Exercises: Introduction to the Command Line

This exercise is part of the course:
"Data Interpretation of RNASeq and Bisulfite Sequencing Data in Cancer Research"
And goes together with the hand-out: Command Line Cheat Sheet

Overview: Starting at home directory

The following is for your information and can be useful anytime during the exercises:

- You will work in your home directory, you can navigate to it with these equivalent commands:
  `cd ~` or `cd /home/username` or even `cd` (without any parameters)
- You will work on data from here:
  `/bfg/data/courses/deNBI2019/Intro_bash`
- Whenever something goes wrong, you can reset your working directory (remove all files and copy exercise-files over) with the following commands:
  `rm -r ~/*; cp -r /bfg/data/courses/deNBI2019/Intro_bash ~/`
  Note: rm -r is extremely powerful, it will not prompt you again for confirmation but simply delete all files in the selected directory. Try to avoid using this one ;)

Now, let's go through the following exercises to familiarize yourself with the command line! The exercises are designed to be more than you can finish, so focus on the fun of learning the command line. Whenever you feel like it, you can play around with the commands you have learned by yourself, or you can strictly follow the instruction below, whatever works best for you. Don't worry too much about finishing it all, as in the last minutes of this session, the most important parts will be explained again with slides.

To get yourself familiar with command line, please type all commands by yourself

Better not to copy-paste the command into the Terminal!!

Explore Files and Folders: commands `cd`, `pwd` and `ls`

We have prepared some files you can explore and play with for you and will now navigate to them. But first, let's find out where you are: type the command `pwd` into your terminal and hit enter. pwd stands for **p**rint **w**orking **d**irectory, and it should tell you where you are in /home/*your-username*. You can **l**i**s**t all files present in this directory with the command `ls`: as you will see, this directory is empty, or contains some standard directories (Desktop, Documents, Downloads, etc.).

Now let's use the ls-command to look at our exercise files. Type in the following:

`ls /bfg/data/courses/deNBI2019/Intro_bash`

This should display three files within that directory. If you use `pwd` again, you can see that you are still in your home directory, and with `ls` see that it's still empty – the ls-command above only showed you the files in /bfg/data/courses/deNBI2019/Intro_bash, it did not navigate to this directory.

We can do that using the **c**hange **d**irectory command 'cd'. Simply type the following to go there:

`cd /bfg/data/courses/deNBI2019/Intro_bash` and verify it worked with `pwd`.

Your current working directory changed, so now typing `ls` without any arguments will list all the files inside, whereas you have to type `ls ~` or `ls /home/`*your-username* to display your (empty) Home directory – try it!

Directory paths, use and display special directories

As in Windows and MacOS, directories are hierarchical: the path

/bfg/data/courses/deNBI2019/Intro_bash

means that the 'bfg'-directory contains the 'data' directory, which contains the 'courses' directory and so on. There are special directories you can change to: type `cd ..` to change to the parent directory, and verify it worked with `pwd`. For completeness, the following table lists special directories in Linux:

.	Current directory
..	Parent directory
~	Your Home directory (/home/*your-username*) *note: the '~'-sign is next to the enter key on a German keyboard (press 'Alt Gr'+'*'), and next to the '1'-key on an English layout*
/	The root directory of the operating system. *Contains all other directories, such as /home/..., /bfg/... and Linux-specific folders.*

Make your life easier – hints and tips for command line

At this point you might think that the command line is all about tedious typing for things that could be done easier and faster with a GUI. But don't worry: once the tasks become more complex, bash's flexibility will convince you. And there are many features that allow you to reduce the things you actually have to type. They are listed in a text-box on the "**Command Line Cheat Sheet**" we handed out to you, but let's quickly go over a few!

If you are not there already, change to your home directory using one of these equivalent commands

`cd ~`, `cd` or `cd /home/your-username`

Let's try to list the contents of both directories (Home and "Linux") again without much

typing. Bash has a **history** that stores your most recent commands. Press the arrow-up key now: you will see the last command you have entered. If you want to execute it, hit enter, or keep pressing arrow-up to go to less recent and arrow-down for more recent commands. Go through the history until you see ' ls /bfg/data/courses/deNBI2019/Intro_bash ' and hit enter to execute it. Easy, right?

