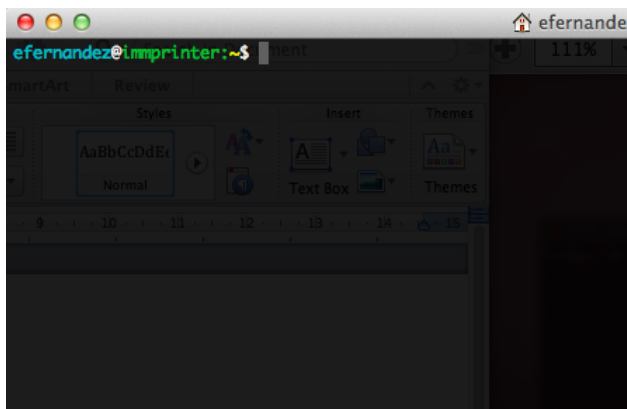


INTRODUCTION TO LINUX

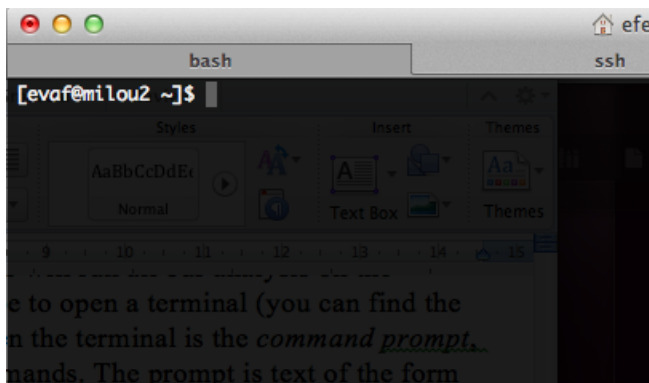
The terminal

In this course we will work on Linux computers and we will run all our analysis on the terminal, in a command-line way. The first thing will be to open a terminal. When we open the terminal we can see the ***prompt***, that indicates that the computer is ready to accept commands. The prompt is a line of text with the following structure
user@computer:directory\$

In the example below, we can see that the user efernandez is on the computer immprinter in the directory ~ (don't worry about the ~ symbol, we will explain that in the next section).



Having this information is very useful to know where you are at every moment. Even more when you are working remotely with multiple computers at the same time (for example, when you are working locally and on Uppmax at the same time). The prompt is configurable and therefore, it might vary slightly from one computer to another.



In this second example we can see that the prompt has changed a bit (now we have the information in between square brackets). In this case we can see that the user evaf is on the milou2 computer in the directory ~.

Some useful commands to get information about your user name and the computer you are using:

whoami	Username
hostname	Computer name
uname	Operating system (uname -a for full details).

Paths

All files and directories in Linux are arranged in a tree-like structure starting at the root. The root is represented as a single slash: `/`. The root contains several directories that at the same time can contain other directories or files. The location of a directory or file can be specified by its *path*. There are two types of path: the absolute and the relative path. The **absolute path** of a file will start at the root and traverse all the intermediate directories to get to the specified file. For example, this is the absolute path in Uppmax to a file that you will access later in the exercises:

```
/proj/genomeanalysis2022/nobackup/linux/transferme.txt
```

The **relative path** will give you the path to a file from where you currently are (it doesn't start from the root). In the previous example, if we were on the *nobackup* folder, the relative path to `transferme.txt` would be `linux/transferme.txt` or `./linux/transferme.txt`.

There are symbols to refer to a few specific directories. It might be handy to know some of them:

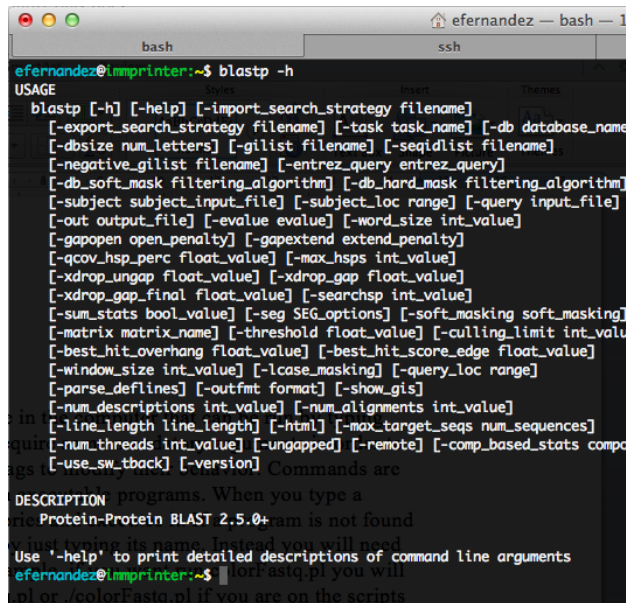
<code>/</code>	Root directory
<code>.</code>	Current directory
<code>..</code>	Parent directory
<code>~</code>	Home directory

Programs and commands

Commands are programs stored somewhere else in the computer and they can be run by typing their name on the terminal. Commands might require some mandatory arguments in order to work. Sometimes they can also have optional flags to modify their behavior.

