Using UNIX.

UNIX is mainly a command line interface. This means that you write the commands you want executed. In the beginning that will seem inferior to windows point-and-click, but in the long run the power of the interface will surface.

Getting unix
There are many other ways to get it than mentioned here, and if you have a better way, please use that.
Windows: Download and install **MobaXTerm**. Run it, too.
Mac: Mac is unix. Use the applications **X11** or **Terminal**.
Linux: You sly dog, you - you have it already ☺
Running the application will start a window, the shell.

The shell
This is what we call the command line interface. There is a prompt that looks different on all unix's, but is easily recognizable, that shows that the machine is awaiting your orders. Some examples;

```
aix4(s010178) $
interaction[pws]:/home/people/pws>
pws@antros:~$
```

There are many different shells and the exact look and feel depends on various factors (like the OS, etc). It makes no sense to describe the details in this guide.

The file system.
In this guide the greater-than and less-than signs around something, for example like this <something> means: you type "something" here. It will be obvious in the actual situation, what you should write. Example: <filename> should be replaced with the actual name of the file: MyFile.txt

File and directory (folder) names should not include spaces, Danish or other outlandish letters etc. The following characters are safe abcdefghijklmnopqrstuvxwz ABCDEFGHIJKLMNOPQRSTUVXWZ 0123456789.-_Others work also, but you might want to stay away from trouble.

There is a limit to how long file names can be, but you won't run into it. Case matters in UNIX. Capital letters are not the same as small letters. Example: MyFile.txt is not the same as myfile.txt.
File extensions (or the .doc, .txt things added to the end of files) do not matter in UNIX, but they are needed for other systems. It is a good idea to use them so you know what sort of file you are using e.g. a file with just text in should have a .txt extension, and other extensions you use should be consistent and relevant to the file.

File permissions is a difficult subject, but important to know in UNIX! Every file and directory is equipped with some controls over who are allowed to do what with it. In the UNIX security model there are three different groups of people; the User (you), your Group (your friends), and the Others (everyone else). The groups have three kinds of access rights to the file; read, write and execute permissions.

Read access means that you can read the file, write means that you can write or change the file, execute means that the file is a program, which you can run or execute. Here's an example: (you get this output on the screen by using the "ls -l" command which lists the access status of the file);

```
-rwxr--r--    1 root     sys       113520 Sep  5 14:15 good_program
```

This file is named good_program, the size is 113520 bytes, it was changed/created at Sep 5 14:15, the owner is named root, and the group is sys. The file permissions are rwxr--r--, which means that the owner can read, write (change), and execute the file (it is a program). The group, sys, can read the file, and so can everyone else. So the owner, root, is the only one who can change and execute the program, but s/he does not mind if everyone else looks at what s/he is doing.

File permissions are simply three triplet of rwx. Now if the file really is a directory, then permissions look like this:

```
drwxr-xr-x    6 gorm     user         72 Jun 26 23:50 gorm_directory
```

Notice the d in the beginning of the permissions, it means that this is a directory. The permissions on directories have a slightly different meaning than on files. Write permission means right to create and delete files in the directory (but not change existing files, THAT is governed by the file permissions). Execute permission means right to list the content of the directory. If the execute permission is not set, then the directory will look empty to you when you list it (ls).

Overview of often used UNIX commands:
Most standard UNIX commands have options which control the different ways a command can work. Look at the man pages for details, see next paragraph. Many commands are abbreviated to just a few (obscure) letters. That is the UNIX way.

Most programs can be terminated with ctrl c (typing c while holding down the ctrl button). If it does not work, harsher measures are available.
Some, like 'less' and 'man', display some info and wait for the user give further 'one key' instructions. Here it is important to know that you quit the command by pressing 'q'.

Getting help.
The way to get help on a command in UNIX is using the man(ual) pages. They can be difficult to read at first, but persist and it will become clear.
man <command you want to know about>
You can get help on man by typing "man man". You can also type 'man <command>' on google.

The list below only show a few UNIX commands and only in very little detail. It is mostly for remembering the name and what it does.

Listing files in a directory.
ls
ls <filename>
A good option to know is -l which also lists details about the file, "ls -l"

Copying files.
cp <filename> <destination>
cp <filename> <newfilename>
Moving (and renaming) files.
mv <filename> <destination> (note the destination path must be very specific-this will be explained)
mv <filename> <newfilename>
This last way to use the command, moves the file from one name to another, effectively renaming the file.

Removing (deleting) files.
rm <filename>

Listing file content.
less <filename>
You can move freely up and down in the file by using the space bar and 'b' key. PageUp, PageDown and arrow keys might work, but this differs from UNIX system to UNIX system. You cannot change anything in the file, only look at it. Remember ‘q’ for quit.

