# Guide to the Linux Terminal and Running Jobs using High Performance Computing Resources

Jennifer Garner

August 2017

# Contents

# 1 Jen's Guide to the Terminal

The linux terminal accepts bash commands for navigating file systems, moving and copying files between directories, editing files, accessing remote servers, and running programs (i.e. job submission). Some helpful terminal commands are: ls, pwd, cd, mkdir, cp, scp, mv, chmod, man, grep, awk, which, and ssh. These will be further explained. See Table 1 for a summary on these commands.

## 1.1 Navigating the Terminal

When the terminal is opened, the user will most likely be in the home directory. This home directory is usually */home/username* - this string of text is called a path. The current path, which can be determined using the command *pwd* or *present working directory*, is the user's location on the computer, starting with the root directory (/). The terminal is case-sensitive, and requires the use of forward-slashes (/) to denote the end of each directory name. Commands are given after the prompt, which is usually the dollar sign ($).

To determine what is in the home directory, the command is *ls*. This command has optional arguments, which are denoted by a hyphen (-). Some common arguments are: *-a* to display hidden files and directories, *-l* to show more information on the contents of a directory, and *-h* to give the file sizes in "human readable" format (i.e. using KB, MB, and GB designations as opposed to size in bytes). Hidden files and directories begin with a dot (.).

```
username@pcname:~$ pwd
/home/username
username@pcname:~$ ls
Documents  Pictures  Templates Downloads  Music
Public     Videos    Desktop
username@pcname:~$ ls -l
total 56
drwxr-xr-x  2 ownername groupname 4096 Apr 17 14:54 Desktop
drwxr-xr-x 11 ownername groupname 4096 Apr 18 15:12 Documents
drwxrwxr-x  2 ownername groupname 4096 Apr 18 14:54 Downloads
drwxr-xr-x  3 ownername groupname 4096 Aug 12  2016 Music
drwxr-xr-x  5 ownername groupname 4096 Apr 17 12:09 Pictures
drwxr-xr-x  2 ownername groupname 4096 Jul 25  2016 Public
drwxr-xr-x  2 ownername groupname 4096 Jul 25  2016 Templates
drwxr-xr-x  2 ownername groupname 4096 Sep 17  2016 Videos
```

The file system on linux can be thought of as a tree, where the root directory is the top of the tree and the subsequent directories are branches from that tree. To access the home directory from any location, use the command *cd $HOME*, where cd stands for *change directory*. Note that after changing the directory to Documents, the length of the prompt increases to include the new path. The tilde key (~) can also be used to reference the home directory; *~/Documents* is the same path as

*/home/username/Documents*.

To go up one directory, the command is *cd ../*. The current directory is usually denoted by *./*; therefore, *cd ./../* and *cd ../* are the same command. When the *cd* command is not preceded by a slash, the *./* is implied. If the *cd* command is used to try and access a file rather than a directory, the error "Not a directory" will appear. If the *cd* command is used to try and access a directory that is not available (located somewhere else on the computer), the error "No such file or directory" will appear. A common error with file navigation is to try a command such as *cd /Documents/*. This commmand will return the "No such file or directory error", because the initial forward slash (/) denotes the root directory, which does not contain a directory called Documents.

```
username@pcname:~$ cd Documents/
username@pcname:~/Documents$ pwd
/home/username/Documents
username@pcname:~/Documents$ cd ../
username@pcname:~$ cd Documents/
username@pcname:~/Documents$ cd $HOME
username@pcname:~$ pwd
/home/username
```

Some linux systems have colouring on the terminal to indicate the difference between a file (white/black, depending on terminal background colour), an executable file (one that can be run - green), a directory (blue), a linked file (cyan), a picture file (purple), and a compressed (zipped) file (red). If colouring is not enabled by default, the command *ls −−color=auto* can be used.

Depeneding on the specific operating system (OS), copying and pasting files from the terminal may require the use of Ctrl-Shift-C and Ctrl-Shift-V instead of the usual Ctrl-C and Ctrl-V.

## 1.2 File Permissions

Recall that executable files are different from regular text files. An executable file will have an 'x' category when the ls -l command is used. The position of the x depends on who can execute the file (or which class - see table below). The three permissions that a class can have are: read (r), write (w), and execute (x). If permission is not given for a class, the hyphen will be present (-).

| | Permissions by Class | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Directory (d) | all (a) | | | | | | | | |
| Not a directory (-) | user (u) | | | group (g) | | | others (o) | | |
| d | r | w | x | r | w | x | r | w | x |

For example, to change the regular file ex1.bash to an executable file, use the command *chmod u+x ex1.bash*. This command makes the file executable (x) for the

current user (u). To remove this same permission, the command *chmod u-x ex1.bash* can be used. The current user can be determined by using the command *echo $USER*. The file can also be made executable for other classes, such as the group class and the others class, by specifying g or o respectively.

```
username@pcname:~/examples$ ls -l
total 16
-rw-rw-r--  1 ownername groupname   33 Apr 21 17:12 exectuable.bash
-rw-rw-r--  1 ownername groupname    0 Apr 21 17:12 textfile.txt
drwxrwxr-x  2 ownername groupname 4096 Apr 24 10:19 example-directory/
username@pcname:~/examples$ chmod u+x exectuable.bash
username@pcname:~/examples$ ls -l
total 4
-rwxrw-r--  1 ownername groupname   33 Apr 21 17:12 exectuable.bash
-rw-rw-r--  1 ownername groupname    0 Apr 21 17:12 textfile.txt
drwxrwxr-x  2 ownername groupname 4096 Apr 24 10:19 example-directory/
```

Sudo is a group that have control over all files on a system.

## 1.3 Aliases

Sometimes a user will want to make "shortcuts" to their frequent commands. These shortcuts - called aliases - allow for short strings of text to be entered in place of longer commands. Aliases are stored in the user's home directory and are contained within hidden files. Using the command *ls -a* from the home directory will determine the default alias files available. In Ubuntu, aliases can be stored within the file *.bashrc*. However, it is recommended that the user make a new file, called *.bash_aliases*, for their personal commands. This file may already be present in the home directory.

To start, open the .bashrc file from the home directory. You can use any text editor to do this, such as gedit, nano, vim, emacs, etc. For example, to open the file using gedit, the command would be *gedit ~/.bashrc*. If not already present, add the following lines to the file:

```
1  # Alias definitions.
2  if [ -f ~/.bash_aliases ]; then
3      . ~/.bash_aliases
4  fi
```

Line 1 of these instructions is a comment (denoted by #), line 2 checks if there is a file called .bash_aliases in the home directory, line 3 executes the .bash_aliases file (only if present), and line 4 concludes the if statement.

Next, the user should make the file .bash_aliases in their home directory. To make a blank file, the command *touch* can be used. Alternatively, the user can simply type out a text editor and the name of the new file. The ampersand symbol (&) is used such that the terminal can accept commands after the given command is executed. Without an ending ampersand, the given command will lock the terminal (the prompt will not be available) until the application has completed its task or is closed. For example:

```
username@pcname:~$ touch ~/.bash_aliases
username@pcname:~$ gedit ~/.bash_aliases &
username@pcname:~$
```

Now that the .bash_aliases file is open for editing, the aliases can be added. The format is as follows. Make sure there aren't any spaces around the equals sign (=).

```
alias new='old command here'
```

A common alias used by default in Ubuntu is *ll*, which is short for *ls -l*. To add this alias here, add:

```
alias ll='ls -l'
```

Aliases are loaded when the terminal is first opened. Therefore, any new aliases will not be available in the current terminal session until the computer is instructed to use them. To update the aliases with the contents of .bash_aliases, use the *source* command.

```
username@pcname:~$ source ~/.bashrc
username@pcname:~$ . ~/.bashrc
```

Aliases can also be enabled by closing and reopening the terminal. Furthermore, a dot (.) is an alias for source, and so replacing source with a dot is an equivalent command.

