Introduction to Unix

1 The command line

When working in bioinformatics, using the command line is essential. In contrast to the usual drag and drop interfaces using the mouse, every command needs to be typed into the shell of the terminal. You can open the terminal for that with `CTRL` + `ALT` + `T`. To specify the parameters belonging to the command, a space is left after the command and then followed by the parameter.

```
> command parameter1 parameter2
```

The `>` is the command prompt, which is just the shells way of asking what it can do for you. It is important to remember that lower case or upper case letters do make a difference and can describe a different command or parameters.

If unsure which parameters can be used within a command, you can type in

```
> command -h
```

In most cases, this will list the help page with all relevant information on the command and the parameters. In case you want more detailed information on a special application of this command or some examples for its use you can also use Google for more information from experienced users. However, as always, be careful with information provided by an unknown internet source.

If you want to stop a command from running, you can press `CTRL` + `C` (alternatively: `CTRL` + `Z`). This will interrupt the execution of the command or program and you will return to the command prompt.

2 Moving along the directory tree

Some of the most basic but important commands are those that help you to navigate through your file and folder system. The following table (table 1) will help you to move between folders (directories), create, delete, copy and move files and folders and look at the content.

Table 1: Basic unix commands

Command	Result
cd *directory1*	Change to *directory1*
ls *directory1*	List content of *directory1*
mkdir *directory1*	Create *directory1*
rmdir *directory1*	Remove empty *directory1*
cp *file1 folder2*	Copy *file1* to *folder2*
mv *file1 folder2*	Move *file1* to *folder2*
rm *file1*	Delete *file1*
pwd	Show current directory

When moving between directories, you are traversing along a directory tree. This means the files are usually in folders, which are part of other folders and so on. At the top of the tree there is the `root` folder, from which the other folders branch off. A folder is usually marked by a slash `/` after the name, which can then followed by the name of a file in the folder. Usually when logging in, you will start in your `home` folder, which branches off the `root` folder (Figure 1).

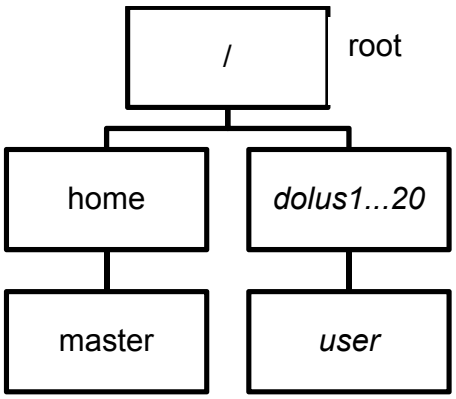

Figure 1: Folder structure within a Unix based system.

There should be a home folder for each user. You will see a description of the folder you are currently residing in in front of the prompt, which will include your user name and the computer name before and after the `@`, respectively, and the current directory path after the `~`. There are some wildcards for parameters (table 2), e.g. for using with the commands listed in table 1 to navigate between directories:

Table 2: Wildcards

Wildcard	Representation
~	Home folder
.	Current directory
..	One directory above the current one
/	Separator between directories
*	Any amount of any symbol (or any file)
?	Any symbol (takes one spot)

When stating paths to files or directories, you can either type in the absolute path from the root directory or you can type in a relative path from the current directory you are residing in (e.g. `cd ../home/master` or `/home/master`). If you do not want to type in the whole name of the folder or file, you can type in the first letters and then press the `TAB` button on your keyboard for auto-completion. If there are several possibilities, another press of the button will list these. However, if there are too many possibilities you will get noticed. Be sure if you really want to get them all listed then!

Exercise 1

Open the terminal and find out in which folder you are. Create a folder with your *user name* in there. Then create a folder called `pflaphy_exercises` in your *user* folder. Subsequently, copy the files from the `master` folder into the folder `pflaphy_exercises`. The `master` folder should be in `/home/` in the directory tree. Move into the `pflaphy_exercises` folder. Show all file names in the current directory and delete the `I_want_to_be_removed.txt` file. Ensure that it is gone from the folder! Then change the name of `I_want_a_new_name.txt` to `HMA4.txt`.

