# Metagenomics Workshop SciLifeLab Documentation

***Release 1.0***

**Johannes Alneberg, Johan Bengtsson-Palme, Ino de Bruijn, Luisa**

October 28, 2015

Contents

This is a three day metagenomics workshop. We will discuss assembly, binning and annotation of metagenomic samples.

Program:

- **Day 1 Time 14-17**
  - Metagenomic Assembly Workshop
  - Comparative Functional Analysis Workshop
  - Comparative Taxonomic Analysis Workshop
- **Day 2 Time 14-17**
  - Metagenomic Binning Workshop
- **Day 3 Time 13.30-17**
  - Metagenomic Annotation Workshop

Contents:

CHAPTER 1

# Metagenomic Assembly Workshop

In this metagenomics workshop we will learn how to:

- Quality trim reads with sickle
- Perform assemblies with velvet
- Map back reads to assemblies with bowtie2

The workshop has the following exercises:

## 1.1 Checking required software

An often occuring theme in bioinformatics is installing software. Here we wil go over some steps to help you check whether you actually have the right software installed. There's an optional excerise on how to install `sickle`.

### 1.1.1 Programs used in this workshop

The following programs are used in this workshop:

- Bowtie2
- Velvet
- samtools
- sickle
- Picard
- Ray

All programs are already installed, all you have to do is load the virtual environment for this workshop. Once you are logged in to the server run:

```
source /proj/g2014113/metagenomics/virt-env/mg-workshop/bin/activate
```

You deactivate the virtual environment with:

```
deactivate
```

NOTE: This is a python virtual environment. The binary folder of the virtual environment has symbolic links to all programs used in this workshop so you should be able to run those without problems.

### 1.1.2 Using which to locate a program

An easy way to determine whether you have have a certain program installed is by typing:

```
which programname
```

where `programname` is the name of the program you want to use. The program `which` searches all directories in `$PATH` for the executable file `programname` and returns the path of the first found hit. This is exactly what happens when you would just type `programname` on the command line, but then `programname` is also executed. To see what your `$PATH` looks like, simply `echo` it:

```
echo $PATH
```

For more information on the `$PATH` variable see this link: http://www.linfo.org/path_env_var.html.

### 1.1.3 Check all programs in one go with which

To check whether you have all programs installed in one go, you can use `which`.

bowtie2 bowtie2-build velveth velvetg shuffleSequences_fastq.pl parallel samtools Ray

We will now iterate over all the programs in calling `which` on each of them. First make a variable containing all programs separated by whitespace:

```
$ req_progs="bowtie2 bowtie2-build velveth velvetg parallel samtools shuffleSequences_fastq.pl Ray"
$ echo $req_progs
bowtie2 bowtie2-build velveth velvetg parallel samtools shuffleSequences_fastq.pl
```

Now iterate over the variable `req_progs` and call which:

```
$ for p in $req_progs; do which $p || echo $p not in PATH; done
/proj/g2014113/metagenomics/virt-env/mg-workshop/bin/bowtie2
/proj/g2014113/metagenomics/virt-env/mg-workshop/bin/bowtie2-build
/proj/g2014113/metagenomics/virt-env/mg-workshop/bin/velveth
/proj/g2014113/metagenomics/virt-env/mg-workshop/bin/velvetg
/proj/g2014113/metagenomics/virt-env/mg-workshop/bin/parallel
/proj/g2014113/metagenomics/virt-env/mg-workshop/bin/samtools
/proj/g2014113/metagenomics/virt-env/mg-workshop/bin/shuffleSequences_fastq.pl
```

In Unix-like systems a program that sucessfully completes it tasks should return a zero exit status. For the program `which` that is the case if the program is found. The `||` character does not mean *pipe the output onward* as you are probably familiar with (otherwise see http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO-4.html), but checks whether the program before it exists succesfully and executes the part behind it if not.

If any of the installed programs is missing, try to install them yourself or ask. If you are having troubles following these examples, try to find some bash tutorials online next time you have some time to kill. Educating yourself on how to use the command line effectively increases your productivity immensely.

Some bash resources:

- Excellent bash tutorial http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html
- Blog post on pipes for NGS http://www.vincebuffalo.com/2013/08/08/the-mighty-named-pipe.html
- Using bash and GNU parallel for NGS http://bit.ly/gwbash

### 1.1.4 (Optional excercise) Install sickle by yourself

Follow these steps only if you want to install `sickle` by yourself. Installation procedures of research software often follow the same pattern. Download the code, *compile* it and copy the binary to a location in your `$PATH`. The code for sickle is on https://github.com/najoshi/sickle. I prefer *compiling* my programs in `~/src` and then copying the resulting program to my `~/bin` directory, which is in my `$PATH`. This should get you a long way:

```
mkdir -p ~/src

# Go to the source directory and clone the sickle repository
cd ~/src
git clone https://github.com/najoshi/sickle
cd sickle

# Compile the program
make

# Create a bin directory
mkdir -p ~/bin
cp sickle ~/bin
```

## 1.2 Quality trimming Illumina paired-end reads

In this excercise you will learn how to quality trim Illumina paired-end reads. The most common Next Generation Sequencing (NGS) technology for metagenomics.

### 1.2.1 Sickle

For quality trimming Illumina paired end reads we use the library sickle which trims reads from 3' end to 5' end using a sliding window. If the mean quality drops below a specified number the remaining part of the read will be trimmed.

### 1.2.2 Downloading a test set

Today we'll be working on a small metagenomic data set from the anterior nares (http://en.wikipedia.org/wiki/Anterior_nares).

So get ready for your first smell of metagenomic assembly - pun intended. Run all these commands in your shell:

```
# Download the reads and extract them
mkdir -p ~/asm-workshop
mkdir -p ~/asm-workshop/data
cd ~/asm-workshop/data
wget http://downloads.hmpdacc.org/data/Illumina/anterior_nares/SRS018585.tar.bz2
tar -xjf SRS018585.tar.bz2
```

If successfull you should have the files:

```
$ ls -lh ~/asm-workshop/data/SRS018585/
-rw-rw-r-- 1 inod inod  36M Apr 18  2011 SRS018585.denovo_duplicates_marked.trimmed.1.fastq
-rw-rw-r-- 1 inod inod  36M Apr 18  2011 SRS018585.denovo_duplicates_marked.trimmed.2.fastq
-rw-rw-r-- 1 inod inod 6.4M Apr 18  2011 SRS018585.denovo_duplicates_marked.trimmed.singleton.fastq
```

If not, try to find out if one of the previous commands gave an error.

Look at the top of the one of the pairs:

```
cat ~/asm-workshop/data/SRS018585/SRS018585.denovo_duplicates_marked.trimmed.1.fastq | head
```

**Question: Can you explain what the different parts of this header mean @HWI-EAS324_102408434:5:100:10055:13493/1?**

### 1.2.3 Running sickle on a paired end library

I like to create directories for specific parts I'm working on and creating symbolic links (shortcuts in windows) to the input files. One can use the command `ln` for creating links. The difference between a symbolic link and a hard link can be found here: http://stackoverflow.com/questions/185899/what-is-the-difference-between-a-symbolic-link-and-a-hard-link. In this case I use symbolic links so I know what path the original reads have, which can help one remember what those reads were:

```
mkdir -p ~/asm-workshop/sickle
cd ~/asm-workshop/sickle
ln -s ../data/SRS018585/SRS018585.denovo_duplicates_marked.trimmed.1.fastq pair1.fastq
ln -s ../data/SRS018585/SRS018585.denovo_duplicates_marked.trimmed.2.fastq pair2.fastq
```

Now run sickle:

```
# check if sickle is in your PATH
which sickle
# Run sickle
sickle pe \
    -f pair1.fastq \
    -r pair2.fastq \
    -t sanger \
    -o qtrim1.fastq \
    -p qtrim2.fastq \
    -s qtrim.unpaired.fastq
# Check what files have been generated
ls
```

Sickle states how many reads it trimmed, but it is always good to be suspicious! Check if the numbers correspond with the amount of reads you count. Hint: use `wc -l`.

**Question: How many paired reads are left after trimming? How many singletons?**

**Question: What are the different quality scores that sickle can handle? Why do we specify -t sanger here?**

## 1.3 Assembling reads with Velvet

In this exercise we will learn how to perform an assembly with Velvet. Velvet takes your reads as input and turns them into contigs. It consists of two steps. In the first step, `velveth`, the de Bruijn graph is created. Afterwards the graph is traversed and contigs are created with `velvetg`. When constructing the de Bruijn graph, a *kmer* has to be specified. Reads are cut up into pieces of length *k*, each representing a node in the graph, edges represent an overlap (some de Bruijn graph assemblers do this differently, but the idea is the same). The advantage of using kmer overlap instead of read overlap is that the computational requirements grow with the number of unique kmers instead of unique reads. A more detailled explanation can be found at http://www.nature.com/nbt/journal/v29/n11/full/nbt.2023.html.

### 1.3.1 Pick a kmer

Please work in pairs for this assignment. Every group can select a kmer of their likings - pick a random one if you haven't developed a preference yet. Write you and your partner's name down at a kmer on the Google doc for this

workshop.

### 1.3.2 velveth

Create the graph data structure with `velveth`. Again like we did with `sickle`, first create a directory with symbolic links to the pairs that you want to use:

```
mkdir -p ~/asm-workshop/velvet
cd ~/asm-workshop/velvet
ln -s ../sickle/qtrim1.fastq pair1.fastq
ln -s ../sickle/qtrim2.fastq pair2.fastq
```

The reads need to be interleaved for `velveth`:

```
shuffleSequences_fastq.pl pair1.fastq pair2.fastq pair.fastq
```

Run velveth over the kmer you picked (21 in this example):

```
velveth out_21 21 -fastq -shortPaired pair.fastq
```

Check what directories have been created:

```
ls
```

### 1.3.3 velvetg

To get the actual contigs you will have to run `velvetg` on the created graph. You can vary options such expected coverage and the coverage cut-off if you want, but we do not do that in this tutorial. We only choose not to do scaffolding:

```
velvetg out_21 -scaffolding no
```

### 1.3.4 assemstats

After the assembly one wants to look at the length distributions of the resulting assemblies. You can use the `assemstats` script for that:

```
assemstats 100 out_*/contigs.fa
```

Try to find-out what each of the stats represent by varying the cut-off. One of the most often used statistics in assembly length distribution comparisons is the *N50 length*, a weighted median, where you weight each contig by it's length. This way you assign more weight to larger contigs. Fifty procent of all the bases in the assembly are contained in contigs shorter or equal to N50 length. Once you have gotten an idea of what it all the stats mean, it is time to compare your results with the other attendees of this workshop. Generate the results and copy them to the doc:

```
assemstats 100 out_*/contigs.fa
```

Do the same for the cut-off at 1000 and add it to the doc. Compare your kmer against the others. If there are very little available yet, this would be an ideal time to help out some other attendees or do the same exercise for a kmer that has not been picked by somebody else yet. Please write down you and your partners name again at the doc in that case.