## 1.4 Environment Variables

In programming languages such as bash, a variable is an identifier for a value. For example, we can set x=4, where x is the variable. We can also assign x to a string x=name, a directory x=/home/username/Documents, or an expression x=$((10+3)). Note that the double parentheses indicates an expression; setting x=10+3 will return 10+3, not 13. To call a variable that has been set (to get the value of the variable), bash requires that a dollar sign be used just prior to the name ($). For example, here is how to interact with a local variable within a terminal:

```
username@pcname:~$ x=yellow
username@pcname:~$ x
x: command not found
username@pcname:~$ $x
yellow: command not found
username@pcname:~$ echo $x
yellow
username@pcname:~$ export x=yellow
```

Here, x is equal to the string yellow. The first assignment makes x a local variable (usable within the current terminal only). By default, calling a variable will attempt to execute the variable. To determine the value of a variable without trying to execute it, the command *echo* is used. To make x an environment variable, the

command *export* is used. Note that *export $x* would not work here - the command must be explicit.

An environment variable is one that is available to the user and any programs that the user might run. For example, some programs may need to know what keyboard configuration is set, and so they will look for an environment variable at start-up. Global environment variables (available for all users) are set in the */etc/environment* file. User-specific environment variables are set in the *∼/.bashrc* file. Some common environment variables are HOME, USER, and PATH - these are set by default. It is possible to overwrite default environment variables, but this change will only apply for the current terminal session (unless changes are made within configuration files). To get a full list of all environemnt variables currently in use, the command is *printenv*. In the example above where x was set to yellow, this environment variable will only apply for that particular terminal (and processes invoked by that terminal). Once the terminal has been closed, then x will no longer be an environment variable.

## 1.5 Adding Executables to PATH

The PATH a set of directories, separated by colons (:), that contain executable files commonly used on a computer. For example, when using commands such as *ls* and *pwd*, the computer looks in the PATH for these executables. If a PATH variable was not set, each time *ls* was called from the terminal, it would also be necessary to specify the location of the executable (i.e. */bin/ls*). To find out where an executable is located on a computer, the command *which* can be used.

```
username@pcname:~$ echo $PATH
/home/username/bin:/home/username/.local/bin:/usr/local/sbin:
/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/
local/games:/snap/bin
username@pcname:~$ export PATH="$PATH:/home/username/scripts"
username@pcname:~$ echo $PATH
/home/username/bin:/home/username/.local/bin:/usr/local/sbin:
/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/
local/games:/snap/bin:/home/username/scripts
username@pcname:~$ which ls
/bin/ls
```

To add a directory to the path, the *export* command is used. Notice that, in the example above, export sets the variable PATH equal to the original PATH variable and appends the new directory name. When the PATH variable is updated in the terminal in this manner, the change is not permanent and only applies for that particular terminal (and processes invoked by that terminal)

To make the changes permanent (for example, when installing a new program), the PATH can be updated in the .bashrc or .profile file. For example, the VMD (Visual Molecular Dynamics) executable - a program for viewing and interacting with molecular dynamics simulations - is located in the /usr/local/bin directory by default.

This directory is already in the PATH, and so does not need to be added. However, if the executable was not located in the PATH - say it was added to /home/username/software instead - this directory would need to be added to the PATH. To add this program to the PATH permanently, open .profile from the home directory (*gedit /.profile*) using a text editor and add the following line:

```
export PATH="$PATH:/home/username/software"
```

Remember to refresh the file after this command has been added (*source ~/.profile*). Note that the name of the executable is not specified in the PATH; rather, the name of the directory containing the executable is given.

## 1.6 Interacting with Servers

### 1.6.1 Secure Shell (ssh) Connection

Accessing the Compute Canada HPC resources requires a secure shell (ssh) connection. For example, to access the system orca from Sharcnet from the local computer, use the following:

```
user@local:~$ ssh username@orca.sharcnet.ca
username@orca.sharcnet.ca's password:
[username@orc-login1 ~]$ ssh orc-dev1
[username@orc129 ~]$ logout
Connection to orc-dev1 closed.
[username@orc-login1 ~]$ logout
Connection to orca.sharcnet.ca closed.
user@local:~$
```

The prompt will usually change to include square brackets when the user sucessfully logs on to a server. Note that some servers, such as Orca, require another ssh login to access development nodes. Check the system documentation to see if this extra step is required. The command to disconnect from a ssh login is *logout*.

### 1.6.2 Secure Copy (scp) for Transferring Files

To copy files and/or directories between a remote server and the local computer, the *scp* (secure copy) command is used. This command is usually executed from the local computer (and not the server). That being said, the *scp* command can be used from a remote server to send files to or receive files from another remote server. The format of the command is: *scp <file/directory origin> <file/directory destination>*. When trying to send a directory and its contents, the argument *scp -r* (recursive) must be specified. For example, to send a directory called test from the home directory on the local computer to Orca, use the command:

```
user@local:~$ scp -r ~/test/ username@orca.sharcnet.ca:/home/username
username@system.ca's password:
test/file1.txt                100%   10KB    9.5MB/s    00:00
```

```
test/file2.txt                    100%   12KB   10.4MB/s    00:00
user@local:~$
```

As the files transfer, the % completed, the amount of the file transfered, the speed of transfer, and the total time elapsed will appear in the terminal window. Note that the *scp* command must specify the destination directory as well as the server.

To get files/directories from a server, first find out where those files/directories are located on the server. Then, logout from the server and navigate on your local computer to where those files should be sent. For example, to retrieve a file called data.csv from the Orca in the directory /work/username/analysis and send it to the home computer in the directory /home/username/from-server, use the following commands:

```
[username@orc129 analysis]$ ls
data.csv
[username@orc129 analysis]$ pwd
/work/username/analysis
[username@orc129 analysis]$ logout
Connection to orc-dev1 closed.
[username@orc-login2 ~]$ logout
Connection to orca.sharcnet.ca closed.
user@local:~$ cd from-server/
user@local:~/from-server$ scp username@orca.sharcnet.ca:/work/username/
    analysis/data.csv ./
username@orca.sharcnet.ca's password:
data.csv                                  100%    0       0.0KB/s    00:00
```

For submitting jobs, please refer to the server-specific guides in Sections 5 (Calcul Québec) and 6 (Sharcnet).