Now press arrow-up again to display the same ls-command, but this time use arrow-left instead of executing it with enter. As you can see, you can go along the command and change it: navigate to the beginning of line (arrow-left multiple times, or ctrl-a once), delete the letters 'ls' and replace it with 'cd' like this:

change ls /bfg/data/courses/ to conveniently obtain cd /bfg/data/courses/

These shortcuts can make your life much easier, but might take some practice until you get used to them. If you like them, however, simply put your "Cheat Sheet" next to your keyboard and play around with the shortcuts now or during the next exercises.

Manage files and directories: cp, mv, rm and mkdir

It is time that you get your own copy of the files now, because the three files we saw so far lie in a public location and are read-only: this means you can open and copy them, but not delete them or change their contents.

Let's copy the **surpriseFile.txt** to your home directory. First change to its location and make sure it's there:

```
cd /bfg/data/courses/deNBI2019/Intro_bash
ls          (should display the file 'surpriseFile.txt', amongst others)
```

Now copy it by typing one of these equivalent commands:

cp surpriseFile.txt ~/ (simplest version of the command, uses same filename for the copy)
cp surpriseFile.txt ~/surpriseFile.txt (explicitly states the filename of the copy you are creating)
cp /bfg/data/courses/deNBI2019/Intro_bash/surpriseFile.txt ~/surpriseFile.txt (absolute paths)

You should now see the file when typing ls ~

Note: The three versions of the command here are shown for your information, illustrating a typical command line feature: usually commands need very specific instructions (as in the third version), but sometimes offer simplifications for the command's most common usages. In the second version above, 'cp' assumes the file you are copying lies in the current working directory unless you specify an absolute path, as this is a very typical situation.

Let's create a directory next and put another copy of the file there. First, change to your home directory with cd ~, then type mkdir secondcopy. As you see when you type ls, this created a directory, and typing ls secondcopy shows you that it is, of course, empty. Try to copy your surpriseFile.txt into the newly created directory with the cp command – it's easy,

but the solution is also on the last page of this document.

Instead of copying files, you can also move them to a new location using the 'mv 'command. This can also be used to rename a file – you can change the file's name by typing this:
`mv secondcopy/surpriseFile.txt secondcopy/newname.txt`

suggestion for typing this command: try out tab-completion, e.g. type 'mv secon' and press the tab key (it's left of the 'q' key). This will auto-complete to 'mv secondcopy/', if you press tab again it will auto-complete to 'mv secondcopy/surpriseFile.txt', as this is the only file in the directory.

Look at your renamed file using the 'ls' command. Now let's remove the new copy and the new directory again:
`rm secondcopy/newname.txt` rm removes files
`rmdir secondcopy` rmdir removes empty directories (warns if not empty).

Look into files – cat, head, tail, less

After the above sections, your home directory should now contain the file 'surpriseFile.txt' – if not, copy it over now. For the next exercise, use what you learned above to also copy the surpriseFile.**csv** from the directory '/bfg/data/courses/deNBI2019/Intro_bash'. Good luck! (it's not hard, but if you need it, the solution can also be found on the last page of this document).

Let's have a look at surpriseFile.txt first. You can print out the entire file by typing `cat surpriseFile.txt`. You will see 26 lines of output, it should look something like this:

```
ProbeName,GeneName,SystematicName,sample1,sample2
A_23_P145330,CCHCR1,NM_019052,1.691658,4.144538
      . . .          . . .          . . .
```

As you can see, the output is hard to read as it is not well formatted. To improve this, you can use the 'column' command and tell it to split columns after every comma to get a nicer output: `column -ts , surpriseFile.txt`

Note: the contents of this file are not important for this course, but in case you are curious: what you see is a small subset of microarray expression data, where each line has information on a single gene (gene name, systematic gene name) and its normalized expression value in two different samples (sample1 and sample2). In a real analysis workflow, the complete file would be read in by an R script, for example, that then computes differentially expressed genes, etc..

Let's look at a few more ways to explore files next – especially if they have a lot of content.

Looking into long files

We now try the same command on the CSV-file we just copied: `column -ts , surpriseFile.csv`. As you can see, the file is much longer, and the output completely fills the terminal. In Bioinformatics, very long files are quite normal: high-throughput methods such as RNA-sequencing require that files store millions of lines with sequence fragments or entire genomes, so that that using 'cat' is not an option.

To explore text-files, and especially very long ones, you can now try the following commands to get an idea of what is in the file:

`cat surpriseFile.csv` prints out the entire file on the command line. Bad for long files.
`head surpriseFile.csv` prints out the **first** 10 lines of the file.
`tail surpriseFile.csv` prints out the **last** 10 lines of the file.
`wc surpriseFile.csv` **w**ord **c**ount: prints out the number of lines, words and characters in the file.