**Question: What are the important length statistics? Do we prefer sum over length? Should it be a combination?**

Think of a formula that could indicate the best preferred length distribution where you express the optimization function in terms of the column names from the doc. For instance only `n50_len` or `sum * n50_len`.

### 1.3.5 (Optional exercise) Ray

Try to create an assembly with Ray over the same kmer. Ray is an assembler that uses MPI to distribute the assembly over multiple cores and nodes. The latest version of Ray was made to work well with metagenomics data as well:

```
mkdir -p ~/asm-workshop/ray
cd ~/asm-workshop/ray
ln -s ../sickle/qtrim1.fastq pair1.fastq
ln -s ../sickle/qtrim2.fastq pair2.fastq
mpiexec -n 1 Ray -k 21 -p pair1.fastq pair2.fastq -o out_21
```

Add the `assemstats` results to the doc as you did for Velvet. There is a separate tab for the Ray assemblies, compare the results with Velvet.

### 1.3.6 (Optional exercise) VelvetOptimiser

VelvetOptimiser is a script that runs Velvet multiple times and follows the optimization function you give it. Use VelvetOptimiser to find the assembly that gets the best score for the optimization function you designed in *assemstats*. It requires `BioPerl`, which you can get on uppmax with `module load BioPerl`.

## 1.4 Mapping reads back to the assembly

### 1.4.1 Overview

There are many different mappers available to map your reads back to the assemblies. Usually they result in a SAM or BAM file (http://genome.sph.umich.edu/wiki/SAM). Those are formats that contain the alignment information, where BAM is the binary version of the plain text SAM format. In this tutorial we will be using bowtie2 (http://bowtie-bio.sourceforge.net/bowtie2/index.shtml).

The SAM/BAM file can afterwards be processed with Picard (http://picard.sourceforge.net/) to remove duplicate reads. Those are likely to be reads that come from a PCR duplicate (http://www.biostars.org/p/15818/).

BEDTools (http://code.google.com/p/bedtools/) can then be used to retrieve coverage statistics.

There is a script available that does it all at once. Read it and try to understand what happens in each step:

```
less `which map-bowtie2-markduplicates.sh`
map-bowtie2-markduplicates.sh -h
```

Bowtie2 has some nice documentation: http://bowtie-bio.sourceforge.net/bowtie2/manual.shtml

**Question: what does bowtie2-build do?**

Picard's documentation also exists! Two bioinformatics programs in a row with decent documentation! Take a moment to celebrate, then have a look here: http://sourceforge.net/apps/mediawiki/picard/index.php

**Question: Why not just remove all identitical pairs instead of mapping them and then removing them?**

**Question: What is the difference between samtools rmdup and Picard MarkDuplicates?**

### 1.4.2 Mapping reads with bowtie2

Take an assembly and try to map the reads back using bowtie2. Do this on an interactive node again, and remember to change the 'out_21' part to the actual output directory that you generated:

```
# Create a new directory and link files
mkdir -p ~/asm-workshop/bowtie2
cd ~/asm-workshop/bowtie2
ln -s ../velvet/out_21/contigs.fa contigs.fa
ln -s ../sickle/pair1.fastq pair1.fastq
ln -s ../sickle/pair2.fastq pair2.fastq

# Run the everything in one go script.
map-bowtie2-markduplicates.sh -t 1 -c pair1.fastq pair2.fastq pair contigs.fa contigs map > map.log 2
```

Inspect the `map.log` output and see if all went well.

**Question: What is the overall alignment rate of your reads that bowtie2 reports?**

Add the answer to the doc.

### 1.4.3 Some general statistics from the SAM/BAM file

You can also determine mapping statistics directly from the bam file. Use for instance:

```
# Mapped reads only
samtools view -c -F 4 map/contigs_pair-smds.bam

# Unmapped reads only
samtools view -c -f 4 map/contigs_pair-smds.bam
```

From: http://left.subtree.org/2012/04/13/counting-the-number-of-reads-in-a-bam-file/. The number is different from the number that bowtie2 reports, because these are the numbers after removing duplicates. The `-smds` part stands for running `samtools sort`, `MarkDuplicates.jar` and `samtools sort` again on the bam file. If all went well with the mapping there should also be a `map/contigs_pair-smd.metrics` file where you can see the percentage of duplication. Add that to the doc as well.

### 1.4.4 Coverage information from BEDTools

Look at the output from BEDTools:

```
less map/contigs_pair-smds.coverage
```

The format is explained here http://bedtools.readthedocs.org/en/latest/content/tools/genomecov.html. The `map-bowtie2-markduplicates.sh` script also outputs the mean coverage per contig:

```
less map/contigs_pair-smds.coverage.percontig
```

**Question: What is the contig with the highest coverage? Hint: use sort -k**

At least a basic knowledge of how to work with the command line is required otherwise it will be very difficult to follow some of the examples. Have fun!

CHAPTER 2

# Comparative Functional Analysis Workshop

In this workshop we will do a comparative analysis of several Baltic Sea samples on function. The following topics will be discussed:

- Find genes on a coassembly of all samples using Prodigal
- Classify the found genes with similar function in Clusters of Orthologous Groups (COG) using WebMGA
- Compare the expression of the different COG families and classes by looking at their coverage in different samples using R

The workshop has the following exercises:

1. Gene Finding Exercise
2. COG Exercise
3. Comprative Functional Analysis Exercise

Contents:

## 2.1 Gene finding

Now that you have assembled the data into contigs next natural step to do is annotation of the data, i.e. finding the genes and doing functional annotation of those. For gene finding a range of programs are available (Metagene Annotator, MetaGeneMark, Orphelia, FragGeneScan), here we will use Prodigal which is very fast and has recently been enhanced for metagenomics. We will use the -p flag which instructs Prodigal to use the algorithm suitable for metagenomic data. We will use a dataset consisting of 11 samples from a time series sampling of surface water in the Baltic Sea. Sequencing was done with Illumina MiSeq here generating on average 835,048 2 x 250 bp reads per sample. The reads can be found here:

```
/proj/g2014113/metagenomics/comparative-functional-analysis/reads
```

The first four numbers in the filename represent a date. All samples are from 2012. R1 and R2 both contain one read of a pair. They are ordered, so the first four lines in R1 are paired with the read in the first four lines of R2. They are in CASAVA v1.8 format (http://en.wikipedia.org/wiki/FASTQ_format).

A coassembly has already been made with Ray using all reads to save you some time. You can find the contigs from a combined assembly on reads from all samples here:

```
/proj/g2014113/metagenomics/cfa/assembly/baltic-sea-ray-noscaf-41.1000.fa
```

They have been constructed with Ray using a kmer of 41 and no scaffolding. Only contigs >= 1000 are in this file. The reason a coassembly is used is that we can get an idea of the entire metagenome over multiple samples. By mapping the reads back per sample we can compare coverages of contigs between samples.

**Question: What could be a possible advantage/disadvantage for the assembly process when assembling multiple samples at one time?**

**Question: Can you think of other approaches to get a coassembly?**

Note that all solutions (i.e. the generated outputs) for the exercises are also in:

```
/proj/g2014113/metagenomics/cfa/
```

In all the following exercises you should again use the virtual environment to get all the necessary programs (unless you already loaded it ofc):

```
source /proj/g2014113/metagenomics/virt-env/mg-workshop/bin/activate
```

It's time to run Prodigal. First create an output directory with a copy of the contig file:

```
mkdir -p ~/metagenomics/cfa/prodigal
cd ~/metagenomics/cfa/prodigal
cp /proj/g2014113/metagenomics/cfa/assembly/baltic-sea-ray-noscaf-41.1000.fa .
```

Then run Prodigal on the contig file (~2m20):

```
prodigal -a baltic-sea-ray-noscaf-41.1000.aa.fa \
         -d baltic-sea-ray-noscaf-41.1000.nuc.fa \
         -i baltic-sea-ray-noscaf-41.1000.fa \
         -f gff -p meta \
         > baltic-sea-ray-noscaf-41.1000.gff
```

This will produce 3 files:

- `-d` a fasta file with the gene sequences (nucleotides)
- `-a` a fasta file with the protein sequences (aminoacids)
- `stdout` a gff file

The gff format is a standardised file type for showing annotations.It's a tab delimited file that can be viewed by e.g.

```
less baltic-sea-ray-noscaf-41.1000.gff
```

Pass the option -S to less if you don't want lines to wrap

An explanation of the gff format can be found at http://genome.ucsc.edu/FAQ/FAQformat.html

**Question: How many coding regions were found by Prodigal? Hint: use grep -c**

**Question: How many contigs have coding regions? How many do not?**

## 2.2 Functional annotation

Now that we have extracted the genes/proteins we want to functionally annotate those. There are a bunch of ways of doing this. We will use webMGA to do do rpsBLAST searches against the COG database. COGs are clusters of orthologs genes, i.e. evolutionary counterparts in different species, usually with the same function (http://www.ncbi.nlm.nih.gov/COG/). Many COGs have known functions and the COGs are also grouped at a higher level with functional classes.

