Comparative Bacterial Genomics

Teacher: Prof. David W. Ussery
Assistant teacher: Tammi Vesth

May 15, 2013

Contents

Day 1

Overview

- Set up a virtual computer, CMG-biotools
- Introduction to basic UNIX commands
- Create an organism specific project folder
- Download a genome sequence from NCBI (one genome)
- Extract the organism name from a GenBank file
- Extract DNA, genes and proteins from a GenBank file
- Organize data
- Annotate genomes, gene-finding
- Remove empty files

1.1 Set up CMG-biotools

See separate manual.

1.2 Introduction to Unix

UNIX is mainly a command line interface. This means that you type in the commands you want executed. In the beginning this might seem inferior to the modern windows "point-and-click", but as you do the exercises, the power of this interface to be able to handle lots of data will become obvious. To type in Unix commands, open the application called `Terminal Emulator` on CMG-biotools. You fin the application in the menu or at the bottom panel on the desktop, the icon is a `black screen`.

1.2.1 Unix - Some useful concepts

This is a a brief overview of some useful concepts in unix.

- `The Shell.` Although Unix has a graphical interface called X Windows, it is often easier and quicker to run programs by typing commands into a terminal window. Access to unix from other operating systems is usually conducted through a terminal client e.g. Putty for windows.
- `Users.` All programs are run as a specific user, so you have to log into the system as that user with a password.
- `Files and processes.` Everything is a file or a process and the input and output from files and processes can be sent to each other (see pipes and redirection).
- `Permissions.` All files, directories and programs have access permissions. A user cannot see the contents of a file or run a program unless the permissions allow.

1.2.2 A brief overview of the command line shell

This is what is run when you open a terminal window. It provides a lot of information and tools to help you run programs.
`The command prompt:` When you open a terminal, the text at the bottom of the screen next to the cursor will look something like this:

```
interaction[maq]:/home/projects/MicrobialGenomicsGroup>
```

This is useful because it tells you who you are and where you are. It can be configured in different ways but the example above shows the machine, the username in square brackets `'[ ]'` and the current directory after the colon `':'`.

1.2.3 Command syntax

The exercises presented in the following modules use the UNIX terminal and command-line syntax to perform bioinformatic analysis. Each analysis involves a number of command-line calls and each analysis will be introduced using the same example syntax. Below is seen an example of how commands will be shown in these exercises. The example task is how to create a folder called using the directory name `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. The dollar sign at the beginning of the line indicates that the following command should be typed on a command-line in the `Terminal` program.

Listing 1.1: Command syntax - create directory example

```
# Syntax:
$ mkdir <directory_name>
# Example:
$ mkdir GenBankDNA
```

The symbol "#" indicates a comment or note and should not be typed into the terminal.
`Command line history and auto-completion:` Previously entered commands can be edited or executed again using the up arrow key.

1.2.4 Directories and the file system

The shell logs you into a directory in the file system. There are some rules for about the file system.

- `Files, directories and executables are case sensitive:` This means that `x.txt` and `X.txt` are two different files.
- `Path delimiter:` The unix shell uses the forward slash `'/'` to separate files and directories. There are some special characters which are used when defining the location of a file, directory or program.
- `The root directory:` The root directory is defined by the single slash `'/'` and represents to the first node of the directory tree. It is similar to `'C:'` on a windows machine.
- `The ''.'' directory:` The ''.'' character is used to define the current directory when it is part of a file path. If you wanted to copy a file to the current directory, you would type "`cp <file> .`" where <file> is the name of the file you want to copy, and ". means copy it to the current directory.
- `The home directory:` Most users are assigned a home directory where files can be created. This can be referenced using the `'~'` character.

Examples on absolute versus relative paths:

```
/usr/bin/perl   #   Absolute paths are defined from the root directory e.g.
./script.pl     #   Relative paths can be defined from the current directory e.g.
~/script.pl     #   Relative paths can be defined from the home directory e.g.
```

Useful command to assist getting around the filesystem:

```
pwd             #   prints the current, or "working" directory
cd /usr/bin     #   changes the current working directory to a new location
mkdir newdir    #   creates a new directory
rmdir newdir    #   removes a directory (directory must be empty)
```

1.2.5 Working with files

Files reside in directories and can contain text or binary information. Files can be created, copied, moved, renamed and deleted with the following commands.

- `ls`: lists the contents of directories. Run just as `ls` the command lists all the contents without any other information. More information can be gained by supplying some arguments.

```
ls -l       #   list showing permissions, user, group. size, modification date and <filename>
ls -lh      #   as above but file sizes are printed in human readable form
ls -lhS     #   as above but sort the results by file size in descending order
ls -lhSr    #   as above but in ascending order
ls -lR      #   list the contents of all directories recursively from the current directory
ls -lt      #   list files in ascending order of last modified
ls -ltr     #   as above but in descending order
```

- `touch`: used with a <filename>. If the file does not already exist, a new one is created. Otherwise the date of the file is changed to the current time.

```
touch <filename>
```

- `cp`: copies <filename1> to <filename2> or into a directory and leaves the original file untouched.

```
cp <filename1> <filename2>  # Copy one file into another
```

- `mv`: moves a file from one to another and deletes the original file. It is used for renaming files. It can also be used to recursively move directories.

```
mv <filename1> <filename2>  # Re-name file
mv <file> <directory>/  # Move file into directory
mv <dir1> <dir2>    # Re-name directory
```

- `rm`: deletes a file, can also be used to delete a directory and the contents. Use with care.

```
rm file
rm -r dir
```

- `File name advice`: It is best not to use spaces or special characters such as `‘ ’ < > @` in <filename>s. Underscores `'_'` and hypens `'-'` are fine.

1.2.6 Reading the contents of text files

The contents of text files (but not binary files) can be read quite easily through the terminal.

- `cat`: appends the contents of one file into another

```
cat <file1> # prints out the entire contents of <file1> to the screen
cat <file1> <file2> # adds the contents of <file1> to the end of <file2> (print to screen)
```

- `more`: shows the contents of a text file. Press `'q'` to return to prompt

```
more <filename>
```

- `head`: shows the first few lines of the given file(s). A hyphen and number can be passed to determine how much of the file is shown

```
head <filename> # Prints the first 10 lines of the file.
head -5 <filename>  # Prints the first 5 lines of the file.
```

- `tail`: shows the last few lines of the given file(s).

```
tail <filename>
tail -n 5 <filename> # -n and a number can be passed to determine how much of the file is shown
```

- `grep`: Searches through text files for a search term and print matching lines. `grep` is a complex command and has many options. Try looking at the manual page for `grep` (`man grep`).

```
grep searchword <file1> <file2>     #   to search two files for a searchword
grep -v <searchterm> <file1> <file2>    #   to search two files to print lines excluding a
    searchword
grep -c <searchterm> <file1> <file2>    #   To count the number of matches per file
grep -c "<searchterm>" <file1> <file2>  #   Search term is a symbol, use " around the term
grep -c "<" <file1> <file2> #   Search term is a symbol, use " around the term
```

- `wc`: counts the number of words or lines a file contains

```
wc -w <filename>    #   print the word count of a file
wc -l <filename>    #   print the line count of a file
```

- `wildcards`: A number of characters are interpreted by the Unix shell before any other action takes place. These characters are known as wildcard characters. Usually these characters are used in place of <filename>s or directory names.

```
*   #   An asterisk matches any number of characters in a <filename>, including none
?   #   The question mark matches any single character
[ ] #   Brackets enclose a set of characters, any one of which may match a single character
-   #   A hyphen used within [ ] denotes a range of characters.
~   #   A tilde at the beginning of a word expands to the home directory
    #   If another user name is appended to the character, it refers to that user's home directory
```

Here are some examples:

```
cat c*          #   Displays any file whose name begins with c including the file c, if it exists.
ls *.c          #   Lists all files that have a .c extension.
cp ../rmt?.     #   Copies files in parent directory to working directory if file matches:
                #   four characters long and begins with rmt.
ls rmt[34567]   #   Lists every file that begins with rmt and has a 3, 4, 5, 6, or 7 at the end.
ls rmt[3-7]     #   Does exactly the same thing as the previous example.
ls ~            #   Lists your home directory.
ls ~hessen      #   Lists the home directory of the guy1 with the user id hessen.
```

1.2.7 Invoking executables

Some files can be marked as executable which means they can run as programs and perform tasks. Executables residing in the `PATH` directories can be invoked without including the path to the executable. If a program called `perl` is found in the folder `/usr/bin` and the folder `/usr/bin` is part of `PATH`, the program can be called without giving the absolute or relative path to the program. Non-`PATH` program must be called giving an absolute or relative path.

```
perl myscript.pl            #   Program found in PATH
/usr/bin/perl myscript.pl   #   Non-PATH program
```

1.2.8 Redirection and pipes

Most unix processes write their output to the standard output (the terminal screen) and take their input from standard input (they keyboard). There is also a standard error which is usually printed to the terminal screen. These inputs and outputs can be redirected to files. The standard output can be redirected using the '>' character

```
ls -l >> list.txt               #   Appending to a file
echo 'hello' > list.txt     #   Redirecting input
sort < filewithdata.txt         #   Sorts lines in file
sort < input.txt > output.txt   #   Both at the same time
```

Pipes ('|') allow processes to direct output directly to other programs. Pipes can be put together to allow complex "pipelines" of commands to be put together. The output of command 1 can be passed to command 2 as follows:

```
ls -1 | grep -v '*.txt' | grep -c '*.coli*'
```

This pipeline will first list the <filename>s in a directory, then exclude (`grep -v`) the once with the file extension (`.txt`) and then count (`grep -c`) the once with he extension (`.coli`).

