Using Mammouth

Maxime Charlebois & Simon Verret
Sherbrooke
September 2017

Abstract

This is intended to help someone who wants to connect to the supercomputer mammouth. It shows how to put jobs in the execution queue, how to follow the progression for those jobs and how to compile and run simple programs.

Contents

1 Linux command line

Before reading this, one should be familiar with Linux terminal (Bash language, same as Mac terminal or Unix terminal). The reader is expected to know the basic commands before reading this document. Here is a list of the most important commands that one should know before reading this document:

```
$ ls                      # list the file in the current directory
$ cd                      # change directory
$ pwd                     # show the current directory
$ mkdir                   # make a new directory
$ cp                      # copy files or directories
$ mv                      # move files or directories
$ rm                      # remove (delete) files or directories
$ sudo                    # do a command in superuser
$ cat                     # show the content of a file
$ man                     # show the manual of a command
$ grep                    # find a text or pattern in one or multiple file
```

In this document, we use "`$`" sign to indicate the terminal prompt and the "`#`" indicates a comment in the bash language. If you need help to get familiar with Linux (Unix), I suggest the first 6 tutorials of this website: `http://www.ee.surrey.ac.uk/Teaching/Unix/index.html`.

Another essential element is to learn how to edit a text file with a terminal text editor like `nano` (basic and simple) or `vi` (complicated and powerful). This will become important when you need to edit a file remotely (on another computer, like mammouth) from your terminal. With the basic setup, these editors won't allow for mouse inputs. You must rely on multiple keyboard shortcuts to edit your file correctly. For the present tutorial, the reader only need to know how to open, edit and save a file.

Another important tip is the copy-paste in command-line, hence if `nano` and `vi`. On a PC keyboard, the Ctrl-c and Ctrl-v probably won't work. You must then use Ctrl-shift-c and Ctrl-shift-v if that is the case. This is useful if, for example, you want to copy the text from this ".pdf" file to the terminal. On Mac, the shortcuts Cmd-c and Cmd-v should remain the same.

1.1 Connect to Mammouth

Once you are familiar with Linux, you can connect to mammouth, the super calculator at Sherbrooke University. Of course, you will need to get a username and a password with the help of your supervisor. Let's suppose here that your supervisor is Andre-Marie Tremblay and that the username you got is `charleb1` . Now go into a terminal and type:

```
local$ ssh -X charleb1@tremblay-ms.ccs.usherbrooke.ca
```

Then enter your password, and that's it, you are connected! The name `local` is the name of the terminal that run on your (local) computer. A different name will be used when we are logged on a remote computer like mammouth. Notice that you are connected to the MS, standing for *Mammouth Serie*. There is also MP (*Mammouth Parallel*), a different cluster with more computing cores, to which you can connect by replacing `tremblay-ms` by `tremblay-mp2` in the command.

1.2 Moving files

Here are some commands that can be useful to move files between your computer and mammouth. You must run them on a local terminal and not on a terminal logged onto mammouth. Indeed, you need to specify an internet address or a ip. Since you most likely don't know the ip address of your computer, it is always better to use these command on a local terminal.

To copy from mammouth to your computer:

```
local$ scp -r charleb1@tremblay-ms.ccs.usherbrooke.ca:~/path/file ./file
```

and from your computer to mammouth

```
local$ scp -r ./file charleb1@tremblay-ms.ccs.usherbrooke.ca:∼/path/file
```

the "-r " (recursively) is to copy folders (like with the `cp` command). Your password will be asked like for `ssh`.

2 Launch jobs on Mammouth

Most of the information in this section were taken from: http://wiki.calculquebec.ca/w/Running_jobs#tab=tab8. Once you are logged into mammouth, you can already execute some commands and codes. But, you are on the interactive node, shared with other users of the department (you can check who with the command `who` and even write to them with the command `write`). Since the node is interactive, you can run your code, interact with it and debug, but it should not be used for more than that. If you want to launch a code and wait for the result, you need to do a job submission. A rule of thumb is: if code that runs for more than 20 minutes or requires more than half of the node's processors (cores), it should be launched with a job submission.

