

The Scientific Communication of Comparative Genomics

Teacher: Prof. David W. Ussery

Assistant teacher and exercises: Tammi Vesth

December 9, 2011

Chapter 1

Day 1, overview

1. Set up virtual computer, Biotoools-xubuntu
2. Create a working directory (folder)
3. Copy project specific data to work directory
4. Download genomes from NCBI using GPID numbers, GenBank file format (1-2 genomes)
5. Rename files to organism names (UNIX *mv* command)
6. Create organism specific genomeatlas configuration file
7. Execute organism specific genome atlas file

1.1 Install VirtualBox and start up Biotools-xubuntu computer

See separate instruction manual.

1.2 Introduction to Unix

See separate Unix guide

1.3 Introduction

Command syntax Below is seen an example of how commands will be shown in these exercises. The example task is how to create a folder, *GenBankDNA*. Whenever a word is marked with <> it indicates that a word should be inserted here WITHOUT the <> signs. Below is shown the syntax that will be used in this document and what the command should look like when used.

```
syntax command: mkdir <directory_name>
typed command: mkdir GenBankDNA
```

1
2

1.3.1 Get acquainted with the NCBI webpage and the information it contains.

Genomes have been sequenced from all around the world, and many of these have been submitted to public databases. A database that collects many genome sequence projects is NCBI, National Center for Biotechnology Information <http://www.ncbi.nlm.nih.gov/>. Although there are several institutions that publish genome sequences, NCBI remains a main source of genomic sequence data. The *Genome* resource at this page takes you to the “NCBI Genome Assemblies and Resources” where a list of “Prokaryotic Genome Projects” can be found <http://www.ncbi.nlm.nih.gov/genomes/lproks.cgi>. This page has information about ongoing projects (unfinished genomes) and finished projects (complete genomes).

1.3.2 Project assignments and initial investigation

Each group has been assigned a project with a list of genomes and a focus question. Take a look at the list of genomes that the group has been given. Does the list of genomes fit the research question? It is expected that the group find one or two genome sequences to be added to the initial genome set. Look at the NCBI *lproks* page and find such two genomes. Note the NCBI project id number and download the genomes. From this point on, these genomes will be referred to as *the additional-genomes*.

Copy the the pre-downloaded files to the *Documents* folder!

```
cp -r <SpeciesName> /home/student/Documents/
```

1

Questions:

1. Investigate the folder structure of this package.
2. How many folders are in this package?

3. How many files are found in each sub folder?
4. What type of files are found in the different folders?

1.3.3 Download genomes from GPID numbers

A program has been written which accesses the NCBI webpage, downloads the individual GenBank files and puts them together. The resulting GenBank file contains multiple GenBank files pasted together, one after another. The program is called *getgbk* and uses a GPID or a NCBI accession number as an argument. The output from the program is a GenBank file equivalent to the one that is found on the webpage. Here we will use the program option *-p* which reads the input as a GPID. Another option is *-a* which reads the input as an NCBI accession number, you will need this to make genome atlases for individual chromosomes. The syntax of the program is shown below. Note the Unix usage of the " > " sign, which is a redirection of the output into a file. If this is not included (*getgbk -p <GPID >*), the program will write the output, the GenBank file, to the screen.

```
# Make sure that these new GenBank files are located in the same directory as the pre-downloaded files!
getgbk -p <GPID> > <GPID>.gbk # NOTE: This step should only be performed on the additional-genomes.
getgbk -a <ACC> > <ACC>.gbk # NOTE: This step should only be performed on the additional-genomes.
```

1.4 Investigate GenBank file format

First, look at the GenBank file. Open the file in a text-editor (xubuntu-biotools has a text-editor called *mousepad*).

```
mousepad <name>.gbk
```

In the beginning of the file is the metadata, names, publications, habitat and similar information. The next part is the annotations, *genes* and *CDS (CoDing Sequences)*. In this section the genes are described by their location, direction, note, and *translation*.

Questions:

1. What information is found in the line marked *LOCUS*?
2. How many lines are marked *LOCUS* and what does this number show (Hint: use Unix command *grep* to find the *LOCUS* lines in a file)?
3. Explain the content in the reoccurring fields marked *source*, *gene* and *CDS*.