1.2.9 File system permissions

File system permissions ensure that file contents or executables can only be examined or invoked by users with the correct authorization. They can often also be a source of problems when using data or programs created by others where you can't access a file or directory due to the permissions set. To view permissions type `'ls -l'` in a directory containing some files, the output will look like this:

```
-rw-r----- 1 user1 cdrom 11802 Jul  9 10:02 file.txt
```

The 10 character string `'-rw-r---'` describes the permissions. The hyphen indicates that the permission has not been granted and `'r'` indicates read permission, `'w'` indicates write and `'x'` indicates that the file can be executed. There are three sets of `'r','w','x'` or `'-'` to control access by the user to whom the file belongs, members of the same group and anyone else. e.g. `'-rwxrwxrwx'` means anyone can read, write and execute the file

```
whoami  #   will display your username
groups  #   will display the groups you are a member of
chmod   #   file permissions can be changed using the chmod command
```

The `chmod` command takes a complex set of arguments. The user, group or other are represented by `'u'`, `'g'` and `'o'`. The letter `'a'` is used to represent all whether permissions are granted or revoked is determined using `'+'` or `'-'` respectively read, write or execute permissions are represented by `'r'`, `'w'` and `'x'`.

```
chmod go-wx data.txt    #   remove write and execute permissions for the group and others try
chmod a+rw data.txt     #   to give everyone read and write permissions
```

Useful pages with Unix introductions include: `http://www.ee.surrey.ac.uk/Teaching/Unix/`

1.2.10 File-formats

The files you will be dealing with here are mostly plain text file, but some formats will be used. Among these is the GenBank and FASTA format (examples shown below). The GenBank sequence format is a rich format for storing sequences and associated annotations.

Listing 1.2: GenBank file format example

```
LOCUS       CAA89576                109 aa            linear   PLN 11-AUG-1997
DEFINITION  CYC1 [Saccharomyces cerevisiae].
ACCESSION   CAA89576
VERSION     CAA89576.1  GI:1015707
DBSOURCE    embl locus SCYJR048W, accession Z49548.1
KEYWORDS    .
SOURCE      Saccharomyces cerevisiae (baker's yeast)
  ORGANISM  Saccharomyces cerevisiae
            Eukaryota; Fungi; Ascomycota; Saccharomycotina; Saccharomycetes;
            Saccharomycetales; Saccharomycetaceae; Saccharomyces.
REFERENCE   1  (residues 1 to 109)
  AUTHORS   Huang,M.E., Chuat,J.C. and Galibert,F.
  JOURNAL   Unpublished
REFERENCE   2  (residues 1 to 109)
  AUTHORS   MIPS.
  TITLE     Direct Submission
  JOURNAL   Submitted (25-SEP-1995) Data collected by MIPS on behalf of the
            European yeast chromosome X sequencing project. MIPS at the
            Max-Planck-Institut fuer Biochemie, Am Klopferspitz 18a D-82152
            Martinsried, FRG; E-mail: Mewes@mips.embnet.org
FEATURES             Location/Qualifiers
     source          1..109
                     /organism="Saccharomyces cerevisiae"
                     /db_xref="taxon:4932"
                     /chromosome="X"
     Protein         1..109
                     /name="CYC1"
     CDS             1..109
                     /gene="CYC1"
                     /coded_by="Z49548.1:954..1283"
                     /note="ORF YJR048w"
                     /db_xref="GOA:P00044"
                     /db_xref="SGD:S0003809"
                     /db_xref="UniProtKB/Swiss-Prot:P00044"
ORIGIN
        1 mtefkagsak kgatlfktrc lqchtvekgg phkvgpnlhg ifgrhsgqae gysytdanik
       61 knvlwdennm seyltnpkky ipgtkmafgg lkkekdrndl itylkkace
//
```

In bioinformatics, FASTA format is a text-based format for representing either nucleotide sequences or peptide sequences, in which nucleotides 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. A sequence in FASTA format begins with a single-line description, followed by lines of sequence data. The description line is distinguished from the sequence data by a greater-than (">") symbol in the first column. The word following the ">" symbol is the identifier of the sequence, and the rest of the line is the description (both are optional). There should be no space between the ">" and the first letter of the identifier. It is recommended that all lines of text be shorter than 80 characters. The sequence ends if another line starting with a ">" appears; this indicates the start of another sequence. A simple example of one sequence in FASTA format:

Listing 1.3: FASTA file format example

```
>gi|5524211|gb|AAD44166.1| cytochrome b [Elephas maximus maximus]
LCLYTHIGRNIYYGSYLYSETWNTGIMLLLITMATAFMGYVLPWGQMSFWGATVITNLFSAIPYIGTNLV
EWIWGGFSVDKATLNRFFAFHFILPFTMVALAGVHLTFLHETGSNNPLGLTSDSDKIPFHPYYTIKDFLG
LLILILLLLLLALLSPDMLGDPDNHMPADPLNTPLHIKPEWYFLFAYAILRSVPNKLGGVLALFLSIVIL
GLMPFLHTSKHRSMMLRPLSQALFWTLTMDLLTLTWIGSQPVEYPYTIIGQMASILYFSIILAFLPIAGX
```

1.2.11 Exercises

1. Use `gedit` to create a file `mycommands.txt` where you write all commands and observations you do in the following exercises.
2. First list the files in the directory
3. Create a file using `touch filename`, open the file using `gedit` and add 3 lines of text to the file
4. Use `head`, `tail` and `wc` to verify the content of the file
5. Copy the file into a new file called `file.copy`
6. Verify that the two files are the same
7. Delete `file.copy`
8. Make a directory `test` and move your file to it
9. Make a directory `data` and move your file to that instead
10. Remove `test` directory
11. Change the directory to data and confirm that you succeeded. Go one directory back again
12. Make three new directories `newtest` - one inside the other, like a russian doll. Your prompt should end up looking something like this:
    `student@student-VirtualBox: /Unix/newtest/newtest/newtest$`
13. Move the data directory to the innermost `newtest` directory.
14. Move to the `data` directory, the prompt should now look something like this:
    `student@student-VirtualBox: /Unix/newtest/newtest/newtest/data$`
15. Confirm that your file has moved along with the data directory
16. Copy the three files to your home (your top directory) (`/home/student/`)
17. Remove all `newtest` directories and `data`
18. Use `grep` to get the second line of your file

1.3 GenBank

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 GenBank [**?**], which is part of the National Center for Biotechnology Information (NCBI) in the U.S. `http://www.ncbi.nlm.nih.gov/`. Although there are several institutions that publish genome sequences, NCBI remains a major source of genomic sequence data. The website `http://www.ncbi.nlm.nih.gov/genome/browse/` lists thousands of Prokaryotic genome sequence projects. Each genome replicon is represented by a unique ID accession number, which is the same across major databases in the US (GenBank) Europe (EMBL) and the DNA Database of Japan (DDBJ). These three databases share the same accession numbers, and is called INSDC, International Nucleotide Sequence Database Collaboration. These numbers can be used to download individual genome sequences for a given organism. The NCBI page has information about on-going projects (unfinished genomes) and finished projects (complete genomes). The sequences handed out during this course were downloaded from this website using INSDC numbers.
For this course, 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 your group has been given. It is expected that each group find at least one additional genome sequences to be added to the initial genome set. Note the GenBank (INSDC) accession number and download the genome. The additional genome will be analyzed along with the remaining sequences. Copy the folder fitting the research question from the teacher (USB stick or other memory device). Use the file-manager to drop and drag the folder on to the CMG-biotools system. Use the basic UNIX commands to move around the folders and look at the content (`cd`, `ls -l`, `head`, `tail`, `cat`). Note have specific file extensions (`*.gbk`, `*.fsa` and so on) are used to indicate the content of the files. These are not strict rules, but they help you keep track of the data.

1.3.1 Exercises

1. What type of files have you been supplied? (Use `head`, `tail` and `gedit`)
2. Does the list of genomes fit the research question?