2.1 Job submissions

There are two ways to submit jobs on mammouth, here I present the basic way with the command `qsub`. There is also a fancier and more powerful way, with the `bqsubmit` command (see section 2.3). When submitting jobs, one first need to ask for the necessary amount of cores and memory. This is done with a system called *Torque* (identified by acronym PBS) which manage the priority of your jobs, given a set of mysterious rules. The idea is to write a file containing the commands to execute as a script, preceded by a PBS header. Let's name this file `sub.pbs`. Here, I want to execute `echo "hello world"` using 2 cores on mammouth (somewhat overkill) and record the terminal output in a file name `ecran_out` (otherwise lost):

Submission file `sub.pbs`:

```
#!/bin/bash
#PBS -N nameToTrackYourJob
#PBS -q qwork@ms
#PBS -l nodes=1:ppn=2,walltime=00:05:00
cd $PBS_O_WORKDIR
echo "hello world" >> output.dat
```

The first line is the usual bash script header. The second line, the first PBS line, determines the name of the job, because keeping good track of the jobs you submit is a really good habit to take. The second PBS line determine the queue in which you want to submit your job. Apart from `qwork@ms` (allows max 120h), there is also `qtest@ms` (max ∼10h) and `qlong@ms` (max 1000h). If you want to launch job on MP, the most common queue is `qwork@mp2`. It is important to stress that you must be logged on MS when you `qsub` a "pbs" file with `qwork@ms` and on MP when you `qsub` a "pbs" file with `qwork@mp2`. The third PBS line specifies the requirements of one job. For example, here we ask for 1 node, 2 cores (ppn for "processor per node"), and an expected maximum time of 00:05 hours (5 minutes). The idea behind this is that our allowed total time per core is counted and determines our priorities in the queues. We don't want to use too much.

The last part is the executed script. The command `cd $PBS_O_WORKDIR` set the "current directory" at the directory from where the submission was called (with the environment variable `$PBS_O_WORKDIR`) when running the script. This is necessary if you want the output files your script to end there.

Finally, one can submit this once and for all:

```
ms$ qsub sub.pbs
```

The job gets a number which will be displayed and then is waiting in the queue.

On MS, there are 8 cores per node and 16 Go of memory. A single node is always reserved to a single user. A good habit is therefore to fill a node completely as often as possible, so that you don't waste the unused core time. So, one should send 8 jobs with node=1 and ppn=1 at a time, or 4 with ppn=2, or any other combination that gives 8. For memory greedy programs, it can be advised to ask for more cores even if the program uses only one of them. That way you won't have memory conflicts.

2.2 Follow your jobs

Once the job is submitted, one can check its status with the `qstat` command. A plain `qstat` will yield a list of all the users. To get your specific jobs, use (-u for users):

```
ms$ qstat -nu charleb1

ms.m:
                                                                              Req'd    Req'd       Elap
Job ID               Username    Queue    Jobname     SessID  NDS   TSK    Memory   Time     S   Time
-------------------- ----------- -------- ----------- ------ ----- ------ ------ --------- - ---------
177107.ms.m          charleb1    qwork    nameToTrack   4059     1      2     --    00:05:00 R  00:00:02
   cs302/0+cs302/1
```

The status of the job will be marked as Q if it is waiting in the queue and as R if running. Once the job is running you can see on which node it is, given that you added the -n option, for "node". Here, `qstat` shows that the nodes `cs302` is used, with processors `0` and `1` . While the code is running, you can visit this particular node with the command `ssh`. Once on the node, you can, for example, check the CPU usage and the memory allocated with the command `top`:

```
ms$ ssh cs302
cs302$ top
```

Finally, if something is wrong, use `qdel` with the number of the job, to kill it:

```
ms$ qdel 177107.ms.m
```

2.3 Submit a lot of calculations

MP and MS offer another tool to submit a lot of jobs at the same time: BqTools. This offer the possibility to launch the same code with different parameters and correctly allocate every job and create every corresponding directory. This is the ideal way to scan multiple parameters, but requires a little more efforts. For now, we recommend the reader to consult `http://wiki.calculquebec.ca/w/BqTools/en`

2.4 Monitor the available nodes

One very useful command to see how many nodes are busy on the cluster is:

```
ms$ bqmon
```

Which will show you each available queue and the number of requests made on that queue.