3 File types

Files usually have a file extension, separated from the filename by a dot, which shows the file type.

> `filename.extension` e.g. `data.txt`

This ensures that the file can be appropriately recognized, opened and interpreted. A simple text file usually has the ending `.txt` and there are no regulations regarding the format. Sequence files (DNA, RNA or protein) are most often in Fasta format (`.fasta` or `.fa`). Fasta files are characterized by starting with a `>`, followed by sequence name, description and putative further information. This is the so called header line. The next line contains the sequence, in which nucleotides or amino acids are represented with

their respective single letter codes. The sequences within a multi sequence fasta-file are separated by the headers which always start with the `>`.

With the new approaches of whole genome and whole transcriptome sequencing, large files containing millions of small sequences were established. These new large datasets, called *libraries*, contain millions of small sequences called *reads*. Those reads can be aligned to a reference sequence. The output, called *alignment*, can be stored in a Sequence Alignment/ Map file [short: SAM-file (`.sam`)]. SAM is a TAB-delimited text format consisting of a header section, where the lines start with `@`, and an alignment section. The header section contains additional information on the aligned sequences and alignment parameters set. The alignment section consists of 11 mandatory fields, e.g. coordinates, the sequence, matches, insertions, deletions or read pairs. Read pairs are corresponding reads from two ends of the same DNA molecule. An example line for the alignment section is given below:

```
r001 99 ref 7 30 8M2I4M1D3M = 37 39 TTAGATAAAGGATACTG *
```

This tells us that read1 (`r001`) is the first of a paired read with the mate properly mapped on the reverse strand (`99`) aligned to the reference `ref` at position `7` with a mapping quality of `30`. The cigar string `8M2I4M1D3M` indicates that there are 8 matches, followed by two insertions to the reference, followed by 4 matches, 1 deletion and another 3 matches. The `=` refers to the reference sequence name of the mate, which is in this case the same as the `ref` here. The mate is aligned at position `37`. The number of bases between the start of read1 and the end of read2 is `39` bases. The line then gives the sequence of read1 and a `*` in place of a base quality, because that information is in this case not available.

Due to storage availability, alignments are mostly converted from SAM format to BAM format (`.bam`). While the SAM format is human readable, the BAM format is not as it is a binary-file, which means that all letters, signs and codes are converted into a sequence of bytes [the binary digits (bits)] grouped in eights.

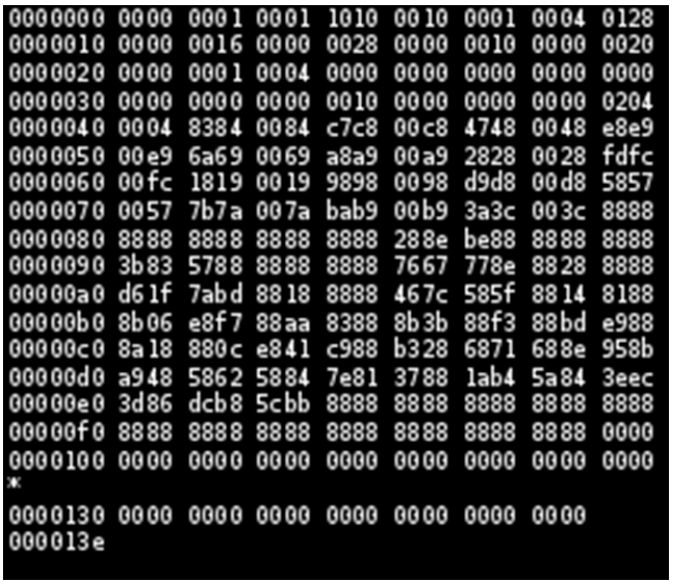

Figure 2: Example of a binary file. First column numerates the lines starting address. (This picture is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license. https://commons.wikimedia.org/wiki/File:Wikipedia_favicon_hexdump.svg)

4 Show and edit files

There are several commands available for opening or manipulating text files (table 3):

Table 3: Example of some commands that can be used to show, edit, and manipulate files.