1.3.2 Download genomes from GenBank

A program has been written which accesses the NCBI webpage, downloads the individual GenBank files from the INSDC numbers. The program is called `gbk_get` and uses a GPID or an NCBI accession number as an argument. The `gbk_get` script uses the Entrez E-utils programmatic interface made available by the NCBI to fetch sequence data. 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 `-s` which reads the input as a INSDC number. 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, the program will write the output, the GenBank file, to the screen.

Listing 1.4: GenBank - download file from NCBI

```
# Syntax:
$ gbk_get -a <INSDC number> > file.gbk
# Example:
$ gbk_get -a AE000511 > AE000511.gbk
```

1.3.3 Obtain data from GenBank files

At this point you should have a pre-downloaded folder with GenBank files along with an additional file which you downloaded as described above. You shall now investigate the GenBank file format (file extension: `*.gbk`) Open the file in a text-editor, either from the file-manager (click `Home` on the desktop) by clicking the file or in the `Terminal` application calling the program `gedit`.

Listing 1.5: Open and investigate GenBank file

```
# Syntax:
$ gedit <INSDC>.gbk
# Example:
$ gedit AE000511.gbk
```

In the beginning of the file is the metadata, which contains 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`.

1.3.3.1 Exercises

1. Download one genome from GenBank.
2. What information is found in the line marked `LOCUS`?
3. How many lines are marked `LOCUS` and what does this number show (use `grep -c`)?
4. Explain the content in the re-occurring fields marked `source`, `gene` and `CDS`.

1.3.4 Extract organism name

To make it easier to recognize files they will now be re-named so they are called an organism name instead of a `INSDC` number. This procedure has already been performed on your large dataset, and you should only run this program on your newly downloaded GenBank file! From this point on, `INSDC` will be replaced with `name` and will refer to the organism name the file is given.

Listing 1.6: GenBank extract - organism name

```
# Syntax:
$ gbk_ExtractName <INSDC>.gbk
# Exampel:
$ gbk_ExtractName AE000511.gbk
# The following file is produced:
Helicobacter_pylori_26695_ID_AE000511.gbk
```

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.

1.3.4.1 Exercises

1. Extract name from one GenBank file
2. What could the `gbk_ExtractName` program be doing? How does the program create the name?
3. To look at the code, open it in the text-editor (`gedit /usr/biotools/gbk_ExtractName`).

1.3.5 Extract DNA

Further analysis of the genomes sequences requires extracting the DNA from the GenBank file. This procedure has already been performed on your large dataset, and you should only run this program on your newly downloaded GenBank file! This can be done using a program called `saco_convert` [?], which, as the name implies, converts one file format into another. Below is shown the syntax of the program (note that the length of the name makes the line wrap around, but the command is still one line):

Listing 1.7: GenBank extract - DNA

```
# Syntax:
$ saco_convert -I genbank -O fasta <name>.gbk > <name>.gbk.dna
# Exampel:
$ saco_convert -I genbank -O fasta Helicobacter_pylori_26695_ID_AE000511.gbk >
    Helicobacter_pylori_26695_ID_AE000511.gbk.dna
```

The file extension is now `*.gbk.dna`, illustrating the the file contains DNA extracted from a GenBank file. You shall now try to run this procedure on all the GenBank files in the `GBK` folder. This can be done using a so called `for-loop`, which runs a specific command a number of times in stead of one. Below is shown to versions, first a `Trial` to illustrate how the loop works and then a `Example` of how the loop looks for `saco_convert`. First try the `Trial`, type `for x in *gbk` on the command-line, this will cause a `>` sign to appear. Type the next commands, and finish each line with `Enter`. The word `done` tells the `Terminal` that the loop is now over and executes the commands typed within the loop. Read the below explanations carefully before you type and do the trials a couple of times before you go on to the `saco_convert` loop.

Listing 1.8: `Introduction - for-loop`

```
# Trial:
$ for x in *gbk
> do
> echo $x
> done
Neisseria_gonorrhoeae_FA_1090_ID_AE004969.gbk
Neisseria_gonorrhoeae_NCCP11945_ID_CP001050.gbk
Neisseria_gonorrhoeae_TCDC-NG08107_ID_CP002440.gbk
Neisseria_meningitidis_053442_ID_CP000381.gbk
Neisseria_meningitidis_8013_ID_FM999788.gbk
Neisseria_meningitidis_alpha14_ID_AM889136.gbk
Neisseria_meningitidis_alpha710_ID_CP001561.gbk
Neisseria_meningitidis_FAM18_ID_AM421808.gbk
......
```

The command `echo` simply writes something to the screen, try typing `echo hello world`. As is seen above, the program `for` looks at the files matching some match criteria, in this case, files with the suffix `*.gbk`. Each of these files, the name of the file, is used as a value of `x` in the loop. More elaborate patterns can also

be used, like `Neisseria_gonorrhoeae*gbk` if you only want to run the loop on a subset of files. Below is shown the loop for extracting DNA from several GenBank files. Note how the extension `*.dna` is added to the value of `x`, which is the GenBank filename. The resulting filename will have the extension `*.gbk.dna`.

Listing 1.9: `GenBank extract - DNA in for-loop`

```
# Example:
$ for x in *gbk
> do
> echo $x
> saco_convert -I genbank -O fasta $x > $x.dna
> done

# Alternative writing
$ for x in *gbk; do saco_convert -I genbank -O fasta $x > $x.dna; done
```

None of the files generated above should be empty as this would indicate that no sequences are found in the GenBank file or that the program is not managing to find the DNA. Verify that your files are not empty using `ls -lh`. Look at the file and make sure that it contains DNA in FASTA format (use `head`, `tail` or `gedit`). Number of ">" FASTA headers should be equal to the number of replicons (chromosomes or plasmids).

1.3.5.1 Exercises

1. Extract DNA from all GenBank files
2. Count the number of `LOCUS` tags in each GenBank file (use `grep -c`)
3. Count number of FASTA headers in the `*.dna` file (use `grep -c`)

1.3.6 Extract genes and proteins + gene-finding

From the initial investigations of the GenBank files, you have probably seen that some files contain genes and proteins. These data are the result of 'gene-finding', where the DNA sequence has been analyzed and searched for possible genes. For some genes there might be some additional experimental verification, but many (most) genes are just predictions. In the following we will extract the nucleotide sequences as well as the corresponding amino acid sequences. The programs is using `BioPerl` modules[**?**] for the handling of GenBank formats. The code is called `gbk_ExtractGeneProt` and the output is two FASTA formatted text files, one for the genes and one for the proteins.

Listing 1.10: GenBank extract - genes and proteins

```
# Syntax:
$ gbk_ExtractGeneProt <name>.gbk
# Example:
$ gbk_ExtractGeneProt Helicobacter_pylori_26695_ID_AE000511.gbk
```

For the genomes/replicons with no published annotation you will run local gene-finding. Gene finding is performed using the program Prodigal[**?**]. The program is wrapped into a formatting program called `prodigalrunner`. The program reformats the raw output of Prodigal to FASTA formatted open reading frames, DNA and amino acids, along with a draft of a GenBank file and a raw general feature formatted

file, a `*.gff` file. The Prodigal program allows for different parameter modifications, including training (`prodigalrunner -t <organism>`) of the gene finder using given data. This feature increases the computation time of the algorithm, but for less known organisms this feature might improve gene finding. It should be noted that the default behavior when encountering N's is not changed - the program treats runs of N's as masked sequence and does not build genes across them. The CMG-Biotools system also comes with the native Prodigal program, which can be used as published[**?**]. Identify the files that are empty and for those DNA files, run the following command:

Listing 1.11: GenBank extract - gene-finding

```
# Syntax:
$ prodigalrunner <DNA FASTA file>
# Example:
$ prodigalrunner Neisseria_gonorrhoeae_FA_1090_ID_AE004969.gbk.dna
# The following files are produced:
Neisseria_gonorrhoeae_FA_1090_ID_AE004969.gff    #   raw prodigal output, you will not use this file
Neisseria_gonorrhoeae_FA_1090_ID_AE004969_prodigal.gbk  # "fake" GenBank file, you will not use this file
Neisseria_gonorrhoeae_FA_1090_ID_AE004969_prodigal.orf.fna  # gene file in FASTA format
Neisseria_gonorrhoeae_FA_1090_ID_AE004969_prodigal.orf.fsa # protein file   in FASTA format
# Remove un-needed files:
$ rm *gff
$ rm *prodigal.gbk
```

Move the gene FASTA (`*.fna`) and protein FASTA (`*.fsa`) to the appropriate folders. Now you can remove the GenBank gene and proteins files that were empty. Move into the folders where the gene FASTA (`*.fna`) and protein FASTA (`*.fsa`) files are stored and run the following commands. Verify the files that will be deleted by first running the `Display` command.

