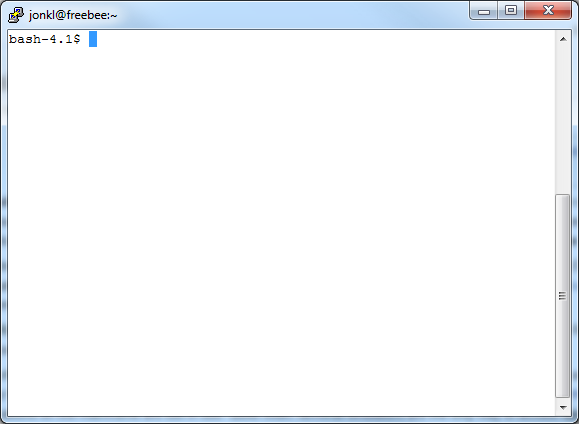
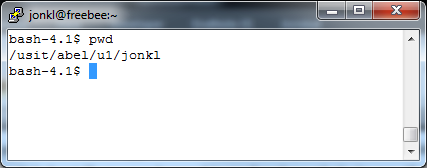
Unix basics exercise – MBV-INFX410

In order to start this exercise, you need to be logged in on a UNIX computer with a terminal window open on your computer. It is best if you are logged in on freebee.abel.uio.no. If you do not know how to open a terminal window, like the one below, on your desktop, please contact one of the instructors and get help. If you are working on a Mac or Linux computer, *make sure you are logged in on freebee* (with ssh) before you start!



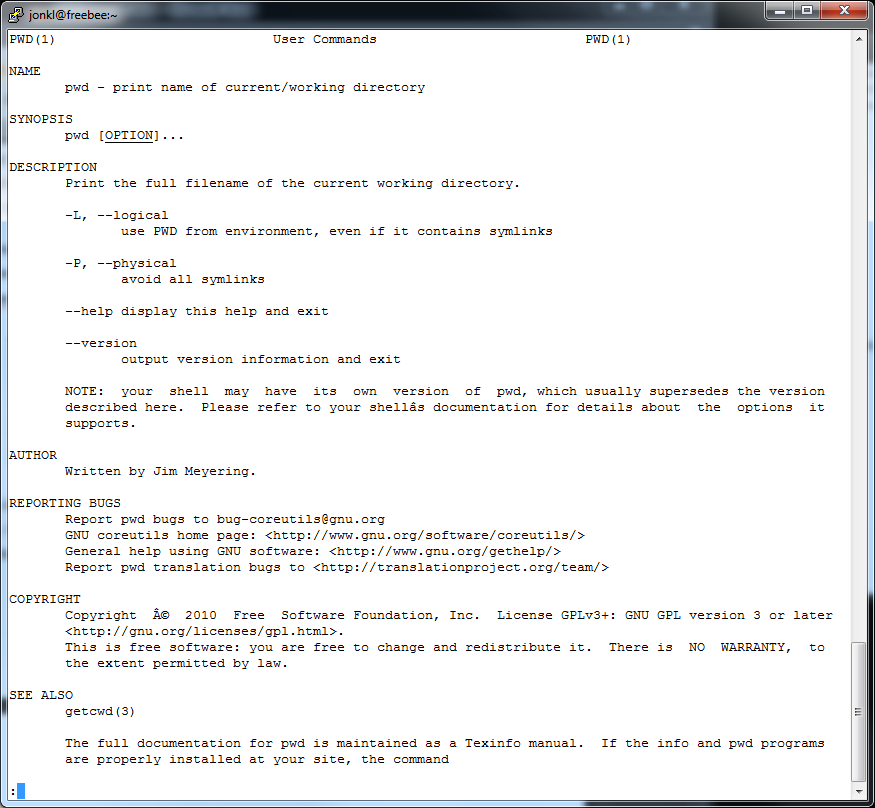
In the terminal window you have a Unix shell open with a command-line (a shell prompt) where you can type your commands. The command-line interpreter will execute your commands. There are several alternative and widely used Unix shells, for example C shell (csh), Bourne shell (sh), tcsh (used to be default at UiO), and Bourne-again shell (bash) (now default at UiO). In this exercise we will use bash. Hopefully you already have a bash shell running. If you are uncertain, type “bash” at the command prompt and press enter. Now you have opened a bash shell inside the other shell, and you are ready to start.

