Bash_Practical_Work

November 28, 2018

Written by Nadège Guiglielmoni.

1 Useful tools

This part aims to introduce you to a variety of tools that are essential in a day to day use of the shell.

1.1 Read The Fucking Manual (RTFM) : MAN

Whenever you want to use a command you don't know, you can use "man nameofcommand" to access its manual page. This page gives details about what this command can do and its options.

```
In [ ]: man echo
```

If you execute this command, you can see that the command echo is used to "display a line of text". man shows the purpose of the command, and how it should be used : how to pass the input, the parameters which can be used...

1.2 ECHO

Display a string (sequence of characters).

This command can be used to display a string passed as an argument.

```
In [1]: echo "Hello world"

Hello world
```

echo can also be used to display the value of a variable.

1.3 Create a variable

A variable can be created by attributing a value to a name, in the following manner : name=variable In shell, you must not put spaces between the name, the symbol =, and the variable. The value can be a number, a string (sequence of characters)...

```
In [2]: var1=1         #integer variable
        var2="IGNITE" #character variable
```

You can display the variable with the command 'echo', for example.

```
In [3]: echo $var1
        echo $var2

1
IGNITE
```

You may have noticed here that to call the variable, it needs to be preceeded by $. In addition, it is good practice to call a variable ${var1} instead of $var1, or ${var2} instead of $var1. It is not necessary in many cases, but if you use ${} and not just $, you are sure that your variable will be called correctly.

```
In [4]: echo ${var1}
        echo ${var2}

1
IGNITE
```

Here is an example of the limits of $:

```
In [5]: echo "$var2 is awesome!"
        echo "This is an $var2mare"

IGNITE is awesome!
This is an
```

And here is how this problem is solved with ${} :

```
In [6]: echo "This is an ${var2}mare"

This is an IGNITEmare
```

With ${} the computer knows where the name of the variable starts and where it ends. This notation is better suited for example when you call variables in a path.

The value attributed to a variable can be modified :

```
In [7]: echo ${var2}
        var2="ITN"
        echo ${var2}

IGNITE
ITN
```

When a variable has been attributed a number as a value, this value can be used in simple operations :

```
In [8]: echo ${var1}
        (( var1 = var1 + 5 ))
        echo ${var1}
        (( var1 = var1 - 2))
        echo ${var1}
        (( var1 = var1 * 3))
        echo ${var1}
```

```
1
6
4
12
```

1.4 LS

Prints the list of elements (files and directories) in a directory.

```
In [9]: ls
```

```
Bash_Practical_Work.ipynb  Lachancea_meyersii        LAKL.ans.cds  test_awk.txt
Lachancea_dasiensis        Lachancea_thermotolerans  LAME.ans.cds
Lachancea_kluyveri         LADA.ans.cds              LATH.ans.cds
```

Here, you can see all the directories and files in the current directory.

1.5 PWD

Print working directory. This command shows the path to the directory you are in.

```
In [10]: pwd
```

```
/home/nadege/ignite_workshop
```

/home/nadege/ designates the user home directory. It can be called with ~ . Here you are located in the directory "ignite_workshop". You can call directly all the files and directories in the directory "ignite_workshop". To access files and directories in other directories, you can either : 1) change directory 2) type the path leading to this element.

~/ignite_workshop and /home/nadege/ignite_workshop are equivalent in this situation (on my computer).

1.6 CD

Change directory. To switch to another directory.

We are currently in the directory "ignite_workshop", and the command ls has showed you all the directories and files in the current directory "ignite_workshop".

Earlier, when using the command ls, you may have noticed that there is a directory called "Lachancea_dasiensis". You can use cd to go in that directory.

```
In [11]: cd Lachancea_dasiensis                    #to get into the directory Lachancea_dasie
         pwd                                       #check what is the current directory
         ls                                        #take a look at the files and directories
                                                   #in this directory
```

```
/home/nadege/ignite_workshop/Lachancea_dasiensis
LADA0A.fsa  LADA0C.fsa  LADA0E.fsa  LADA0G.fsa
LADA0B.fsa  LADA0D.fsa  LADA0F.fsa  LADA0H.fsa
```

The result of the command ls is different from the one you had earlier. This is because you changed directory, and you now have directly access to all the files and directories in the directory "Lachancea_dasiensis".

