Bash: cheatsheet

Some important note before start:

1- **Know your tools**: Linux is Unix-based operating system (OS), born to be free, open source, fast and safe. It is not a windows system, so if you are looking for main stream commercial software you will probably not find them (MS office, photoshop, etc.). There are alternatives to these tools, but are developed from a volunteer community, and so it often lacks of function compared to commercial alternatives. However, Linux systems wins in terms of pure performances, and their full support to many programming languages make it the first solution in term of high performance computing solution and, of course, for bioinformatics.

2- **From great powers…**: generally, a Linux distribution is completely customizable. You can remove all files and folders, and install whatever you want. This means that is also easy to remove system folders or files and damage your OS, and it become even easier using bash. Be aware of that, and think twice before launch any command!

3- **Security**: unlike Windows and, partially, Mac OS systems, Linux has very few, if not at all, problems with viruses, worms, malwares, etc. This because this system is supported by both companies and community, which are actively covering any security problem that may arise.

4- **Remember the community:** ***BASH*** (*<u>**B**</u>ourne <u>**A**</u>gain <u>**SH**</u>ell*), the most common Linux shell, is widely documented online, where you can find any solution to any problem you may find.

5- **Linux gets annoyed easily**: you will encounter A LOT of errors during your work with bash. This often happens because of common typing errors. Remember: Linux is ***case sensitive***, and so the same thing written with upper or lower case means **two different things to BASH**.

6- **Software, file & folders**: operations you can do on a folder, on a file or with a software are different. You can't write text on a folder, you can't try to find a file in text document and, of course, you can't create a subfolder in a program. A very common errori is try to use a command on the wrong data type. So remember: every command have an helper (*Command_name -h*) and a manual (*man command_name*).

7- **Tab is a great friend**: to avoid typing errors, the **Tab** key is almost mandatory. In BASH, this key means **self-completion**. For example, if I dial 'p' followed by a tab Linux will try to complete with a file, folder or software that start with a p (of course it will look for contextually to what you are doing). If there are more than one choices, it will not complete automatically. In this case, just giving a double-tab will ask Linux to show all possible choices. When I detect what I want to keep, I just have to add few character and then complete again with the tab key.

8- **Special characters**: remember that some characters have special meanings for Linux. The most common is the empty space " ", that is used as a command and argument separator. Another character is the "*****" that means "everything". So, giving "rm hi*" will remove all files that start with hi. The character "\" is used to precede empty spaces or special characters and tells Linux to consider them as normal character (eg. having a file named "file 1.doc" is common in windows; to remove it in bash is necessary a command like "rm file\ 1.doc").

Short Linux commands for bash	
pwd	Acronym of ‘***p****rint* ***w****orking* ***d****irectory*’, it shows in which folder you are (the ***working directory***).
cd	Acronym of ‘***c****hange* ***d****irectory*’, the command requires you to specify the path to the destination folder. Some examples: - ‘./’ -> this path means “this folder” (in fact, the command “cd ./” do not have any effect); - ‘../’ -> it means “the folder *above* this”, so if, for example, I’m in a folder “/home/user/documents”, the command “cd ../” will bring me to the “user” folder; - To access to a subfolder, just give folder name after the cd (e.g. ‘cd ./subfolder_name/’). - The ‘/’ after cd will point at the origin of all folders (also known as *root*, it’s the same as “C:\” in Windows systems). Remember that the folder that contains another is the **parent directory**, whereas the one inside another is a **subfolder**.
mkdir	Literally ‘***m****a****k****e* ***dir****ectory*’, it makes directory (what else…). The command requires you to specify the new folder name (eg. *“mkdir Goofy”* will make a folder named Goofy into the working folder I am). It is possible to create folder in other places just by specifying the path **BEFORE** the new folder name without separators (eg. “*mkdir ../Goofy”* will make the folder Goofy in the parent directory).
rmdir	Literally ‘***r****e****m****ove* ***dir****ectory*’, it removes EMPTY folders (if you have any file inside, i twill raise an error).
cp	Command to copy file and directory, you have to specify file(s)/folder(s) to copy. If you copy more than one file, after the full list of them you have to put the destination path, where files should be cloned. Examples: cp file1 ../target_directory cp file1 file2 file3 ../target_directory it also allows to copy and rename a file (for single file, no multiple), giving the new name after the path to the destination folder. Example. cp file1 ../target_directory/file_renamed
mv	Command to move file(s) and folder(s), works as cp. It is possible to use this command to rename files without create a copy, just by moving it in the same place, eg: mv file1 ./newname
rm	Command to remove files. It can remove multiple files just by listing them after the rm command. It is also possible to remove non-empty folders by the command: rm -r folder_2_remove/ the -r options means ‘*recursively*’, so it **recursively** removes all files, subfolder and the specified folder itself.
ls	It means ‘*list*’, and it list all files in the folder. It has several options that can be added after the command, such as: - -a : means ‘show all files, also hidden one’; - -l: means ‘show file details’; - -h: after -l, makes all details more intelligible (eg. Expresses the file size Mbyte o Gbyte instead of bytes, choosing by the file size).
head	**To use on files only**, by default shows the first ten lines in a file. It is possible to show a different number of lines just adding the ‘-‘ followed by the number of lines to show (‘-20’, ‘-30’, and so on) between the head and the file name. Example head -25 file1 Shows the first 25 lines in a file.