To download the protein sequences that Prodigal generated, open a local terminal and type:

```
mkdir -p ~/metagenomics/cfa/prodigal
cd ~/metagenomics/cfa/prodigal
scp username@milou.uppmax.uu.se:~/metagenomics/cfa/prodigal/baltic-sea-ray-noscaf-41.1000.aa.fa .
```

To get COG classifications of your proteins, go to webMGA http://weizhong-lab.ucsd.edu/metagenomic-analysis/ and select Server / Function annotation / COG. Upload the protein file (`baltic-sea-ray-noscaf-41.1000.aa.fa`) and use the default -e value cutoff. rpsBLAST is used, which is a BLAST based on position specific scoring matrices (pssm). For each COG, one such pssm has been constructed. These are compiled into a database of profiles that is searched against. http://www.ncbi.nlm.nih.gov/staff/tao/URLAPI/wwwblast/node20.html. rpsBLAST is more sensitive than a normal BLAST, which is important if genomes in your metagenome are distant from existing sequences in databases. It is also faster than searching against all proteins out there.

When the search is done you get a zipped folder. On milou, create the directory:

```
mkdir -p ~/metagenomics/cfa/wmga-cog
```

Use wget or curl to download the zip file on uppmax or use scp to upload it to that folder i.e.:

```
scp output.zip username@milou.uppmax.uu.se:~/metagenomics/cfa/wmga-cog
```

Then unzip the file on kalkyl:

```
cd ~/metagenomics/cfa/wmga-cog
unzip output.zip
```

Have a look at the README.txt to see what all the files represent. The file output.2 includes detailed information on the classifications for every protein with a hit below the -e value cutoff. View them with:

```
less README.txt
less -S output.2
```

NOTE: If the queueing takes too much time you can also just copy the results from the project dir:

```
cp -r /proj/g2014113/metagenomics/cfa/wmga-cog/ ~/metagenomics/cfa/
```

**Question: What seem to be the 3 most abundant COG classes in our combined sample (not taking coverage into account)?**

## 2.3 Determine gene coverage in metagenomic samples

Ok, now that we know what functions are represented in the combined samples (we could call it the Baltic meta-community, i.e. a community of communities), we may want to know how much of the different functions (COG families and classes) are present in the different samples, since this will likely change between seasons. To do this we first map the reads from the different samples against the contigs. We will use the mapping script that we used this morning. First create a directory and cd there:

```
mkdir -p ~/metagenomics/cfa/map
cd ~/metagenomics/cfa/map
```

Copy the contig file and build an index on it for bowtie2:

```
cp /proj/g2014113/metagenomics/cfa/assembly/baltic-sea-ray-noscaf-41.1000.fa .
bowtie2-build baltic-sea-ray-noscaf-41.1000.fa baltic-sea-ray-noscaf-41.1000.fa
```

You will end up with various baltic-sea-ray-noscaf-41.1000.fa.*.bt2 files that represent the index for the assembly. It allows for faster alignment of sequences, see http://en.wikipedia.org/wiki/Burrows%E2%80%93Wheeler_transform for more information.

Now we will use some crazy bash for loop to map all the reads. This actually prints the mapping command instead of executing, because it takes too much time to run it, it does however create the directories:

```
for s in /proj/g2014113/metagenomics/cfa/reads/*_R1.fastq.gz; do
    echo mkdir -p $(basename $s _R1.fastq.gz)
    echo cd $(basename $s _R1.fastq.gz)
    echo map-bowtie2-markduplicates.sh -ct 1 \
        $s ${s/R1/R2} pair \
        ~/metagenomics/cfa/map/baltic-sea-ray-noscaf-41.1000.fa asm \
        bowtie2
    echo cd ..
done
```

The for loop iterates over all the first mates of the pairs. It then creates a directory using the basename of the pair with the directory part and the postfix removed, goes to that dir and runs `map-bowtie2-markduplicates.sh` on both mates of the pair. Try to change the for loop such that it only maps one sample.

**Question: Can you think of an easy way to parallelize this?** .. Add an & after bowtie2

For more examples on how to parallelize this check: http://bit.ly/gwbash

If you sort of understand what's going on in the for loop above you are welcome to copy the data that we have already generated:

```
cp -r /proj/g2014113/metagenomics/cfa/map/* ~/metagenomics/cfa/map/
```

Take a look at the files that have been created. Check `map-bowtie2-markduplicates.sh -h` for an explanation of the different files.

**Question what is the mean coverage for contig-394 in sample 0328?**

Next we want to figure out the coverage for every gene in every contig per sample. We will use the bedtools coverage command within the BEDTools suite (https://code.google.com/p/bedtools/) that can parse a SAM/BAM file and a gff file to extract coverage information for every gene:

```
mkdir -p ~/metagenomics/cfa/coverage-hist-per-feature-per-sample
cd ~/metagenomics/cfa/coverage-hist-per-feature-per-sample
```

Run bedtools coverage on one sample (~4m):

```
for s in 0328; do
    bedtools coverage -hist -abam ~/metagenomics/cfa/map/$s/bowtie2/asm_pair-smds.bam \
        -b ../prodigal/baltic-sea-ray-noscaf-41.1000.gff \
        > $s-baltic-sea-ray-noscaf-41.1000.gff.coverage.hist
done
```

Copy the other ones:

```
cp /proj/g2014113/metagenomics/cfa/map/coverage-hist-per-feature-per-sample/* .
```

Have a look at which files have been created with less again. The final four columns give you the histogram i.e. coverage, number of bases with that coverage, length of the contig/feature/gene, bases with that coverage expressed as a ratio of the length of the contig/feature/gene.

Now what we want to is do is to extract the mean coverage per COG instead of per gene. Remember that multiple genes can belong to the same COG so we will take the sum of the mean coverage from those genes. We will use the script `br-sum-mean-cov-per-cog.py` for that. First make a directory again and go there:

```
mkdir -p ~/metagenomics/cfa/cog-sum-mean-cov
cd ~/metagenomics/cfa/cog-sum-mean-cov
```

The script expects a file with one samplename per line so we will create an array with those sample names (http://www.thegeekstuff.com/2010/06/bash-array-tutorial/):

```
samplenames=(0328 0403 0423 0531 0619 0705 0709 1001 1004 1028 1123)
echo ${samplenames[*]}
```

Now we can use process substitution to give the script those sample names without having to store it to a file first.

**Question: What is the difference between the following statements?**:

```
echo ${samplenames[*]}
cat <(echo ${samplenames[*]})
cat <(echo ${samplenames[*]} | tr ' ' '\n')
```

Run the the script that computes the sum of mean coverages per COG (~2m47):

```
br-sum-mean-cov-per-cog.py --samplenames <(echo ${samplenames[*]} | tr ' ' '\n') \
    ../prodigal/baltic-sea-ray-noscaf-41.1000.gff ../prodigal/baltic-sea-ray-noscaf-41.1000.aa.fa \
    ../wmga-cog/output.2 ../coverage-hist-per-feature-per-sample/*.gff.coverage.hist \
    > cog-sum-mean-cov.tsv
```

Have a look at the table with less -S again.

**Question: What is the sum of mean coverages for COG0038 in sample 0423?**

## 2.4 Comparative functional analysis with R

Having this table one can use different statistical and visualisation software to analyse the results. One option would be to import a simpler version of the table into the program Fantom, a graphical user interface program developed for comparative analysis of metagenome data. You can try this in the end of the day if you have time.

But here we will use the statistical programming language R to do some simple analysis. cd to the directory where you have the cog-sum-mean-cov.tsv file. Then start R:

```
cd ~/metagenomics/cfa
R
```

and import the data:

```
tab_cog <- read.delim("cog-sum-mean-cov/cog-sum-mean-cov.tsv")
```

Assign the different columns with descriptors to vectors of logical names:

```
cogf <- tab_cog[,1] # cog family
cogfd <- tab_cog[,2] # cog family descriptor
cogc <- tab_cog[,3] # cog class
cogcd <- tab_cog[,4] # cog class descriptor
```

Make a matrix with the coverages of the cog families:

```
cogf_cov <- as.matrix(tab_cog[,5:ncol(tab_cog)]) # coverage in the different samples
```

And why not put sample names into a vector as well:

```
sample <- colnames(cogf_cov)
sample
```

Let's clean the sample names a bit:

```
for (i in 1:length(sample)) {
    sample[i] <- matrix(unlist(strsplit(sample[i],"_")), 1)[1,4]
}
```

Since the coverages will differ depending on how many reads per sample we have we can normalise by dividing the coverages by the total coverage for the sample (only considering cog-annotated genes though):

```
for (i in 1:ncol(cogf_cov)) {
    cogf_cov[,i] <- cogf_cov[,i]/sum(cogf_cov[,i])
}
```

The cogf_cov gives coverage per cog family. Let's summarise within cog classes and make a separate matrix for that:

```
unique_cogc <- levels(cogc)
cogc_cov <- matrix(ncol = length(sample), nrow = length(unique_cogc))
colnames(cogc_cov) <- sample
rownames(cogc_cov) <- unique_cogc
for (i in 1:length(unique_cogc)) {
    these <- grep(paste("^", unique_cogc[i],"$", sep = ""), cogc)
    for (j in 1:ncol(cogf_cov)) {
        cogc_cov[i,j] <- sum(cogf_cov[these,j])
    }
}
```

OK, now let's start playing with the data. We can for example do a pairwise plot of coverage of cog classes in sample1 vs. sample2:

```
plot(cogc_cov[,1], cogc_cov[,2])
```

or make a stacked barplot showing the different classes in the different samples:

```
barplot(cogf_cov, col = rainbow(100), border=NA)
barplot(cogc_cov, col = rainbow(10), border=NA)
```

The vegan package contains many nice functions for doing (microbial) ecology analysis. Load vegan:

```
install.packages("vegan") # not necessary if already installed
library(vegan)
```

If installing doesn't work for you have a look here http://www.stat.osu.edu/computer-support/mathstatistics-packages/installing-r-libraries-locally-your-home-directory

We can calculate pairwise distances between the samples based on their functional composition. In ecology pairwise distance between samples is referred to as beta-diversity, although typically based on taxonomic composition rather than functional:

```
cogf_dist <- as.matrix(vegdist(t(cogf_cov), method="bray", binary=FALSE, diag=TRUE, upper=TRUE, na.rm
cogc_dist <- as.matrix(vegdist(t(cogc_cov), method="bray", binary=FALSE, diag=TRUE, upper=TRUE, na.rm
```

You can visualise the distance matrices as a heatmaps:

```
image(cogf_dist)
image(cogc_dist)
```

Are the distances calculated on the different functional levels correlated?:

```
plot(cogc_dist, cogf_dist)
```