### 1.6.3 Using sshfs to edit remote files

As an alternative to editing files directly on the remote system through ssh, a remote directory can be mounted onto a home computer using *sshfs*. The following instructions were taken from the following video https://www.youtube.com/watch?v=VONgiHWc-gE, beginning at timestamp 4:33. First, install sshfs (assumes the user has sudo access):

```
user@local:~$ sudo apt install sshfs
```

Next, create a directory on the local computer. This will serve as a location to mount the remote directory. For example, the following instruction will create a directory called **mount-point-1** in the home directory:

```
user@local:~$ mkdir ~/mount-point-1
```

Then, determine the directory of the remote server that you would like to mount, and use *sshfs* to mount that directory on the empty directory that was created on the local computer.

```
user@local:~$ sshfs username@system.ca:/path/to-mount ~/mount-point-1
```

This command may require you to login to the remote system. Note that the command has a similar structure to the *scp* (secure copy) command, where you must first specify the name of the remote system, followed by a colon and the path to the directory. After this command has been executed, the user should be able to access the contents of the remote directory (to-mount) on the local computer. Editing these files on the local computer (with a text editor, for instance) will also update the files on the remote system.

When the user has finished editing or viewing files on the home computer, the directory should be unmounted as so:

```
user@local:~$ fusermount -u ~/mount-point-1
```

Where mount-point-1 should be replaced by the name of the directory on the home computer.

### 1.6.4 Modules

Sometimes jobs will require certain modules - these can be thought of as programs or supporting packages. To see the available modules currently loaded on a system, the command is *module list*. To see packages that can be loaded to a system, the command is *module avail*. Finally, to load a module, the command is *module load <modulename>*, and to unload a module, the command is *module unload <modulename>*. Another command is *module show <modulename>*, which provides information about a module. To remove all of the currently loaded modules, use the command *module purge*. Note that some packages may have dependencies (need other modules to be loaded before they can be run), or conflicts (two modules cannot be loaded at the same time). This information can usually be found in the Sharcnet Documentation or in Calcul Québec's Wiki pages.

```
[username@orc131 ~]$ module show vmd/notachyon/1.9.2
-------------------------------------------------------------------
/opt/sharcnet/modules/vmd/notachyon/1.9.2:

module-whatis   Provides vmd 1.9.2 (2014-12-29) LINUX_64 OpenGL, CUDA.
conflict    vmd
prepend-path    PATH /opt/sharcnet/vmd/1.9.2/notachyon/bin
-------------------------------------------------------------------
```

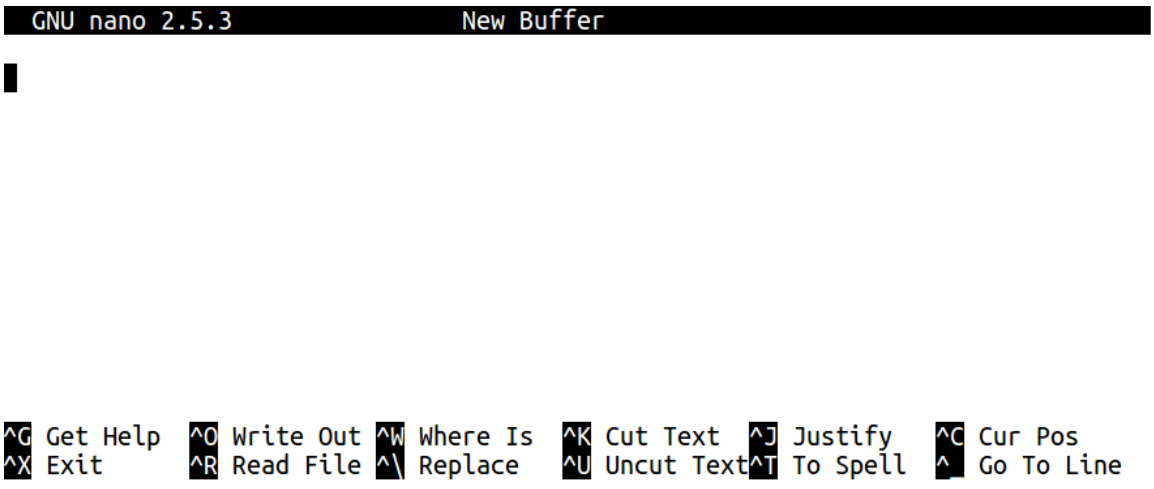
```
[username@orc131 ~]$ module list
Currently Loaded Modulefiles:
  1) torque/2.5.13                5) mkl/10.3.9
  2) moab/7.1.1                   6) sq-tm/2.5
  3) intel/12.1.3                 7) ldwrapper/1.1
  4) openmpi/intel/1.6.2          8) user-environment/2.0.1
```

## 1.7 Using the Nano Text Editor

Most text editors, such as notepad and gedit, include a graphical user interface (GUI). A GUI has buttons and is made for interaction via the mouse and keyboard. Others, such as nano, are based entirely around a command line interface (CLI). A terminal is an example of a CLI, where the commands are written in the scripting language bash. CLI-based programs are focused primarily on interaction via a keyboard, and do not contain buttons or support the use of a mouse (some enable the use of a scroll wheel). Connection to a remote server does not enable the user to run GUI programs, unless the server is intended for visualisation purposes (see Section 6.4). Therefore, nano is a necessary program to learn in order to edit files on the server.

The nano text editor can be configured using the file */etc/nanorc*. To enable options, remove the comment symbol (hash/pound #). The nano text editor is called from the terminal using the command *nano <filename>*. If a filename is not specified, a blank file will open (Figure 1), and the user can name the file before it is closed. If the file is not in the current directory, the path can be specified to the file (i.e. *nano /home/username/filename*).