You can type cd .. to go to the upper directory.

```
In [12]: cd ..                                     #to go to the upper folder
         pwd                                       #check what is the current directory
         ls                                        #take a look at the files and directories
                                                   #in this directory
```

```
/home/nadege/ignite_workshop
Bash_Practical_Work.ipynb  Lachancea_meyersii         LAKL.ans.cds  test_awk.txt
Lachancea_dasiensis        Lachancea_thermotolerans  LAME.ans.cds
Lachancea_kluyveri         LADA.ans.cds              LATH.ans.cds
```

The symbol ~ is used to designate the user home directory. The command cd ~ switches the environment to the user home directory.

```
In [13]: cd ~                                      #to go to the home directory
         pwd                                       #check what is the current directory
```

```
/home/nadege
```

Here, the working directory is "/home/nadege" which is the same as ~ .

To go in a specific directory, you can use cd with full path leading to this directory.

```
In [14]: cd ~/ignite_workshop/Lachancea_dasiensis/  #to go to Lachancea_dasiensis with
                                                   #full path
         pwd                                       #check what is the current directory
```

```
/home/nadege/ignite_workshop/Lachancea_dasiensis
```

To go back to the directory "ignite_workshop", you can use cd and the path to the directory "ignite_workshop".

```
In [15]: cd ~/ignite_workshop/                     #to go to ignite_workshop with full path
         pwd                                       #check what is the current directory
```

```
/home/nadege/ignite_workshop
```

Since you were located in the directory "Lachancea_dasiensis" which is a subdirectory of "ignite_workshop", you could have used cd .. to go back to the upper folder "ignite_workshop", and the result would have been the same.

1.7 TREE

List contents of directories in a tree-like format.

Directories are organised as an arborescence, where subdirectories and files of a directory are its ramifications. The command tree can be used to visualize these ramifications.

```
In [16]: tree
```

```
.
 Bash_Practical_Work.ipynb
 Lachancea_dasiensis
ãã  LADA0A.fsa
ãã  LADA0B.fsa
ãã  LADA0C.fsa
ãã  LADA0D.fsa
ãã  LADA0E.fsa
ãã  LADA0F.fsa
ãã  LADA0G.fsa
ãã  LADA0H.fsa
 Lachancea_kluyveri
ãã  SAKL0A.fsa
ãã  SAKL0B.fsa
ãã  SAKL0C.fsa
ãã  SAKL0D.fsa
ãã  SAKL0E.fsa
ãã  SAKL0F.fsa
ãã  SAKL0G.fsa
ãã  SAKL0H.fsa
 Lachancea_meyersii
ãã  LAME0A.fsa
ãã  LAME0B.fsa
ãã  LAME0C.fsa
ãã  LAME0D.fsa
ãã  LAME0E.fsa
ãã  LAME0F.fsa
ãã  LAME0G.fsa
ãã  LAME0H.fsa
 Lachancea_thermotolerans
ãã  CU928165.fsa
ãã  CU928166.fsa
ãã  CU928167.fsa
ãã  CU928168.fsa
```

```
ãã  CU928169.fsa
ãã  CU928170.fsa
ãã  CU928171.fsa
ãã  CU928180.fsa
 LADA.ans.cds
 LAKL.ans.cds
 LAME.ans.cds
 LATH.ans.cds
 test_awk.txt

4 directories, 38 files
```

Here you can see all the subdirectories as well as the files in the directory and in the subdirectories.

1.8 TOUCH

This command is described as "change file timestamps", but it is often used to create a file.

```
In [17]: touch test.txt #to create the file named "test.txt"

         ls             #to see the list of elements in the directory
                        #and check that the file has been created
```

```
Bash_Practical_Work.ipynb  Lachancea_meyersii        LAKL.ans.cds  test_awk.txt
Lachancea_dasiensis        Lachancea_thermotolerans  LAME.ans.cds  test.txt
Lachancea_kluyveri         LADA.ans.cds              LATH.ans.cds
```

Here you can see that the file "test.txt" now appears in the list of elements in the directory.