Now let's cluster the samples based on the distances with hierarchical clustering. We use the function "agnes" in the "cluster" library and apply average linkage clustering:

```
install.packages("cluster") # not necessary if already installed
library(cluster)

cluster <- agnes(cogf_dist, diss = TRUE, method = "average")
plot(cluster, which.plots = 2, hang = -1, label = sample, main = "", axes = FALSE, xlab = "", ylab =
```

Alternatively you can use the function heatmap, that calculates distances both between samples and between features and clusters in two dimensions:

```
heatmap(cogf_dist, scale = "none")
heatmap(cogc_dist, scale = "none")
```

And let's ordinate the data in two dimensions. This can be done e.g. by PCA based on the actual coverage values or by e.g. PcOA or NMDS (non-metrical dimensional scaling). Let's do NMDS:

```
mds <- metaMDS(cogf_dist)
plot(mds$points[,1], mds$points[,2], pch = 20, xlab = "NMDS1", ylab = "NMDS2", cex = 2)
```

We can color the samples according to date (provided your samples are ordered according to date). There are some nice color scales to choose from here http://colorbrewer2.org/:

```
install.packages("RColorBrewer") # not necessary if already installed
library(RColorBrewer)
color = brewer.pal(length(sample), "Reds") # or select another color scale!

mds <- metaMDS(cogf_dist)
plot(mds$points[,1], mds$points[,2], pch = 20, xlab = "NMDS1", ylab = "NMDS2", cex = 5, col = color)
```

Let's compare with how it looks if we base the clustering on COG class coverage instead:

```
mds <- metaMDS(cogc_dist)
plot(mds$points[,1], mds$points[,2], pch = 20, xlab = "NMDS1", ylab = "NMDS2", cex = 5, col = color)
```

In addition to these examples there are of course infinite ways to analyse the results in R. One could for instance find COGs that significantly differ in abundance between samples, do different types of correlations between metadata (nutrients, temperature, etc) and functions, etc. Leave your R window open, since we will compare these results with taxonomic data in a bit.

CHAPTER 3

# Comparative Taxonomic Analysis Workshop

In this workshop we will do a comparative analysis of several Baltic Sea samples on taxonomy. The following topics will be discussed:

- Taxonomic annotation of rRNA reads using sortmeRNA
- Visualizing taxonomy with KRONA
- Comparative taxonomic analysis in R

The workshop has the following exercises:

1. rRNA Annotation Exercise
2. KRONA Exercise
3. Comprative Taxonomic Analysis Exercise

Contents:

## 3.1 Extracting rRNA encoding reads and annotating them

Taxonomic composition of a sample can be based on e.g. BLASTing the contigs against a database of reference genomes, or by utilising rRNA sequences. Usually assembly doesn't work well for rRNA genes due to their highly conserved regions, therefore extracting rRNA from contigs will miss a lot of the taxonomic information that can be obtained by analysing the reads directly. Analysing the reads also has the advantage of being quantitative, i.e. we don't need to calculate coverages by the mapping procedure we applied for the functional genes above. We will extract rRNA encoding reads with the program sortmeRNA which is one of the fastest software solutions for this. The program sortmeRNA has built-in multithreading support so this time we use that for parallelization. These are the commands to run:

```
mkdir -p ~/metagenomics/cta/sortmerna
cd ~/metagenomics/cta/sortmerna
samplenames=(0328 0403 0423 0531 0619 0705 0709 1001 1004 1028 1123)
for s in ${samplenames[*]}
    do sortmerna -n 2 --db \
    /proj/g2014113/src/sortmerna-1.9/rRNA_databases/silva-arc-16s-database-id95.fasta \
    /proj/g2014113/src/sortmerna-1.9/rRNA_databases/silva-bac-16s-database-id85.fasta \
    --I /proj/g2014113/metagenomics/cta/reads/${s}_pe.fasta \
    --accept ${s}_rrna \
    --other ${s}_nonrrna  \
    --bydbs  \
    -a 8  \
    --log ${s}_bilan  \
```

```
    -m 5242880
done
```

Again, this command takes rather long to run (~5m per sample) so just copy the results if you don't feel like waiting:

```
cp /proj/g2014113/metagenomics/cta/sortmerna/* ~/metagenomics/cta/sortmerna
```

It outputs the reads or part of reads that encode rRNA in a fasta file. These rRNA sequences can be classified in many ways, and again blasting them against a suitable database is one option. Here we use a simple and fast method (unless you have too many samples), the classifier tool at RDP (ribosomal database project). This uses a naive bayesian classifier trained on many sequences of defined taxonomies. It gives bootstrap support values for each taxonomic level; usually the support gets lower the further down the hierarchy you go. Genus level is the lowest level provided. You can use the web service if you prefer, and upload each file individually, or you can use the uppmax installation of RDP classifier like this (~4m):

```
mkdir -p ~/metagenomics/cta/rdp
cd ~/metagenomics/cta/rdp
for s in ../sortmerna/*_rrna*.fasta
    do java -Xmx1g -jar /glob/inod/src/rdp_classifier_2.6/dist/classifier.jar \
    classify \
    -g 16srrna  \
    -b `basename ${s}`.bootstrap  \
    -h `basename ${s}`.hier.tsv  \
    -o `basename ${s}`.class.tsv  \
    ${s}
done
```

## 3.2 Visualising taxonomy with KRONA

To get a graphical representation of the taxonomic classifications you can use KRONA, which is an excellent program for exploring data with hierarchical structures in general. The output file is an html file that can be viewed in a browser. Again make a directory for KRONA:

```
mkdir -p ~/metagenomics/cta/krona
cd ~/metagenomics/cta/krona
```

And run KRONA, concatenating the archaea and bacteria class files together at the same time like this and providing the name of the sample:

```
ktImportRDP \
<(cat ../rdp/0328_rrna.silva-arc-16s-database-id95.fasta.class.tsv ../rdp/0328_rrna.silva-bac-16s-dat
<(cat ../rdp/0403_rrna.silva-arc-16s-database-id95.fasta.class.tsv ../rdp/0403_rrna.silva-bac-16s-dat
<(cat ../rdp/0423_rrna.silva-arc-16s-database-id95.fasta.class.tsv ../rdp/0423_rrna.silva-bac-16s-dat
<(cat ../rdp/0531_rrna.silva-arc-16s-database-id95.fasta.class.tsv ../rdp/0531_rrna.silva-bac-16s-dat
<(cat ../rdp/0619_rrna.silva-arc-16s-database-id95.fasta.class.tsv ../rdp/0619_rrna.silva-bac-16s-dat
<(cat ../rdp/0705_rrna.silva-arc-16s-database-id95.fasta.class.tsv ../rdp/0705_rrna.silva-bac-16s-dat
<(cat ../rdp/0709_rrna.silva-arc-16s-database-id95.fasta.class.tsv ../rdp/0709_rrna.silva-bac-16s-dat
<(cat ../rdp/1001_rrna.silva-arc-16s-database-id95.fasta.class.tsv ../rdp/1001_rrna.silva-bac-16s-dat
<(cat ../rdp/1004_rrna.silva-arc-16s-database-id95.fasta.class.tsv ../rdp/1004_rrna.silva-bac-16s-dat
<(cat ../rdp/1028_rrna.silva-arc-16s-database-id95.fasta.class.tsv ../rdp/1028_rrna.silva-bac-16s-dat
<(cat ../rdp/1123_rrna.silva-arc-16s-database-id95.fasta.class.tsv ../rdp/1123_rrna.silva-bac-16s-dat
```

The `<()` in bash can be used for process substitution (http://tldp.org/LDP/abs/html/process-sub.html ). Just for your information, the above command was actually generated with the following commands:

```
cmd=`echo ktImportRDP; for s in ${samplenames[*]}; do echo '<('cat ../rdp/${s}_rrna.silva-arc-16s-dat
echo $cmd
```

Copy the resulting file rdp.krona.html to your local computer with scp and open it in firefox.

## 3.3 Comparative taxonomic analysis with R

KRONA is not very good for comparing multiple samples though. Instead we will use R as for the functional data. First combine the data from the different samples with the script sum_rdp_annot.pl (made by us) into a table. By default the script uses a bootstrap support of 0.80 to include a taxonomic level (this can be changed easily by changing the number on row 16 in the script). You may need to change the input file names in the beginning of the script ($in_file[0] = ...). The script will only import the 16S bacterial data:

```
cd ~/metagenomics/cta/rdp
sum_rdp_annot.pl > summary.rrna.silva-bac-16s-database-id85.fasta.class.0.80.tsv
```

Let's import this table into R:

```
tab_tax <- read.delim("summary.rrna.silva-bac-16s-database-id85.fasta.class.0.80.tsv")
```

And assign the descriptor column to a vector:

```
tax <- tab_tax[,1]
```

And put the counts into a matrix:

```
tax_counts <- tab_tax[,2:ncol(tab_tax)] # counts of taxa in the different samples
```

Since you will compare this dataset with the functional dataset you generated before it's great if the samples come in the same order. Check the previous order:

```
sample
```

And the current order:

```
colnames(tax_counts)
```

if they are not in the same order contact an assistant for help. Otherwise:

```
colnames(tax_counts) <- sample
rownames(tax_counts) <- tax
```

Make a normalised version of tax_counts:

```
norm_tax_counts <- tax_counts
for (i in 1:ncol(tax_counts)) {
    norm_tax_counts[,i] <- tax_counts[,i]/sum(tax_counts[,i])
}
```

What different taxa do we have there?:

```
tax
```

From the tax_counts matrix we can create new matrices at defined taxonomic levels. If you open the text file /proj/g2013206/metagenomics/r_commands.txt you can copy and paste all of this code into R (or use the source command) and this will give you the matrices and vectors below (check carefully that you didn't get any error messages!):

```
phylum_counts             matrix with counts for different phyla
norm_phylum_counts          normalised version of phylum_count
phylum               vector with phyla (same order as in phyla matrix)

class_counts             matrix with counts for different classes
norm_class_counts        normalised version of class_count
class                  vector with classes

phylumclass_counts         matrix with counts for different phyla and proteobacteria classes
norm_phylumclass_counts    normalised version of phylumclass_count
phylumclass              vector with phyla and proteobacteria classes
```

The "other" clade in each of the above sums reads that were not classified at the defined level. Having these more well defined matrices we can compare the taxonomic composition in the samples. We can apply the commands that we did for the functional analysis:

```
library(vegan)
library(RColorBrewer)
```

Barplots:

```
par(mar=c(1,2,1,22)) # Increase the MARgins, to make space for a legend

barplot(norm_phylum_counts, col = rainbow(11), legend.text=TRUE, args.legend=list(x=ncol(norm_phylum_
barplot(norm_phylumclass_counts, col = rainbow(15), legend.text=TRUE, args.legend=list(x=ncol(norm_ph
barplot(norm_class_counts, col = rainbow(18), legend.text=TRUE, args.legend=list(x=ncol(norm_phylum_c
```

If you can't see the legends, they're just in a bad position. Try altering the x and y parameters in the args.legend.