As you can see, there are over 2,000 lines in this file.

To navigate through the entire text file, type `less surpriseFile.csv`. The program 'less' starts and shows you the file's first page. You can quit the program anytime by pressing 'q', or move through the file page-by-page (press `space-bar` repeatedly) or line-by-line (press `arrow-up/down` keys). Less can do more than that, e.g. search for words or go to a specific page number, but for now, close it by pressing `q` on the keyboard.

Maximum flexibility on the command line using pipes

So far, the things you did on the command line could be done similarly in a GUI, and without learning so many commands by heart. We now come to combine these rather simple commands to perform increasingly complex actions.

One of the most powerful aspects of the command line is that you can use one commands output as the input for the next one, arranging commands into a 'pipe'. The character to do this is the vertical bar '|'.

Note: the '|'-sign is next to the 'y'-key on a German keyboard (press 'Alt Gr'+'<'), and next to the enter-key on an English layout (press shift + '\').

Let's say we wanted to only print the first lines of surpriseFile.csv, as done by 'head', but with nice column formatting, as done by 'column'. You can achieve this by piping the output of 'column' into 'head':

```
column -ts , surpriseFile.csv | head
```

Another example is to count all files in the current directory:

```
ls /bfg/data/courses/deNBI2019/Intro_bash | wc -l
```

In the above command, the 'ls' command lists all files in the '*Linux*' directory. Its output shows one filename per line, and these three lines are counted with 'wc -l'.

These are examples for pipes constructed with two very simple commands. It is possible to combine all kinds of commands in pipes with two or more commands in a row. The reason the command line is still so popular in the 21st century is that this enormous flexibility can solve potentially annoying, repetitive tasks with a single line of code – so you can spend your time on the important stuff!

Redirecting Output into Files: '>>', echo and '>'

Commands, and especially pipes, can fulfil complex tasks and then produce valuable output. Instead of piping it into another command or displaying it in the terminal, you can also 'redirect' the output into a file. Let's take the pipe from above, and **redirect** it into the file **fileCount.txt**:

```
ls  /bfg/data/courses/deNBI2019/Intro_bash | wc -l >> ~/fileCount.txt
```

Use `cat ~/fileCount.txt` to print out the file, then execute the above command again, and again use the 'cat' command. You should see that after the second execution of the pipe, two lines have been added to fileCount.txt. This is because the operator '>>' does not only redirect the pipe's output to a file, it also checks whether it exists and if so, **appends** the output.

A very easy way of appending something to a file is with the 'echo' command. Try the following:

`echo “third append” >> ~/fileCount.txt` and look at it with
`cat ~/fileCount.txt` You should now have three lines in the file.

If you actually do want to overwrite the file, you can use the '>' operator instead:

`echo “some Text” > ~/fileCount.txt`

Note that this difference between '>' and '>>' only exists if the file *fileCount.txt* was there before. If not, both will produce exactly the same result.

Advanced Topics

The above exercises cover already more than you need, but if you complete before time and are curious, you can look over these advanced topics.

Advanced Topics - Wildcards, xargs, grep, xargs and sed as great pipe components

Let's quickly look at wildcards and three commands that will substantially enhance the repertoire of pipes you can write.

Wildcards

You can use the asterisk symbol '*' as *wildcard*. This means that '*' will replace any filename. For example, in the command `ls ~/*`, all files inside the Home directory are selected and passed to 'ls' – so it's the same as executing `ls ~`. The great thing about wildcards is that * can also replace *parts* of filenames: type `ls ~/surprise*` and see what happens. The 'surprise*' is read as both surpriseFile.txt and surpriseFile.csv. For more usage examples, look at the second textbox on your Cheat Sheet.

xargs

Xargs is a bit complicated to understand, but shortly mentioned here for completeness, as it is ideal for building pipes. Just to illustrate how 'xargs' can come in handy, compare these two commands' outputs:

`ls surpriseFile.* | head` (displays the first lines of the file list) [there are <10, so here 'head' is superfluous]
`ls surpriseFile.* | xargs head` (displays the first ten lines of **files in** the file list)

grep

Grep on its own searches through a given file and 'grabs' all lines in which a specified pattern occurs. Type `grep GOLGA surpriseFile.csv`. This gives you all lines of the CSV-file in which the pattern 'GOLGA' occurs – this way, you can quickly look up the expression values of this protein family. Now let's see **grep in a pipe**, together with xargs:

```
ls surpriseFile.* | xargs grep GOLGA
```

This pipe searches for the GOLGA pattern in both input files (CSV and txt). As you can see, there is six lines with it in the CSV-file, but only one in the txt-file.

