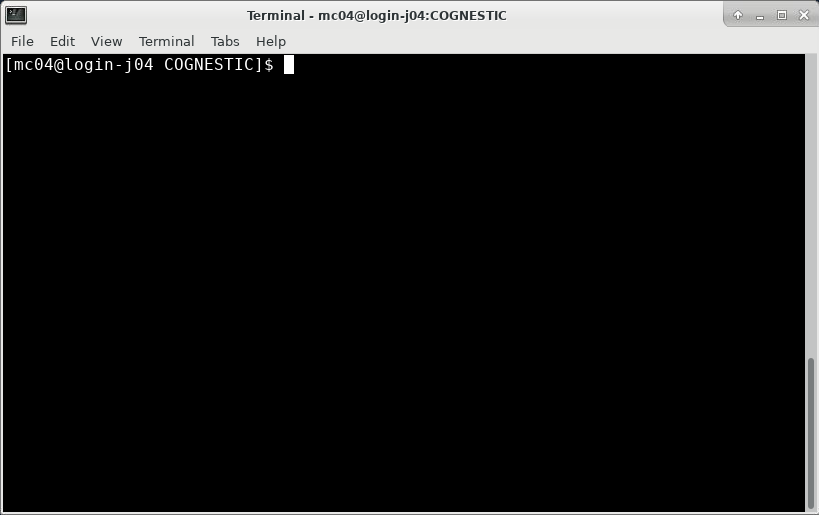
**GOGNESTIC 2023: Introduction to the command line and bash scripts**

**The Command Line**

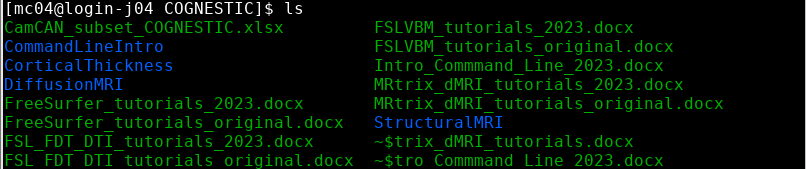
Command prompt: [user@machine name\_of\_current\_working\_directory]



**pwd**: print working directory



**ls**: list directory contents



Command argument: provide and input to a given command. For example, using command ls with an argument:



**cd**: change directory



**mkdir**: make new directory



**cd ..** : go back one directory

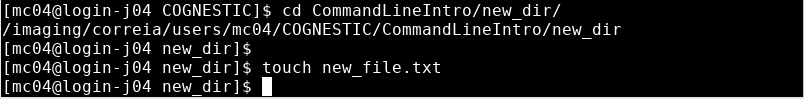


Other examples:





**touch**: create new file



Text editors: **vim**, **gedit**





**cat**: view text file



Other options for larger files: **less**, **more**

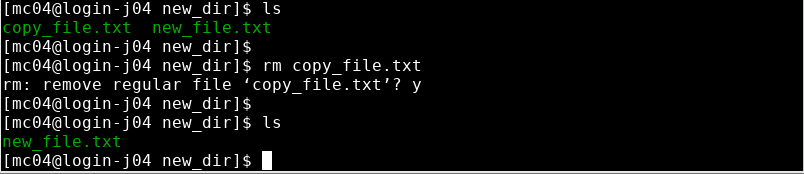
**cp**: copy file or directory



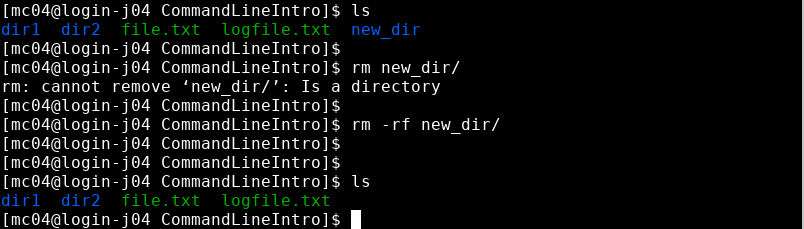
**mv**: move file or directory



**rm**: remove file, directory, etc

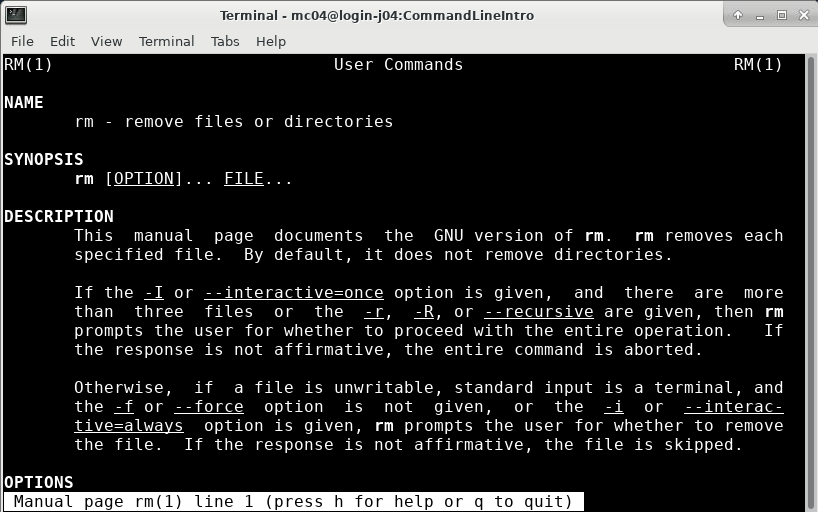


rm folder gives an error so we need to use flags –r –f (or –rf): the flag r will recursively delete the entire directory, subdirectories and all files they contain; the flag f will execute the command without prompt.

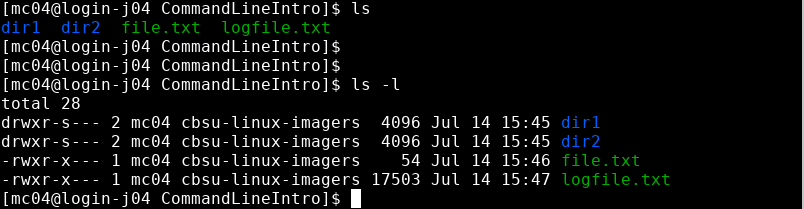


**man**: look at command documentation.





Another example of using a command with a flag:



**echo:** print text or string data

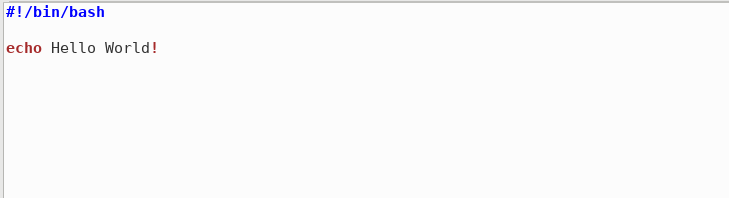


**Bash scripting**

**Getting started**

Start by creating a new scrip file using a text editor of your choice (e.g., vim, gedit).





The text **#!/bin/bash** at the start of the file is simply providing the path to the shell interpreter (bash in this case) to ensure the correct interpreter will be used.

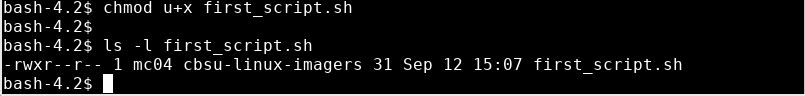
You can display the contents of your scrip file:



However, we don’t yet have permission to execute it as in linux we need to give files permission to act as an executable file.



In order to change this we can use the command chmod u+x first\_script.sh which will give the owner user permission to execute the file:

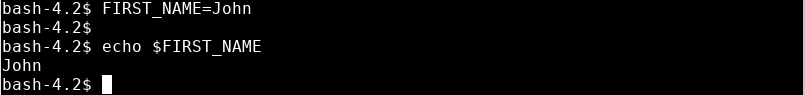


Now we can run our script by typing **./first\_script.sh:**

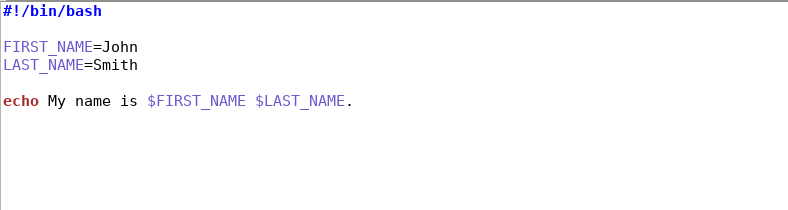


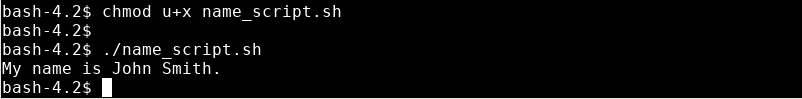
**Variables**

We can define variable to contain a string of text, for example:

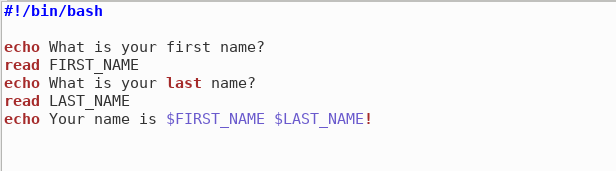


And we can include variables in our script, for example:

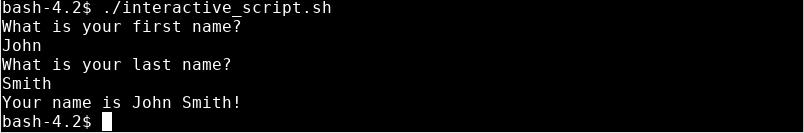




We can also create an interactive script that ask the user for their name:



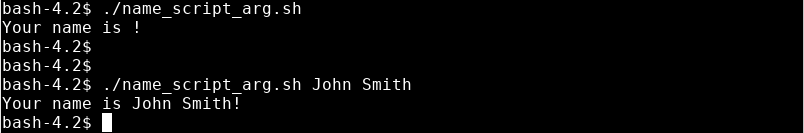
And when you execute this script the **read** command will wait until the user enters an answer:



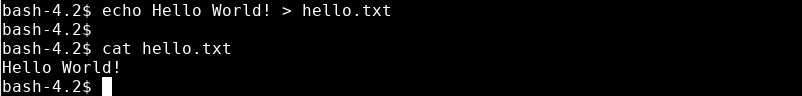
**Handling inputs/outputs**

The variables can also be passed into the script as positional arguments:

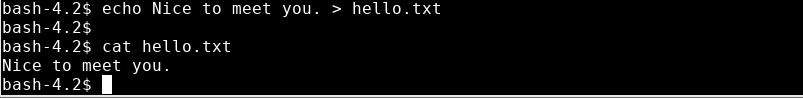




The output of a command can be stored in a text file by using the syntax > and **>>**. For example:



But if you use the syntax > again you will find that your original file is overwritten (which may or may not be the desired effect):



If you want to append to the file rather than overwriting it, you can use the syntax >>:



Some commands can also take files as input using the < syntax. For example the **wc** (word count) command works as follows:



**For loops**

**For** loops are one of the most useful tools in bash scripting, and especially so for neuroimaging data analysis, where we will often repeat the same pre-processing and model fitting steps for each of the subjects in our datasets.

We can start off by defining a **list**:



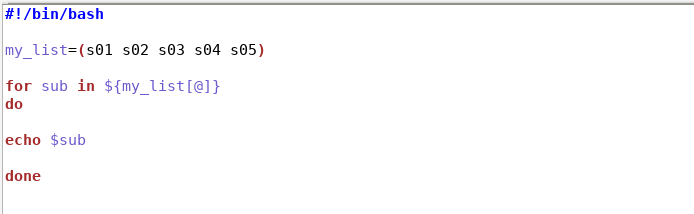
We can now inspect the contents of that list using **echo:**

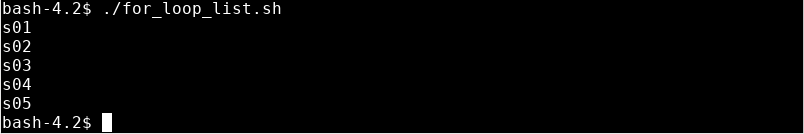


If you want to see all entries in your list you need to use the following syntax:

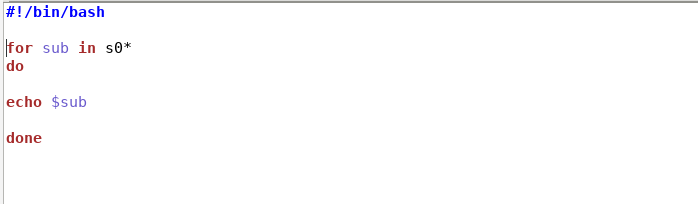


You can now use a **for** loop to execute the same steps or commands for each of the entries of your list. We can do that in a new script:





Alternatively, if you already have a folder containing all the files you want to loop through (e.g. s01, s02, s03, etc), you can also call each file within your **for** loop as follows:



The asterisk (\*) can represent any number of characters, so this option will work as long as there is a common part to the name of all files/folders you want to loop through.