1. Your first command will be *pwd* (short for *print working directory*). It will tell you exactly where in the file system hierarchy you are, *i.e.* the full pathname of the directory you are in. Type *pwd* and press enter.



The command-line interpreter have interpreted you command. It knows that *pwd* is a program and runs that program. The output from the program was sent to your screen and the interpreter is waiting for a new command. What is your “current directory”? This is “Unix-speak” for “the directory you are in right now”.

1. Your second Unix command is *man*. It will show the manual pages for other Unix commands. Type *man pwd*. Then press enter. (You always press enter to make the interpreter start working on your command. Therefore we will not repeat this below!)



You get a lot of information about the *pwd* command. Press the letter “q” on your keyboard to leave the so-called man-page (the manual page for *pwd*).

1. There is also a man-page for the command *man*. Try typing “*man man”*. You will see that while *pwd* was rather simple, *man* has many options and you can give it many different arguments. You can go up and down in the manual page by using “arrow up”, “arrow down”, “page up” and “page down” on your keyboard. Try this. There is a “-h” option for *man*. What will it do? Leave the man-pages (by pressing “q”) and try it out by typing “*man -h”* on the command-line. What do you get?

The man-pages are very useful if you know the command you want to use, but you have forgotten some details about how to use it or what options you may use. Make sure you use the *man* command often later in this exercise!

1. Very useful and important commands are *ls* (lists all files in a directory), *cd* (change directory, go to a different directory), and *mkdir* (make directory, create a new directory)

Write these commands (with enter after each line)

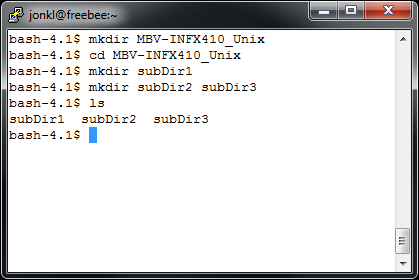
*mkdir MBV-INFX410\_Unix*

*cd MBV-INFX410\_Unix*

*mkdir subDir1*

*mkdir subDir2 subDir3*

*ls*



You have made a new directory called MBV-INFX410\_Unix, you went into that directory and created first 1 and then 2 new directories. Finally you listed all files in your “current directory”, *i.e.* the directory you are in.

1. Here are some other commands with explanations

*cd ..* (you will go from the directory you are in and up one level in the directory tree)

*cd ../..* (you will go up 2 levels in the directory tree)

*cd .* (nothing happens! You changed directory to the directory you are in)

*cd* ~/ (go to your “home directory”, the directory you came to when you logged in)

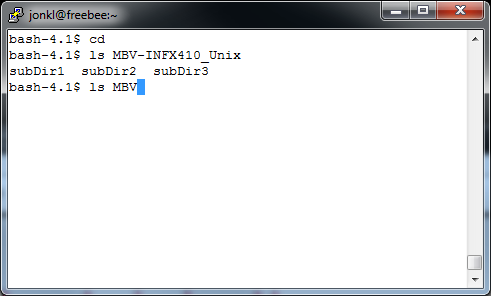
*cd* (this will do exactly the same as the previous line. If you “get lost” you can always type *cd* to go home...)

*ls -l* (shows all files in your current directory, but with more information)

*ls -la* (also shows “hidden files”, *i.e.* files with a name that starts with a “.”. These are often configuration files that you do not want to see)

Try out these commands. Experiment with walking around in the directory tree and see what you can find out.

1. Type *cd* to get to your home directory and type *ls MBV-INFX410\_Unix* to see what is in this directory. It is a lot of work, and a waste of time, to type the whole “*MBV-INFX410\_Unix*”. If you type *ls MBV* and press the Tab key, the shell will try to finish what you are writing. Most likely you only have one directory and “*ls MBV”* turns into “*ls MBV-INFX410\_Unix*” after you have pressed Tab.



Now go into the MBV-INFX410\_Unix directory by using *cd*, but do not type the full directory name. Use the Tabbing-trick instead. Ok?