Commands are stored in special directories dedicated to contain executable programs. When you type a command, the computer looks into those directories and executes it. If a program is not found in those directories, you won't be able to run it by just typing its name. Instead you will need to provide its path (absolute or relative). For example, if you want to run the script `colorFastq.pl` you will need to type something like `/scripts/colorFastq.pl` or `./colorFastq.pl` if you are on the `scripts` folder.

In order to get information about the behavior and options of a command you can check its MANual. For typical Unix commands you can simply type `man` followed by the name of the command. For example: `man ls` will show all the information about the `ls` command. The `man` command won't work for non-built-in commands. Non-built-in programs usually provide an option (`-h` or `--help` in many cases) to print this type of information. Others provide this information when you type their name without any argument (for example, `bwa` or `samtools`)



```
efernandez@imprinter:~$ blastp -h
USAGE
blastp [-h] [-help] [-import_search_strategy filename]
        [-export_search_strategy filename] [-task task_name] [-db database_name]
        [-dbsize num_letters] [-gilist filename] [-seqidlist filename]
        [-negative_gilist filename] [-entrez_query entrez_query]
        [-db_soft_mask filtering_algorithm] [-db_hard_mask filtering_algorithm]
        [-subject subject_input_file] [-subject_loc range] [-query input_file]
        [-out output_file] [-evalue evalue] [-word_size int_value]
        [-gapopen open_penalty] [-gapextend extend_penalty]
        [-qcov_hsp_perc float_value] [-max_hsps int_value]
        [-xdrop_ungap float_value] [-xdrop_gap float_value]
        [-xdrop_gap_final float_value] [-searchsp int_value]
        [-sum_stats bool_value] [-seg SEG_options] [-soft_masking soft_masking]
        [-matrix matrix_name] [-threshold float_value] [-culling_limit int_value]
        [-best_hit_overhang float_value] [-best_hit_score_edge float_value]
        [-window_size int_value] [-lcase_masking] [-query_loc range]
        [-parse_deflines] [-outfmt format] [-show_gis]
        [-num_descriptions int_value] [-num_alignments int_value]
        [-line_length line_length] [-html] [-max_target_seqs num_sequences]
        [-num_threads int_value] [-ungapped] [-remote] [-comp_based_stats compo
        [-use_sw_tback] [-version]
Commands are
DESCRIPTION programs. When you type a
Protein-Protein BLAST 2.5.0+
ram is not found
Just typing its name. Instead you will need
Use '-help' to print detailed descriptions of command line arguments
efernandez@imprinter:~$
```

Comments

When running a script or code in Bash (the scripting language most commonly used for working in the terminal), the symbol `#` indicates the beginning of a comment. Comments are lines of text that won't be executed. It is advisable to use them to make notes or clarify certain parts of the code.

Example:

```
# This is a comment. I can write whatever here
```

Moving around

You will need to move through files and directories during this course, and you need to keep track of where you are at every moment. In order to know your current location on the terminal you can run the `pwd` (Print Working Directory). This will show the absolute path of the directory you are in.

You can then LiSt the contents of a directory by typing `ls`:

```
ls      # List files
ls -a   # Also show hidden files (those whose name begins with
        ## a period).
ls -l   # Show more information about each file (permissions,
        ## owner, group, time and date of last modification).
```

You can also change to another directory by typing `cd` (Change Directory) and giving a new path:

```
cd path      # Change working directory to path
cd           # Change to home directory
cd ..       # Change to previous directory
```

Working with files and directories

You can CoPy or MoVe a file from a directory to another by using the commands `cp` or `mv`. `mv` can also be used to rename files:

```
cp file1 file2          # copy file1 to file2
cp file1 directory/     # copy file1 into directory. The
                        ## copy of the file has path
                        ## directory/file1
cp file1 file2 directory/ # copy file1 and file2 to
                        ## directory. When copying
                        ## multiple files, the
                        ## destination must be a path to
                        ## a directory.

mv file1 file2          # Rename file1 to file2
mv file1 directory/     # Move file to directory
mv file1 file2 directory/ # Move files to directory
```