**Figure 1:** A blank nano file.

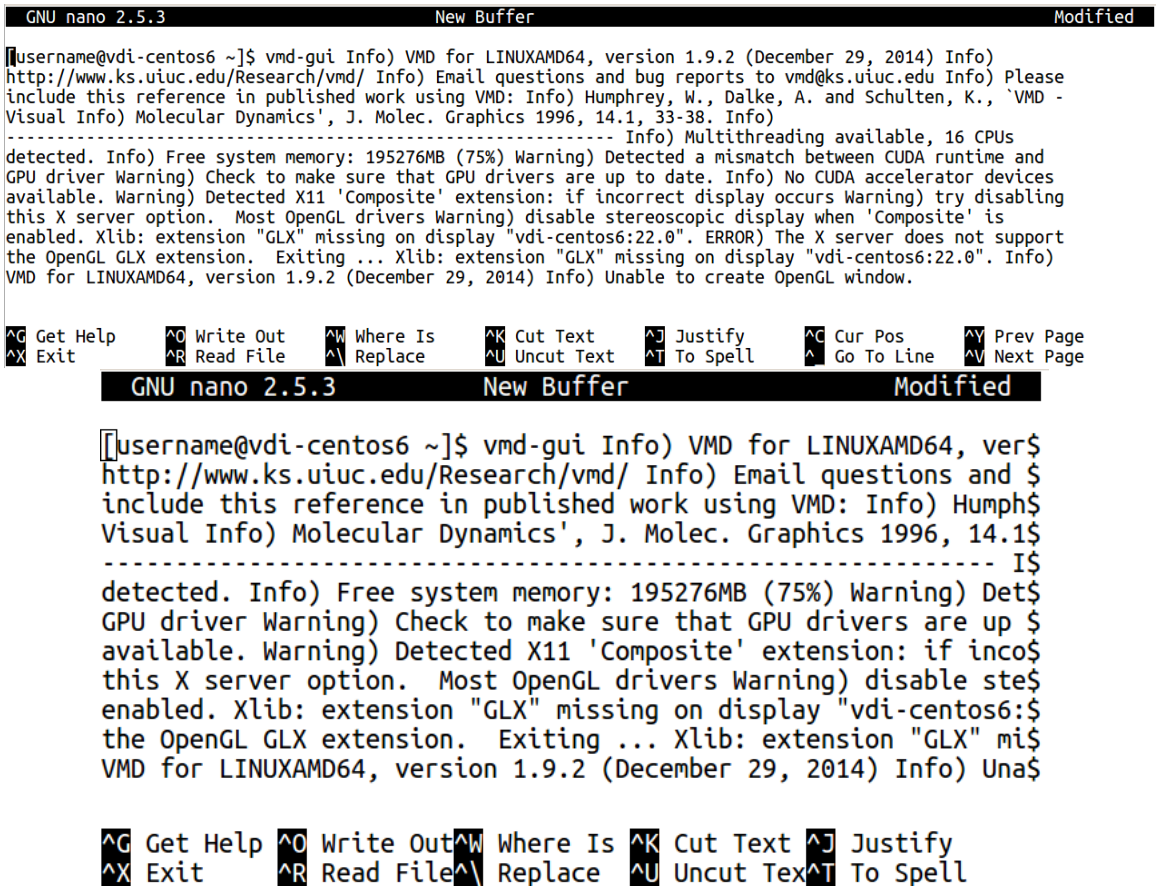



The options available in the nano text editor are listed along the bottom of the terminal window. The caret (∧) refers to the Ctrl button. The most important options are: **Ctrl-X** (close the program) and **Ctrl-O** (save file). Note these commands just need the Ctrl key and the letter (no shift key). **Ctrl-G** will display the help menu. Sometimes nano will prompt for user input (for example, what filename to save as). These prompts will appear just above the options list. For example, if a text file was edited, but not saved, nano will prompt the user to save the file upon closing. The file can be saved under a new name if needed.

Nano can take a bit of getting used to with regards to interactivity. Clicking on the text editor with the mouse does not move the cursor (blinking black object). Rather, the arrow keys or mouse wheel must be used to change the cursor's location.

Other options include **Ctrl-R** (Read File), which allows the user to open a file from the current directory (or peruse the file structure - **Ctrl-T** or To Files). **Ctrl-W** (Where Is) will search the text file that is currently open for a string of characters. If it finds the string, it will highlight the start of that string with the cursor. The **Ctrl-J** (Justify) option only really applies if the terminal window has been resized. An example of where **Ctrl-J** can be used is shown in Figure 2. To unjustify the text, use **Ctrl-U**. **Ctrl-C** (Cur Pos) shows where the cursor is currently in the file, including line number, column number, and character number. **Ctrl-K** and **Ctrl-U** can be used together to cut and paste (Uncut) text. **Ctrl-**\ can be used to find and replace strings of text. Nano will prompt yes or no for each matching instance that it finds.

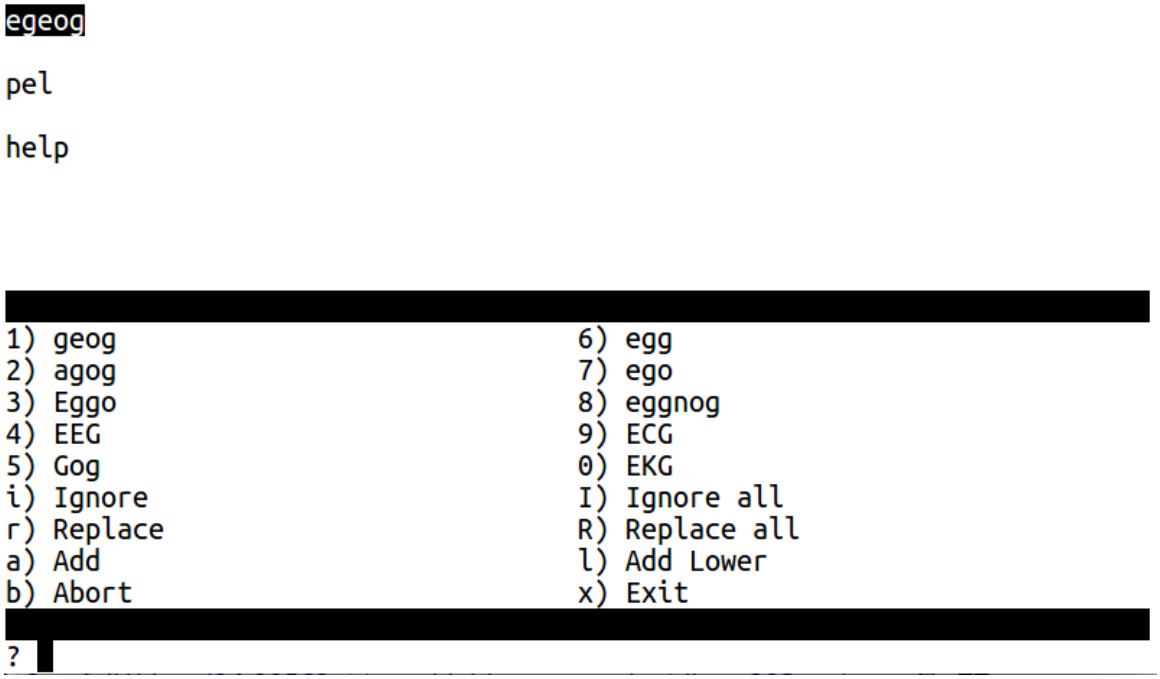
**Figure 2:** If the terminal is resized, some text may no longer show on one line. The justify option in nano should be used when the cursor is at the start of a paragraph (top left) to set the text to the same width as the terminal window. Lines that are too long for the window are indicated by a dollar sign ($). Note that justifying text does not create line breaks.




**Ctrl-T** starts the spell-checker in nano (Figure 3), but this feature may not work by default. There is a line in the configuration file that can be enabled to add a spell-checker (shown below). However, to edit the */etc/nanorc* file, the user needs sudo access. If sudo access is not available, the user can make a file in the home directory (∼*/.nanorc*) and add commands to this file instead.

```
## Use this spelling checker instead of the internal one. This option
## does not properly have a default value.
set speller "aspell -x -c"
```

**Figure 3:** An example of the spell-checker in nano, after enabling the set speller line in the */etc/nanorc* file.




## 1.8 Other File Manipulation Options

Sometimes, it is helpful to know what is in a file, but opening the file in nano is pointless. Instead, there is an option to use the command *cat*. The *cat* command is used for outputting the contents of a text file, or multiple text files (concatenation), to the terminal. For example, if there were two grocery lists, the cat command could show both:

```
username@oryx:~/Documents$ cat grocery1.txt grocery2.txt
Grocery List 1:
- beans
- cheese
- apple cider vinegar
- Smarties
- white bread
- BBQ sauce
- parchment paper
Grocery List 2:
- crayons
- butter
```

```
- chicken legs
- cheese
```

Another reason to use cat would be to make a new text file containing the contents of multiple text files. For example, to put both grocery lists in one file, use the right angle bracket (>, not hardware related) in the direction shown. The right angle bracket is the standard output.

```
username@oryx:~/Documents$ cat grocery1.txt grocery2.txt > groceries.txt
username@oryx:~/Documents$ cat groceries.txt
Grocery List 1:
- beans
- cheese
- apple cider vinegar
- Smarties
- white bread
- BBQ sauce
- parchment paper
Grocery List 2:
- crayons
- butter
- chicken legs
- cheese
```