Calculate beta-diversity based on class-level taxonomic counts:

```
class_dist <- as.matrix(vegdist(t(norm_class_counts[1:(nrow(norm_class_counts) - 1),]), method="bray"
```

Note that by "[1:(nrow(norm_class_counts) - 1),]" we exclude the last row in norm_class_counts when we calculate the distances because this is the "others" column that contains all kinds of unclassified taxa.

Hierarchical clustering:

```
library(cluster)
cluster <- agnes(class_dist, diss = TRUE, method = "average")
plot(cluster, which.plots = 2, hang = -1)
```

Heatmaps with clusterings:

```
heatmap(norm_class_counts, scale = "none")
heatmap(norm_phylumclass_counts, scale = "none")
```

And ordinate the data by NMDS:

```
color = brewer.pal(length(sample), "Reds")
mds <- metaMDS(class_dist)
plot(mds$points[,1], mds$points[,2], pch = 20, xlab = "NMDS1", ylab = "NMDS2", cex = 5, col = color)
```

Does the pattern look similar as that obtained by functional data?:

```
mds <- metaMDS(cogc_dist)
plot(mds$points[,1], mds$points[,2], pch = 20, xlab = "NMDS1", ylab = "NMDS2", cex = 5, col = color)
```

We can actually check how beta diversity generated by the two approaches is correlated:

```
plot(cogf_dist, class_dist)
cor.test(cogf_dist, class_dist)
```

(For comparing matrices it is common to use a mantel test, but the r-value (but not the p-value) is in fact the same.)

Finally, let's check how alpha-diversity fluctuates over the year and compares between taxonomic and functional data. Since alpha-diversity is influenced by sample size it is advisable to subsample the datasets to the same number of reads. We can make a subsampled table using the vegan function rrarefy:

```
sub_class_counts <- t(rrarefy(t(class_counts), 100))
```

This will be difficult to achieve for the functional data at this point, however, so let's skip that for the functional data.

Let's use Shannon diversity index since this is pretty insensitive to sample size. Shannon index combines richness (number of species) and evenness (how evenly the species are distributed); many, evenly distributed species gives a high Shannon. There is a vegan function for getting shannon:

```
class_shannon <- diversity(class_counts[1:(nrow(norm_class_counts) - 1),], MARGIN = 2)
sub_class_shannon <- diversity(sub_class_counts[1:(nrow(norm_class_counts) - 1),], MARGIN = 2)
cogf_shannon <- diversity(cogf_cov, MARGIN = 2)
```

How does subsampling influence shannon?:

```
plot(class_shannon, sub_class_shannon)
```

Is functional and taxonomic shannon correlated?:

```
plot(sub_class_shannon, cogf_shannon)
cor.test(sub_class_shannon, cogf_shannon)
```

CHAPTER 4

# Metagenomic Binning Workshop

In this metagenomics workshop we will learn how to:

- Perform unsupervised binning with concoct
- Evaluate binning performance

The workshop has the following exercises:

1. Setup Environment
2. Running Concoct
3. Evaluate Clustering Using Single Copy Genes
4. Phylogenetic Classification using Phylosift

At least a basic knowledge of how to work with the command line is required otherwise it will be very difficult to follow some of the examples. Have fun!

Contents:

## 4.1 Setup Environment

This workshop will be using the same environment used for the assembly workshop. If you did not participate in the assembly workshop, please have a look at the introductory setup description for that.

### 4.1.1 Programs used in this workshop

The following programs are used in this workshop:

- CONCOCT
- Phylosift
- Blast

All programs and scripts that you need for this workshop are already installed, all you have to do is load the virtual environment. Once you are logged in to the server run:

```
source /proj/g2014113/metagenomics/virt-env/mg-workshop/bin/activate
```

If you'd wish to inactivate this virtual environment you could run:

```
deactivate
```

NOTE: This is a python virtual environment. The binary folder of the virtual environment has symbolic links to all programs used in this workshop so you should be able to run those without problems.

### 4.1.2 Check that the programs are available

After you have activated the virtual environment the following commands should execute properly and you should be able to see some brief instructions on how to run the different programs respectively.

CONCOCT:

```
concoct -h
```

BLAST:

```
rpsblast --help
```

## 4.2 CONCOCT - Clustering cONtigs with COverage and ComposiTion

In this excercise you will learn how to use a new software for automatic and unsupervised binning of metagenomic contigs, called CONCOCT. CONCOCT uses a statistical model called a Gaussian Mixture Model to cluster sequences based on their tetranucleotide frequencies and their average coverage over multiple samples.

The theory behind using the coverage pattern is that sequences having similar coverage pattern over multiple samples are likely to belong to the same species. Species having a similar abundance pattern in the samples can hopefully be separated by the tetranucleotide frequencies.

We will be working with an assembly made using only the reads from this single sample, but since CONCOCT is constructed to be ran using the coverage profile over multiple samples, we'll be investigating how the performance is affected if we add several other samples. This is done by mapping the reads from the other samples to the contigs resulting from this single sample assembly.

### 4.2.1 Getting to know the test data set

Today we'll be working on a metagenomic data set from the baltic sea. The sample we'll be using is part of a time series study, where the same location have been sampled twice weekly during 2013. This specific sample was taken march 22.

Start by copying the contigs to your working directory:

```
mkdir -p ~/binning-workshop
mkdir -p ~/binning-workshop/data
cd ~/binning-workshop/
cp /proj/g2014113/nobackup/concoct-workshop/Contigs_gt1000.fa data/
cp /proj/g2014113/nobackup/concoct-workshop/120322_coverage_nolen.tsv data/
```

You should now have one fasta file containing all contigs, in this case only contigs longer than 1000 bases is included to save space, and one comma separated file containing the coverage profiles for each contig. Let's have a look at the coverage profiles:

```
less -S ~/binning-workshop/data/120322_coverage_nolen.tsv
```

Try to find the column corresponding to March 22 and compare this column to the other ones. Can you draw any conclusions from this comparison?

We'd like to first run concoct using only one sample, so we remove all other columns in the coverage table to create this new coverage file:

```
cut -f1,3 ~/binning-workshop/data/120322_coverage_nolen.tsv > ~/binning-workshop/data/120322_coverage
```

### 4.2.2 Running CONCOCT

CONCOCT takes a number of parameters that you got a glimpse of earlier, running:

```
concoct -h
```

The contigs will be input as the composition file and the coverage file obviously as the coverage file. The output path is given by as the -b (–basename) parameter, where it is important to include a trailing slash if we want to create an output directory containing all result files. Last but not least we will set the length threshold to 3000 to speed up the clustering (the less contigs we use, the shorter the runtime):

```
mkdir -p ~/binning-workshop/concoct_output
concoct --coverage_file ~/binning-workshop/data/120322_coverage_one_sample.tsv --composition_file ~/b
```

This command will normally take a couple of minutes to finish. When it is done, check the output directory and try to figure out what the different files contain. Especially, have a look at the main output file:

```
less ~/binning-workshop/concoct_output/3000_one_sample/clustering_gt3000.csv
```

This file gives you the cluster id for each contig that was included in the clustering, in this case all contigs longer than 3000 bases.

For the comparison we will now run concoct again, using the coverage profile over all samples in the time series:

```
concoct --coverage_file ~/binning-workshop/data/120322_coverage_nolen.tsv --composition_file ~/binnin
```

Have a look at the output from this clustering as well, do you notice anything different?

### 4.2.3 Evaluating Clustering Results

One way of evaluating the resulting clusters are to look at the distribution of so called Single Copy Genes (SCG:s), genes that are present in all bacteria and archea in only one copy. With this background, a complete and correct bin should have exactly one copy of each gene present, while missing genes indicate an inclomplete bin and several copies of the same gene indicate a chimeric cluster. To predict genes in prokaryotes, we use Prodigal that we then use as the query sequences for an RPS-BLAST search against the Clusters of Orthologous Groups (COG) database. This RPS-BLAST search takes about an hour and a half for our dataset so we're going to use a precomputed result file. Copy this result file along with two files necessary for the COG counting scripts:

```
cp /proj/g2014113/nobackup/concoct-workshop/Contigs_gt1000_blast.out ~/binning-workshop/data/
cp /proj/g2014113/nobackup/concoct-workshop/scg_cogs_min0.97_max1.03_unique_genera.txt ~/binning-work
cp /proj/g2014113/nobackup/concoct-workshop/cdd_to_cog.tsv ~/binning-workshop/data/
```

With the CONCOCT distribution comes scripts for parsing this output and creating a plot where each cog present in the data are grouped accordingly to the clustering results, namely COG_table.py and COGPlot.R. These scripts are added to the virtual environment, try check out their usage:

```
COG_table.py -h
COGPlot.R -h
```

Let's first create a plot for the single sample run:

```
COG_table.py -b ~/binning-workshop/data/Contigs_gt1000_blast.out -c ~/binning-workshop/concoct_output
COGPlot.R -s ~/binning-workshop/cog_table_3000_single_sample.tsv -o ~/binning-workshop/cog_plot_3000_
```

This command should have created a pdf file with your plot. In order to look at it, you can download it to your personal computer with scp. OBS! You need to run this in a separate terminal window where you are not logged in to Uppmax:

```
scp username@milou.uppmax.uu.se:~/binning-workshop/cog_plot_3000_single_sample.pdf ~/Desktop/
```

Have a look at the plot and try to figure out if the clustering was successful or not. Which clusters are good? Which clusters are bad? Are all clusters present in the plot? Now, lets do the same thing for the multiple samples run:

```
COG_table.py -b ~/binning-workshop/data/Contigs_gt1000_blast.out -c ~/binning-workshop/concoct_output
COGPlot.R -s ~/binning-workshop/cog_table_3000_all_samples.tsv -o ~/binning-workshop/cog_plot_3000_al
```

And download again from your separate terminal window:

```
scp username@milou.uppmax.uu.se:~/binning-workshop/cog_plot_3000_all_samples.pdf ~/Desktop
```

What differences can you observe for these plots? Think about how we were able to use samples not included in the assembly in order to create a different clustering result. Can this be done with any samples?