You can use `mkdir` (MaKe DIRectory) to create new directories:

```
mkdir path              # Make directory described by path
mkdir -p directory1/directory2 # Make the directory
                        ## described and all
                        ## directories leading to
                        ## it (its Parents) if
                        ## necessary.
```

And of course you can remove those directories with `rmdir` (ReMove DIRectory). This command can only delete empty directories:

```
rmdir directory        # Remove directory
```

If you want to remove files or non-empty directories you would need to use `rm` (ReMove). Be extremely careful when you are deleting files in the terminal. The deletion on Unix is permanent and instantaneous. This means that **you won't be able to recover a deleted file**. There is no trash directory or anything like that where you can go back in case you made a mistake. So be extra careful.

```
rm file                # Remove file
rm directory           # Fails, can't remove directories
rm -r directory        # Recursively descend into directory and
                        ## delete everything, including other
                        ## directories inside of it (hence
                        ## recursively). This will remove the
                        ## directory. WARNING! A mistake using this
                        ## command could accidentally remove several
                        ## files that you didn't want to delete!
```

Escaping characters

Certain characters have a special meaning on the terminal, but sometimes you want them to be literal. For example / is the root, but you might have a file that includes this symbol. In that case you can “escape” that character by adding the prefix ‘\’ (without quotes). This will tell the terminal that the following character has to be interpreted literally and not with any special meaning. This is also useful when you have file/directory names with spaces. In the same way you could escape the space by using ‘\ ’ before it.

In any case, it is advisable to avoid special characters in file/directory names. This way you avoid possible future mistakes and gain speed.

Hidden files

In Unix, hidden files have a dot (‘.’) at the beginning of their name. They won’t appear in the output of ls unless it is specified. Hidden files are usually configuration files containing information that is important for the computer but not for the user.

Multiple files

When you have to work with several files at the same time it is handy to do it at once instead of having to type almost the same over and over. To work with multiple files we can give a pattern to the computer that will be interpreted and converted into a list that includes all those files containing the pattern.

Here we provide a brief list of special symbols we can use to create patterns:

```
*           # Match anything, including match "nothing"
?           # Match any character exactly once
[abc]       # Match exactly one of 'a', 'b' or 'c'
[^abc] or [!abc] # Match any character but 'a', 'b' or 'c'
[c-y]       # Match any character between 'c' and 'y'.
# Note: [-a] is defined to mean a match with '-' or 'a'.
{pattern1,pattern2} # Combines the result of pattern1 and
                    ## pattern2 together. Note if a file is
                    ## matched by pattern1 and pattern2, it
                    ## is returned twice.
```

For example: `ls Sample_*.fasta` will list all the files that start with Sample_ and end with .fasta.

Stdout and pipes

In Unix, programs usually take an input, do something and produce an output. That input in Unix is the “*stdin*” (STanDard INput) and the output is written into the “*stdout*” (STanDard OUTput). By default, stdout will write into the terminal where you are running the command. However, the output can be redirected into a file using the symbol “>” followed by a filename. For example: `ls > ls_output.txt`, will run the ls command and save the output in a file called ls_output.txt. Note that this file will be created every time that you run this command. This means that if you run the same command a second time, this file will be re-created and all previous information inside it will be lost.

If you want to append new lines to a file you need to use ‘>>’ instead of ‘>’. `ls >> ls_output.txt` will append new lines to the ls_output.txt file.

You could also use the stdout of a command as the input of another one. We use *pipes* “|” to concatenate multiple commands. For example: `ls | grep 'Lab1'` will first run the “ls” command and then it will search in the ls output for a line containing “Lab1”

Compressing files

Some biological files can occupy a large amount of space in disk. That is why, in many cases, we will store them as compressed files.

There are many different compression formats, but we will just see a few of them. The most common tool to compress files is `gzip` (to compress) and `gunzip` (to uncompress) .gz files.

You can also compress multiple files at the same time using `tar`. To create an archive containing file1, file2, and file3, and then compress it, use

```
$ tar -cf files.tar file1 file2 file3
$ gzip files.tar
```

To unpack the **files.tar.gz** file use

```
$ tar -xzvf files.tar.gz
```

The command will always be the same for all tar.gz files you want to unpack. -xzvf means eXtract from a Zipped file, Verbose (prints the name of the file being unpacked), from the specified File (f must always be the last of the letters).

To unpack the **files.tar.bz2** file use

```
$ tar -xjvf files.tar.bz2
```

In this case the -j indicates that the file has been compressed in the bz2 format. Modern versions of tar detect compression automatically, so the z and j flags can typically be omitted.

Working remotely

In this course we will work remotely on Uppmax. We can do that using a program called `ssh`. Secure SHell is a method of connecting to other computers and giving access to a command-line on them; once we have a command-line we can interact with the remote computer just like we interact with the local one using the command-line.