Type *cd s* and press Tab. Now there are several options, and the shell will not know if you want to go into subDir1, subDir2, or subDir3. If you press Tab twice, you get the options listed. Type *2* and enter and you are in ~/MBV-INFX410\_Unix/subDir2. Type *pwd* to find out where you are in the complete directory tree. This shows you the “full path” or “absolute path”.

Make sure you use the Tab key a lot. *It will save you a lot of time and typing!*

1. Other useful things you can do are:

Press “Ctrl” key and then (without letting go of Ctrl) “e”. This will take you to the end of the command line. Make sure you have written something after the prompt. If not, you are already at the end! This combination of keystrokes would usually be written “<ctrl>-e” or just “^e”. You can also go back and forth in the text on the command line by using the “arrow left” and “arrow right” keys. And you can delete text with the “Delete” and “Backspace” keys. Try this out!

Other useful stuff:

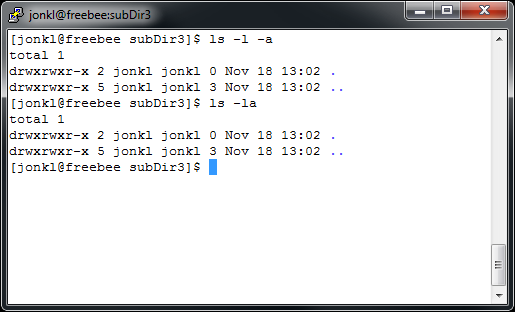
<ctrl>-a: Go to start of line, just after the prompt

<ctrl>-k: Delete everything after the cursor

<ctrl>-c: Kills (stops) a job that is running

<ctrl>-z: Suspends a job that is running. A suspended job can be restarted.

1. Go into the directory ~/MBV-INFX410\_Unix/subDir3 and make sure it is empty by running the command *ls -l -a*.

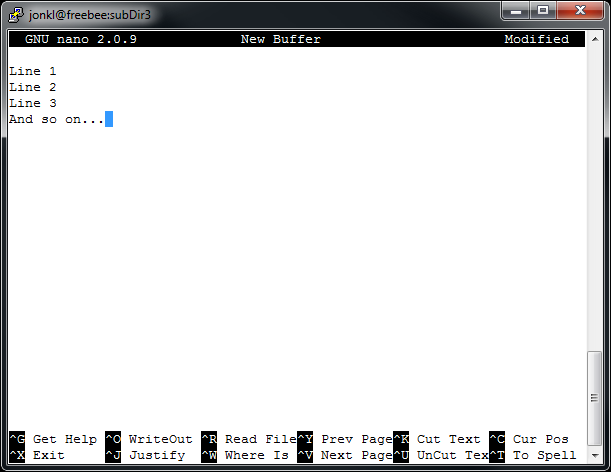
**

Above, you see that subDir3 is empty. There might appear to be two “files” there, but they are “.”, the directory you are in, and “..”, the directory above. Also note that the commands *ls -la* and *ls -l -a* will do exactly the same thing. For *ls* you can combine the arguments like this. This is not the case for all commands.

Now we want to open a text editor. Type *nano*. Inside the text editor you can type text, go up and down with the arrow keys and use delete and backspace. At the bottom of the editor, you see some keystrokes you may use to do various tasks. For example, “^x” or <ctrl>-x will exit the text editor and bring you back to the shell command line. Try this (but do not save any files yet)!

1. Nano is an extremely simple text editor. If you are more serious about using Unix you should learn something else, most likely emacs or vi. We will use nano for this exercise, but if you have a different favourite text editor, use that.

Open nano and type in the text shown below.



Save this text as a text file by typing <ctrl>-o and calling the file “fileOne”. Save it again, but this time call it “fileTwo”. Now exit nano (<ctrl>-x). Use *ls -l* to list the files you just wrote and the information about them.