Before adding these two lists together, it might be helpful to see if there are any duplicates (i.e. cheese). For examples that are longer and less trivial, this command can really help to achieve an objective quickly. The *diff* command will be helpful in comparing the contents two files. The argument *-y* is used to show the outputs of both files side-by-side.

```
username@oryx:~/Documents$ diff -y grocery1.txt grocery2.txt
Grocery List 1:                 |  - Grocery List 2:
- beans                         |  - crayons
                                >  - butter
                                >  - chicken legs
- cheese                           - cheese
- apple cider vinegar           <
- Smarties                      <
- white bread                   <
- BBQ sauce                     <
- parchment paper               <
```

Another helpful command is *tail*. This command outputs the last 10 lines of a text file to the terminal. It is helpful when analysing a really large file. In the case of MD simulations, the programs often write data to a log file. At the end of the log file, the completion status of the program is shown (did the job finish successfully?). Rather than using cat to output the entire contents of the log file, tail can be used instead. This is what a successful job completion looks like for a NAMD job on Orca (Sharcnet):

```
[username@orc131 namd]$ tail step7.1_production.log
```

```
WallClock: 42244.085938  CPUTime: 42244.085938  Memory: 934.199219 MB
[Partition 0][Node 0] End of program
--- SharcNET Job Epilogue ---
               job id: 5896362
          exit status: 0
         elapsed time: 11.7h / 13.0h (90 %)
       virtual memory: 6.3G / 1.8G (349 %)

Job completed successfully
```

And this is what a successful job completion looks like for a NAMD job on Guillimin (Calcul Québec):

```
[username@lg-1r17-n02 nsep20025]$ tail step5.1_production.log
FINISHED WRITING RESTART VELOCITIES
WRITING EXTENDED SYSTEM TO OUTPUT FILE AT STEP 500000
CLOSING EXTENDED SYSTEM TRAJECTORY FILE
WRITING COORDINATES TO OUTPUT FILE AT STEP 500000
CLOSING COORDINATE DCD FILE
WRITING VELOCITIES TO OUTPUT FILE AT STEP 500000
=======================================================

WallClock: 16467.189453  CPUTime: 16467.189453  Memory: 259.929688 MB
End of program
```

Similar to cat, there are other commands for showing the contents of text files - *less* and *more*. These two commands work in a similar fashion, and show the full output of a file to the terminal.

## 1.9 Compressed Files

Usually it is necessary to compress files (sometimes called adding to an archive) before they are sent to or received from a remote server. Compression reduces how much has to be sent over the network and therefore speeds up file transfer. Furthermore, if the transfer is interrupted before completion, the compressed archive can be deleted and resent more cleanly and easily. Compressed files are usually highlighted in red in the terminal, and can have extensions such as .tgz, .tar.gz, .zip, .gz, .7z, and .rar. Depending on the type of archive, a different command will be needed to interact with these files.

.tgz or .tar.gz can be easily worked with using linux machines. To create such an archive, use the following command: *tar -czvf name-of-archive.tar.gz inputfile1.txt inputdirectory*, where c specifies creation of an archive. To decompress this archive, use the command: *tar -xzvf name-of-archive.tar.gz*, where x specifies extraction of an archive. When creating or opening an archive, the -zvf arguments are: f for filename to work with, v for verbose (show files as they are being processed), and z means to work with a compressed archive. These commands retain the directory structure, and when archives are extracted, they will extract to the current directory.

# 2 Summary Table of Bash Commands

**Table 1:** A basic list of helpful terminal commands.

| Command | Description | Notes |
|---|---|---|
| pwd | present working directory | get current path |
| ls | list directory contents | -a (hidden files), -h (human readable), -l (more details) |
| cd | change directory | cd <full path> or cd <subdirectory> |
| cp | copy | use -r (recursive) for copying directories |
| scp | secure copy | over ssh connection; use -r (recursive) for copying directories |
| mv | move | cut and paste file/directory to new location; can also rename file |
| exit | close a terminal | |
| ssh | secure shell | sign-in to remote host (ex: ssh username@system.ca) |
| logout | disconnect from remote host | |
| man | manual entry | followed by name of command, example: *man ls*. can also try <command> −−help |
| chmod | change mode | change permissions for file, example: *chmod u+x file.txt* |
| chown | change owner | change who owns/created the file, may require sudo access. example: *chown user:user file.txt* |
| export | set environment variable | example: *export x=/home/user/Documents* |
| which | find location of executable | example: *which ls* |
| grep | search | example: *grep -i 'query' file.txt*; -i means case insensitive |
| awk | file manipulation | example: *awk '{print $2}' file.txt*; $2 refers to column 2 in file.txt |

| Command | Description | Notes |
|---|---|---|
| echo | print to terminal | example: *echo $PATH* |
| nano | CLI text editor | open a blank file, or use *nano /path/to/filename* |
| cat | concatenate files and print on the standard output | show output of text file, example: *cat file.txt* |
| tail | print last ten lines of file | example: *tail file.txt* |
| more \| less | scan through files | *more file.txt* or *less file.txt* |
| diff | compare two files | determines if there are differences between two files given. use -y argument to show files side-by-side, with differences denoted by \|. example: *diff -y file1.txt file2.txt* |
| clear | clear terminal | puts prompt at the top of the terminal window; can still access previous output by scrolling upwards |
| reset \| tput reset | clear terminal | prevents scrollback, refreshes terminal window while keeping set variables (local/environment) |
| module | interact with modules on HPC system | examples: *module list, module avail, module load <modulename>, module unload <modulename>, module show <modulename>.* |
| rm | remove | PERMANENTLY REMOVES FILES, or remove directories (permanently) with -r. do not use unless absolutely sure you want to remove something - rm does not send files to a recycle bin!!! |
| tar -czvf | create compressed archive | example: *tar -czvf archive-name.tar.gz directory-name* |
| tar -xzvf | extract compressed archive to current directory | example: *tar -xzvf archive-name* |
| sshfs | mount a remote directory on the local system | example: *sshfs user@remotehost.ca:/path/to-mount /~/local-dir* |
| fusermount | unmount remote directories | example: *fusermount -u /~/local-dir* |

# 3 Globus File Transfer

As an alternative to scp (secure copy) - see Section 1.6.2, Globus can be used to transfer files between servers and personal computers. The user will first need to register for a Globus account. Use the following link to their website: https://www.globusid.org/login.