3 Compile on mammouth

When you develop a code, you cannot compile it on your personal computer and copy the binary file on mammouth. It will most probably won't work, and if it works, it will most probably not be optimal. You must then learn to compile your code directly on the version of mammouth you want to run your code (MS or MP).

3.1 Your first code

A very basic source code can be obtained at the adresse:
www.physique.usherbrooke.ca/quantique-udes/afmconductivities.tar.gz. You can move it on mammouth (with `scp`) and uncompress it with:

```
ms$ tar -xzf afmconductivities.tar.gz
```

You can go then in the newly created directory `afmconductivities` and view the information in the README file. In this directory, the code should compile directly with the command:

```
ms$ make
```

This command will read the `makefile` and generate locally a executable binary file `afmCond`. To run this code, you can type the command:

```
ms$ ./afmCond
```

This will read the `model.dat` file present in the directory and compute various physical quantities (check the README for more information about the physics). With the present `model.dat` file, this should run under one minute.

3.2 The bin directory

The memory space on mammouth is expensive. It is a good practice to avoid to copy-pasting your executables in every directory you want to use them. A good practice is to use the `bin` directory.

In your home directory (the one you access by doing the `cd` alone, or `cd ~`), you should have a directory named `bin`. If it is not there, create it with `mkdir bin`. This is the directory where you want to move (`mv`) your binary files in order to have access to this command from anywhere on your computer. Avoid using binary names that corresponds to already existing terminal command, otherwise it could make this command (or your program) unreachable.

The `bin` directory is automatically added to the executable `PATH` on mammouth due to the lines

```
export PATH=$PATH:$HOME/bin
```

in `~\.bash_profile` file. You can create your own `bin` directory by adding a similar line in your `~\.bash_profile` file or `~\.bashrc` file.

3.3 The module command

In order to avoid that every user install the same library multiple times, some program on mammouth compiled and shared for all users. These are called "modules" and can be managed with the `module` command.

Let's download another source code that requires some modules at the adresse: www.physique.usherbrooke.ca/quantique-udes/onebody.tar.gz and "untar" it using the same procedure. If you go into the directory of `onebody` and type `make`, you will most likely receive an error message. This is because this code requires a more recent `gcc` compiler and some external libraries (Cuba and Lapack). On MS, the libraries you need to load are obtained with the command:

```
ms$ module add gcc/6.4.0 cuba/4.0 intel64/12.1.3.293
```

and on MP the libraries are:

```
mp2$ module add gcc/6.1.0 cuba/4.0 intel64/13.1.3.192
```

With these modules loaded, you can compile (`make`) and run (`oneBody_ms` or `oneBody_mp`) the code.

Here are some more commands you can use to play with the modules:

```
ms$ module list                # list the modules loaded
ms$ module avail               # modules available on the current computer
ms$ module add mymodules       # you already know
ms$ module rm mymodules        # remove a loaded module
ms$ module initadd mymodules   # add module to the default modules list
ms$ module initrm mymodules    # rm module to the default modules list
```

The last two lines are very important. Indeed, most of the time, you need modules not only for the compilation, but also for the execution of the binary. If you try to logout from mammouth and come back, the code you compiled won't run anymore unless you used `initadd` with the correct modules. These commands edit the file `~\.modules` (or `~\.modules_mp2` on MP). You can edit these files manually, but there is a good risk that these two commands won't work anymore. If that is the case, you can always delete them, and the system will create fresh new `.modules` files.

3.4 Plotting some results using `gnuplot`

It can be useful to rapidly plot some results obtained. As long as the result are in text format, it should be possible to plot using `gnuplot`. Both codes downloaded in this tutorial provide some plotting example in their README

To get a pretty Fermi surface, you can go to the directory `onebody` and run these commands:

```
ms$ ./oneBody_ms mdc
ms$ module add gnuplot/4.5
ms$ gnuplot
gnuplot> plot 'mdc.dat' matrix with image
gnuplot> q
```

The last line is to exit gnuplot. You can also plot a more traditional plot (x-y axis) with:

```
ms$ ./oneBody_ms l
ms$ gnuplot
gnuplot> plot 'n.out' u 1:2 w lp
```

`gnuplot` is much more powerful and configurable. But this gives a good starting point to visualize your data on mammouth.