In the above window you see information about these two files in my current shell. Here is what it all means:

|  |  |  |
| --- | --- | --- |
| Number/position | Value | Explanation |
| 1, 1st letter | - | - means regular file  d means directory |
| 1, 2nd - 4th letter | rw- | Access rights for the file owner (I can both read the file and modify/write to it, as well as delete it) |
| 1, 5th - 7th letter | r-- | Access for all members of the group (here: mikrobi). The group members can open the file but not modify or delete it |
| 1, 8th - 10th letter | --- | Access for everyone else. Users that are not members of the user group mikrobi cannot open the file (or modify it) |
| 2 | 1 | Number of hard links to the file (ignore this) |
| 3 | jonkl | User name of the file owner |
| 4 | mikrobi | Group of users that the file belongs two |
| 5 | 35 | Size of the file in bytes |
| 6 | Nov 18 13:26 | Date and time the file was last modified |
| 7 | fileOne | File name |

By modifying the accession rights you can make files and directories accessible to everyone on the computer, to only certain groups or to only yourself. The command *groups* will show which groups you are a member of. Try it! These are also useful:

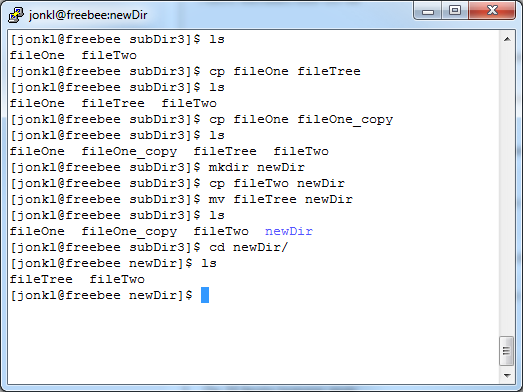
* *chmod* will modify the accessions, *i.e.* the string “-rw-r-----“ to something else
* *chown* will change the owner and/or group of a file

We do not have time to try this now, but you can use the man-pages and experiment later.

Note how useful this can be! For example, if you are involved in two (top secret) projects, you can make two user groups, group1 and group2. You can then have files and directories that can be viewed (or modified) by only yourself, only members of group1, only members of group2, or all users on the computer. Maybe you will only allow group1 members to run a certain program, while only group2 members are allowed to see your holiday photos? The system is very flexible!

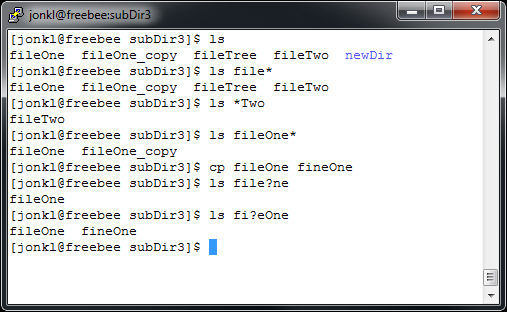
1. The commands *cp* (make a copy of a file), *mv* (move a file), and *rm* (remove a file) are very useful. By the way, to remove a directory use *rmdir*.

Go to ~/MBV-INFX410\_Unix/subDir3 where you still have the two files fileOne and fileTwo. Try the commands in the window below.



Make sure you understood what you did! Also make sure you use the Tab-key in order to avoid a lot of typing.

1. Learn to use the wildcards “\*” (matches zero or more characters) and “?” (matches any single character). For examples, see the window below.



1. Create some more directories, copy and move files, and finally remove some of the directories and files you created.

Be very careful with *rm* and *rmdir*. If you remove files and directories, they are gone, and you will not get them back!

1. Copying and pasting text is of course very useful. In order to copy text from somewhere in the terminal window, click and drag to mark the text you want to copy. Then right click on the command line to paste in this text. (This might work slightly differently depending on your laptop and/or mouse. If you are stuck, ask one of the teachers)

When you have learned to copy and paste within the terminal window you are ready to continue!

1. Open this webpage in a web browser: <http://www.uniprot.org/uniprot/O15527> Click on the “Format” box near the top, and choose “FASTA (canonical)” to get to the following page: <http://www.uniprot.org/uniprot/O15527.fasta>

Copy the whole Fasta-sequence including the header. On a Microsoft Windows laptop you would mark the text and press <ctrl>-c. Open the nano text editor in your terminal window and paste in the text by right clicking in that window. You might have to experiment a bit, as copying and pasting will depend on computer/mouse and other settings. If you get stuck, as a teacher!