Command	Result
less *file1*	Show content of *file1* one page at a time
more *file1*	Show content of *file1* one page at a time
cat *file1 > file2*	Save *file1* in *file2*
cat *file1 >> file2*	Concatenate *file1* and *file2*
nano *file1*	Open *file1* in the text editor Nano
head -n 10 *file1*	Show the first 10 lines of *file1*
tail -n 10 *file1*	Show the last 10 lines of *file1*
wc -l	Counts the number of lines in a file

If you want to open a file to look at the content without editing, you can use the command **less** or **more**

```
> less file1
```

or

```
> more file1
```

The main difference between the two is that you can scroll backwards with less. Both will display the text on the command window and break the text into pages if it exceeds the screen size. You can then move to the next page by pressing the `SPACE BAR` or using the `DOWN ARROW`.

The command **`cat`** will also open a file, so that the content of the file can then be used with other commands or stored in another file. Here, the `>` redirects the output of a command, in this case the opened file, to a specific file. The similar `>>` also directs the output of the last command to the specified file, but appends to this file instead of replacing possible contents. Always be aware that copying to a certain file or directing output to a file will replace the contents of this file if it already exists.

If you want to edit a file, one example of a text editor working from the command line is `Nano`. Once you have opened a file in there, you can change the text by moving with the arrows and typing on the keyboard. Copying is achieved by marking the text with the mouse, pasting by using a right click or a click on the middle wheel of the mouse. You can save the altered file by pressing `CTRL` and `X` and `Nano` will then give you the choice between altering the current file or saving it under a new name. If you want to save it under the current name, you can then just press `ENTER`, otherwise you write the new name and press enter and then confirm with `Y`.

If you only want to display the first lines or last lines of a file, you can use **`head`** or **`tail`**. If you do not add the `-n #` parameter, the default is to show the first or last 10 lines, respectively. However, you can add e.g. the `-n 12` parameter to show the first or last 12 lines. This is especially helpful for files that store meta information (additional information e.g. on the creation of the file) in the header, or for error messages, that often give a final statement (done or error) at the bottom.

The **`wc`** command can be helpful if you want to know the number of words or the number of lines of a file. **`wc`** `-l` *`file`* will give you the number of lines specifically, **`wc`** `-c` *`file`* the number of words and **`wc`** *`file`* will print the number of lines, words and digits. This can give you a first impression of the size of a text file. You can use this e.g. to determine if your concatenation has been successful.

Exercise 2

Look at the first page of the `HMA4_1.fasta` file. What do you notice about the setup of a fasta file? Do the same for `HMA4_2.fasta`. Then concatenate the 2 (`HMA4_1.fasta` before `HMA4_2.fasta`) into `HMA4.fasta`. Look at the first and last 10 lines of this new file and count the lines of `HMA4_1.fasta`, `HMA4_2.fasta` and `HMA4.fasta` to ensure

your concatenation worked. Open `HMA4.fasta` in nano and change the "`Halleri`" in line 1 to "`halleri`". Save this change.

5 Searching text files or directories

There are several commands available for finding files, text in files or to replace this text with another text (table 4). In order to find a file you can use the find command, e.g. if you are not sure if the file is contained in this specific folder. This can be useful instead of the **`ls`** command for large folders. To find text within a file, the **`grep`** command is very common. This will print all the lines of a file containing the text you have searched for. If you want to replace the text with another text, you can use **`sed`**. For extracting certain columns of a file, the **`cut`** command is helpful.