## 4.3 Phylogenetic Classification using Phylosift

In this workshop we'll extract interesting bins from the concoct runs and investigate which species they consists of. We'll start by using a plain'ol BLASTN search and later we'll try a more sophisticated strategy with the program Phylosift.

### 4.3.1 Extract bins from CONCOCT output

The output from concoct is only a list of cluster id and contig ids respectively, so if we'd like to have fasta files for all our bins, we need to run the following script:

```
extract_fasta_bins.py -h
```

Running it will create a separate fasta file for each bin, so we'd first like to create a output directory where we can store these files:

```
mkdir -p ~/binning-workshop/concoct_output/3000_all_samples/fasta_bins
extract_fasta_bins.py ~/binning-workshop/data/Contigs_gt1000.fa ~/binning-workshop/concoct_output/300
```

Now you can see a number of bins in your output folder:

```
ls ~/binning-workshop/concoct_output/3000_all_samples/fasta_bins
```

Using the graph downloaded in the previous part, decide one cluster you'd like to investigate further. We're going to use the web based BLASTN tool at ncbi, so lets first download the fasta file for the cluster you choose. Execute on a terminal not logged in to UPPMAX:

```
scp username@milou.uppmax.uu.se:~/binning-workshop/concoct_output/3000_all_samples/fasta_bins/x.fa ~/
```

Before starting to blasting this cluster, lets begin with the next assignment, since the next assignment will include a long waiting time that suits for running the BLASTN search.

### 4.3.2 Phylosift

Phylosift is a software created for the purpose of determining the phylogenetic composition of your metagenomic data. It uses a defined set of genes to predict the taxonomy of each sequence in your dataset. You can read more about how this works here: http://phylosift.wordpress.com I've yet to discover how to install phylosift into a common bin, so in order to execute phylosift, you'd have to cd into the phylosift directory:

```
cd /proj/g2014113/src/phylosift_v1.0.1
```

Running phylosift will take some time (roughly 45 min) so lets start running phylosift on the cluster you choose:

```
mkdir -p ~/binning-workshop/phylosift_output/
./phylosift all -f --output ~/binning-workshop/phylosift_output/ ~/binning-workshop/concoct_output/3
```

While this command is running, go to ncbi web blast service:

http://blast.ncbi.nlm.nih.gov/Blast.cgi?PROGRAM=blastn&PAGE_TYPE=BlastSearch&LINK_LOC=blasthome

Upload your fasta file that you downloaded in the previous step and submit a blast search against the nr/nt database. Browse through the result and try and see if you can do a taxonomic classification from these.

When the phylosift run is completed, browse the output directory:

```
ls ~/binning-workshop/phylosift_output/
```

All of these files are interesting, but the most fun one is the html file, so lets download this to your own computer and have a look. Again, switch to a terminal where you're not logged in to UPPMAX:

```
scp username@milou.uppmax.uu.se:~/binning-workshop/phylosift_output/x.fa.html ~/Desktop/
```

Did the phylosift result correspond to any results in the BLAST output?

CHAPTER 5

# Metagenomic Annotation Workshop

In this part of the metagenomics workshop we will:

- Translate nucleotides into amino acid seuquences using EMBOSS
- Annotate metagenomic reads with Pfam domains
- Discuss and perform normalization of metagenomic counts
- Take a look at different gene abundance analysis

The workshop has the following exercises:

1. Translation Exercise
2. HMMER Exercise
3. Normalization Exercise
4. Differential Exercise
5. BonusExercise: Metaxa2

At least a basic knowledge of how to work with the command line is required otherwise it will be very difficult to follow some of the examples. Have fun!

Contents:

## 5.1 Checking required software

Before we begin, we will quickly go through the required software and datasets for this workshop. For those who are already command-line-skilled there will also be a possibility to install the Metaxa2 tool for rRNA finding, but this is not required to complete the workshop.

### 5.1.1 Programs used in this workshop

The following programs are used in this workshop:

- EMBOSS (transeq)
- HMMER
- R
- Optionally: Metaxa2

Since we are going to use the plotting functionality of R, we need to login to Uppmax with X11 forwarding turned on. In the Unix/Linux terminal this is easily achieved by adding the `-X` (captal X) option. All programs but Metaxa2 are already installed, all you have to do is load the virtual environment for this workshop. Once you are logged in to the server run:

```
source /proj/g2014113/metagenomics/virt-env/mg-workshop/bin/activate
```

You deactivate the virtual environment with:

```
deactivate
```

NOTE: This is a python virtual environment. The binary folder of the virtual environment has symbolic links to all programs used in this workshop so you should be able to run those without problems.

### 5.1.2 Check all programs in one go with which

To check whether you have all programs installed in one go, you can use `which` to test for the following programs:

```
hmmsearch
transeq
R
blastall
```

### 5.1.3 Data and databases used in this workshop

In this workshop, we are (due to time constraints) going to use a simplified version of the Pfam database, including only protein families related to plasmid replication and maintenance. This database is pre-compiled and can be downloaded from http://microbiology.se/teach/scilife2014/pfam.tar.gz Download it using the following commands:

```
mkdir -p ~/Pfam
cd ~/Pfam
wget http://microbiology.se/teach/scilife2014/pfam.tar.gz
tar -xzvf pfam.tar.gz
cd ~
```

In addition, you will need to obtain the following data sets for the workshop:

```
/proj/g2014113/metagenomics/annotation/baltic1.fna
/proj/g2014113/metagenomics/annotation/baltic2.fna
/proj/g2014113/metagenomics/annotation/indian_lake.fna
/proj/g2014113/metagenomics/annotation/swedish_lake.fna
```

We are going to use two data sets from the Baltic Sea, one from a Swedish lake and one from an Indian lake contaminated with wastewater from pharmaceutical production. For the same of time, I have reduced the data sets in size dramatically prior to this workshop. You can create links to the above files using the `ln -s <path>` command. Use it on all the four data sets.

### 5.1.4 (Optional excercise) Install Metaxa2 by yourself

Follow these steps only if you want to install `Metaxa2` by yourself. The code for Metaxa2 is available from http://microbiology.se/sw/Metaxa2_2.0rc3.tar.gz You can install Metaxa2 as follows:

```
# Create a src and a bin directory
mkdir -p ~/src
mkdir -p ~/bin
```

```
# Go to the source directory and download the Metaxa2 tarball
cd ~/src
wget http://microbiology.se/sw/Metaxa2_2.0rc3.tar.gz
tar -xzvf Metaxa2_2.0rc3.tar.gz
cd Metaxa2_2.0rc3

# Run the installation script
./install_metaxa2

# Try to run Metaxa2 (this should bring up the main options for the software)
metaxa2 -h
```

If this did not work, you can try this manual approach:

```
cd ~/src/Metaxa2_2.0rc3
cp -r metaxa2* ~/bin/

# Then try to run Metaxa2 again
metaxa2 -h
```

If this brings up the help message, you are all set!

## 5.2 Translating nucleotide sequences into amino acid sequences

The first step before we can annotate the metagenomes with Pfam domains using HMMER will be to translate the reads into amino acid sequences. This is necessary because HMMER (still) does not translate nucleotide sequnces into protein space on the fly (like,for example, BLAST). For completing this task we will use `transeq`, part of the EMBOSS package.

### 5.2.1 Running `transeq` on the sequence data sets

To run `transeq`, take a look at its available options:

```
transeq -h
```

If you have trouble getting `transeq` to run, try to run:

```
module load emboss
```

A few options are important in this context. First of all, we need to supply an input file, using the (somewhat bulky) option `-sequence`. Second, we also need to specify an output file, otherwise transeq will simply write its output to the screen. This is specified using the `-outseq` option.

However, if we just run `transeq` like this we will run into two additional problems. First, `transeq` by default just translate the reading frame beginning at the first base in the input sequnece, and will ignore any bases in the reading frames beginning with base two and three, as well as those on the reverse-complementary strand. Second, the software will add stop characters in the form of asterixes `*` whenever it encounters a stop codon. This will occasionally cause HMMER to choke, so we want stop codons to instead be translated into X characters that HMMER can handle. The following excerpt form the HMMER creator's blog on this subject is one of my personal all-time favorites in terms of computer software documentation:

> There's two ways people do six-frame translation. You can translate each frame into separate ORFs, or you can translate the read into exactly six "ORFs", one per frame, with * chararacters marking stop codons. HMMER prefers that you do the former. Technically, * chararacters aren't legal amino acid residue codes, and the author of HMMER3 is a pedantic nitpicker, passive-aggressive, yet also a pragmatist: so while

> HMMER3 pragmatically accepts * chararacters in input "protein" sequences just fine, it pedantically relegates them to somewhat suboptimal status, and it passively-aggressively figures that any suboptimal performance on *-containing ORFs is your own fault for using *'s in the first place.

To avoid making Sean Eddy angry and causing other problems for our HMMER runs, we will use the `-frame 6` option to `transeq` in order to get translations of all six reading frames, and the `-clean` option to convert stop codons to X instead of *.

That should give us the command:

```
transeq -sequence <input file> -outseq <output file> -frame 6 -clean
```

Now run this command on all four input files that we have created links to. When the command has finished for all files, we can move on to the actual annotation.

## 5.3 Search amino acid sequences with HMMER against the Pfam database

It is time to do the actual Pfam annotation of our metagenomes!

### 5.3.1 Running `hmmsearch` on the translated sequence data sets

Before we run `hmmsearch`, we will look at its available options:

```
hmmsearch -h
```

As you will see, the program takes a substantial amount of arguments. In this workshop we will work with the table output from HMMER, which you get by specifying the `--tblout` option together with a file name. We also want to make sure that we only got statistically relevant matches, which we can do using the E-value option. The E-value (Expect-value) is an estimation of how often we would expect to find a similar hit by chance, given the size of the database. To avoid getting a lot of noise matches, we will specify and E-value of 10^-5, that is that we would by chance get a match with a similarly good alignment in 1 out of 100000 cases. This can be set with the `-E 1e-5` option. Finally, to speed up the process a little, we will use the `--cpu` option to get multi-core support. On the Uppmax machines you can use up to 16 cores for the HMMER runs.