Listing 1.12: Remove empty files

```
# Display empty files:
$ find . -type f -empty
# Remove empty files:
$ find . -type f -empty -exec rm {} \;
```

For each of the non-empty gene and protein FASTA files, count the number of sequences and store the results in a file. These numbers describe the size of the proteome for each chromosome/plasmid and genome. The number of sequences in the protein/gene files should be the same, as all genes should be translated.

Listing 1.13: Count number of proteins/genes - loop

```
for x in *fna
> do
> echo $x
> echo $x >> proteinCounts.txt
> grep -c ">" $x >> proteinCounts.txt
> done
# If you need to run this loop again, delete the proteinCounts.txt file first
```

1.3.6.1 Exercises

1. Extract genes and proteins from all GenBank files

2. Some of the gene and protein files might be empty (use `ls -lh` to verify), can you think of a reason why?
3. Remove empty files (use `find`)
4. Make sure that all files are put in a folder corresponding to file type, for example, make a folder called `FSA` and move all files with the extension `*.fsa` to that folder.

Day 2

Overview

- Calculate basic statistics for genomes.
- Construct a genome atlas.
- Create a table with basic statistics for each genome.
- Locate the 16S rRNA sequence in the chromosomal DNA.
- Extract the best 16S rRNA sequence from each genome.
- Do a complete multiple alignment for all 16S rRNA sequences.
- Construct a phylogenetic distance tree from alignment results.

2.1 Calculate basic statistics

The next part is an analysis performed on a FASTA formatted file containing complete genomic DNA (`*.dna`), not genes or proteins. 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`).

Listing 2.1: Calculate basic genomic DNA statistics

```
# Syntax:
$ stats_genomeDNA <name>.gbk.dna
# Example:
$ stats_genomeDNA Neisseria_meningitidis_Z2491_ID_AL157959.gbk.dna
Filename    TotalBases: Per.AT: StDevAT:    ContigCount:    Per.Unknowns:   Per.LargestSeq  N25 N50 N75
Neisseria_meningitidis_Z2491_ID_AL157959.gbk.dna    2184406 48.19   0.0000  1   0.0000  100.000 2184406
    2184406 2184406
```

Output is by default written to the screen. You can run the command in a `for`-loop and store the data from each genome in a text file.

Listing 2.2: Calculate basic genomic DNA statistics - loop

```
$ for x in *.gbk.dna
> do
> echo $x
> stats_genomeDNA $x >> genomeStats.all
> done
```

Note the ">>" signs, for appending to a file. 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).

2.1.0.2 Exercises

1. Calculate basic statistics for all `*.gbk.dna` files and store results in file
2. Which genome/organism has the highest AT content?
3. Which genome/organism has the lowest AT content?
4. Which genome/organism is the largest?
5. Which genome/organism is the smallest?

2.2 Genome atlas

The genome atlas presented here is an implementation of the atlas presented earlier by Jensen et al. 1999. Below is a short description of each of the parameters shown in the genome atlases. Color scales for all parameters follow the same system. The DNA sequence is read and an output file is generated for the various calculated parameters. For each nucleotide in the genome a numerical value is calculated. This file is then read by the `GeneWiz` program, which calculates the average and standard deviation for each parameter, if the average value of the window is more than 3 standard deviations on either side of the overall average the window is maximally colored. In order to plot the data on a circular map a "window size" is used for longer genomes, which effectively smooths the data for better graphics. For the parameters `Stacking Energy`, `Position Preference` and `Intrinsic Curvature`, the window is 0.002 x genome length. The window is 0.001 x genome length for `Percent AT` and `GC skew`. Each of these are calculated separately, wrapped into a pipeline and visualized in a circular plot, called an atlas. The gene annotations are taken directly from a GenBank coding regions; if no such information is found the `CDS-/+` lanes will be blank. The following lists explanations to each of the lanes in a genome atlas: `Percent AT` is the percent of A's and T's in the genome. `GC skew` is calculated as *((G-C)/(G+C))*, with a window size of 10 000 bp and is useful for determining the origin and terminus of replication [**?**, **?**]. `Global Direct Repeats` and `Global Inverted Repeats` refer to a sequence that is present in at least two copies on the same or opposite strands, respectively. `Intrinsic Curvature` is a measure of DNA curvature and is calculated using the CURVATURE program [**?**, **?**]. The values are scaled from 0 (e.g. no curvature) to 1, which is the curvature of DNA when wrapped around the nucleosome. `Stacking Energy` is derived from the di-nucleotide values provided by Ornstein et al. [**?**]. The scale is in kcal/mol, and the di-nucleotide values range from -3.82 kcal/mol (will unstack easily) to -14.59 kcal/mol (difficult to unstack). A positive peak in base-stacking (i.e., numbers closer to zero) reflects regions of the helix which would de-stack or melt more readily. Conversely, minima (larger negative numbers) in this plot would represent more stable regions of the chromosome. `Position Preference` is a measure of preferential location of sequences within nucleosomal core sequences [**?**]. The tri-nucleotide values range from essentially zero (0.003, presumably more flexible), to 0.28 (considered rigid). Since very few of the tri-nucleotide have values close to zero (e.g. little preference for nucleosome positioning), this measure is considered to be more sensitive towards the low ("flexible") end of the scale.

Now you will construct a genome atlas (See Figure 3.2). 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 cannot 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. For each genome, investigate if the genome contains more than replicon (use `grep` to locate how many `LOCUS` lines are in the GenBank files). Move to the directory where you have the GenBank files. First, it is necessary to construct a set-up file from which to draw the atlas. This file is called a `configuraton` file and holds informations on which calculations to run and which files to use as input. The file is then used as an input to the program `atlas` which draws the circular genome

atlas figure.

Listing 2.3: Create genome atlas

```
# Syntax:
$ atlas_createConfig -ref <name>.gbk
# The file <name>.gbk.atlas.cf is automatically generated

$ atlas
# USAGE:
#   atlas -f <genbankFile> -c <configFile> -o <outputFile> -tidy
#
# Creates a genome/blast atlas of the organism given in <genbankFile> based on the setup in the config
    file.
#
# -f    The name of the gbk file to use <REQUIRED>
# -c    The name of the config file created by atlasCf <REQUIRED>
# -o    The name of the plot file to create.#   If not specified no plot will be made.

$ atlas -f <name>.gbk -c <name>.gbk.atlas.cf -o <name>.gbk.atlas.ps

# Example:
$ atlas_createConfig -ref Neisseria_gonorrhoeae_FA_1090_ID_AE004969.gbk

# The file Neisseria_gonorrhoeae_FA_1090_ID_AE004969.gbk.atlas.cf is generated
# Following command must be one one command-line
$ atlas -f Neisseria_gonorrhoeae_FA_1090_ID_AE004969.gbk
-c Neisseria_gonorrhoeae_FA_1090_ID_AE004969.gbk.atlas.cf
-o Neisseria_gonorrhoeae_FA_1090_ID_AE004969.gbk.atlas.ps
```

Using `ls -l` you will see that a number of files have been generated. These should be kept as they will be used later. Find the postscript file `*.ps` in the file-manager and open it, save as a PDF. To do a zoom of a specific region, open the file called <name>.genomeatlas.cf and add the line described below. Note that this procedure should be run in the same directory as the whole chromosome (not-zoomed) atlas, as it uses the same files.

Listing 2.4: Create zoom of genome atlas

```
$ cp <name>.genomeatlas.cf <name>.zoom.genomeatlas.cf
$ gedit <name>.zoom.genomeatlas.cf
# Syntax:
circlesection <start> <end>;
# Example:
circlesection 515000 535000;
# This line should be added under the line that looks like this:
circletics auto;
# Save the file as something else, like
# Now re-run the atlas picture
$ genewiz -p <name>.zoom.genomeatlas.ps <name>.zoom.genomeatlas.cf
```

Find the postscript file `*.ps` in the file-manager and open it. Save the plots as a PDF format and open a word processor (Word, Pages) or presentation software (PowerPoint, Keynote) on your LOCAL computer. Get the PDF from the shared folder and put the picture into your presentation.

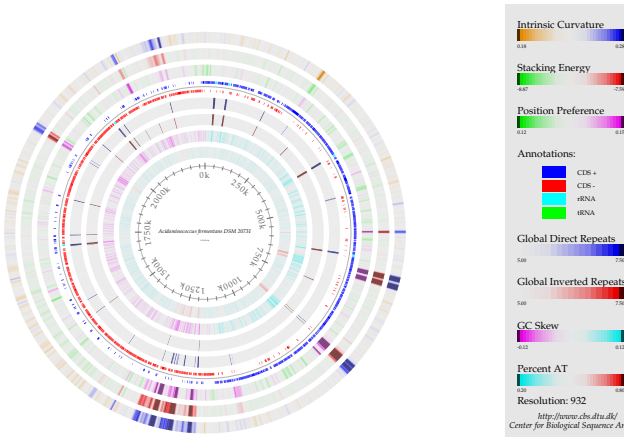

Figure 2.1: **Genome atlas, DNA structures.** A DNA structural atlas. DNA, RNA and gene annotations are from the published GenBank data. Each lane of the circular atlas shows a different DNA feature. From the innermost circle: size of genome (axis), percent AT (red = high AT), GC skew (blue = most G's), inverted and direct repeats (color = repeat), position preference, stacking energy and intrinsic curvature.