Save the file by pressing <ctrl>-x, type “y” (yes) and type a file name (call it “O15527.fasta”). Exit nano and make sure you have created a file with the correct name.

Repeat the procedure with the text here:

<http://www.uniprot.org/uniprot/Q92736.fasta>

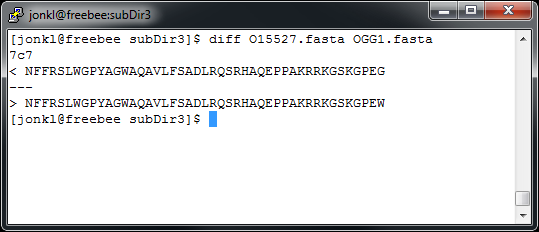
Save this as a file named RYR2.fasta.

1. A useful command to just list all the text in a text file is *cat*. Try *cat O15527.fasta* and *cat RYR2.fasta*. What happens?
2. Note that you, on the command line, can go back and forth in old commands by using the “arrow up” and “arrow down” keys. In this way you can access old commands and run them again. Or you can modify them and run. Try it out. This also saves a lot of typing!
3. The *cat* command will show you the whole file. If the file is long, the top will just disappear out of view at the top of the screen. A useful alternative is the *more* command. Try *more RYR2.fasta*. When you press “enter” *more* will show you one more line of text. Pressing the space bar, gives you one extra page of text. Does it work?
4. In the Unix shell we have something called “standard input” (STDIN) and “standard output” (STDOUT). Put simply, STDIN is what you type on the keyboard, while STDOUT is what is shown on the screen. This is the default. You can redirect STDIN and STDOUT. First try, as before, *cat O15527.fasta*. Then try

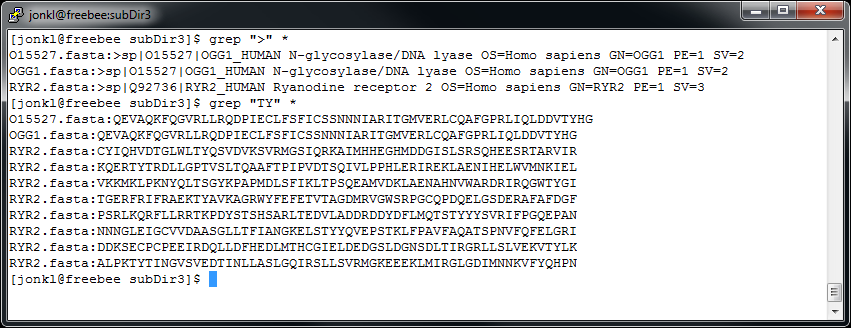
*cat O15527.fasta > OGG1.fasta*

Apparently nothing happens, but this is because the output wasn’t sent to the screen. Instead it was redirected into the new file OGG1.fasta. Do *cat* on that file to check. Ok?

1. Open the file OGG1.fasta in nano and change the C-terminal G (glycine) to W (tryptophan). A mutation, right! Now save the file again with the modification.
2. Another useful command is *diff*. Try *diff* *O15527.fasta OGG1.fasta*. What happens? Use the manual pages for *diff* if you didn’t get it or you want to know more.



1. The command *grep* will show you all the lines in one or several text files that contains a pattern, for example like this:



1. We want to get access to some GenBank data from the GenBank ftp site. Google “genbank ftp site” to find where it is. Google can be useful...

Hope you found it? It is here <ftp://ftp.ncbi.nih.gov/genbank>

Open this link in your browser. You find an awful lot of files, some of them are quite big. You can find information about the data here

<ftp://ftp.ncbi.nih.gov/genbank/gbrel.txt>

<ftp://ftp.ncbi.nih.gov/genbank/README.genbank>

Click on the file gbphg1.seq.gz in the GenBank ftp directory in your web browser and then save it to ~/MBV-INFX410\_Unix/subDir1. This means you have to find this directory on your laptop. Hope you have mounted your UiO home area on your laptop... If you are having problems, ask one of the teachers!