1.4.1 Re-name GenBank files from GPID numbers to organism names.

To make it easier to recognize files they will now be renamed so they are called an organism name instead of a GPID number. From this point on, *<GPID >* will be replaced with *<name >* and will refer to the organism name the file is given.

```
extractname <GPID>.gbk # NOTE: This step should only be performed on additional-genomes.
# Do NOT change the names of the files in the project package!
```

Note that the files are not moved, but rather, they are copied into a new file. Delete the numbered files using the command *rm*. The new files will from here on be referred to as *< name > .gbk* in the command syntax.

Questions:

1. What could the *extractname* program be doing? Where does the program get the name from?
2. To look at the code, open it in the texteditor *mousepad /usr/biotools/extractname*.

1.5 Genome atlas

Now you will construct a visual representation of the data along with a visual analysis of the DNA structures. A genome atlas is a visual representation of genome properties, genes/proteins and patterns in DNA associated with DNA structures, helix, repeats and so on. A genome atlas can be made from a GenBank file and uses the gene/protein annotations published with the genome DNA sequence. It is important to have only one replicon in your GenBank file (count number of *LOCUS* if you are not sure). The reason for this is that multiple replicons can not be visualized in a single atlas and the program will not execute. In order to construct an atlas, the DNA sequence is scanned for all kinds of patterns. This means that it takes time to prepare the files necessary for a genome atlas. Select around 4 genomes for which to construct atlases. For each of these genomes, investigate if the genomes consists of multiple replicons (use *grep* to locate how many *LOCUS* lines are in the GenBank files). If the genomes have more than one replicon (ex. one chromosome and one plasmid), find the NCBI accession number for the chromosome (found in *LOCUS* line).

```
mkdir GenomeAtlas<acc>          # Create folder to hold the files for the genome atlas, put the NCBI
    accession number in the directory name
cd GenomeAtlas<acc> # Enter the atlas specific folder
getgbk -a <acc> > <acc>.gbk # Download the chromosome as a separate GenBank file.
sed s/XXX/<acc>/g /usr/biotools/genomeAtlas > <name>.genomeatlas.sh
chmod +x <acc>.genomeatlas.sh
./<acc>.genomeatlas.sh
```

Chapter 2

Day 2, overview

1. Extract DNA sequences from the GenBank file
2. Calculate basic statistics for genomes
3. Create table with basic statistics for each genome (Gnumeric, spreadsheet editor on xubuntu)
4. Locate 16S rRNA sequence in DNA
5. Extract the best 16S rRNA sequence from each genome
6. Do complete multiple alignment for all 16S rRNA sequences
7. Construct phylogenetic distance tree from alignment results
8. View tree and save as PDF

2.1 Extract DNA, from GenBank files and save in fasta files.

From the first module, we learned that the GenBank file format contains the raw DNA sequence for a given genome project. In this modules the DNA will be extracted and searched for patterns similar to rRNA sequences. The extraction of DNA from GenBank could in theory be done manually, but there are much faster ways. Here it is done using a small extraction program called *saco_convert* which takes a GenBank file as input and stores the DNA in a FASTA format.

```
saco_convert -I genbank -O fasta <name>.gbk > <name>.fna # NOTE: This step should only be performed on
additional-genomes. 1
```

Look at the file and make sure that it contains DNA in FASTA format (use *head*, *tail*, *cat*, *less* or the *texteditor*). Number of ” > ” FASTA headers should be equal to the number of replicons (chromosomes or plasmids). Count the number of replicons using *grep*:

```
grep -c '>' <name>.fna # NOTE: This step should be performed on all genome DNA files 1
# The above can be wrapped in for-loop to make it faster 2
for x in *fna 3
> do 4
> echo $x >> proteinCounts.txt 5
> grep -c '>' $x >> proteinCounts.txt 6
> done 7
# If you need to run this loop again, delete the proteinCounts.txt file first 8
9
```

Questions:

1. Why is it that “Number of ” > ” FASTA headers should be equal to the number of replicons (chromosomes or plasmids).” for complete genomes?
2. What does the **fna* indicate? (Hint: which files does the loop run through?)