2.2.1 Exercises

1. Create a genome atlas for 2 different replicons, plasmids/chromosomes
2. Create a zoom of one of the atlases
3. Save plots as PDF and insert into presentation

2.3 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 [**?**]. 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 utilizing 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.

Listing 2.5: Identify 16S rRNA sequences in genomic DNA

```
# Syntax:
$ rnammer -S bac -m ssu -f <name>.rrna <name>.gbk.dna
# Example, one line command:
$ rnammer -S bac -m ssu -f Neisseria_meningitidis_Z2491_ID_AL157959.gbk.dna.rrna
     Neisseria_meningitidis_Z2491_ID_AL157959.gbk.dna
```

Note that this takes time to go through a single genome, and if there is more than one rRNA operon (which is true for many bacteria), then multiple 16S rRNA sequenced will be found per genome. See the example below, where first the *N. meningitidis* genomes are done, and then the *N. gonorrhoeae* genomes are done.

Listing 2.6: Identify 16S rRNA sequences in genomic DNA - loop

```
for x in Neisseria*gbk.dna
> do
> echo $x
> rnammer -S bac -m ssu -f $x.rrna $x
> done
```

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.

2.3.1 Exercises

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

2.4 Multiple sequence alignment of 16S rRNA sequences

One way of comparing the 16 sRNA sequences is to do a multiple sequence alignment. Multiple alignments are often used in identifying conserved sequence regions across a group of sequences hypothesized to be evolutionarily related. If two sequences in an alignment share a common ancestor, mismatches can be interpreted as point mutations and gaps as "indels" (that is, insertion or deletion mutations) introduced in one or both lineages in the time since they diverged from one another. 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 operon. In order to do a 16S rRNA tree you should pick one sequence. Here we choose the one that has the highest score according to the `rnammer` models. The program `rnammer_extractseqs` takes all files in a directory named `*.rrna` 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.

Listing 2.7: Identify 16S rRNA sequences in DNA - select sequences

```
# The following code works on all files in the current working directory.
# Syntax:
$ rnammer_extractseqs <allfile>.RRNA
$ rnammer_extractseqs_allLengths <allfile>.RRNA
# Example:
$ rnammer_extractseqs all.RRNA
$ rnammer_extractseqs_allLengths allLengths.RRNA
```

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.

Listing 2.8: Multiple alignment of 16S rRNA sequences

```
# Syntax:
$ clustalw <allfile>.RRNA
# Example:
$ clustalw all.RRNA
# The following files are created:
all.aln
all.dnd
```

Next we create a distance tree for the alignment, using a bootstrap value of 1000. Here is a quote from 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.

Listing 2.9: Multiple alignment of 16S rRNA sequences - bootstrap

```
# Syntax:
$ clustalw <allfile>.RRNA -bootstrap=1000
# Example:
$ clustalw allLengths.RRNA -bootstrap=1000
# The following files are created:
all.phb
```

The tree construction is simply a drawing program called `njplot` [?].

Listing 2.10: View 16S rRNA tree

```
$ njplot all.RRNA.phb
```

Open the bootstrap tree file `*.phb` and tick the `Display` setting by clicking `Bootstrap values`. Click `"File"` and `"Save Rooted tree"`. Output file is names `all_root.phb`. Under `"File"`, save the tree as a PDF format and open a word processor (Word, Pages) or presentation software (PowerPoint, Keynote) on your LOCAL computer. Get the PDF from the shared folder and put the picture into your presentation. Add colors to indicate taxonomic groupings or others clusters (See Figure 2.2).

2.4.1 Exercises

1. Extract the best scoring sequences within the length criteria for all `*.rrna` files and store in file `all.RRNA`
2. How many sequences are there in the `all.RRNA` file?
3. Extract the best scoring sequences for all `*.rrna` files and store in file
4. How many sequences are there in the `all.RRNA` file?
5. Run a multiple alignment on selected sequences
6. Generate a phylogenetic tree
7. Save tree as PDF and insert into presentation

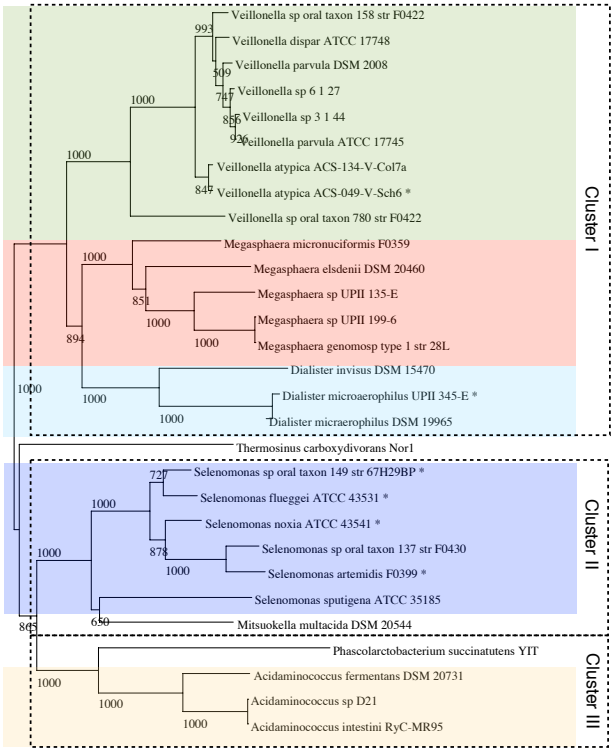

Figure 2.2: **16S rRNA tree.** Each genome sequence was searched for 16S rRNA patterns and candidate sequences were extracted. The best sequence from each genome was selected. For two genomes, no sequences were found, *Centipeda periodontii* DSM 2778, *Megamonas hypermegale* ART12 1. For 6 additional genomes, the located sequences were shorter than the default acceptable length. The short sequences sequences are marked with a "*". Length criteria was changed from minimum 1 400 to 1 100 and maximum 1 800 unchanged. The distance tree was made with 1 000 bootstraps.

Day 3

Overview

- Genome comparison: Amino acid and codon usage
- Genome comparison: BLAST atlas
- Proteome comparison: BLAST matrix

3.1 Amino acid and codon usage

The amino acid and codon usage is calculated using `BioPerl` modules, and is a simple calculation of the fraction of each amino acid or codon count of the total count of amino acids or codons. The bias in third position is found by counting the number of each base on each position in each codon, divided by the total number of codons. The bias in the third position between `G/C` and A/T is calculated as `sum(GC)-sum(AT)`, so that 100 % GC in third codon position is +1 and -1 for 100 % AT. The calculations have been implemented in a program called `stats_usage` and takes gene FASTA files as input.

Listing 3.1: Calculate amino acid and codon usage

```
# Syntax:
stats_usage <name>.gbk.fna /usr/bin/gnuplot
# Example:
stats_usage Neisseria_gonorrhoeae_FA_1090_ID_AE004969.gbk.fna /usr/bin/gnuplot
# Loop
$ for x in *fna; do stats_usage $x /usr/bin/gnuplot; done
```

The output is a number of images files and a text file. Open the image files in the file manager and investigate the content. Use `gedit` to investigate the text file.

Listing 3.2: Calculate amino acid and codon usage - output

```
# Image files
Neisseria_gonorrhoeae_FA_1090_ID_AE004969.gbk.fna.aa-usage.pdf
Neisseria_gonorrhoeae_FA_1090_ID_AE004969.gbk.fna.codon-usage.pdf
Neisseria_gonorrhoeae_FA_1090_ID_AE004969.gbk.fna.basicInfoPS.pdf
Neisseria_gonorrhoeae_FA_1090_ID_AE004969.gbk.fna.all.pdf
Neisseria_gonorrhoeae_FA_1090_ID_AE004969.gbk.fna.bias.pdf
# Text file
Neisseria_gonorrhoeae_FA_1090_ID_AE004969.gbk.fna.CodonAaUsage
```

3.1.1 Exercises

1. Calculate the codon and amino acid usage for all genomes
2. Create a folder for the PDF files (`PDF`) and one for the usage text files (`USAGE`)
3. Move all `*pdf` files to the `PDF` folder and all `*.CodonAaUsage` to the `USAGE` folder

3.1.2 Compare amino acid and codon usage

Comparing the amino acid and codon usage between many genomes also usually involves clustering genomes with similar usage. It is therefore useful to compare these numbers using a so-called heat-map constructed in `R` (The R Project for Statistical Computing). First we prepare the data from the `*CodonAaUsage` files and collect them into one file. You can of course choose to only include some of your file if you wish to do a smaller comparison, or more data if you want.