tail	Works as *head*, but shows the bottom *n* lines (default 10)
less	Shows the whole file. It is possible to scroll inside it with the mouse or with the keyboard. Do NOT allow file editing. The '-S' (uppercase S) option, allow to show each line keeping the format (useful with files with long lines, such as genotype or sequencing data).
grep	Find a pattern in a file. By default, if looks also for partial occurrencies (eg., looking for the word 'bees' in the file will also return 'honeybees' because the word is **included**). If I want to find only perfect match, the option '-w' followed by the pattern is needed.
wc	Acronym of ***w**ord **c**ount*, it counts words in a file. If the '-l' option is given it count lines instead of words in a file.
chmod	Acronym of ***ch**ange **mod**e*, it allows to change permission of a file (read, write, execute). The following letters can be used to change mode of a file to: - Allow (+r) or deny (-r) to read file; - Allow (+w) or deny (-w) to modify file; - Allow (+x) or deny (-x) to execute a file as a program; I can combine more of them (+rw add read and write permissions, -x remove execution permission). A more tricky way is based on the number (4 to read, 2 to write and 1 to execute) and can be applied to the owner, to the group or to all possible users. For example, the number 751 add read+write+executable option to the owner (4 + 2 + 1 = 7), read and execution to the group (4 + 1 = 5) and only execution to all other users (1). The order **must** be always the same (owner, group and all other users). Two simple examples: chmod +x file1 # Make file1 executable; chmod 754 file # give full permission to owner, read + write to group and read-only to all other.

Some special characters	
“ “	The space is the simplest special character, and is used to separate the parameter of a command line.
“\n”	The new line special character is used to tell bash to run a command
*	The asterisk special character act as a string wildcard, and it can indicate one or more characters.
|	This is the ‘*pipe*’ character, and allow to redirect output of a command to another directly. For example, the command ‘*grep -w HI File1.txt | wc -l*’ will first look for all HI occurrence in a file, and then will count for the number of lines that include that words. The final output will be the number of lines that include the word HI in *File1.txt*.
&&	Put between two command, it allows to run the second command only if the first end without errors. Eg. ‘*mkdir Goofy && mv File1.txt Goofy*’ will create the folder Goofy; if the creation will conclude with no errors, then the second command will be run moving the File1.txt in the goofy folder.
\	Known as the “escape symbol”, says to the bash console to consider a special character as a normal one. As example, if I have a file with spaces inside the name (“File 1.txt”), linux will not recognize it if written as it is. This because the space is a special character too, and serves to separate two parameter. If I want to remove the file previously said, I have to put the “\” just before the space: rm File\ 1.txt
Quotes/ Double quotes	The quote are used in a variety of function that will not discussed here. However, the simplest use of the quotes is to avoid the use of the \ character, as example: rm “File 1.txt”

Loops and control flows	

For loops	For loops allow the user to process multiple folders, range of values, list of words etc. in a one-after-another fashion.
	Here some examples:
	1. Process all subdirectories in a parent directory
	
	for subfolder in \`ls FOLDER\`; do
	   echo FOLDER/$subfolder
	done
	
	2. Extract all primary chromosomes from the cattle genome:
	
	for chromosome in {1..29} X Y MT; do
	   samtools faidx genome.fa $i > chr${chromosome}.fa
	done

While loops	While loop allow to perform an operation until a negative condition is found (i.e. while true do something, otherwise stop).
	An example of this is processing a file line by line. Assume we have a list of samples (samples.txt) to extract from a bigger csv table of data (all data.csv):
	
	while read sample; do
	   grep -w $sample data.csv
	done < samples.txt

Conditional operation	Conditional operation are used to control when do something and when not. This is done using the “if-elif-else” statements, where “if” define an initial condition, “elif” one or more alternative condition(s) and “else” what to do if no conditions are matched.
	Of these, only the “if” is mandatory, the other are optional in case complex conditions are expressed.
	The generic syntax is represented here:
	
	if [CONDITION 1]; then
	   DO SOMETHING
	elif [CONDITION 2]; then
	   DO SOMETHING ELSE
	elif [CONDITION 3]; then
	   DO SOMETHING DIFFERENT
	else;
	   WHAT TO DO IF NOTHING MATCHES
	fi
	
	Conditional operation cover a complex syntax, which can easily accessed here:
	https://linuxhint.com/bash_conditional_statement/