This file is in a “packed” format. It has been packed with the gzip program. To unpack, type *gunzip gbphg1.seq.gz*. Type *ls -lh* to see some information about the file. The *-h* option means that you get the file size in a more convenient format. The file is very big, 239 Mb! *This value was 204 Mb in 2012 and will change every time GenBank is updated.*

***PLEASE! When you have finished doing this exercise, delete this file (and other big files). We do not want to fill up your home area and the UiO computers with data that you absolutely do not need!!***

The file contains a large part of the phage data in GenBank. The rest is in gbphg2.seq and gbphg3.seq, but we will not look into that now. Do *more gbphg1.seq* to start reading from the top. What is the accession number of the first entry? (AB000833)

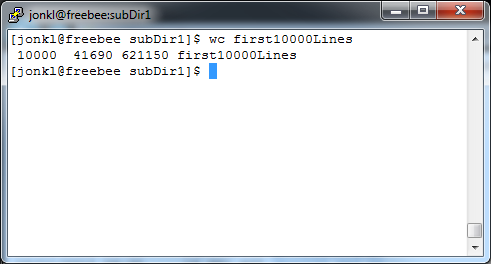
1. Are there anyone called Hansen that submitted any of the data? Try *grep Hansen gbphg1.seq*.
2. The commands *head* *-n* *20* and *tail* *-n* *50* will print out the first 20 and the last 50 lines of a given file (but you can of course choose the number of lines you want). Try

*head -n 200 gbphg1.seq*

*tail -n 60 gbphg1.seq*

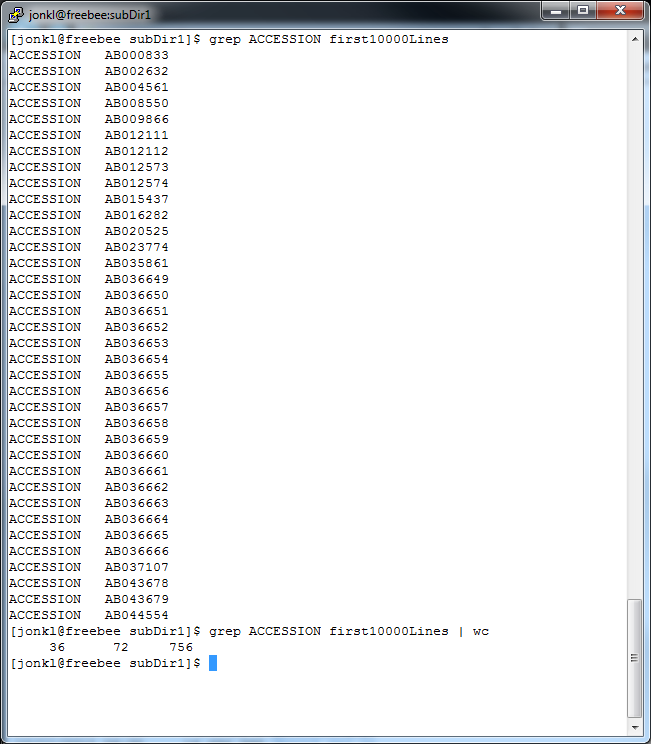
*head -n 10000 gbphg1.seq > first10000Lines*

The command *wc* counts the number of words in a file, for example *wc first10000Lines*:



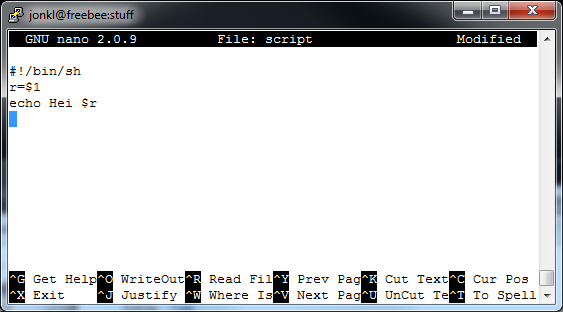
There are 10000 lines, 41690 words and 621150 bytes of information in the file. *Actually, it is 41687 and 621140 in November 2013, and this will most likely change when GenBank is updated next time.*

The “|” (the “pipe”) will send the STDOUT of one command into the STDIN of the next. What happens here?:



First, all the lines with “ACCESSION” is printed out. Then, instead of printing this out to the screen, the output is sent to the *wc* command. It counts 36 lines. It is 36 “ACCESSION” in the file. Easier than counting manually! How many lines have “ACCESSION” in them in gbphg1.seq?