To connect to Uppmax from linux you need to open a terminal and type:

```
$ ssh -X username@rackham.uppmax.uu.se
```

Replace **username** with your uppmax user name. -X means that X-forwarding is activated on the connection, which means graphical data can be transmitted if a program requests it, i.e. programs can use a graphical user interface (GUI) if they want to.

After introducing your password you should see something like:

```
[dahlo@dahlo ~]$ ssh dahlo@kalkyl.uppmax.uu.se
Warning: Permanently added the RSA host key for IP address '130.238.136.99' to the list of known hosts.
dahlo@kalkyl.uppmax.uu.se's password:
Last login: Wed Sep 19 13:42:22 2012 from array.medsci.uu.se
```

```
| System:    kalkyl3.uppmax.uu.se  
| User:      dahlo  
| Jobs:      0 running  
| Queue:     0 pending
```

#####

User Guides: <http://www.uppmass.uu.se/support/user-guides>
FAQ: <http://www.uppmass.uu.se/faq>

Write to support@suppmax.uu.se, if you have questions or comments.

```
[dahlo@kalkyl3 work]$
```

Running programs on Uppmax

To run a program on Uppmax (**just on Uppmax, not locally**) we first need to upload its module, otherwise the program won't be found by the computer. For example if we want to run `samtools` we need to type first:

```
$ module load bioinfo-tools samtools  
$ samtools
```

NOTE: All modules are unloaded when you disconnect from UPPMAX, so you will have to load the modules again every time you log in. If you load a module in a terminal window, it will not affect the modules you have loaded in another terminal window, even if both terminals are connected to UPPMAX. Each terminal is independent of the others.

To view which module you have loaded at the moment, type

```
$ module list  
  
[dahlo@kalkyl3 old_courses]$ module list  
Currently Loaded Modulefiles:  
  1) uppmx               2) gcc/4.6               3) bioinfo-tools        4) samtools/0.1.12-10  
[dahlo@kalkyl3 old_courses]$
```

Let's say that you want to make sure you are using the latest version of `samtools`. Look at which version you have loaded at the moment (`samtools/0.1.12-10` in the example above).

Now type:

```
$ module avail
```

to see which programs are available at UPPMAX.

To change which `samtools` module you have loaded, you have to unload the module you have loaded and then load the other module. To unload a module, use

```
$ module unload <module name>
```

Look in the list from `$ module list` to see the name of the module you want to unload.

Transferring files between computers

There are several programs that you can use to transfer files between computers, but here we will use `rsync`.

The general way to do it is:

```
rsync file.txt user@machine:path    # To transfer a local file to a
```



```

                                ## remote computer
or
rsync user@machine:file.txt destination_path    # To transfer a file
                                                ## in a remote
                                                ## computer to our
                                                ## local machine.

```

If we want to transfer from uppmx to our local computer we will do something like:

```

rsync
username@rackham.uppmx.uu.se:/proj/genomeanalysis2022/nobackup/linux/transferme.txt .

```

This will transfer the file *transferme.txt* into the current directory on our local computer.

Note that you always have to run this on the terminal **on your local computer** (not on Uppmax).

Most used commands

Here we list some of the most important commands, that you would have to use during this course:

`echo "Any text."` – Displays a text

`echo -e "\nAny\text."` – Enable interpretation of backslash.

NOTE: `\t` represents a tab and `\n` a new line

`head` – show the first lines of a document (10 by default)

`head -5` – prints the first 5 lines of a document

`head -n 1` – prints the first line of a document

`tail` – show the last lines of a document

`tail -2` – shows the last 2 lines of a document

`tail -n +2` – shows the whole document but the first line

`cat file1 file2 ...` – concatenate multiple files

`cat file1` – show the content of file1

`less file1` – browse content of a file

`grep "pattern" file1` – search into a file

`grep -i "pattern" file1` – ignore case

`grep -c "pattern" file1` – gives you the number of lines which has the desired pattern

`clear` – clear screen

`wc file1` – count the number of lines, words and characters in a file

`wc -m file1` – count the number of characters in a file

`tr "x" "y"` – translate characters

`tr "\t" ";"` – translate tabs into semicolon

`tr -d "\n"` – delete newlines

`touch file1` – update the access and modification time of a file. If the file doesn't exist it creates it

`exit` – end the application

Type efficiently on the terminal:

Control-a	Move to beginning of line
Control-e	Move to end of line
Alt-f Move	forward one word
Alt-b Move	backwards one word
Control-l	Clear screen, leaving current line
Tab	Try to automatically complete path (one tab will complete it until ambiguity, pressing twice will provide a list of possible completions)