2.2 Calculate basic statistics for published genome.

The next part is an analysis performed on a FASTA formatted file containing complete genomic DNA (<name>.fna), not genes or proteins (<name> proteins.fsa). Calculate the AT content (*Per.AT*), number of replicons (*ContigCount*), deviation of AT across replicons (*StDevAT*), percentage of unknown bases (*Per.Unknowns*) and total size in bp (*TotalBases*).

```
genomeStatistics <name>.fna # NOTE: This step should be performed on all genome DNA files 1
```

Output is by default written to the screen. You can copy the output from the screen window into a spreadsheet. The command line can be used in a *for* loop, but the amount of data written to the screen might be overwhelming. In that case the result can be redirected to an output file.

```
for x in <name1> <name2> <name3> <name4> 1
> do 2
> genomeStatistics $x.fna >> genomeStats.all 3
> done 4
```

Note the " >> " signs, this means "append" instead of redirect. When using redirect, " > ", a file is created if not already found or overwritten if already found. When appending, the output is added if the file is not found or appended if the file is already found. Copy the genome statistics into a spreadsheet (*Gnumeric*).

Questions:

1. Which genome/organism has the highest AT content?
2. Which genome/organism has the lowest AT content?
3. Which genome/organism is the largest?
4. Which genome/organism is the smallest?

2.3 Run algorithm to identify rRNA sequences in DNA.

For identifying rRNA sequences in DNA we will use *rnammer*, a program that implements an algorithm designed to find rRNA sequences in DNA ¹. The program was made by modeling a large number of known rRNA sequences and making them into generalized patterns. These patterns are then used as search models in a new DNA sequence. If a part of the DNA matches the model, the sequence is extracted as a likely rRNA sequence. The help page for *rnammer* has the following description for the program:

```
" RNAmmer predicts ribosomal RNA genes in full genome sequences by utilising two levels of Hidden Markov Models: An initial spotter model searches both strands. The spotter model is constructed from highly conserved loci within a structural alignment of known rRNA sequences. Once the spotter model detects an approximate position of a gene, flanking regions are extracted and parsed to the full model which matches the entire gene. By enabling a two-level approach it is avoided to run a full model through an entire genome sequence allowing faster predictions.
```

```
RNAmmer consists of two components: A core Perl program, 'core-rnammer', and a wrapper, 'rnammer'. The wrapper sets up the search by writing on or more temporary configuration(s). The wrapper requires the super kingdom of the input sequence (bacterial, archaeal, or eukaryotic) and the molecule type (5/8, 16/17s, and 23/28s) to search for. When the configuration files are written, they are parsed in parallel to individual instances of the core program. Each instance of the core program will in parallel search both strands, so a maximum of 3x2 hmmsearch processes will run simultaneously. The input sequences are read from sequence and must be in Pearson FASTA format. "
```

The program is run as follows, specifying the taxonomical kingdom (*bac*) and the type of molecules to search for (*tsu* for 5/8s rRNA, *ssu* for 16/18s rRNA, *lsu* for 23/28s rRNA). The parameter *-f* specifies the name of the output file.

```
rnammer -S bac -m ssu -f <name>.rrna.fsa <name>.fna # NOTE: This step should be performed on all genome DNA files
```

Wrapping all your genomes in a single *for* loop for *rnammer* might, therefore, not a good idea. Make 2-3 loops and wait for each of them to finish before you start the next one.

```
for x in <name1> <name2> <name3> <name4>
do
rnammer -S bac -m ssu -f $x.rrna.fsa $x.fna
done
```

¹Lagesen, K. et al. (2007). RNAmmer: consistent and rapid annotation of ribosomal RNA genes. *Nucleic Acids Research*

Have a look at the FASTA header line for the 16S rRNA sequences and look at the score and the lengths. Use *grep* to get the header lines for each file.

Questions:

1. The output files from RNAmmer are FASTA formatted; count the number of sequences in each file, why are there multiple entries?
2. Does the program only find 16 S rRNA sequences? What type of molecules has the algorithm found?
3. Are there some files that are empty? What does it mean if the file is empty?
4. What kind of information can you get from the header lines of these FASTA files?