Changing file permissions.
chmod <options> <filename>
Making a file executable: chmod u+x <filename>
Making a file readable by everyone: chmod a+r <filename>
Making a directory readable by everyone: chmod a+rx <directoryname>

Creating a directory (folder):
mkdir <directoryname>

Removing a directory.
rmdir <directoryname>
Only empty directories may be removed.

Changing directory.
cd <directoryname>
Going up (back) i the directory hierarchy: cd ..

What is the current directory (print working directory).
pwd

Counting lines in a file.
wc <filename>
Typical output looks like this:
```
   17078     38712     939401 link.test
```
The numbers are lines, words and bytes in the file.

Outputting a file.
cat <filename>
This command pours the content of a file (or files) to the screen, or other files. This example copies two files into a third: cat file1.txt file2.txt > result.txt

Finding specific lines in a file.
grep <word> <filename>
This is an incredibly useful command, very versatile.
"grep promoter genome.txt" list all lines with the word promoter in the file genome.txt.

Seeing the beginning of a file.
head <filename>
Options can determine how many lines to see.

Seeing the end of a file.
tail <filename>
Options can determine how many lines to see.

Extrating columns of data from a file.
cut <options> <filename>
Extremely useful command for getting data out of tabular files.
If the file content is constructed like this;

```
SwissProt   GenBank         Cat1    Cat2    Cat3
K6PL_HUMAN  X15573.CDS.1     0.6284  0.9183 -2.9719
G6PI_HUMAN  K03515.CDS.1     1.0377  1.9856 -1.1401
G6PD_HUMAN  L44140.CDS.10    3.1217  6.8903 -4.3967
```

If you are only interested in Category 2 numbers, you get them like this; cut -f 4 prediction.txt
-f 4 means field 4. You can write -f 2-5 (fields 2 to 5) or -f 2,4 (2 and 4). cut has a weakness, though. It operates best on tab delimited files.

Merging files.
paste <filename1> <filename2>
Concentrates corresponding lines of the given input files file1, file2, and so on. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging).

Sorting files.
sort <filename>
Sorts the lines in the file alphabetically. Some options will make the sorting be numerically and/or on certain columns in the file. Extremely useful.

Downloading files from the internet.
wget <URL>
An URL is the link you see in the browser. wget will download the page/data for you in current directory.

Printing text to the screen.
echo <text>

Getting time and date information.
date

Stopping runaway programs.
kill <PID>
Killing the program is quite often the way to handle a badly behaving program. Be glad we don't follow this procedure, when it comes to humans. Sometimes a program won't die. Then use the sure kill; kill -9 <PID>

Editing text files.
nedit <filename>
This is a very user friendly text editor. It is easy and intuitive to use and it has menus and all. It is used for making Perl programs, too. There are other editors like gedit or emacs. You can use the one you are used to. However, avoid Notepad or word processors like MS Word.
Logging in to remote computers.
ssh -X <username>@<hostname>
This will start a shell on the remote computer. You need to have an account on the remote machine first.

Transferring files to and from other computers.
ftp <hostname>
After issuing the command, you will be asked for your username and password on the remote computer. You use the keywords "put" and "get" for file transfer, and "help" for getting help.

An encrypted and therefore secure alternative is sftp.
sftp <username>@<hostname>
You will be prompted for password, but the functionality of ftp and sftp is the same.

Redirection and pipes.
Every program/command has something called STDIN, STDOUT and STDERR.
This is shorthand for standard input, standard output and standard error. The default STDIN is the keyboard, default SDTOUT and default STDERR are both the terminal window (screen). STDIN, STDOUT and STDERR can be redirected to someplace else.

Redirecting STDOUT is most often used, as you often want to save the output of some program to a file. It is done with the ">" character.

Example: paste GB.lst pred.res > GBpred.lst

Using “>” overwrites the file (or creates it if it does not exists). Sometimes it is desirable to append (add) to the file instead. This is done by using two greater-than’s; “>>”. The file just grows with the additional data.

Example: paste GB.lst pred.res >> GBpred.lst

Most UNIX commands which takes a filename as an argument (for processing the file content) can also handle input directly by redirecting STDIN.
This character used for redirecting STDIN is "<".

Example: grep promoter < genome.txt

Often the output from one command is to be used as input for another command. It is obvious that you can save the output in a file, and then use it as input for the next command, but there is an even smarter method called piping. Piping redirects STDOUT from one program to STDIN to the next. You use "|" for piping.
In this example you want to find lines which contain the words "promoter" and "region".

Example: grep promoter genome.txt | grep region | less

First "grep promoter genome.txt" finds the lines containing the word "promoter". These lines are sent as input to "grep region", which finds the lines containing the word "region" (those lines already contain the word "promoter", as we ensured). Now the resulting lines are sent to "less", which simply lists the lines in a sensible manner.

Practice exercises can be found at http://teaching.bioinformatics.dtu.dk/36610/

PWS 1/12-2017