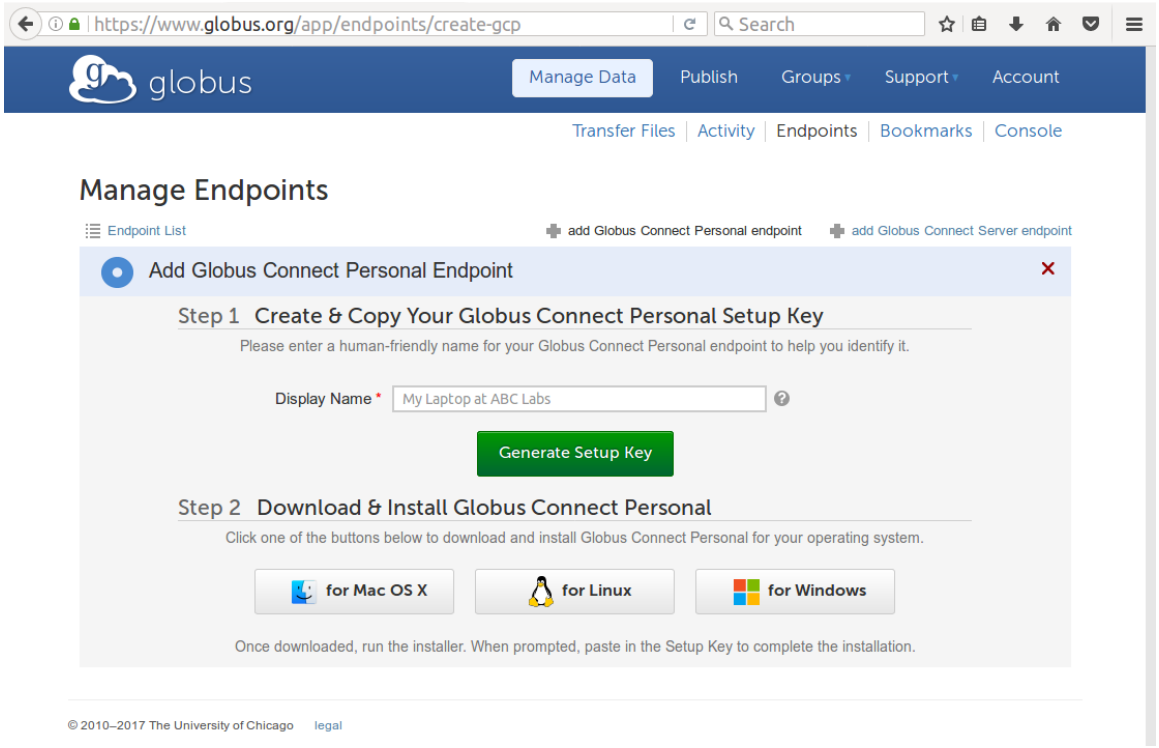
If you have yet to create a personal endpoint (and would like to transfer files between a home computer and a server), follow the instructions in Section 3.1. To transfer files between servers (or if you already have a personal endpoint), skip to Section 3.3.

## 3.1 Setting up Globus Connect Personal for the first time

These instructions are only for those who wish to transfer files between their personal computers and a server. These instructions need only be followed once to setup an endpoint.

First, login to Globus: https://www.globusid.org/login. While still logged in, the user should generate a Setup Key from the website: https://www.globus.org/app/endpoints/create-gcp.

**Figure 4:** To set-up a Globus endpoint on your personal computer, a name and Setup Key are required. The name should be an identifier for the computer to which the endpoint is connected.




The user should then download the Globus Connect Personal Package, either using the link shown in Figure 4 above, or by entering the following into a terminal:

```
user@localpc:~$ wget https://s3.amazonaws.com/connect.globusonline.org/
    linux/stable/globusconnectpersonal-latest.tgz
```

The file should be extracted into an easily accessible directory on the local computer.

```
user@localpc:~$ tar -xzvf globusconnectpersonal-latest.tgz
```

By extracting this archive, a directory should have appeared that looks similar to *globusconnectpersonal-x.y.z*, where x.y.z is the version number that was downloaded. Go into that directory:

```
user@localpc:~$ cd globusconnectpersonal-x.y.z
```

Then enter the setup key by running the globusconnectpersonal executable file. An example setup key is shown below. Replace this key with the one that was generated using your Globus account.

```
user@localpc:~/globusconnectpersonal-x.y.z$ ./globusconnectpersonal -
    setup 224532bb-8a4b-4d32-8995-e1fb442be98e
```

If everything goes smoothly, this should enable the globusconnectpersonal executable. The configuration files are saved in the home directory under / /.globusonline/lta.

This concludes the instructions for setting up Globus on a home computer. To transfer files using Globus, see Section 3.3.

## 3.2 Removing Globus Connect Personal

To remove Globus from a home computer, first kill any processes that Globus is running:

```
user@localpc:~$ killall gc-ctrl.py
```

Then remove the installation files from your home pc. You may get an error if these files have been moved or renamed. For example, if they were downloaded to your home directory, cd to $HOME and type:

```
user@localpc:~$ rm -r globusconnectpersonal-x.y.z globusconnectpersonal-
    latest.tgz
```

The last thing to do is remove the configuration files, which are stored in a hidden folder in your home directory:

```
user@localpc:~$ rm -r ~/.globusonline/
```

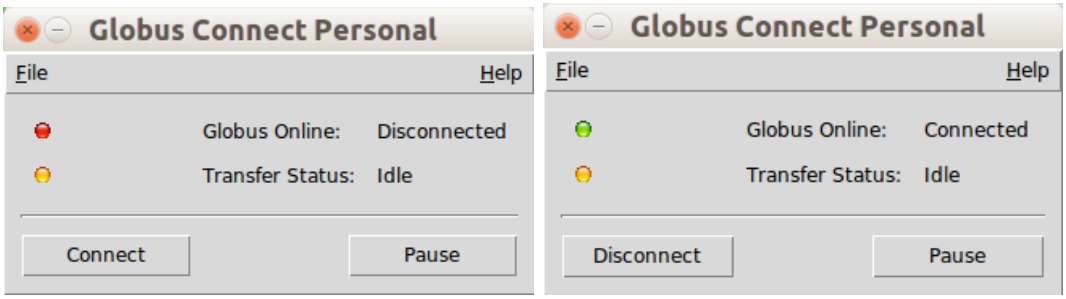
## 3.3 Using Globus to transfer files

If transferring files using Globus Connect Personal (i.e. between your home pc and a server), first activate the endpoint. Go to where your Globus files are stored on your home computer, and run the globusconnect executable. Note: the tailing ampersand enables the use of the terminal while the Globus GUI is running.

```
user@localpc:~$ cd globusconnectpersonal-x.y.z
user@localpc:~/globusconnectpersonal-x.y.z$ ./globusconnect &
```