1.9 MKDIR

To create a directory.

```
In [18]: mkdir test #to create the directory

         ls         #to check the list of elements in the current directory
                    #and check that the new directory has been created
```

```
Bash_Practical_Work.ipynb  Lachancea_meyersii        LAKL.ans.cds  test
Lachancea_dasiensis        Lachancea_thermotolerans  LAME.ans.cds  test_awk.txt
Lachancea_kluyveri         LADA.ans.cds              LATH.ans.cds  test.txt
```

mkdir -p creates a directory and does not display an error if the directory already exists.

1.10 RM

To remove an element.

To remove a directory, the option -r has to be added.

```
In [19]: rm test.txt #delete the file test.txt

         rm -r test  #delete the directory test

         ls          #check that they have been deleted
```

```
Bash_Practical_Work.ipynb  Lachancea_meyersii        LAKL.ans.cds  test_awk.txt
Lachancea_dasiensis        Lachancea_thermotolerans  LAME.ans.cds
Lachancea_kluyveri         LADA.ans.cds              LATH.ans.cds
```

rm should be used carefully : when a file or a full directory is deleted with rm, it is gone. Good luck retrieving it.

1.11 Send the output to a file

You can send the output of a command to a file with > after the command.

```
In [20]: ls > test.txt
```

You can use >> to simply add to a file. If you use > and the file already exists, the existing file is going to be replaced by the new file.

1.12 Visualize a file's content

You can see what is in a file by using more.

```
In [21]: more test.txt
```

```
Bash_Practical_Work.ipynb
Lachancea_dasiensis
Lachancea_kluyveri
Lachancea_meyersii
Lachancea_thermotolerans
LADA.ans.cds
LAKL.ans.cds
LAME.ans.cds
LATH.ans.cds
test_awk.txt
test.txt
```

The commands head and tail allow you to get only the first lines or the last lines of a file. You can add a parameter to indicate how many lines you want to see.

```
In [22]: head -2 test.txt

Bash_Practical_Work.ipynb
Lachancea_dasiensis


In [23]: tail -2 test.txt

test_awk.txt
test.txt
```

1.13 MV

To move an element somewhere else.

For example, the file called test.txt is moved to the directory "Lachancea_dasiensis" :

```
In [24]: mv test.txt Lachancea_dasiensis #move test.txt to Lachancea_dasiensis

         cd Lachancea_dasiensis          #switch to Lachancea_dasiensis directory

         ls                              #check that test.txt is in Lachancea_dasiensis

LADA0A.fsa  LADA0C.fsa  LADA0E.fsa  LADA0G.fsa  test.txt
LADA0B.fsa  LADA0D.fsa  LADA0F.fsa  LADA0H.fsa
```

Here you can see that if you change your directory to the directory "Lachancea_dasiensis", you can see that you now find the file test.txt in this directory.

mv can also be used to change a file's name.

```
In [25]: mv test.txt foo.txt
         ls

foo.txt     LADA0B.fsa  LADA0D.fsa  LADA0F.fsa  LADA0H.fsa
LADA0A.fsa  LADA0C.fsa  LADA0E.fsa  LADA0G.fsa
```

The file test.txt is now found under the name foo.txt.

1.14 CP

Copy files and directories.

Here to make a copy of the file foo.txt in the file foo2.txt :

```
In [26]: cp foo.txt foo2.txt #make a copy
         ls                  #check the existence of foo2.txt

foo2.txt  LADA0A.fsa  LADA0C.fsa  LADA0E.fsa  LADA0G.fsa
foo.txt   LADA0B.fsa  LADA0D.fsa  LADA0F.fsa  LADA0H.fsa
```

1.15 GREP

grep is used to find a pattern in a string.

```
In [27]: grep Lachancea foo.txt #find the lines that match the pattern "Lachancea"
                                #in the file foo.txt

Lachancea_dasiensis
Lachancea_kluyveri
Lachancea_meyersii
Lachancea_thermotolerans
```

grep -v is used to show only the lines that do not match the pattern.

```
In [28]: grep -v Lachancea foo.txt #find the lines that do not match the pattern
                                   #"Lachancea" in the file foo.txt

Bash_Practical_Work.ipynb
LADA.ans.cds
LAKL.ans.cds
LAME.ans.cds
LATH.ans.cds
test_awk.txt
test.txt
```