2.4 Do multiple sequence alignment of selected 16S rRNA sequences.

The way of comparing the 16 sRNA sequences is to do a multiple sequence alignment. In this course we use *clustal* to align the sequences and find the distance/differences between them. The greater the distance between two sequences the greater the difference between the organisms from which the sequences came. The *rnammer* program finds all possible rRNA sequences in a genome. Some, and indeed many, genomes have more than one copy of this gene. In order to do a 16S rRNA tree you should pick one of these sequences. Here we choose the one that has the highest score according to the *rnammer* models. The program *extractseqs* takes all files in a directory name *something.rrna.fsa* and selects the best sequence within each file (each organism). For some organisms, *rnammer* might not find a sequence with a good enough score. This means that the model does not find a sequence that looks sufficiently like a rRNA sequence. These sequences, and organisms, will be excluded from the analysis.

The sequences will now be evaluated based on fixed criteria for a 16S rRNA sequence. These criteria include length and fitness score to the *rnammer* models. This extraction procedure will only work for 16s rRNA sequences that fulfill the criteria.

```
# The following code workis on all files in the current working directory.  
extractseqs all.rrna.fsa
```

1
2

The output is a FASTA formatted file with rRNA genes in DNA code. Count the number of sequences in this FASTA file (using *grep*). Try using *grep* without the *-c* option and look at what the headers of this FASTA file looks like. Look at the header lines for the selected sequences. The alignment is performed using *clustalw* and the file holding one rRNA sequence from each organism.

```
clustalw all.rrna.fsa
```

1

Next we create a distance tree for the alignment, using a bootstrap value of 1000. Wiki about bootstrapping in statistics:

" In statistics, bootstrapping is a computer-based method for assigning measures of accuracy to sample estimates (Efron and Tibshirani 1994). This technique allows estimation of the sample distribution of almost any statistic using only very simple methods (Varian 2005). Generally, it falls in the broader class of resampling methods. Bootstrapping is the practice of estimating properties of an estimator (such as its variance) by measuring those properties when sampling from an approximating distribution. One standard choice for an approximating distribution is the empirical distribution of the observed data. In the case where a set of observations can be assumed

to be from an independent and identically distributed population, this can be implemented by constructing a number of resamples of the observed dataset (and of equal size to the observed dataset), each of which is obtained by random sampling with replacement from the original dataset. "

When using bootstrapping, different versions of the distance tree will be constructed and each time a branching point will be recorded. With a bootstrap value of 1000, the number on the tree will indicate how many, out of 1000 trees, have this branching. The closer to 1000 the more sure we are of that branch.

```
clustalw all.rrna.fsa -bootstrap 1000
```

Questions:

1. How many sequences are there in the *all.rrna.fsa* file?

2.5 Construct phylogenetic tree from 16S rRNA alignment.

The tree construction is simply a drawing program called *njplot*. Open the bootstrap tree file **.phb* and tick the *Display* setting by clicking *Bootstrapvalues*.

```
njplot all.rrna.phb
```

Try to change the settings for the tree, rooted/unrooted and so on.

Chapter 3

Day 3, overview

1. Genefinding on 1-2 genomes, using Prodigal (prodigalRunner is an implementation of this algorithm)
2. Count number of proteins found in each genome
3. Amino acid and codon usage

3.1 Run genefinding algorithm on extracted DNA (hypothetical genes/proteins).

Now we will run our own genefinding algorithm on the DNA sequence of the genome. This is very often the same that the publishers of the genome has done. The good thing about running one algorithm on all the genomes is that the results will be standardized which is not the case with published annotations. Genefinding will only be done for the additional genomes, as it has already been performed for the precomputed genome set.

```
prodigalrunner <name>.fna # NOTE: This step should only be performed on the additional-genomes. 1
```

Output files include:

```
<name>.gff => raw prodigal output, you will not use this file 1
<name>_prodigal.orf.fsa => protein file in FASTA format 2
<name>_prodigal.orf.fna => gene file in FASTA format 3
<name>_prodigal.gbkb => "fake" GenBank file, you will not use this file 4
```

3.2 Count number of hypothetical genes/proteins, from FASTA files (amino acids).

This procedure is something you have done before. Be aware that you are now working on protein files from your local genefinding. The number of genes/proteins might be different than the numbers you obtained from the published genome data. Compare the numbers you obtained here with the previous numbers for the same genome.

