Basic bash commands

We provide below a set of basic commands to be used in the shell together with an example application. To test the effect of these command, download the file ejemplo.txt below, and give it a try. Note, the example file seems to contain useless information. Still, once you understood what the commands are doing, they will help you during the course to extract and format information from sequence files, even if they contain millions of lines.

ejemplo.txt

```
XXXXXXXXXXXXX
aaaaa   xxxxx
xxxxx   bbbbb
ccccc   xxxxx
xxxxx   ddddd
eeeee   xxxxx
aaaaa   bbbbb
ddddd   fffff
axaxa   bxbxb
XXXXXXXXXXXXX
```

Play with the following commands. If necessary, make changes to find out what they are doing. Remember to call the file only in the first command given: cat ejemplo.txt | grep “xxxxx”

Tip: You can create this file with:

```
nano ejemplo.txt
```

paste the contents of the ejemplo file into the nano editor and close it using STRG/CMD + X

Alternatively you can download the file by clicking on its name and do the exercises in a linux environment on your local computer (for example using mobaXterm on windows, or in the command line on macOS).

Reading files + counting lines

- `cat ejemplo.txt`
- `less ejemplo.txt` (press q to exit)
- `head -n3 ejemplo.txt` (returns the first three lines of the file)
- `tail -n2 ejemplo.txt` (returns the last two lines of the file)
- `cat ejemplo.txt | wc -l` (counts the number of lines in the file)
- `cat ejemplo.txt | uniq` (displays the content of the file by removing successive identical lines)

- `cat ejemplo.txt | sort | uniq` (sorts the content of the file, removes successive identical lines, and displays the output)
- `cat ejemplo.txt | sort | uniq -c` (counts the number of unique lines in the file)

Wildcards

The commands will apply to all the files in the current directory that match the pattern. “*” is any number of any character. For example, the following command will concatenate all the files that have the ending “.txt”:

```
cat *.txt
```

“?” works as any single character:

```
cat ejemplo.t?t
```

Tab completion

You can auto-complete paths by pressing the tab key once or twice anytime while writing a path. If the auto-completion does not work, there might be something wrong with your path..

You can check out this video to see how it works.

Pattern matching using grep

- `grep “xxxxx” ejemplo.txt` (search for the pattern 'xxxxx' in the file ejemplo.txt)
- `grep “xxxxx” ejemplo.txt | wc -l` (counts the number of lines in ejemplo.txt that contain the pattern 'xxxxx')
- `grep -c “xxxxx” ejemplo.txt` (also counts the number of lines in ejemplo.txt that contain the pattern 'xxxxx')
- `grep -v “x” ejemplo.txt` (returns the lines that **do not contain** the pattern)
- `grep -i “x” ejemplo.txt` (makes the pattern matching case insensitive)
- `grep -o “aaaaa” ejemplo.txt` (returns only the matching pattern)
- `grep -A2 “c” ejemplo.txt` (returns the matching plus the next two lines)
- `grep -B2 “c” ejemplo.txt` (returns the matching plus the two preceding lines)
- `grep -E “aaaaa|ccccc” ejemplo.txt` (returns lines that contain **either** 'aaaaa' or 'ccccc'

What would you have write to find lines that contain both “a” and “b”?

Regular expressions

- `grep “^x” ejemplo.txt` (returns all lines starting with an 'x')
- `grep “x$” ejemplo.txt` (returns all lines ending with an 'x')
- `grep “aaaaa…” ejemplo.txt` (How many characters are returned?)
- `grep “a*” ejemplo.txt` (returns all lines)
- `grep “a.*” ejemplo.txt`
- `grep “a[^a]” ejemplo.txt` (returns all lines containing a pattern where at least one 'a' is followed by a different character)
- `grep “a[ax]” ejemplo.txt` (returns all lines where an 'a' is either followed by an 'a' or by an 'x')

Note that the regular expressions work on “file contents”. Do not confuse with wildcards.

Regular expressions can be combined with the previous options, particularly with -o to extract characters after a pattern. Try extracting only the first 3 characters of lines that start with “a”

Text editing using sed

```
sed s/x/i/ ejemplo.txt
sed s/x/i/g ejemplo.txt
sed s/axaxa/kkkkk/ ejemplo.txt
```

Sed and special characters. (An special character might be a space, a tab(\t), a symbol reserved for regular expressions ($), etc.):

```
sed s/\t// ejemplo.txt
sed -e 's/\t//' ejemplo.txt
```

Combine with regular expressions → Try converting the first 3 characters into “iii”

Cut

Extract columns from a table

```
cut -f2 ejemplo.txt
```

Extract the first column of lines that contain the character “d”