grep -r searches the pattern in all the files in your directory, including the files in the subdirectories.

```
In [29]: grep -r Lachancea #searches "Lachancea" in all the files in the directory
                           #and subdirectories

LADA0H.fsa:>LADA0H Lachancea dasiensis CBS10888 chromosome H, complete assembly
LADA0F.fsa:>LADA0F Lachancea dasiensis CBS10888 chromosome F, complete assembly
LADA0B.fsa:>LADA0B Lachancea dasiensis CBS10888 chromosome B, complete assembly
foo.txt:Lachancea_dasiensis
foo.txt:Lachancea_kluyveri
foo.txt:Lachancea_meyersii
foo.txt:Lachancea_thermotolerans
foo2.txt:Lachancea_dasiensis
foo2.txt:Lachancea_kluyveri
foo2.txt:Lachancea_meyersii
foo2.txt:Lachancea_thermotolerans
LADA0A.fsa:>LADA0A Lachancea dasiensis CBS10888 chromosome A, complete assembly
LADA0D.fsa:>LADA0D Lachancea dasiensis CBS10888 chromosome D, complete assembly
LADA0C.fsa:>LADA0C Lachancea dasiensis CBS10888 chromosome C, complete assembly
LADA0E.fsa:>LADA0E Lachancea dasiensis CBS10888 chromosome E, complete assembly
LADA0G.fsa:>LADA0G Lachancea dasiensis CBS10888 chromosome G, complete assembly
```

grep -c shows the number of lines that match the pattern in a file.

```
In [30]: grep Lachancea foo.txt -c
```

```
4
```

There are 4 lines in foo.txt that matched the pattern "Lachancea".

```
In [31]: grep -r Lachancea -c
```

```
LADA0H.fsa:1
LADA0F.fsa:1
LADA0B.fsa:1
foo.txt:4
foo2.txt:4
LADA0A.fsa:1
LADA0D.fsa:1
LADA0C.fsa:1
LADA0E.fsa:1
LADA0G.fsa:1
```

There are 4 lines in foo.txt that matched the pattern "Lachancea", while only 1 line matched in the other files.

1.16 WC

wc prints the number of lines, words, and bytes in a file.

```
In [32]: wc foo.txt
```

```
 11  11 183 foo.txt
```

wc -l only shows the number of lines.

```
In [33]: wc -l foo.txt
```

```
11 foo.txt
```

wc -c only shows the number of words.

```
In [34]: wc -c foo.txt
```

```
183 foo.txt
```

1.17 Pipe |

The symbol | is used to send directly the output from a command to another command, and it is called pipe.

For example, you can use 'ls' to list the files in your directory, and then use 'grep' to select the ones that match the pattern "fsa".

```
In [35]: ls | grep fsa
```

```
LADA0A.fsa
LADA0B.fsa
LADA0C.fsa
LADA0D.fsa
LADA0E.fsa
LADA0F.fsa
LADA0G.fsa
LADA0H.fsa
```

You may notice that the input was not indicated in the 'grep' command because the input is the output from 'ls'.

1.18 AWK

We will not go into details about awk, because awk is a mini-programming language. But it is commonly used for columns selection.

The command more shows you what is in the file "test_awk.txt" :

```
In [36]: cd .. #switch to upper directory ignite_workshop

         more test_awk.txt #show test_awk.txt
```

```
LADA0A LADA0A00210g 16 15850 17448
LADA0A LADA0A00254g 16 19637 20305
LADA0A LADA0A00276g 0 20619 22100
LADA0A LADA0A00298g 0 23104 25068
LADA0A LADA0A00320g 0 25282 25926
```

Then you can choose to select a specific column in this file, here the second column :

```
In [37]: awk '{print $2}' test_awk.txt #print the second column of test_awk.txt
```

```
LADA0A00210g
LADA0A00254g
LADA0A00276g
LADA0A00298g
LADA0A00320g
```

You can select several columns at the same time, here the first and second colums :

```
In [38]: awk '{print $1,$2}' test_awk.txt #print first and second columns

LADA0A LADA0A00210g
LADA0A LADA0A00254g
LADA0A LADA0A00276g
LADA0A LADA0A00298g
LADA0A LADA0A00320g
```

1.19 CUT

Remove sections for each line of files.

cut can be used in a similar as what was done previously to select columns with awk. cut cuts a string on a specific delimiter.