Listing 3.3: Comparing amino acids and codon usage - preparations

```
grep aa *CodonAaUsage > aaUsage.all
```

```
grep Total *CodonAaUsage > statistics.all
cut -f2,3,4,5,6,7,8 statistics.all > tmp.all
mv tmp.all statistics.all
grep codon *CodonAaUsage > codonUsage.all
```

Start R in the directory where the statistics files are stored and read in the data. Drawing heat-maps of amino acid and codon usage (See Figure 3.1):

Listing 3.4: Comparing codon usage - heat-map

```
$ R # This is typed on the prompt
# As a result R opens it's own prompt with the ''>'' symbol in the start of the line
# On this prompt only R commands work and NOT the normal unix commands

library(gplots)
codon <- read.table("codonUsage.all")
colnames(codon) <- c( 'Name', 'codon', 'score', 'count')
codon <- codon[1:3]
test <- reshape(codon, idvar="Name", timevar="codon", direction="wide")
codonMatrix <- data.matrix(test[2:length(test)])
rownames(codonMatrix) <- test$Name

# R allows you to run one long command like the following
codon_heatmap <- heatmap.2(codonMatrix, scale="none", main="Codon usage",
xlab="Codon fraction", ylab="Organism", trace="none", margins=c(8, 25)) # Command finished

dev.print(postscript, "codonUsage.ps", width = 25, height=25)
dev.off()
```

Listing 3.5: Comparing amino acids usage - heat-map

```
library(gplots)
aa <- read.table("aaUsage.all")
colnames(aa) <- c( 'Name', 'aa', 'score')
test <- reshape(aa, idvar="Name", timevar="aa", direction="wide")
aaMatrix <- data.matrix(test[2:length(test)])
rownames(aaMatrix) <- test$Name

# R allows you to run one long command like the following
stat_heatmap <- heatmap.2(aaMatrix, scale="none", main="Amino acid usage", xlab="Amino acid fraction",
ylab="Organism", trace="none", margins=c(8, 25), col = cm.colors(256))  # Command finished

dev.print(postscript, "aaUsage.ps", width = 25, height=25)
dev.off()
```

3.1.2.1 Exercises

1. Create heat-maps for amino acid and codon usage
2. Save images as PDF and insert into presentation

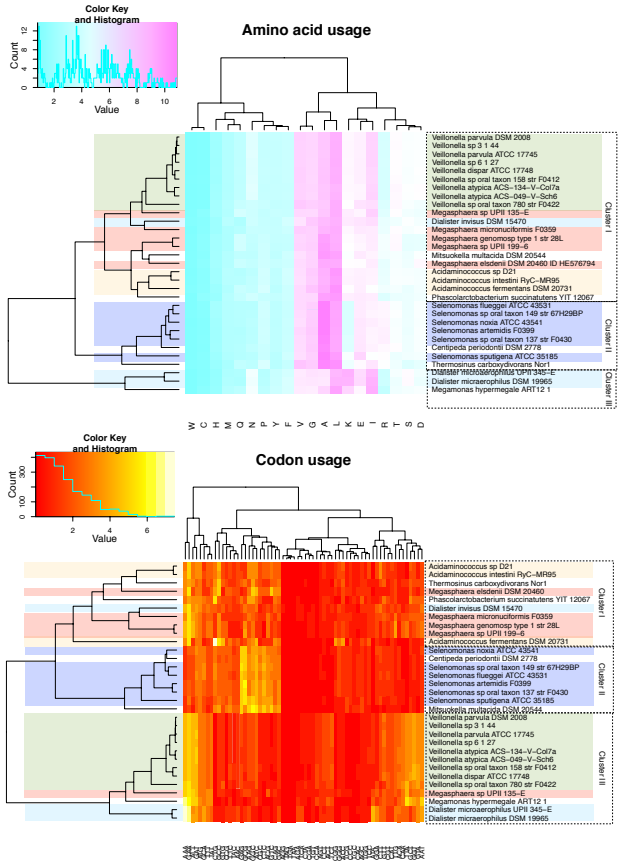

Figure 3.1: **Amino acid and codon usage heat-maps.** Amino acid and codon usage were for 31 genomes calculated based on the genes identified by gene finding (Prodigal). The percentage of codon and amino acid usage was plotted in two heat-maps using `R`. The heat-maps were clustered in 2D, thus reordering the organisms and the amino acids/codon to show the shortest distance between them. Dendograms are drawn for both and can be used to visualize the difference in usage between organisms.

3.2 BLAST atlas

A BLAST atlas is a way of comparing a number of proteomes to a reference genome and visually displaying the similarity. A blast matrix is always made with a reference organism in the ”middle”. All genomic properties that are shown in the atlas relate to this one organism. Next, other organisms that you wish to compare to the reference organism are searched for genes that are similar to those found in the reference organism. These genomes are called `queries`. These hits are then shown in the atlas as lines where regions in the reference organism have been found to have a match in the searched organism. One lane per searched organism is shown. The BLAST atlas is an extended version of the genome atlas which you have already used. The atlas configuration files is altered, adding data to the file and re-creating the figure. To keep some order in your files, follow this produre:

- Choose a reference genome for the atlas, `ref`
- Create a directory to hold the atlas files, `ATLAS`
- Copy the GenBank for the reference into the `ATLAS` directory
- Select 5 genomes to compare to the reference, `queries`
- For each query, copy the protein FASTA file into the `ATLAS` directory
- Construct the set-up file for the reference
- Then add a BLAST lane for each `query` genome using the `atlas_addBlastlane` command
- Use `grep` to verify that your query files have been added to the set-up file
- Run the `atlas` script with the configuration file

Listing 3.6: Create BLAST atlas

```
# Names have been shortened to make the guide more readable
$ mkdir ATLAS
$ cp GBK/Neisseria_gonorrhoeae_FA_1090_ID_AE004969.gbk ATLAS/
# Neisseria_gonorrhoeae_FA_1090_ID_AE004969 re-named to FA_1090
$ cp FSA/Neisseria_gonorrhoeae*fsa ATLAS/
$ cd ATLAS
$ atlas_createConfig -ref FA_1090.gbk
# This file is generated:
FA_1090.gbk.atlas.cf

$ addBlastlane
#
# USAGE: addBlastlane -f <query fasta file> -c <config file> -col <colour gradient> -name <query name>
# Adds BLAST lane to atlas configuration file for a blast atlas.
# This code can only add one lane at a time. Run in loop to add multiple lanes.
# -f    The name of the fasta file to add <REQUIRED>
# -c    The name of the config file created by blastAtlasCf <REQUIRED>
# -col  The colour of the blast lane given as RRGGBB, with RR being level of red and so forth. Each can
    go from 00 to 10 (Default is 060010)
# -name The name for each of the given fasta files that should appear in the legend of the plot (Default
    is the filename)

$ atlas_addBlastlane -f FA_1090.gbk.fsa -c FA_1090.gbk.atlas.cf -col 000090 -name FA1090
```

```
$ grep "FA_1090.gbk.fsa" FA_1090.gbk.atlas.cf
# dat FA_1090.gbk.fsa.genomemap.gz 1 0.0 0.0 0.0 "FA1090";
# circle FA_1090.gbk.fsa.genomemap.gz 1 "101010_000090.cm2" by 0.00 100.0;
# file FA_1090.gbk.fsa.genomemap.gz dat;

$ atlas_addBlastlane -f NCCP11945.gbk.fsa -c FA_1090.gbk.atlas.cf -col 009000 -name NCCP11945

$ grep "NCCP11945.gbk.fsa" FA_1090.gbk.atlas.cf
# dat NCCP11945.gbk.fsa.genomemap.gz 1 0.0 0.0 0.0 "NCCP11945";
# circle NCCP11945.gbk.fsa.genomemap.gz 1 "101010_009000.cm2" by 0.00 100.0;
# file NCCP11945.gbk.fsa.genomemap.gz dat;

$ atlas -f FA_1090.gbk -c FA_1090.gbk.atlas.cf -o FA_1090.gbk.blastatlas.ps
```

Open the image through the file browser.