3.3 Count number of proteins.

Look at the FASTA formatted files created by Prodigal (use *head*, *tail*, *cat*, *less* or the *texteditor*). The wiki entry about FASTA says:

```
" In bioinformatics, FASTA format is a text-based format for representing either nucleotide sequences or peptide sequences, in which base pairs or amino acids are represented using single-letter codes. The format also allows for sequence names and comments to precede the sequences. The format originates from the FASTA software package, but has now become a standard in the field of bioinformatics. http://en.wikipedia.org/wiki/FASTA\_format "
```

Underneath here is shown an example of a FASTA file, with a description header and an amino acid sequence. The sequence lines within a FASTA entry (header + sequence) make up the entire sequence of a gene.

```
>SEQUENCE_1 1
MTEITAAMVKELRESTGAGMMDCKNALSETNGDFDKAVQLLREKGLGKAARKADRLAEEG 2
LVSVKVSDDFITIAAMRPSYLSYEDLDMTFVENEYKALVAELEKENEERRRLKDPNKPEHK 3
IPQFASRKQLSDAILKEAEEKIKEELKAQGKPEKIWDNIIPGKMNSFIADNSQLDLSKLTLL 4
MGQFYVMDDKKTVEQVIAEKEKEFGGKIKIVEFICFEVGEGLKKTEDFAAEVAAQL 5
>SEQUENCE_2 6
```

```
SATVSEINSETDFVAKNDQFIALT KDTTAHIQSNSLQSV EELHSSTINGVKFEEYLK SQI
ATIGENLVVRRFATLKAGANGV VNGYIHTNGRVGVVIAAACDSAEVASKSRDLLRQICMH
```

7
8

From this point forward that the number of genes/proteins is equivalent to the number of headers. As such, counting the number of annotated genes in a GenBank file, can be done by counting the number of ">" signs in the corresponding FASTA file. A simple Unix tool for finding sequence patterns in a file is the program *grep*. Here we will use *grep* to count how many times the ">" sign is found in a given FASTA file.

```
grep ">" <name>.orf.fsa

# It is important to remember the " signs, otherwise the sign will be viewed as a output redirect.
# grep will only find the line, with the search-pattern.
# To count the number of times the pattern is found grep has a -c option, which will print the number on
  the screen.

grep -c ">" <name>.orf.fsa
```

1
2
3
4
5
6
7

To wrap this command in a *for* loop it is necessary to add an extra command that will show the filename (which is the organism name). The command is called *echo*. The *for* loop will look like this:

```
for x in <name1> <name2> <name3> <name4>
> do
> echo $x
> grep -c ">" <name>.orf.fsa
> done

# The following will allow you to run a command on all files with the extension $.orf.fsa$ without
  specifying the names of the files.
# The counts will be written to the screen along with the name of the file.

for x in *orf.fsa;
> do
> echo $x
> grep -c ">" $x
> done
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14

3.4 Calculate amino acid and codon usage for hypothetical genes/proteins.

These basic statistics for each genome are visualized in a PDF summary with three plots and a text summary. The calculations have been implemented in a program called *basicgenomeanalysis* and takes gene FASTA files as input. In this setup the gene FASTA files have the suffix **.orf.fna*. Run the analysis as follows, note that the name is WITHOUT the suffix:

```
basicgenomeanalysis <name>_prodigal /usr/bin/gnuplot
```

1

Chapter 4

Day 4, overview

1. Construct pan and core genome plot from hypothetical genes/proteins.
2. BLAST matrix

4.1 Construct pan and core genome plot from hypothetical genes/proteins.

The pancore genome plot is a visual representation of how a pan and a core genome is calculated for a set of genomes. The algorithm is dependent on BLAST and has been implemented in the program *pancoreplot*. First we construct an input file for the *pancoreplot* program, note that is is the number *one* in the *ls* command and a *pipe* character before the *gawk* command.

```
ls -1 *orf.fsa | gawk '{print $1 "\t" $1}' > pancore.list
```

Take a look at the file and note that the look of it (*cat*, *less*, *head*, *tail*). The first column will become the name in the plot image. You can freely edit this column using the texteditor (*mousepad*). The second column, separated by tab, is the filename and must NOT be edited.