The option -d indicates at what character (delimiter) the string should be cut. The option -f indicates which field to select after cutting.

```
In [39]: cut -f 2 -d " " test_awk.txt

LADA0A00210g
LADA0A00254g
LADA0A00276g
LADA0A00298g
LADA0A00320g
```

Here the lines from the file test_awk.txt are cut at each space " ". Then, the second field is selected. As a result, the second column is displayed.

You can also select several fields :

```
In [40]: cut -f 1-2 -d " " test_awk.txt

LADA0A LADA0A00210g
LADA0A LADA0A00254g
LADA0A LADA0A00276g
LADA0A LADA0A00298g
LADA0A LADA0A00320g
```

Here the lines from the file test_awk.txt are cut at each space " ". Then, the first and second fields (1-2) are selected. As a result, the first and second columns are displayed.

But strings can also be cut at another character, for example here the lines are cut on the delimiter "L" and the third field is selected :

```
In [41]: cut -f 3 -d L test_awk.txt
```

```
ADA0A00210g 16 15850 17448
ADA0A00254g 16 19637 20305
ADA0A00276g 0 20619 22100
ADA0A00298g 0 23104 25068
ADA0A00320g 0 25282 25926
```

1.20 SORT

Sort lines of text files.

By default, the lines are going to be sorted according to the first column.

```
In [42]: sort test_awk.txt
```

```
LADA0A LADA0A00210g 16 15850 17448
LADA0A LADA0A00254g 16 19637 20305
LADA0A LADA0A00276g 0 20619 22100
LADA0A LADA0A00298g 0 23104 25068
LADA0A LADA0A00320g 0 25282 25926
```

In the file test_awk.txt, the values in the first column are all identical (LADA0A). Hence the file is sorted according to the second column.

The user can specify on which column the file should be sorted, with the option -k :

```
In [43]: sort -k 3 test_awk.txt
```

```
LADA0A LADA0A00276g 0 20619 22100
LADA0A LADA0A00298g 0 23104 25068
LADA0A LADA0A00320g 0 25282 25926
LADA0A LADA0A00210g 16 15850 17448
LADA0A LADA0A00254g 16 19637 20305
```

The file is sorted according to the third column. The lines with the value "0" are put first, and the lines with the value "16" are put after.

1.21 GZIP

You can compress a file with gzip, and then decompress with gunzip.

```
In [44]: gzip test_awk.txt #compress file
         ls
```

```
Bash_Practical_Work.ipynb  Lachancea_thermotolerans  LATH.ans.cds
Lachancea_dasiensis        LADA.ans.cds              test_awk.txt.gz
Lachancea_kluyveri         LAKL.ans.cds
Lachancea_meyersii         LAME.ans.cds
```

```
In [45]: gunzip test_awk.txt.gz #decompress file
         ls
```

```
Bash_Practical_Work.ipynb  Lachancea_meyersii        LAKL.ans.cds  test_awk.txt
Lachancea_dasiensis        Lachancea_thermotolerans  LAME.ans.cds
Lachancea_kluyveri         LADA.ans.cds              LATH.ans.cds
```

2 Get lazy with *

The symbol * can be interpreted as "all" or "anything". The symbol is replaced by any possible combination.

For example, if you want to get the number of lines in all the files with extension .cds in the directory, you designate these files by *.cds.

```
In [46]: wc -l *.cds
```

```
  4814 LADA.ans.cds
  4931 LAKL.ans.cds
  4847 LAME.ans.cds
  4879 LATH.ans.cds
 19471 total
```

Here, all these files were treated by the command 'wc -l' because they all matched .cds .