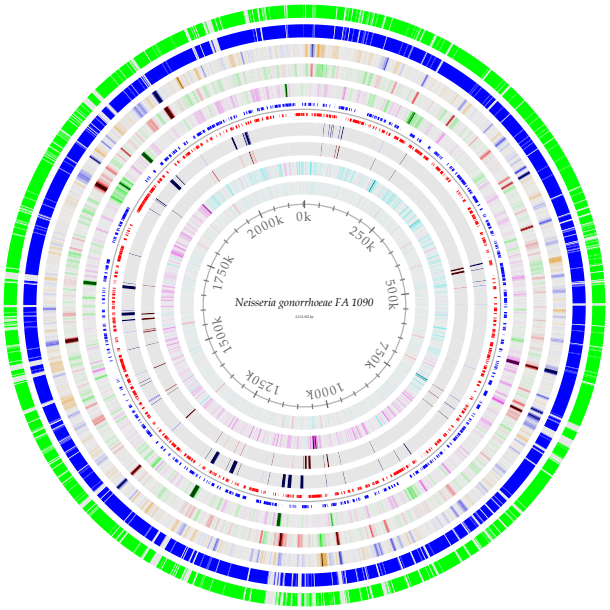

Figure 3.2: **Genome atlas, DNA structures.** A BLAST atlas. DNA, RNA and gene annotations are from the published GenBank data. Each lane of the circular atlas shows a different DNA feature. From the innermost circle: size of genome (axis), percent AT (red = high AT), GC skew (blue = most G's), inverted and direct repeats (color = repeat), position preference, stacking energy and intrinsic curvature.

3.2.1 Exercises

1. Create a BLAST atlas using one reference and 5 queries, use different colors for each query
2. Save images as PDF and insert into presentation

3.3 Proteome comparison - BLAST matrix

The BLAST matrix is a visual presentation of a pairwise proteome comparison using BLAST (Basic Local Alignment Tool). All sequences are compared to each other and a BLAST hit is significant when 50 % of the alignment is identical matches and the length of the alignment is 50 % of the longest gene in the comparison. If two sequences are similar according to the cutoff, they are collected in one "protein family". For the comparison of two genomes, protein families are built through single linkage, so that each shared connection must be between sequences from different genomes (shaded green). Paralogs are traditionally defined as a gene which has undergone duplication before speciation; in the BLAST matrix, an internal hit significantly similar to the query protein is grouped into the same gene family. The bottom row of the matrix shows the number of proteins that have homologous hits within the proteome itself (shaded red). The color scales are set automatically from the highest to lowest value observed, but can be changed manually.
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 (See Figure 3.3). We will construct a matrix from the `*.fsa` files created by prodigal The BLAST matrix algorithm has been implemented in a program called `matrix`. 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 `matrix_createConfig` construct a set-up file from all the `*.fsa` files (protein files in FASTA format) in a directory. Make sure that you have the right files in the current working directory.

Listing 3.7: Creating a BLAST matrix set-up file

```
# Syntax:
$ matrix_createConfig > <file>.xml
# Example:
$ matrix_createConfig > matrix.xml
```

The BLAST matrix is then generated using the `matrix` program with the file `matrix.xml` as input.

Listing 3.8: Creating a BLAST matrix from set-up file

```
# Syntax:
$ matrix -cpu 5 <file>.xml > <file>.ps
# Example:
$ matrix -cpu 5 matrix.xml > blastmatrix.ps
```

Error messages related to "illegal division by zero" can be due to faulty gene-finding (`prodigalrunner`). In some cases, the gene-finder 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.

Listing 3.9: Remove FASTA entries with no sequence

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

Once the matrix is done, open the file in the file manager and save as a PDF. If the order of the genomes is not how you want them, you might want to group your genomes differently, open the `*.xml` file and change the order of the organisms. Each organism is represented by a number of lines, see below. To alter the order of an organism you must move the entire entry for the organism. An entry starts with `<entry>` and ends with `</entry>`. Save the file as a new name, `newmatrix.xml`, and re-run the `matrix` command.

Listing 3.10: Organism entry in matrix set-up file

```
    <source>Neisseria_meningitidis_Z2491_ID_AL157959.gbk.fsa</source>
    <title>Neisseria_meningitidis_Z2491_ID_AL157959.gbk.fsa</title>
    <subtitle></subtitle>
</entry>
<entry>
    <length>unused</length>
    <source>Neisseria_meningitidis_Z2491_ID_AL157959_prodigal.orf.fsa</source>
    <title>Neisseria_meningitidis_Z2491_ID_AL157959_prodigal</title>
    <subtitle></subtitle>
</entry>
</sources>
```

3.3.1 Modify names in BLAST matrix `*.ps` file

HASH lines: If the matrix has a line for each organism saying something like

Listing 3.11: HASH line in matrix figure

```
(HASH\(0x954dba8\))  -45 rotate   dup stringwidth pop neg 0 rmoveto show  45 rotate
```

A Post script file is actually a text file that gets interpreted as a picture. You can open the `*.ps` file in the text editor an see what it looks like. To remove the `HASH` lines from the picture, we must remove a part of the text in the file. Fortunately, one command-line can do this for us. This can be done for all entries in the `*.ps` file as follows:

Listing 3.12: Removing HASH lines from matrix figure

```
$ awk '/HASH/{c=1;next} c--<0 && p{print p} {p=$0} END{print p} ' test.ps > new.ps
```

Bounding box: If the names of the organisms reach outside the border of the picture, change the numbers in the `BOUNDING BOX` field of the `*.ps` file. Each of these numbers correspond to different points in a coordinate system:

Listing 3.13: Bounding box in matrix post script file

```
%%BoundingBox: 0 0 2212 1110
%%BoundingBox: llx lly urx ury
```

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 click the file again.

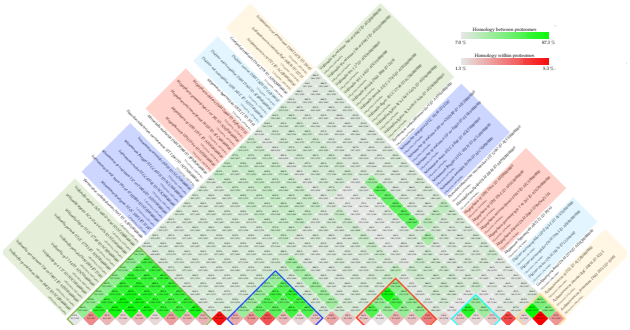

Figure 3.3: **BLAST matrix.** An all against all protein comparison was performed using BLAST to define homologs. A BLAST hit is considered significant if 50 % of the alignment consists of identical matches and the length of the alignment is 50 % of the longest gene. Internal homology (paralogs) is defined as proteins within a genome matching the same 50-50 requirement as for between-proteome comparisons. Self-matches are here ignored.

3.3.2 Exercises

1. Create a matrix for all the protein FASTA files
2. Re-order organisms and re-run `matrix`
3. Remove `HASH` lines from `*.ps` file
4. Save image as PDF and insert into presentation

Day 4

Overview

- Construct pan- and core-genome plot
- Calculate subset specific pan- and core-genomes
- Construct pan-genome tree

4.1 Proteome comparison - pan- and core-genome plot

The pan- and core-genome plot is a different use of BLAST for comparing proteomes (using the 50/50 cutoff as described above). The core-genome consists of protein families with representatives found in all investigated genomes. The pan-genome is the entire set of protein families from all genomes in the comparison. The first genome in the analysis has a core-genome equal to the pan-genome. The addition of an second genome reduces the core-genome of the two genomes and increases the pan-genome. Each sequence of a new genome is compared to a representative from each of the existing gene-families. If the new sequence matches, the family is a core-family, if the sequence does not match a family it becomes a new protein family. When all new sequences have been compared to existing gene-families, core families that did not have a representative in the latest added genome are removed from the core-genome of the genome comparison. The change in the pan- and core-genome is followed as two lines (blue and red, respectively, see Figure 4.1). The number of new proteins, along with how many new protein families that corresponds to, is indicated as gray bars on the plot.

The algorithm is dependent on BLAST and has been implemented in the program `pancoreplot`. First we construct a set-up file (configuration file) for the pan-core-genome plot (program: `pancoreplot_createConfig`). This file will contain a two column list of the files that should be compared in the pan- and core-genome analysis. Run the script in directory with protein FASTA files (`*.fsa`). Make sure that non of the protein FASTA files contain nucleotide sequences and that the files are not empty.

```
# Syntax:
$ pancoreplot_createConfig > <file>.list
# Example:
$ pancoreplot_createConfig > pancoreplot.list
```

Open the file `pancoreplot.list` using `gedit` and investigate the output. The two columns should be identical. The first column contains the name of the organism which will be displayed on the final plot, you can modify this if you like. The second column is the actual filename, this should not be changed as it refers to the actual file.

