A non-comprehensive list of bash commands

I have listed a number of common bash commands that come in handy when working in a shell. You can try the following basic commands on the example file to understand what they are doing.

Feel free to complement your notes on the

compilation of basic commands

.

Slightly more advanced bash

AWK

This command is also considered a programming language on its own. It is particularly useful when you need to process the elements of a table. The basic syntax is as follows:

```
awk -F "\t" '{print $2 "\t" $1} file.txt
```

- -F to indicate the delimiter of your table as tabs (default is space).
- '{print}' to select with “$” the column number you want in the output.
- Text added to the output must be written inside quotation marks; in this case, the text is just the addition of a new tab.
- You can also process columns by mathematical operations. For instance, '{print $1 + $2}'

```
awk -F "\t" '$3 > 10 {print $0}'
```

- You can indicate you want only the rows in which a column meets certain condition. For example, the column 3 requires a value greater than 10.
  - Equal ==
  - Non equal =!
  - > Greater than
- '{print $0}' will print all the columns of the table

Sort

```
sort -g -u file.txt
```

- -r to sort in descending order
- -g to sort by number
- -k1,1 to sort by a specific column, in this example, first column only.
- -u make values unique
- -t ' ' to select a non-default field separator (default is tab, and in this case I changed to a space)

Translate

```
tr '[ATGC]' '[TACG]' | rev
```

- This is a trick if you are working on the complementary strand.
- It will convert each “A” into “T” and so on.
- rev will make everything read backwards.

---

sed (slightly more advanced)

```
sed -n -e '/AAA/,/BBB/ p' file.txt
```

- This will find AAA, and keep all the lines in a file until it reaches BBB. Pro tip: use this one to extract a sequence in a multi-line fasta.
- Note that using variables inside a sed command requires double quotation marks “ instead of single '.

Working with lists and tables

Play with the following sample files.

comm_join_example.tar.gz

comm

To compare contents of both files (in this case, the identifiers of the first column of the two files):

```
comm <() <()
```

- Within each ”<()“ we place the command of the input to compare.
- These should be sorted out
- I use the second part of the command (the sed) to adjust the output to have the correct number of columns

```
comm <(cut -f1 1_table.txt | sort) <(cut -f1 2_table.txt | sort) | sed -e
's/$/\t\t/' | cut -f1,2,3
```

- Output:
- First column: identifiers exclusive of the table in input 1
- Second column: identifiers exclusive of the table in input 2
- Third column: identifiers present in both tables

---

join

Join two tables based on a column in common.

```
join -t $'\t' <() <()
```

- Input within the "<()" must be sorted out.
- I highly recommend using only the first column with the identifiers to join.
- If one of the tables has repeated identifiers, the output will generate all combinations possible.
- The standard output will display only lines with columns in common. We can add option -a1 or -a2 to also include the entries of one of the tables, with no joined values from other. Do not use both.

```
join -t $'\t' -a1 <(sort -k1,1 1_table.txt) <(sort -k1,1 2_table.txt)
```