Now, we want to compute the number of lines in all the files with the extension .fsa, in all the subdirectories.

```
In [47]: wc -l */*.fsa
```

```
 13914 Lachancea_dasiensis/LADA0A.fsa
 17783 Lachancea_dasiensis/LADA0B.fsa
 19803 Lachancea_dasiensis/LADA0C.fsa
 20799 Lachancea_dasiensis/LADA0D.fsa
 23510 Lachancea_dasiensis/LADA0E.fsa
 25749 Lachancea_dasiensis/LADA0F.fsa
 26368 Lachancea_dasiensis/LADA0G.fsa
 30446 Lachancea_dasiensis/LADA0H.fsa
 15859 Lachancea_kluyveri/SAKL0A.fsa
 18659 Lachancea_kluyveri/SAKL0B.fsa
 20876 Lachancea_kluyveri/SAKL0C.fsa
 21491 Lachancea_kluyveri/SAKL0D.fsa
 21594 Lachancea_kluyveri/SAKL0E.fsa
 23089 Lachancea_kluyveri/SAKL0F.fsa
 28956 Lachancea_kluyveri/SAKL0G.fsa
 38584 Lachancea_kluyveri/SAKL0H.fsa
 13283 Lachancea_meyersii/LAME0A.fsa
 13659 Lachancea_meyersii/LAME0B.fsa
```

```
  16381 Lachancea_meyersii/LAME0C.fsa
  19559 Lachancea_meyersii/LAME0D.fsa
  22901 Lachancea_meyersii/LAME0E.fsa
  33554 Lachancea_meyersii/LAME0F.fsa
  34150 Lachancea_meyersii/LAME0G.fsa
  34221 Lachancea_meyersii/LAME0H.fsa
  11463 Lachancea_thermotolerans/CU928165.fsa
  14897 Lachancea_thermotolerans/CU928166.fsa
  16656 Lachancea_thermotolerans/CU928167.fsa
  25227 Lachancea_thermotolerans/CU928168.fsa
  25364 Lachancea_thermotolerans/CU928169.fsa
  27217 Lachancea_thermotolerans/CU928170.fsa
  28669 Lachancea_thermotolerans/CU928171.fsa
  23733 Lachancea_thermotolerans/CU928180.fsa
 728414 total
```

Here, the first * has been replaced by all the possible subdirectories, and then for each directory the second * has been replaced by all the possible files with the extension .fsa . But you should that not all commands could handle a group of files instead of a single file. In these instances, you can use a loop.

3 Get super lazy with loops

With loops, you can automatize tasks to treat plenty of files while you just lay back and watch your computer work (or attend to other tasks).

You can use the loops 'while' and 'for' as you would in other languages.

3.1 WHILE

The 'while' loop is written in the following way :

```
while [ condition ]
do
  task
done
```

If the condition is true, then the loop keeps going; if the condition is false, it stops.

```
In [48]: i=0

         while [ ${i} -le 5 ]
         do
             echo "The number is ${i}"
             ((i++)) #to increment i
         done
```

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
```

Here, the condition that has been tested is whether i is less or equal to 5 :
-le : less or equal
-lt : less than
-ge : greater or equal
-gt : greater than

3.2 FOR

The 'for' loop picks variables in a list and treats them iteratively. It is written in the following way :

---

for i in list
do
  task

---

done

```
In [49]: #create list where variables will be picked
         list="I G N I T E"

         for i in $list
         do
             echo ${i}
         done
```

```
I
G
N
I
T
E
```

'for' loops can be used to iterate on a set of files.

```
In [50]: for i in *
         do
             echo ${i} #display file name
         done
```

```
Bash_Practical_Work.ipynb
Lachancea_dasiensis
Lachancea_kluyveri
Lachancea_meyersii
Lachancea_thermotolerans
LADA.ans.cds
LAKL.ans.cds
LAME.ans.cds
LATH.ans.cds
test_awk.txt
```

```
In [51]: for i in *.cds
         do
             echo ${i}    #display file name
             head -1 ${i} #display first line of the file
         done
```

```
LADA.ans.cds
LADA0A LADA0A00210g 16 15850 17448
LAKL.ans.cds
LAKL0A SAKL0A00132g 16 9038 9973
LAME.ans.cds
LAME0A LAME0A00298g 16 21142 21381
LATH.ans.cds
CU928165 KLTH0A00308g 0 23428 25053
```

4 Get even lazier with tests

4.1 IF ELSE ELIF

You can use tests to execute a task only if the test is true. For example, you could imagine to test whether a file exists before executing a task. This can be useful to not execute a task if the output already exists, or if the input does not exist. The structure is as follow :

---

if [condition]
then
  task

---

fi
Here we test if 10 is greater than 1.

```
In [52]: if [ 10 -gt 1 ] #if 10>1
         then
             echo "10 is greater than 1"
         fi
```

```
10 is greater than 1
```

The condition is true so the command echo has been executed.