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 (`gedit`). The second column, separated by tab, is the filename and must NOT be edited. Running the program with the option `-keep <directory>` creates a folder and stores the raw output from the program. This is needed for further analysis.

```
# Syntax:
$ pancoreplot -keep <directory> <file>.list > <file>.ps
# Example:
$ pancoreplot -keep blastOutPut pancoreplot.list > pancoreplot.ps
```

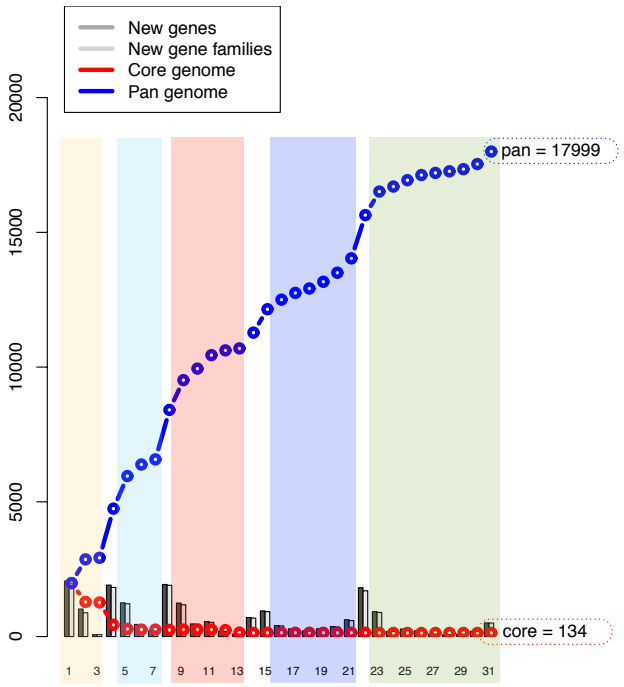

Figure 4.1: **Core and pan genome using BLAST.** A pan- and core-genome calculation was performed using BLAST. A BLAST cutoff of 50 % identity and 50 % coverage of the longest gene was used. If two proteins within a genome matched according to the 50/50 % cutoff, they were clustered into one protein family. Protein families were extended via single linkage clustering. If a protein family includes proteins from all genomes in the comparison, the family is a core protein family.

4.1.1 Exercises

1. Create a pan- and core-genome set-up file
2. Change the order of the genomes in the plot by changing the set-up file
3. Create a pan- and core-genome plot
4. Save image as PDF and insert into presentation

4.2 Subset genes from pan- and core-genome plot

The analysis of "specific" genes is performed on the clustering results from the pan- and core-genome calculations. The analysis will output the actual sequences but for the purpose of this workshop you will only look at the sizes of the subsets. The procedure is based on mathematical set theory and works with intersections, unions and complementary data-sets. Each genome is treated as a set and the intersection (`-i`) is the gene families that two or more sets have "in common". The intersection of genome A and B, is the set of all gene families which are found in both A and B. The union of two or more sets refers to the gene families which

are found in either genome A or B. Calculating the complimentary families of a genome refers to the set of all families which are members of A but not members of B. In the comparative genomic analysis, the sets usually consists of more than one genome, such as the intersection of genome A, B and C while not found (complimentary, `-c`) in genome D, E and F. This will give families that are found in A, B and C but not found in any of D, E or F. It is also possible to calculate the situation where families are found in A, B and C but not found in the intersection of D, E and F, this is referred to as the "compinter" (`-ci`). The last option is to extract the union (`-u`) gene families, all families found in a genome set. This procedure outputs the genes/gene families in common or complementary between genomes in a pan- and core-genome plot. The `<directory>` argument must be of the type created as temporary directory by the `pancoreplot` script (`-keep blastOutPut`).

Listing 4.1: Pan- and core-genome subset - Examples

```
# Syntax:
$ pancoreplot_subsets <-option> <value> <directory>

# Example:
# Gives you the core gene families of genomes 1, 2, 3, 5, 6, and 7.
$ pancoreplot_subsets -i 1:3,5:7 blastOutPut
# Example output:
Calculating the intersection between genomes
Extracting the gene sequences from the data
Outputting 1582 gene sequences

# Gives the core gene families of all genomes in the set. This is used if no options are given.
$ pancoreplot_subsets -i 1: blastOutPut

# Gives the core gene families of genomes 1, 3, 4 and 5 which are not present in any of the genomes from
    6 and on to the last.
# In this command, genome 2 is not considered at all.
$ pancoreplot_subsets -i 1,3:5 -c 6: blastOutPut

# Gives the core-genome of organisms 1, 2 and 3 as well as the core-genome of 5, 6 and 7.
# This is a larger set different from '-i 1:3,5:7' above.
$ pancoreplot_subsets -i 1:3 -i 5:7 blastOutPut

# Gives the part of the pan-genome of organisms 1 through 5 which is neither in the core-genome of 7, 8
    and 9 or in the core-genome of 8, 9 and 10.
$ pancoreplot_subsets -u 1:5 -ci 7:9 -ci 8:10 blastOutPut

# To output actual sequences use \texttt{-p 1} and redirect the output into a file:
$ pancoreplot_subsets -p 1 -i 1:3,5:7 blastOutPut > subset.fsa
```

4.2.1 Exercises

1. Calculate the sizes of relevant subgroups of your genome set and store results in table
2. Save table and add presentation

4.3 Pan-genome tree

The pan-genome tree creates a phylogenetic tree based on shared gene families as defined in the pan- and core-genome analysis. The program uses the output stored using `-keep` and output is a picture.

Listing 4.2: Pan- and core-genome tree

```
# Syntax:
$ pancoreplot_tree <directory name> > <tree>.ps
# Example:
$ pancoreplot_tree blastOutPut > tree.ps
```

4.3.1 Exercises

1. Construct pan-genome tree for genome set
2. Save plot and add to presentation

4.3.2 Gene frequency plot from a pan-core genome output

Before performing this analysis you must read section **??** on pan- and core-genome construction. Find BLAST output reports from pan-core genome plot and locate the last group file (*group_ lastX.dat*).
Run following command in the `Terminal`:

```
# Syntax
gawk '{print NF-2}' group_<lastX>.dat | gawk '{arr[$1]++} END {for(i in arr) print i,arr[i]}' | sort -n > frec.txt
# Example
gawk '{print NF-2}' group_3.dat | gawk '{arr[$1]++} END {for(i in arr) print i,arr[i]}' | sort -n > frec.txt
```

Start R

Read in file:

```
frec <- read.table("/username/path/frec.txt", sep=" ", dec=",")             # Syntax
frec <- read.table("/home/student/Desktop/frec.txt", sep=" ", dec=",")      # Example
```

Create plot:

```
# R allows you to run one long command like the following
barplot(frec$V2, main="Gene frequency",
xlab="Gene occurrence count, genes can be found multiple times in one genome",
ylab="Frequency of gene count, how often does a gene get counted x times",
ylim=c(0,max(frec$V2)*1.1)) # Command finished
```

Save plot:

```
dev.print(pdf, file = "/username/path/frec.pdf", width = 15, height=8)          # Syntax
dev.print(pdf, file = "/home/student/Desktop/frec.pdf", width = 15, height=8)   # Example
```

4.4 Tips and Tricks

4.4.1 Remove FASTA entries with no sequence

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

4.4.2 Modify names in BLAST matrix *.ps file

4.4.2.1 HASH lines

If the matrix has a line for each organism saying something like

```
(HASH\(0x954dba8\))  -45 rotate   dup stringwidth pop neg 0 rmoveto show  45 rotate
```

A Post script file is actually a text file that gets interpreted as a picture. You can open the *.ps file in the text editor an see what it looks like. To remove the HASH lines from the picture, we must remove a part of the text in the file. The entire block must be deleted for each genome for the lines to disappear.

```
183.497474683058 ux 111.639610306789 uy moveto
(HASH\(0x954dba8\))  -45 rotate   dup stringwidth pop neg 0 rmoveto show  45 rotate
newpath
```

Fortunately, one commandline can do this for us. This can be done for all entries in the *.ps file as follows:

```
awk '/HASH/{c=1;next} c--<0 && p{print p} {p=$0} END{print p} ' test.ps > new.ps
```

4.4.2.2 Bounding box

If the names of the organisms reach outside the border of the picture, change the numbers in the BOUNDING BOX field of the *.ps file. Each of these numbers correspond to different points in a coordinate system:

```
%%BoundingBox: 0 0 2212 1110
%%BoundingBox: llx lly urx ury
```

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 click the file again.

4.4.3 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 ps2pdf again. It takes a couple of tries the first time, but then it is easy enough to set the size of the picture.

4.4.4 Rename files

Files with wrong ending can be renamed the following type of loop:

```
for x in *gbk.fna
do
newx=`echo $x | sed "s/.gbk.fna/.fna/"`
mv $x $newx
done
```

Or another example:

```
for x in *gbk.proteins.fsa
do
newx=`echo $x | sed "s/.gbk.proteins.fsa/.proteins.fsa/"`
mv $x $newx
done
```

4.4.5 Delete empty files

Always list the files before using the removing command so you do not delete fields unintentionally.

```
find . -type f -empty    # Display empty files
find . -type f -empty -exec rm {} \;    # Remove empty files
```