the pipeline.

***dbcAmplicons classify***

Once the reads have been overlapped, they can be classified by 16S, ITS, or fungal LSU sequence using the latest version of the RDP classifier. The *classify* process will check for the RDP classifier similarly to how the *join* process will check for the flash2 executable. If the “classifier.jar” file is not found somewhere on the path or is incorrectly specified within the command the program will return an error. Sequences are classified to their closest taxonomic rank that meets a 50% bootstrap match. Usage is as follows:

dbcAmplicons classify [-h] [-b BATCHSIZE] [-q MINQ] [-l MINL]

[-p PROCS] --rdpPath PATH [-g <arg>] [-t TRAIN]

[-O outputPrefix] [-1 read1 [read1 ...]]

[-2 read2 [read2 ...]] [-U single [single ...]][-v]

[--debug]

Within the **tests** directory, execute the *classify* command:

dbcAmplicons classify -b 7500 -q 10 -l 200 -O join/classify

-U join/match\_twoprimer.extendedFrags.fastq.gz

--rdpPath /PATH/TO/RDP\_CLASSIFIER

-p 4 -1 join/match\_twoprimer.notCombined\_1.fastq.gz

-2 join/match\_twoprimer.notCombined\_2.fastq.gz

The above command processes reads in batches of 7500 (-b), trimming the ends to a minimum q-score of 10 (-q) and only keeping reads above 200 bp (-l). The user will determine if using the non-overlapped sequences to identify organisms is appropriate given the expectations of the sample data and the results up to that point.

Depending on the speed of the user's system and how many processors are available, the classification may take a significant amount of time. The RDP process to standard out will appear similar to the lines below:

Starting rdp for file join/classify.7500.fasta

Finished processing join/classify.7500.fasta in 1.43 minutes

Starting rdp for file join/classify.15000.fasta

Finished processing join/classify.15000.fasta in 1.49 minutes

Starting rdp for file join/classify.16175.fasta

Finished processing join/classify.16175.fasta in 0.3 minutes

Starting rdp for file join/classify.21570.fasta

Finished processing join/classify.21570.fasta in 1.1 minutes

Combining temporary files

21570 reads processed in 4.34 minutes

Cleaning up.

The output from the *classify* process is a single fixrank file that can then be processed with *dbcAmplicons abundance.* The file lists each sequence ID on a single line, along with the lowest level of classification that matched the minimum (or default) bootstrap level. Three lines of the classify.fixrank file are shown below, with all three sequences classified to the genus level:

HWI-M01380:41:000000000-A5H69:1:1101:19971:1659|106:16S

Bacteria domain 1.0 Firmicutes phylum 1.0 Bacilli class 1.0

Lactobacillales order 1.0 Lactobacillaceae family 1.0

Lactobacillus genus 1.0

HWI-M01380:41:000000000-A5H69:1:1101:14038:1660|109:16S

Bacteria domain 1.0 "Chloroflexi" phylum 0.98 Caldilineae class 0.98 Caldilineales order 0.98 Caldilineaceae family 0.98

Caldilinea genus 0.69

HWI-M01380:41:000000000-A5H69:1:1101:12610:1953|105:16S

Bacteria domain 1.0 "Proteobacteria" phylum 0.77 Deltaproteobacteria class 0.72 Myxococcales order 0.3 Polyangiaceae family 0.28 Byssovorax genus 0.19

***dbcAmplicons abundance***

Within the *abundance* process, the classify.fixrank file is reformatted to produce three outputs that parse the classified data according to user-specified criteria. Usage is as follows:

dbcAmplicons abundance [-h] [-r <arg>] [-t VALUE] [-m VALUE]

[-M VALUE][-S FILE] [-O FILE\_PREFIX] -F FILE [FILE ...][-b] [-v] [--debug]

The user will determine the lowest classification levels that are determined, the largest and smallest amplicon sizes to be considered within the classification, and the threshold bootstrap value for a classification cutoff. These values are dependent on the quality and diversity of the samples as well as the user's own criteria. The minimum input would be the fixrank file if the user does not want to change any of the default settings. A sample process is listed below, with the command listing an output prefix and the path to the fixrank file.

dbcAmplicons abundance -O join/abundance -F join/classify.fixrank --debug

36570 lines processed in 0.01 minutes

Classification numbers (reads):

domain: 129

phylum: 33

class: 89

order: 294

family: 2082

genus: 33943

species: 0

isolate: 0

Writing output

Writing abundance file to: join/abundance.abundance.txt

Writing proportions file to: join/abundance.proportions.txt

finished in 0.01 minutes

Cleaning up.

Standard out will show a summary of the lowest levels of classifications within the process, and paths to the output files. The outputs include the following three text files: The abundance.txt file lists the taxon abundance with the read counts. The proportions.txt lists the proportions of reads per sample, and the taxa counts.txt lists a summary of the lowest level classification and the number of counts for that level.

The *abundance* step also has the ability to output files in the biom format (<http://biom-format.org/>) if the user specifies:

sample table length: 208, and 5 projects.

36570 lines processed in 0.01 minutes

Classification numbers (reads):

domain: 129

phylum: 33

class: 89

order: 294

family: 2082

genus: 33943

species: 0

isolate: 0

Writing output

Writing json formatted biom file to: join/abundance.biom

finished in 0.02 minutes

Cleaning up.

**abundance.txt:**



Lists all taxa for which at least 1 read was found, the level to which the read was classified, and the bootstrap value for the level. All values are in total number of reads per sample.

**proportions.txt:**

OLE-object

The same information as in abundance.txt, but with the proportion of the sample that the number of reads consist.

**table.taxa\_counts.txt**



A summary table of all taxa found, with the number of reads per level.

All analysis files can be loaded into a spreadsheet editor and manipulated further as necessary.