1. Use the man-pages for *grep* command to find out what “*grep -n*” will do. Can you find the amino acid string/sequence PEECPEE (Pro-Glu-Glu-Cys-Pro-Glu-Glu) anywhere in the file gbphg1.seq? Yes, it is found on one line. On which line?
2. What does the option “*-B*” do for the *grep* commands? Use the man-pages to find out. Use the commands you have learnt to find out which species/phage that encodes a protein with the sequence PEECPEE (Solution: Salmonella phage vB\_SenS\_AG11).
3. Finally in this exercise we will try some very simple shell scripts. Shell scripts are small programs that are doing tasks that also could have been done step by step by a person on the command line. Go into ~/MBV-INFX410\_Unix/subDir2, which should be empty (or else remove the files that are there). Copy fileOne from ../subDir3 here.
4. Open nano and type the following,



What does this mean?

#!/bin/sh The rest of this script should be interpreted by the sh shell

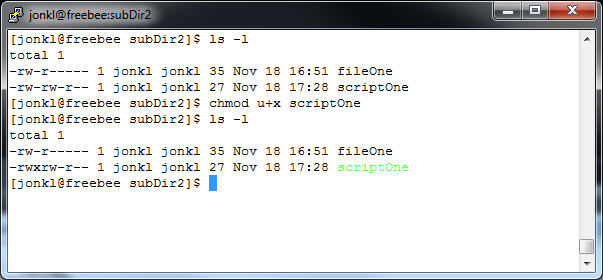
r=$1 The variable r (that is $r) gets the contents of the first

argument on the command line

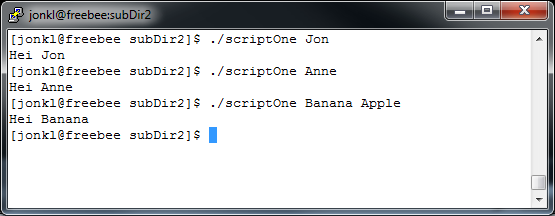
echo Hei $r “Hei” and then the variable content is written to STDOUT

Save the script as “scriptOne”. The “#!” on the first line, two first characters, is called a shebang (from “hash” and “bang”, *i.e.* exclamation mark). It tells the shell that the rest of the script should be interpreted by the program /bin/sh (Bourne shell). If it had been “#!/usr/bin/python” or “#!/usr/bin/perl” that would have indicated that this instead was a python or perl script, respectively.

Most likely you have a problem, since scriptOne is not an executable program that the shell thinks it can run as a program. However, the command *chmod u+x scriptOne* will fix that:



The “rw-“ in position 2-4 is changed to rwx, for the user (that is where the “u” comes from). The user can now, in addition to reading from and writing to the file, execute it (*i.e.* run it as a program). Now let us do that:



You have to put the “./” in front of the name of your script. If you do not, (most likely) the shell will not know where to look for it. As you see, you type the name of the executable file (including the file path to it) and some words. It will repeat back to you “Hei” and the first word. Not very advanced, but you first script!