```
pancoreplot -keep blastOutPut pancore.list > pancoreplot.ps
```

This program runs a series of BLAST comparisons using the *blastp* program and will take time. In class you should do this for a small set of your genomes. Homework will be to start this plot for all of your organisms and let it run overnight.

4.2 Extract subset genes from pan and core genome plot.

This procedure outputs the genes/gene families in common or complementary between genomes in a coreplot. The directory argument must be of the type created as temporary directory by the coregenome script. If “-o” is unspecified (the default), the program will print one representative for each gene family, selected somewhat randomly. Due to speed issues, the program tries to extract the genes from as few genomes as possible. If ‘-o’ is used, all members of the relevant gene families will be extracted from the genomes specified.

SYNOPSIS

```
ProgName [Options] [Directory  
specificGenes [-i, -c ..... ] blastOutPut
```

DESCRIPTION

```
# NOTE ON '[integers]': Genomes must be specified by their number in the order displayed on the  
coreplot (because that is how they are named in the coregenome script).  
# Individual values can be separated by commas, while ranges can be specified using ':' or '-'.  
# The final value in a range can be omitted, letting the range terminate at the last genome in the set  
(or, if '-l' was specified, whatever genome was given there).  
# The various options can be specified several times and will be interpreted as a request for several  
core genomes simultaneously, one from each subset indicated.  
  
# DEFAULT: If neither '-u' nor '-i' is specified, the default is to select the intersection (i.e. core  
gene families) of all genomes up to and including the last genome in the set (or, if '-l' was  
specified, whatever genome was given there).
```

OPTIONS

```
-c or -cu or --complementary [integers]  
# If set, output only from gene families which are complementary to (i.e. not present in) the union of  
these genomes. See note above on [integers] for how to specify genomes.  
  
-ci or --compinter [integers]  
# If set, output only from gene families which are complementary to the intersection of these genomes -  
i.e. not present in the core of these genomes. See note above on [integers] for how to specify  
genomes.  
  
-d or --dispensable  
# If set, reverses the outputting behavior so that genes formerly to be outputted are discarded and the  
genes which would normally have been discarded are instead outputted. The option is named for its  
ability to output the dispensable/auxiliary genes # instead of the core genes.  
  
-i or --intersection [integers]  
Find intersecting (i.e. common) gene families between these genomes. Accepts same format as '-c'. See  
note above on [integers] for how to specify genomes.  
  
# NOTE: Specifying both '-i' and '-u' is valid, but if the sets overlap, then those genes obeying both  
restrictions will be extracted twice. This is considered a feature and not a bug :-).  
  
-l or --last [integer]  
The index of the last organism from the core genome plot to include in the calculation of the core genome  
. Defaults to the last organism in the plot.  
  
-o or --organisms [integers]  
# The index of the organisms for which to output the genes.  
# If specified the program outputs all genes from selected gene families and specified organism(s).  
# See note above on [integers] for how to specify genomes.  
  
# If unspecified, the program will print one representative for each gene family, selected somewhat  
randomly.  
# Due to speed issues, the program tries to extract the genes from as few genomes as possible.  
  
-s or --slack [integer]  
# Number of genomes a gene is allowed to be missing from and still be considered part of the core  
genome.  
# This would normally only be set if the slack option was used with the 'coregenome' script when the  
input directory was created, and then only to the value used there.
```

```

-u or --union [integers] 29
# Find the union of gene families (i.e. all genes) from these genomes. See note above on [integers] for 30
# how to specify genomes. 31
# NOTE: Specifying both '-i' and '-u' is valid, but if the sets overlap, then those genes obeying both 32
# restrictions will be extracted twice. 33
# This is considered a feature and not a bug :-). 34
-h or --help 35

```

EXAMPLES

```

ProgName -i 1:3,5:7 [Directory] 1
# Gives you the core gene families of genomes 1, 2, 3, 5, 6, and 7. 2
3
ProgName -i 1: [Directory] 4
# Gives the core gene families of all genomes in the set. This is actually the default, used if no 5
# options are given. 6
7
ProgName -i 1,3:5 -c 6: [Directory] 7
# Gives the core gene families of genomes 1, 3, 4 and 5 which are not present in any of the genomes 8
# from 6 and on to the last. 9
# In this command, genome 2 is not considered at all. 10
11
ProgName -i 1:3 -i 5:7 [Directory] 11
# Gives the core genome of organisms 1, 2 and 3 as well as the core genome of 5, 6 and 7. 12
# This is a larger set different from '-i 1:3,5:7' above. 13
14
ProgName -u 1:5 -ci 7:9 -ci 8:10 [Directory] 15
# Gives the part of the pan-genome of organisms 1 through 5 which is neither in the core genome of 7, 8 16
# and 9 or in the core genome of 8, 9 and 10. Yes, subsets can overlap.

```

4.3 How to construct a gene frequency plot from a pan-core genome output.

Find BLAST output reports from pan-core genome plot. Find last group file (group_lastX.dat).

Run following command in the *Terminal*:

```

gawk '{print NF-2}' group_lastX.dat | gawk '{arr[$1]++} END {for(i in arr) print i,arr[i]}' | sort -n > 1
frec.txt

```

Start R

Read in file:

```

frec <- read.table("/username/path/frec.txt", sep=" ", dec=",") 1

```

Create plot:

```

barplot(frec$V2, main="Gene frequency", xlab="Gene occurrence count, genes can be found multiple times in 1
one genome", ylab="Frequency of gene count, how often does a gene get counted x times", ylim=c(0,
max(frec$V2)*1.1))

```

Save plot:

```

dev.print(pdf, file = "/username/path/frec.pdf", width = 15, height=8) 1

```

4.4 Construct BLAST matrix from hypothetical genes/proteins.

A BLAST matrix is a comparison of proteomes (proteins from a genome) used to estimate how many proteins is found in common between two genomes. We will construct a matrix from the **.orf.fsa* files created by *prodigal*. The BLAST matrix algorithm has been implemented in a program called *blastmatrix*. First we will construct an input file for this program. The input file must be of the format *XML* which is a nice computer-reading format but not very friendly to human eyes. A small program called *makebmdest* construct a *XML* file from all the **orf.fsa* files (protein files in FASTA format) in a directory. Make sure that you have the right files in the current working directory. In class you will run this procedure on 3 genomes (< *genome1* >, < *genome2* >, < *genome3* >). Select 3 genomes and create a folder for them:

```
mkdir TestMatrix
# Copy the needed files into the test directory:
cp <genome1>_prodigal.orf.fsa TestMatrix/
cp <genome2>_prodigal.orf.fsa TestMatrix/
cp <genome3>_prodigal.orf.fsa TestMatrix/
# Enter the $TestMatrix$ directory:
cd TestMatrix
# Create BLAST matrix XML file from the protein FASTA files within the TestMatrix directory (current
working directory):
makebmdest . > bmdest.xml
```

The "." indicates that the path to the files is the path to the current working directory. If this dot is not included the code will create a *XML* file with no references to the actual protein files. the BLAST matrix is then generated using the *blastmatrix* program with the file *bmdest.xml* as input.

```
blastmatrix bmdest.xml > blastmatrix.ps
```

While the matrix is running you will encounter an error message about a MySQL database. This error is related to the original usage of the script, where a CBS database was searched to see if the calculation had been made before. Because the biotools-xubuntu system does not have a common database the program complains that it can not find it.

Error messages related to "illegal division by zero" can be due to faulty genefinding (*prodigal*). In some cases, the genefinder will locate a gene but then afterwards not include that gene for other reasons. This will cause the program to make a header for a gene but no sequence. If you encounter this problem, run the following commands in the directory where you have the amino acid FASTA files.

```
for x in *.proteins.fsa
do sacco_convert -I Fasta -O tab $x > $x.tab
sed -r '/\t\t\t/d' $x.tab > $x.temp.tab
sacco_convert -I tab -O Fasta $x.temp.tab > $x
rm *tab
end
```

In class you should do this for a small set of your genomes. Homework will be to start this plot for all of your organisms and let it run overnight.

4.5 Convert **ps* file to **pdf*.

To use your figures in a paper or presentation, it is handy to have the picture as a **pdf*. For this you can use the program *ps2pdf*. The syntax is simple:

```
ps2pdf -dEPSCrop file.ps file.pdf
```

You might experience that the **pdf* only contains part of the figure. If this is the case, right click the **ps* file and open it in *mousepad*. Find a line that looks something like `%%BoundingBox: 0 0 2212 1110`. The numbers correspond to the size of the picture, and are coordinates *llx lly urx ury* (lower left corner (x,y), upper right corner (x,y)). Change the coordinates, save the **ps* file and run the *ps2pdfagain*. It takes a couple of tries the first time, but then it is easy enough to set the size of the picture.