To specify the HMM-file database and the input data set, we just type in the names of those two files at the end of the command. Finally we add in the `> /dev/null` string, to avoid getting the screen cluttered with sequence alignments that HMMER outputs. That should give us the following command:

```
hmmsearch --tblout <output file> -E 1e-5 --cpu 8 ~/Pfam/Pfam-mobility.hmm <input file (protein format
```

Now run this command on all four input files that we just have downloaded. When the command has finished for all files, we can move on to the normalization exercise.

## 5.4 Normalization of count data from the metagenomic data sets

An important aspects of working with metagenomics is to apply proper normalization procedures to the retrieved counts. There are several ways to do this, and in part the method of choice is dependent on the research question investigated, but in part also based on more philosphical considerations. Let's start with a bit of theory.

### 5.4.1 Why is normalization important?

Generally, sequencing data sets are not of the same size. In addition, different genes and genomes come in different sizes, which means that *at equal coverage, the number of mapped reads to a certain gene or region will be directly dependent on the length of that region*. Luckily, the latter scenario is not a huge issue for Pfam families (although it exists), and we will not care about it more today. We will however care about the size of the sequencing libraries. To make relatively fair comparisons between sets, we need to normalize the gene counts to something. Let's begin with checking how unequal the librairies are. You can do that by counting the number of sequences in the FASTA files, by checking for the number of ">" characters in each file, using `grep`:

```
grep -c ">" <input file>
```

As you will see, there are quite substantial differences in the number of reads in each library. How do we account for that?

### 5.4.2 What normalization methods are possible?

The choice of normalization method will depend on what research question we want to ask. An easy way of removing the technical bias related to different sequencing effort in different libraries is to simply divide each gene count with the total library size. That will yield a relative proportion of counts to that gene. To make that number easier to interpret, we can multiply it by 1,000,000 to get *the number of reads corresponding to that gene or feature per million reads*.

(counts of gene X / total number of reads) * 1000000

This is a quick way of normalizing, but it does not consider the composition of the sample. Say that you are interested in studying bacterial gene content within e.g. different plant hosts. Then the interesting changes in bacterial composition might be drowned by genetic material from the host plant. That will then have a huge impact on the gene abundances of the bacteria, even if those abundances are actually the same. The same applies to complex microbial communities with both bacteria, single-cell eukaryotes and viruses. In such cases, it might be better to consider a normalization to the number of bacteria in the sample (or eukaryotes if that is what you want to study). One way of doing that is to count the number of reads mapping to the 16S rRNA gene in each sample. You can then divide each gene count with the number of 16S rRNA counts, to yield a genes per 16S proportion.

(counts of gene X / counts of 16S rRNA gene)

There is a few problems with using the 16S rRNA gene in this way. The most prominient one is that the gene exists in a single copy in some bacteria, but in multiple (sometimes >10) copies in other species. That means that this number will not truly be a per-genome estimate. Other genetic markers, such as the *rpoB* gene has been suggested for this, but has not yet taken off.

Finally, we could imagine a scenario in which you are only interested in the proportion of different annotated features in your sample. One can then instead divide to the total number of reads mapped to *something* in the database used. That will give relative proportions, and will remove a lot of "noise", but will have the limitation that only the well-defined part of the microbial community can be studied, and the rest is ignored.

(counts of gene X / total number of mapped reads)

### 5.4.3 Trying out some normalization methods

We are now ready to try out these methods on our data. Let's begin generating the numbers we need for normalization. We begin with the library sizes. As you remember, those numbers can be generated using `grep`:

```
grep -c ">" <input file>
```

To get the number of 16S rRNA sequences, we will use Metaxa2. If you did not install it, you can "cheat" by getting the numbers from this file: `/proj/g2014113/metagenomics/annotation/metaxa2_16S_rRNA_counts.txt`. If you installed it previously, you can test it out using the following command:

```
metaxa2 -i <input file> -o <output file> --cpu 16 --align none
```

Metaxa2 will take a few minutes to run. You will then be able to get the number of bacterial 16S rRNA sequences from the file ending with .summary.txt.

Finally, we would like to get the number of reads mapping to *any* Pfam family in the database. To get that number, we can again use `grep`. This time however, we will use it to *remove* the entries that we are not interested in, and counting the rest. This can be done by:

```
grep -c -v "^#" <hmmer output file>
```

That will remove all lines beginning with a # character, and count all remaining lines. Write all the numbers down that you have got during this exercize, we will use them in the next step!

## 5.5 Estimating differentially abundant protein families in the metagenomes

Finally, we are about to do some real analysis of the data, and look at the results! To do this, we will use the R statistical program. You start the program by typing:

```
R
```

To get out of R, you type `q()`. You will then be asked if you want to save your workspace. Typing "y" (yes) might be smart, since that will remember all your variables until the next time you use R in the same directory!

### 5.5.1 Loading the the count tables

We will begin by loading the count tables from HMMER into R:

```
b1 = read.table("baltic1.hmmsearch", sep = "")
```

To get the number of entries of each kind, we will use the R command `rle`. We want to get the domain list, which is the third column. For `rle` to be able to work with the data, we must also convert it into a proper vector.:

```
raw_counts = rle(as.vector(b1[,3]))
b1_counts = as.matrix(raw_counts$lengths)
row.names(b1_counts) = raw_counts$values
```

Repeat this procedure for all four data sets.

### 5.5.2 Apply normalizations

We will now try out the three different normalization methods to see their effect on the data. First, we will try by normalizing to the number of reads in each sequencing library. Find the note you have taken on the data set sizes. Then apply a command like this on the data:

```
b1_norm1 = b1_counts / 118025
```

You will now see counts in the range of 10^-5 and 10^6. To make these numbers more interpretable, let's also multiply them by 1,000,000 to yield the counts per million reads:

```
b1_norm1 = b1_counts / 118025 * 1000000
```

Do the same thing for the other data sets.

We would then like to compare all the four data sets to each other. Since R's merge function really suck for multiple data sets, I have provided this function for merging four data sets. Copy and paste it into the R console:

```
merge_four = function(a,b,c,d,names) {
m1 = merge(a,b,by = "row.names", all = TRUE)
row.names(m1) = m1[,1]
m1 = m1[,2:3]
m2 = merge(c, m1, by = "row.names", all = TRUE)
row.names(m2) = m2[,1]
m2 = m2[,2:4]
m3 = merge(d, m2, by = "row.names", all = TRUE)
row.names(m3) = m3[,1]
m3 = m3[,2:5]
m3[is.na(m3)] = 0
colnames(m3) = c(names[4], names[3], names[1], names[2])
return(as.matrix(m3))
}
```

You can then try it by running this command on the raw counts:

```
norm0 = merge_four(b1_counts,b2_counts,swe_counts,ind_counts,c("Baltic 1","Baltic 2","Sweden", "India
```

You should then see a matrix containing all counts from the four data sets, with each row corresponding to a Pfam family. Next, run the same command on the normalized data and store the output into a variable, called for example `norm1`. The total abundance of mobility domains can then be visualzied using the following command:

```
barplot(colSums(norm1))
```

We can then repeat the normalization procedure, by instead normalizing to the number of 16S rRNA counts in each library. This can be done similarly to the division by total number of reads above:

```
b1_norm2 = b1_counts / 21
```

This time, we won't multiply by a million, as that would make numbers much larger (and harder to interpret).

Follow the above procedure for all the data sets, and finally store the end result from `merge_four` into a variable, for example called `norm2`.

Finally, we will do the same for the third type of normalization, the division by the mapped number of reads. This can, once more, be done as above:

```
b1_norm3 = b1_counts / 22
```

Follow the above procedure for all the data sets, and store the final result from `merge_four` into a variable, for example called `norm3`.

### 5.5.3 A note on saving plots

Note that if you would like to save your plots to a PDF file you can run the command:

```
pdf("output_file_name.pdf", width = 10, height = 10)
```

and then you can just run all the R commands as normal. Instead of getting plots printed on the screen, all the plots will be output to the specified PDF file, and can later be viewed in e.g. Acrobat Reader. When you are finished plotting you can finalize the PDF file using the command:

```
dev.off()
```

This closes the PDF and enables other software to read it. Please note that it will be considered a "broken" PDF until the `dev.off()` command is run!

### 5.5.4 Comparing normalizations

Let us now quickly compare the three normalization methods. As a quick overview, we can just make three colorful barplots next to each other, each representing one normalization method:

```
layout(matrix(c(1,3,2,4),2,2))
barplot(norm0, col = 1:nrow(norm1), main = "Raw gene counts")
barplot(norm1, col = 1:nrow(norm1), main = "Counts per million reads")
barplot(norm2, col = 1:nrow(norm2), main = "Counts per 16S rRNA")
barplot(norm3, col = 1:nrow(norm3), main = "Relative abundance")
```

As you can see, each of these plots will tell a slightly different story. Let's take a closer look at how normalization affect the behavior of some genes. First, we can see if there are any genes that are present in all samples. This is easily investigated by the following command, which takes counts if a value is larger than zero, counts the number of occurences per per row (rowSums), and finally outputs all the rows from `norm1` where this sum is exactly four:

```
norm1[rowSums(norm1 > 0) == 4,]
```

If that didn't give you much luck, you can try if you can find any genes that occur in at least three samples:

```
norm1[rowSums(norm1 > 0) >= 3,]
```

Select one of those and find out its row number in the count table. Hint: `row.names(norm1)` will help you here! Now lets make boxplots for that row only:

```
x = <insert your selected row number here>
layout(matrix(c(1,3,2,4),2,2))
barplot(norm0[x,], main = paste(row.names(norm1)[x], "- Raw gene counts"))
barplot(norm1[x,], main = paste(row.names(norm1)[x], "- Counts per million reads"))
barplot(norm2[x,], main = paste(row.names(norm2)[x], "- Counts per 16S rRNA"))
barplot(norm3[x,], main = paste(row.names(norm3)[x], "- Relative abundance"))
```

You can now try this for a number of other genes (by changing the value of `x`) and see how normalization affects your story.