1. Now try typing in this script

#!/bin/sh

prefix=$1

echo All files will start with $prefix

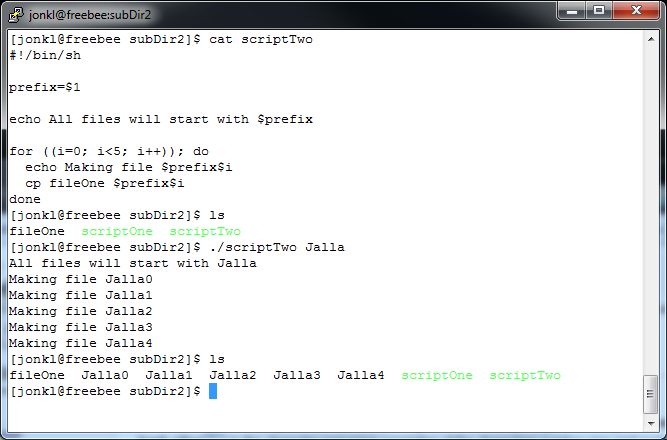
for ((i=0; i<5; i++)); do

echo Making file $prefix$i

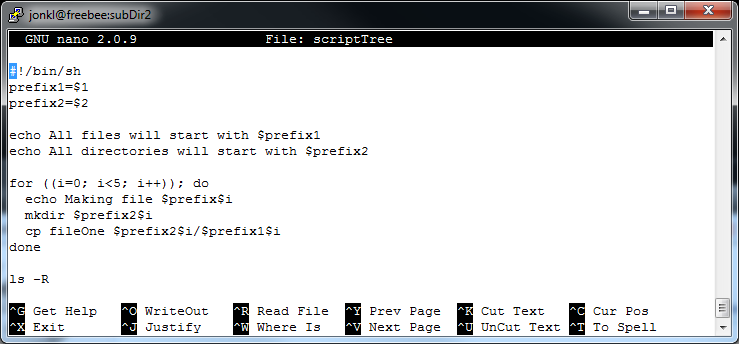
cp fileOne $prefix$i

done

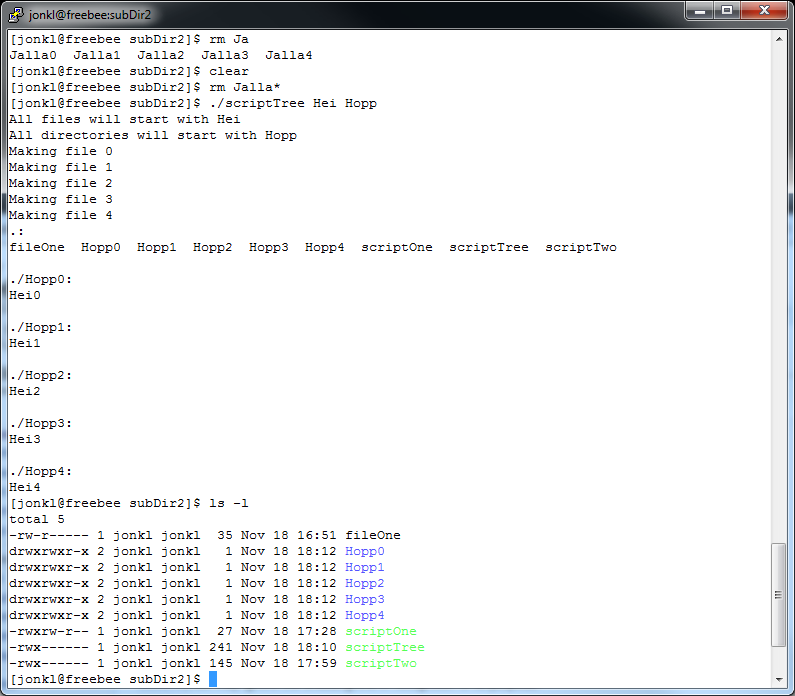
Save it as scriptTwo, make it executable (with *chmod*) and run it with an argument. What do you think it will do?



The script makes 5 copies of fileOne, each called Jalla0, Jalla1 etc, or whatever prefix you entered on the command line. A final example here,



Here it runs...



What does it do?

(When I run it with the arguments Hei and Hopp, it creates directory Hopp0 with a single copy of fileOne called Hei0, Hopp1 with Hei1 and so on... Finally it does an *ls -R*) Check the man pages for *ls* to find out what *-R* is doing.

1. To exit the shell, type *exit* or press <ctrl>-d.

Get more inspiration here

* Trond Hasle Amundsens Linux/UNIX guide for UiO:

<http://www.uio.no/tjenester/it/maskin/linux/hjelp/tips/guide.html>

* bash programming in INF3331 by Hans Petter Langtangen and co-workers:

<http://www.uio.no/studier/emner/matnat/ifi/INF3331/h12/bash.pdf>

* EMBnet “A Quick Guide UNIX”:

<http://www.embnet.org/sites/default/files/quickguides/guideUNIX.pdf>

* Or just search the web. There is lots of useful tutorials for basic and advanced Unix and scripting

Thanks to Trond Hasle Amundsens Linux/UNIX guide and Merete Molton Worrens “Basic Unix commands” for inspiration for this exercise!