Note: if you want to drill down into how this pipeline works, enter the same command without 'xargs'. Why does it not find anything? Check out the solutions for the answer.

sed

sed is a stream editor. In a pipe of the structure `cat file.txt | sed someFilter`, the entire text from the file 'streams' through sed, which filters and/or changes parts of it. Many different ways to filter exist. Sed is well beyond the scope of this exercise, and the following is shown only for illustration:

```
cat surpriseFile.txt | sed "s/PAXBP/my favorite gene/" >> changedFile.txt
```

This basically copies the surpriseFile.txt to the file changedFile.txt, but at the same time **s**ubstitutes 'PAXBP' with 'my favorite gene' – cool stuff!

Advanced Topics – man pages

By typing man *command*, you will get the manual of the given command. You can navigate these pages as in the 'less' program, i.e. with arrow keys and space bar (=page down), and quit by typing 'q'. Furthermore, you can search by typing */searchterm* followed by the enter key and then go forward/backward through the search results by pressing n ('next') and p ('previous').

Most man pages are not easy to get accustomed to, but they always have the same sections, which makes it easier in time. If you're curious, give it a try: type man head and try to find out how to print out the first 5 lines instead of the first 10. The solution to this is on the last page of this document.

Advanced Topics – Bash programming

In bash, you can define variables to store values like this:

```
DIR='/bfg/data/courses/deNBI2019/Intro_bash '
```

and use them with a dollar sign '$' as prefix in commands:

```
ls $DIR
```
(equal to /bfg/data/courses/deNBI2019/Intro_bash)

More complex programming is also possible, you could e.g. construct a programming loop to create directories for all days of this course (day0, day1, day2, day3):

```
for N in 0 1 2 3
do
mkdir day$N
done
```

Advanced Topics – Math

Bash programming language can do very basic arithmetic with **expr** (expressions) command.

```
expr 5 + 2
expr 5 - 2
expr 5 \* 2          notice the escape the star(*) character
expr 5/2
```

The arithmetic operators that you're familiar with for addition (+), subtraction (-), and multiplication (*) work like you would expect them to. Only division operator (/) does not work as expected. Together with for loop above, now you can print out the multiplication table of number 2 (from 2 x 1 to 2 x 12). The solution to this is on the last page of this document.

Solutions: Introduction to the Command Line

These are the solutions for exercises which are part of the course:
"Data Interpretation of RNASeq and Bisulfite Sequencing Data in Cancer Research"
And goes together with the hand-out: Command Line Cheat Sheet

The exercises consist of mostly entering in the printed commands, but sometime you are challenged to try coming up with the commands yourself. If you can not figure those out, here are the solutions.

Solutions for section "Manage files and directories: cp, mv, rm and mkdir"

`cd ~; ls` make sure you're in the right directory and check if the file surpriseFile.txt is there

`cp surpriseFile.txt secondcopy/` copies the file, using the same filename for the copy

Solutions for section "Look into files – cat, head, tail, less"

You can copy the file surpriseFile.csv to your Home directory like this:
`cp /bfg/data/courses/deNBI2019/Intro_bash/surpriseFile.csv ~/`

Solutions for section "Advanced Topics – Wildcards, xargs, grep, xargs and sed as great pipe components

`ls surpriseFile.* | xargs grep GOLGA` does not produce anything if you leave out the xargs (`ls surpriseFile.* | grep GOLGA`) – why is that? To answer this question, put in the pipe's first part only (ls surpriseFile.*) and look at the output: it is two file names. Remember that this output – two filenames-- is given to grep to search on. None of them contains the pattern 'GOLGA' – so the output of the pipe is empty.

Solutions for section "Advanced Topics – man pages"

You can get the first 5 rows instead of 10 with the 'head' command by typing `head -n 5 filename`. You can of course type any number, and the same applies to the 'tail' command.

Solutions for section "Advanced Topics - Math"

```
for N in 1 2 3 4 5 6 7 8 9 10 11 12; do
expr 2 \* $N
done
```