Save/overwrite output

(ACHTUNG: do not give same name as input file)

```
cut -f2 ejemplo.txt > copy_ejemplo.txt
To save without overwriting (lines get added to file):
grep "X" ejemplo.txt >> copy_ejemplo.txt
```

Other commands

Echo: Print something on the terminal

```
echo "Witness me"
```

Translate: Convert one character to another.

```
cut -f1 ejemplo.txt
cut -f1 ejemplo.txt | tr -d '\n'
Notice anything in the output? Allow me to fix it:
cut -f1 ejemplo.txt | tr -d '\n' | sed -e 's/$/\n/'
```

Rev: print each line backwards

```
echo "a b c d e"
echo "a b c d e" | rev
```

Common errors

```
grep "XXX" ejemplo.txt | sed -e 's/X/V/g' ejemplo.txt
```

The output of the first part of the command is not getting passed as the input to the second part of the command, since the file is being read again.

```
cat ejemplo.txt > copy.txt
grep "aaa" copy.txt > copy.txt
cat copy.txt
```

Input file is also the output. The contents of input file will be brutally deleted.

```
cat ejemplo.txt > copy.txt
mv copy.txt folder
```

```
cd folder
cat folder
```

If a directory does not exist, 'mv' renames the file instead

Trying to find all “>” symbols in the following file:

important_sequence.fasta

```
>Important_sequence1
AAATTCTCACCCCTCAGAAA
>Important_sequence2
ACCTCAGAAAAATTCTCACC
```

```
grep > important_sequence.fasta
```

The crocodile “>” without quotations will be interpreted as “save file” instead of “find”. The original file will be overwritten with nothing.

```
sed -e 's/a*//' ejemplo.txt
```

Note that “*” has a different meaning as a wildcard and as a regular expression. Wildcard “*” = Regex “.*”

Some naming schemes can make your life as a programmer a living hell.

output file final final definitive 6.fasta

```
aaa
bbb
ccc
```

```
rename -e 's/_/ /g' output_file_final_final_definitive_6.fasta
cat output\ file\ final\ final\ definitive\ 6.fasta
```

Slightly more advanced bash commands

For-loop

First we will create another file. Observe differences in output when concatenating both files and

when working with them separately in a for-loop:

```
grep "fffff" ejemplo.txt > copy_ejemplo.txt
cat ejemplo.txt | wc -l
cat copy_ejemplo.txt | wc -l
```

Both files

```
cat *plo.txt | wc -l
```

For-loop

for i in *plo.txt; do echo $i; done

- A list of files that you want to process; namely, the files ejemplo.txt and copy_ejemplo.txt
- Instruction for each element of your list; in this case, print said element on the terminal. "$i" is each element of list "i"; note that the variable name is specified by the user, so I could create the list "k" and instruct each element "$k".
- The rest is part of the basic sintax of a for-loop.

We next change the instruction from the example above, to reading the file and counting its lines:

```
for i in *plo.txt; do cat $i | wc -l; done
```

Which file is which?

```
for i in *plo.txt; do cat $i | wc -l | sed -e "s/^/$i\t/"; done
```

Find the lines that contain "a", "b", and "c" in ejemplo.txt, and save them into their respective files (a.txt, b.txt, c.txt). This means, each file should look like this:

a.txt

```
aaaaa   xxxxx
aaaaa   bbbbb
axaxa   bxbxb
```

Hint: let me start the command for you:

```
for i in a b c; do ...
```

AWK

This command is also considered a programming language on its own. It is particularly useful when

needing to process the elements of a table. The basic syntax is the following:

awk -F “\t” '{print $2 “\t” $1}'

- Indicate the delimiter of the table. Here it is specified to tabs, but the default is space.
- $ indicates the number of column that you wish to return
- Text added to the table must be written inside quotation marks; in this case the text is the addition of a tab character.
- Columns can also be processed with mathematical operations. For instance, print $1 + $2.

```
grep "aaaaa" ejemplo.txt
grep "aaaaa" ejemplo.txt | awk -F "\t" '{print $2 "\t" $1}'
```

grep “aaaaa” ejemplo.txt | awk -F “\t” '$2 == “xxxxx” {print $0}'

- You can indicate to only print the rows where the column meets a certain condition. In this case, column 2 must be equal to “xxxxx”.
  - Equal ==
  - Not equal !=
  - Greater than >
- $0 prints all columns of the table

If / Else

```
cat ejemplo.txt | wc -l
if [[ $(cat ejemplo.txt | wc -l) -ge 9 ]] ; then echo "File has nine or more
lines"; else echo "File has less than nine lines"; fi
```

To compare numbers:

- -eq = is equal to
- -ne = is not equal to
- -ge = is greater than or equal to
- -le = is lesser than or equal to
- -gt = is greater than
- -lt = is lesser than