Table 4: Commands for searching a text or directory

Command	Result
find *directory1* -name '*file1*'	Find *file1* in *directory1*
grep -n '*Text*' *file1*	Show line numbers and lines containing '*Text*' in *file1*
sed -i 's/*Text1*/*Text2*/g' *file1*	Find all occurrences of '*Text1*' in *file1* and replace with '*Text2*'
cut -f1,2 -d ' ' *file1*	Take the first and second column of *file1*, which are separated by a space
awk '*pattern* {*action*}' *file1*	Perform *action* on *file1* if *pattern* is fulfilled

Most of these actions can also be achieved with the **`awk`** command, however **`awk`** is an own language and is therefore more complex.

To replace the **`grep`** command, you could also type

```
> awk '/Text/{print $0}' file1
```

To replace the **`sed`** command you could type

```
> awk '{gsub (/Text1/, "Text2"); print}' file1
```

And to replace **`cut`**

```
> awk -F' ' '{print $1,$2}' file1
```

Exercise 3

Search for the pattern "`Halleri`" in `HMA4.fasta`. If exercise 2 worked, your search will be unsuccessful. Then search for "`halleri`" and print the lines containing "`halleri`" to the screen. Replace all occurrences of "`sequence`" with "`Sequence`".

6 Piping

Sometimes it can be useful to combine commands, e.g. redirect the output of one command to another command. This can save memory if the output of the first command is only needed as input for the second command and only the result of the second command needs to be stored or printed to the screen. Additionally we can save some typing. The sign for this is the `|`. One example would be, if you want to count the number of text files in your directory:

```
> ls *.txt | wc -l
```

This assumes that all text files end with `.txt`. The ls command returns a list of all the files ending with `.txt`. This list is then the input for a line count.

Exercise 4

Find out how many files in the current directory end with `.fasta`. Print the first 2 words of `HMA4.fasta` after the "`>HE….1`". List in which line number "`>`" occurs in `HMA4.fasta` and save only these numbers in `Exercise4.txt`.

7 Run a program

After you have installed a program, you can run it by typing the name of the program or by typing `./program file` (if you want to run a certain version of a program), followed by the parameters and input file. You can direct the output via `>` to your desired output file and the error file with `2>` to your desired error file. The error file contains a log of the program, informing you about intermediate steps or if an error occurred. Most error files have a final statement at the bottom, which is either an error message or a "`Done.`" or another statement of success.

8 Write an executable script

If you want to use a combination of commands several times, it can be useful to put these into a script. That way you note which commands you used in what order and you

can run them the same way again. This is especially helpful, if you want to run a certain pipeline. You just have to change the sample names in the file and start it again.

Bash files always start with a line

```
#!/bin/bash
```

This tells the interpreter (a “translator” from your commands to the action of the computer) that this is a bash file. You can then add a comment about what this file does, who created it, when it was created and what versions of a program are used. Commands have a hashtag `#` in front of the line to signal to the interpreter, that this is not part of the script and should not be run as a command. This meta information is very important to include for your own documentation and also when changes are made or the script is shared with someone else.

After this you can add the command you want to run or the program you want to run.

You should save the file with the file extension `.sh` as another way to signal the interpreter that this is a bash file. If you write the file in `Nano`, it will automatically show a color code, e.g. for arguments in quotation marks, that will make it easier for you to keep an overview over your script.

You can write several commands in one line and separate them by a semicolon `;` to execute them in order, but for readability and understanding it can be easier to write every command into a new line. Try to make the script as easy to understand as possible by adding comments also behind commands to indicate their function (separated by `#` from the command) and grouping commands with empty lines to indicate new paragraphs of a script.

You can run the script by typing

```
> ./Script.sh
```

Exercise 5

Write a script `Script1.sh` that returns the number of lines and words of `HMA4.fasta` and the lines starting with `>`. Run the script. If it worked, replace `HMA4.fasta` with `HMA4_2.fasta` and save it as a new script `Script2.sh` and run the second script. Remember to add comments about the date you created the script, your name and what it does.