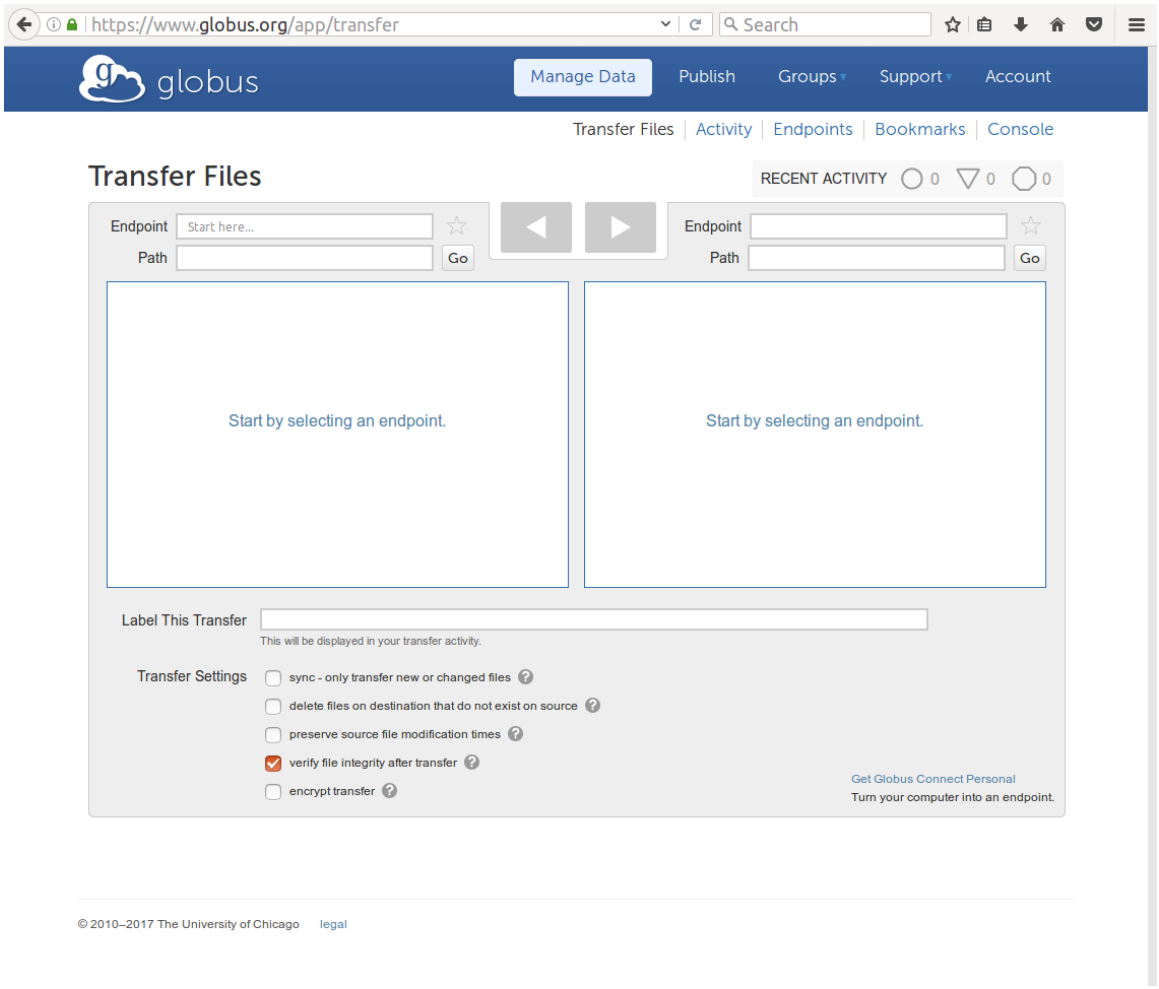
Then select *Connect*. The Globus Online status should be green with a note that says "Connected".

**Figure 5:** Before transferring files between a home computer and a server, the globus endpoint must be activated from the home computer.



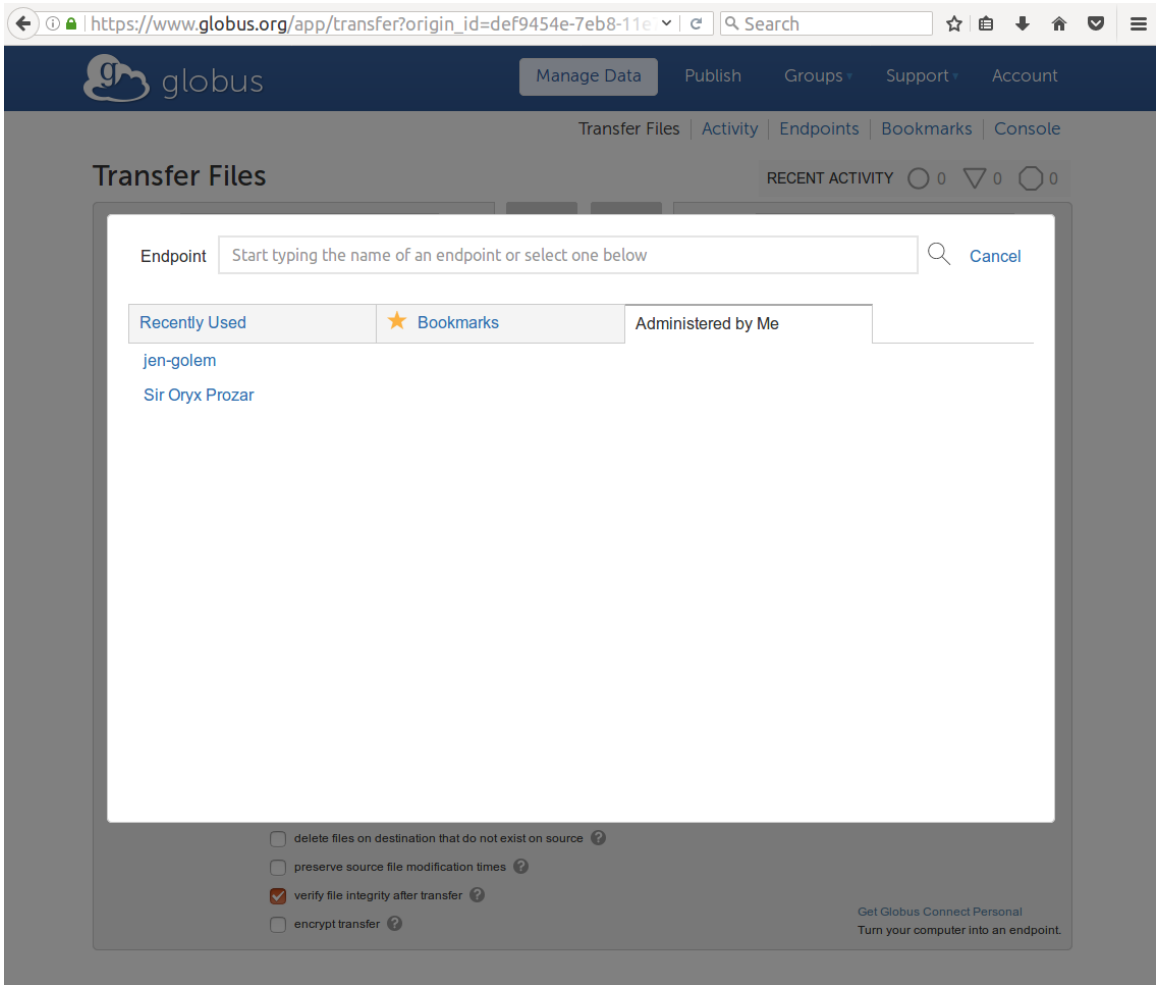
To transfer files, logon to the Globus website https://www.globusid.org/login and go to transfer files https://www.globus.org/app/transfer.

**Figure 6:** This is the main screen for transferring files using Globus.




Then, select the white box on either the left or right side that says *Endpoint*. If you want to select your home computer as an endpoint, select *Administered by Me* and choose the name that was chosen when Globus was set-up.

**Figure 7:** Any endpoints that have been set-up by your account on a home computer will be listed under the "Administered by Me" tab.