**Question: Which normalization method would be most suitable to use in this case? Why?**

### 5.5.5 Visualizing differences in gene abundance

One neat way of visualizing metagenomic count data is through heatmaps. R has a built-in heatmap function, that can be called using the (surprise...) `heatmap` command. However, you will quickly notice that this function is rather limited, and we will therefore install a package containing a better one - the `gplots` package. You can do this by typing the following command:

```
install.packages("gplots")
```

Just answer "yes" to the questions, and the package will be installed locally for your user. After installation you load the package by typing:

```
library(gplots)
```

After this, you will be able to use the more powerful `heatmap.2` command. Try, for example, this command on the data:

```
heatmap.2(norm1, trace = "none", col = colorpanel(255,"black","red","yellow"), margin = c(5,10), cex
```

The trace, margin, cexCol and cexRow options are just there to make the plot look better (play around with them if you wish). The `col = colorpanel(255,"black","red","yellow")` option creates a scale from black to yellow where yellow means highly abundant and black lowly abundant. To make more clear which genes that are not even detected, let's add a grey color to that for genes with zero count:

```
heatmap.2(norm1, trace = "none", col = c("grey",colorpanel(255,"black","red","yellow")), margin = c(5
```

You will now notice that it is hard to see the differences for the lowly abundant genes. To aid in this, we can add a variance-stabilizing transform (fancy name for squareroot) to the data:

```
norm1_sqrt = sqrt(norm1)
```

You can then re-run the `heatmap.2` command on the newly created `norm1_sqrt` variable.

Sometimes, it makes more sense to apply a logarithmic transform to the data instead of the squareroot. This, however, is a bit more tricky since we have zeros in the data. For fun's sake, we can try:

```
norm1_log10 = log10(norm1)
heatmap.2(norm1_log10, trace = "none", col = c("grey",colorpanel(255,"black","red","yellow")), margin
```

This should give you an error message. The easiest way to solve this problem is to add some small number to the matrix before the `log10` command. Since we will display this number with grey color anyway, it will in this case, and for this application, matter much exactly what number you add. You can, for example, choose 1:

```
norm1_log10 = log10(norm1 + 1)
heatmap.2(norm1_log10, trace = "none", col = c("grey",colorpanel(255,"black","red","yellow")), margin
```

Before we end, let's also try another kind of commonly used visualization, the PCA plot. Principal Component Analysis (PCA) essentially builds upon projecting complex data onto a 2D (or 3D) surface, while trying to separate the data points as much as possible. This can be useful for finding groups of observations that fit together. We will use the built-in PCA command called `prcomp`:

```
norm1_pca = prcomp(norm1_sqrt)
```

Note that we used the data created using the variance stabilizing transform. There are more sophisticated ways of reducing the influence of very large values, but many times the squareroot is sufficient. We can visualize the PCA using a plotting command called `biplot`:

```
layout(1)
biplot(norm1_pca, cex = 0.5)
```

To see the proportion of variance explained by the different components, we can use the normal plot command:

```
plot(norm1_pca)
```

We want the first two bars to be as large as possible, since that means that the dataset can be easily simplified to two dimensions. If all bars are of roughly equal height, the projection to a 2D surface has caused a loss of much of the information of the data, and we can not trust the patterns in the PCA plot as much.

If we do the PCA on the relative abundance data (normalization three), we can get a view of which Pfam domains that dominate in these samples:

```
norm3_pca = prcomp(norm3)
biplot(norm3_pca, cex = 0.5)
```

And that's the end of the lab. If you have lots of time to spare, you can move on to the bonus excersize, in which we will analyze the 16S rRNA data generated by Metaxa2 further, to understand which bacterial species that are present in the samples.

## 5.6 Bonus exercise: Using Metaxa2 to investigate the taxonomic content

Now when we are familiar to using R, we can just as well use it to go through another type of output generated by the Metaxa2 software. Metaxa2 does classification of rRNA at different taxonomic levels, assigning each read to a taxonomic affiliation only if it is reliably able to (given conservation between taxa etc.) You can read more about Metaxa2 here: http://microbiology.se/software/metaxa2

### 5.6.1 Install Metaxa2

For this exercise to work, you need Metaxa2 installed. If you did not do this earlier, here are the installation instructions again. If you did install Metaxa2 at the beginning of the workshop, you can skip this step and move straight to the next heading!

The code for Metaxa2 is available from http://microbiology.se/sw/Metaxa2_2.0rc3.tar.gz You can install Metaxa2 as follows:

```
# Create a src and a bin directory
mkdir -p ~/src
mkdir -p ~/bin

# Go to the source directory and download the Metaxa2 tarball
cd ~/src
wget http://microbiology.se/sw/Metaxa2_2.0rc3.tar.gz
tar -xzvf Metaxa2_2.0rc3.tar.gz
cd Metaxa2_2.0rc3

# Run the installation script
./install_metaxa2

# Try to run Metaxa2 (this should bring up the main options for the software)
metaxa2 -h
```

If this did not work, you can try this manual approach:

```
cd ~/src/Metaxa2_2.0rc3
cp -r metaxa2* ~/bin/

# Then try to run Metaxa2 again
metaxa2 -h
```

If this brings up the help message, you are all set!

### 5.6.2 Generating family level taxonomic counts

If you have already run Metaxa2 to get the number of 16S rRNA sequences, you can use the output of those runs. Otherwise you need to run the following command on all the raw read data from all libraries:

```
metaxa2 -i <input file> -o <output file> --cpu 16 --align none
```

To get counts on the family level from the metaxa2 output, we will use another tool bundled with the Metaxa2 package; the Metaxa2 Taxonomic Travesal Tool (`metaxa2_ttt`). Take a look at its options by typing:

```
metaxa2_ttt -h
```

In this exercise we are interested in bacterial counts only , so we will use the `-t b` option. Since we are only interested in family abundance (we have too little data to get any good genus or species counts), we will only output the data for phyla, classes, orders and families, that is we will use the `-m 5` option. As input files, you should use the files ending with ".taxonomy.txt" that Metaxa2 produced as output. That should give you a command looking like this:

```
metaxa2_ttt -i <metaxa .taxonomy.txt file> -o <output file> -m 5 -t b
```

Run this command on the taxonomy.txt files from all input libraries. It should be really quick. If you type `ls` you will notice that `metaxa2_ttt` produced a bunch of .level_X.txt files. Those are the files we are going to work with next.

### 5.6.3 Visualizing family level taxonomic counts

To visualize the family level counts, we will once again use R. Fire it up again and load in the count tables from Metaxa2:

```
R
b1_fam = read.table("baltic1.level_5.txt", sep = "\t", row.names = 1)
```

Repeat this procedure for all four data set. If you saved your workspace, the merge_four function should still be available. You can try it out on the taxonomic counts:

```
all_fam = merge_four(b1_fam,b2_fam,swe_fam,ind_fam,c("Baltic 1","Baltic 2","Sweden", "India"))
```

Let's load in the `gplots` library again, and make a heatmap of the raw data:

```
library(gplots)
heatmap.2(all_fam, trace = "none", col = c("grey",colorpanel(255,"black","red","yellow")), margin = c
```

As you will notice, we will need to do some tweaking to fit in the taxonomic data:

```
heatmap.2(all_fam, trace = "none", col = c("grey",colorpanel(255,"black","red","yellow")), margin = c
```

### 5.6.4 Apply normalizations

As you might already have guessed, taxonomic count data suffers from the same biases from, for example, sequencing library size as other gene data. To account for that, we will apply a normalization procedure. Please note that the normalization methods 2 and 3 (number of 16S and number of total matches to database) would in this case be the same. In other words, the both yield the relative abundances of the taxa. We will therefore only look at two normalization procedures in this part of the lab.

First, we will normalize to the number of reads in each sequencing library. Find the note you have taken on the data set sizes. Then apply a command like this on the data:

```
b1_fam_norm1 = b1_fam / 118025 * 1000000
```

That will give you the 16S rRNA counts for the different families per million reads. Do the same thing for the other data sets.

Next, we will do the same for the other type of normalization, the division by the mapped number of reads/total number of 16S rRNA. This can, once more, be done by dividing the vector by its sum:

```
b1_fam_norm2 = b1_fam / sum(b1_fam)
```

Follow the above procedure for all the data sets, and store the final result from `merge_four` into a variable, for example called `fam_norm2`.

### 5.6.5 Comparing taxonomic distributions

Next we will compare the taxonomic composition of the four environments. Let's start out by just using a barplot. To get the different taxa on the x-axis, we will transform the matrix with normalized counts using the `t()` command. But first we need to set the margins to fit the taxonomic names:

```
par(mar = c(25, 4, 4, 2))
barplot(t(fam_norm1), main = "Counts per million reads", las = 2, cex.names = 0.6, beside = TRUE)
```

We can then do the same for the relative abundances:

```
barplot(t(fam_norm2), main = "Relative abundance", las = 2, cex.names = 0.6, beside = TRUE)
```

To only look at families present in at least two samples, we can use the following command for filtering:

```
fam_norm1_filter = fam_norm1[rowSums(fam_norm1 > 0) >= 2,]
barplot(t(fam_norm1_filter), main = "Counts per million reads", las = 2, cex.names = 0.6, beside = TR
```

**Question: Which normalization method would be most suitable to use in this case? Why?**

We can also look at the differences in taxonomic content using a heatmap. As before, we will use the squareroot as a variance stabilizing transform:

```
heatmap.2(sqrt(fam_norm1), trace = "none", col = c("grey",colorpanel(255,"black","red","yellow")), ma
```

Finally, we can of course also use PCA on taxonomic abundances. We will turn back to the `prcomp` PCA command:

```
fam_norm1_pca = prcomp(sqrt(fam_norm1))
```

We can visualize the PCA using the `biplot` command:

```
biplot(fam_norm1_pca, cex = 0.5)
```

To see the proportion of variance explained by the different components, we can use the normal plot command:

```
plot(fam_norm1_pca)
```

**Question: Can you think about any other type of problem with the data we are using now? This problem applies to both kinds of data, but should be particularly problematic with the taxonomic counts...**

Enjoy!