```
In [53]: if [ 10 -lt 1 ] #if 10<1
         then
             echo "10 is lower than 1"
         fi
```

Here, the message is not displayed because the condition is false.
You can introduce an alternative if the condition is false. The structure is :

---

if [condition]
then
  task
else
  task
fi

Here we test whether 10 is less than 1. If it is, the message "10 is lower than 1" is displayed. If it is not, the message "10 is not lower than 1" is displayed.

```
In [54]: if [ 10 -lt 1 ] #if 10<1
         then
             echo "10 is lower than 1"
         else
             echo "10 is not lower than 1"
         fi
```

```
10 is not lower than 1
```

The task in the alternative else has been executed.
You can introduce alternatives with other tests :

---

if [condition]
then
  task
elif
  task
else
  task
fi

The variable called var is attributed the value "a". The following tests check whether the variable is "a", "b" or "c". If none of these tests is true, then the message "I have no idea what this letter is" is displayed.

```
In [55]: var="a"

         if [ ${var} = "a" ]
         then
             echo "The letter is a"
         elif [ ${var} = "b" ]
         then
             echo "The letter is b"
         elif [ ${var} = "c" ]
         then
             echo "The letter is c"
         else
             echo "I have no idea what this letter is"
         fi
```

```
The letter is a
```

4.2 Tests in loops

Tests can be added to loops. In the following example, the for loop iterates over the characters in the list contained in the variable var. i is going to successively take the values "a", "z", "e", "c", "t", "y", "b", "b", "e", "a", and each time the tests to check whether i is "a", "b", "c" or something else are ran.

```
In [56]: var="a z e c t y b b e a"

         for i in ${var}
         do
             if [ ${i} = "a" ]
             then
                 echo "The letter is a"
             elif [ ${i} = "b" ]
             then
                 echo "The letter is b"
             elif [ ${i} = "c" ]
             then
                 echo "The letter is c"
             else
                 echo "I have no idea what this letter is"
             fi
         done
```

```
The letter is a
I have no idea what this letter is
I have no idea what this letter is
The letter is c
I have no idea what this letter is
I have no idea what this letter is
The letter is b
The letter is b
I have no idea what this letter is
The letter is a
```

4.3 Test for the existence of a file or a directory

You can test for the existence of a file with -f or a directory with -d.

The following example tests for the existence of the file test_awk.txt :

```
In [57]: if [ -f test_awk.txt ]
         then
             echo "The file exists."
         fi

The file exists.
```

This example tests for the non existence of the file test_awk.txt :

```
In [58]: if [ ! -f test_awk.txt ] #test that it does not exist
         then
             echo "The file does not exist"
         fi
```

The file exists so the condition is not true.

Then we test for the existence of the file ignite.txt. In the case where ignite.txt would not exist, the message "The file does not exist." is displayed.

```
In [59]: if [ -f ignite.txt ]
         then
             echo "The file exists."
         else
             echo "The file does not exist."
         fi

The file does not exist.
```

The following example tests for the existence of the directory "Lachancea_dasiensis" :

```
In [60]: if [ -d Lachancea_dasiensis ]
         then
             echo "The directory exists."
         fi

The directory exists.
```

The following example tests for the existence of the directory "Lachancea_cidri" :

```
In [62]: if [ -d Lachancea_cidri ]
         then
             echo "The directory exists."
         fi
```

The message is not displayed because the file does not exist.

5 Exercises

The files that have been provided are :

- .fsa : the sequences for each chromosome of each Lachancea species; the lines starting with ">" indicate the name of the sequence, and the other lines are the sequence
- .cds : the CDS (Coding DNA Sequences) for each Lachancea species; the first column is the chromosome name; the second column is the CDS name; the third column indicates whether the CDS is on the + or - strand; the following numbers are the start position and the end position of the CDS

1. Delete the file "test_awk.txt".
2. How many .cds files are there ?
3. Compress all the .cds files.
4. How many lines are there in each .cds files ?
5. How many CDS are there on the chromosome LADA0A in LADA.ans.cds ?
6. Create a file with the name of all the CDS of all species.
7. How many files are there in each Lachancea folder ?
8. What are the names of all the chromosomes in all the .fsa sequence files?
9. How many times is the restriction site "ATGC" found in the chromosome LADA0A ?
10. Create a file which indicates the name of each chromosome and its length.