To select a server as an endpoint, enter the name of the server into the top white box shown in Figure 7 that says *Start typing the name of an endpoint or select one below*. Common server names are:

| Server Name | Endpoint Name |
|---|---|
| Graham (Compute Canada) | computecanada#graham-dtn |
| Orca (Sharcnet) | computecanada#sharcnet-dtn1 |
| Briaree (Calcul Québec) | computecanada#briaree |
| Guillimin (Calcul Québec) | computecanada#guillimin |

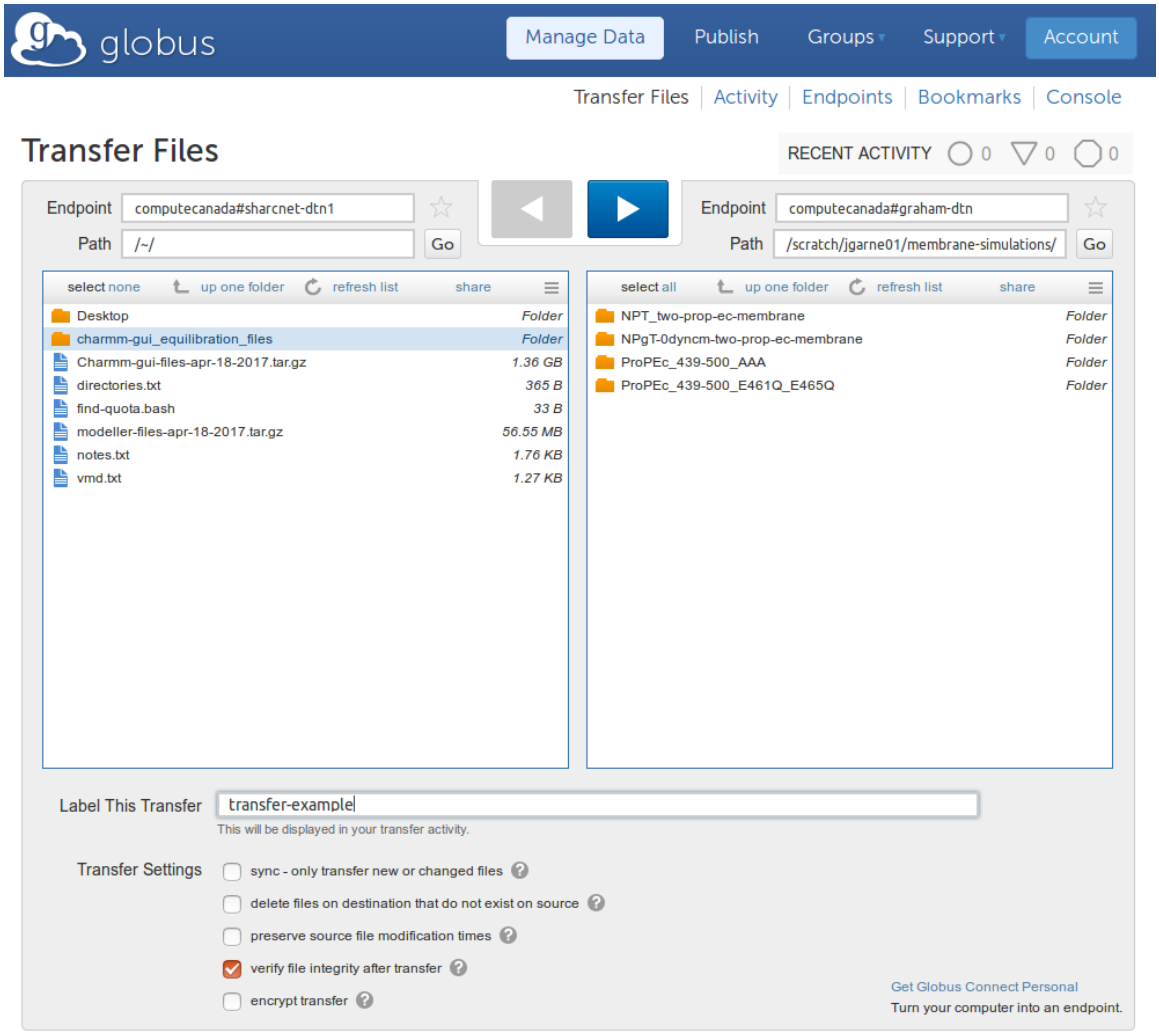
It should be noted that since most of the Sharcnet systems are linked under a common storage space, the endpoint would be the same for other Sharcnet systems

(for example: Monk) as for Orca. When you select these endpoints, you will be required to sign in using your corresponding credentials.

Another consideration is the directory from which you wish to transfer files. The default directory that is opened is the home directory (/∼/); however, most jobs are stored in the project or scratch directories. You will need to enter the corresponding directory into the white box labelled *Path* - see Figure 6. For example, to access the scratch directory in Graham, enter /scratch/user (where user is replaced by the real username) into the Path box.

Once two endpoints have been chosen (note, remote servers may require login credentials), use the arrow keys to transfer files (Figure 8). Optionally, you may give your transfer a name in the *Label This Transfer* box. To see past transfers, click the **RECENT ACTIVITY** link. Globus should email you after transfers have successfully completed.

**Figure 8:** Shows an example of a transfer of the charmm-gui_equilibration_files directory from the home directory on Sharcnet to the scratch directory on Graham. The name of the transfer is *transfer-example*. The transfer begins once the blue arrow is pressed.

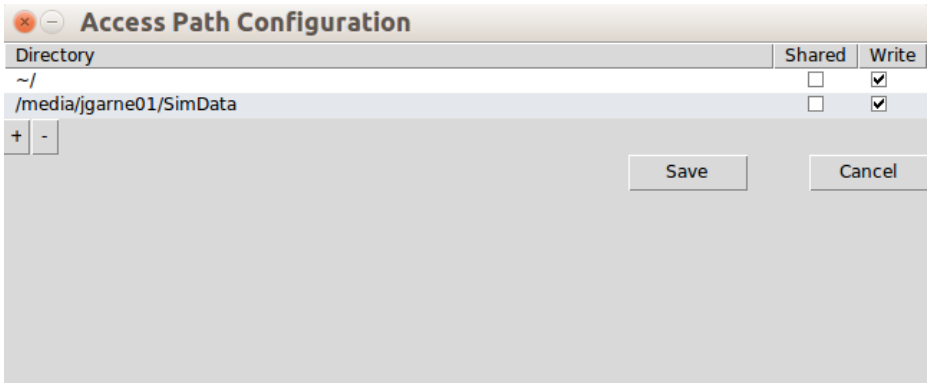



## 3.4 How to add a new path to Globus

By default, Globus Connect Personal only has access to your home directory (/∼/). If you have an external hard drive or USB device, these are not likely to be mounted in the home directory. Therefore, you must give Globus permission to access them. Usually, such external drives are stored in Linux machines under the /media/username directory. You can add this path (to enable access to all mountable media) or a specific path to Globus by opening the Globus Connect Personal GUI window (Figure 5)

and selecting File → Preferences. A new window should open called *Access Path Configuration*, as shown in Figure 9.

**Figure 9:** An example of an external hard drive whose path was added to the list of writable directories, as accessible from the Globus website.




Select the plus (+) button on the left, type the path to your files, and select **Save**. If you want to add files to this new path from elsewhere (i.e. from a server), make sure to make the directory writable by ticking the **Write** box beside the corresponding directory. You should now be able to access the contents of this directory from the Globus website.

# 4 Compute Canada - Graham

This section will be devoted to the new system called Graham. It uses a scheduler called slurm https://slurm.schedmd.com/. The help email for this system is: *support@computecanada.ca*.

## 4.1 Logging On and Off

To access Graham from a terminal, use the *ssh* command and your Compute Canada credentials (note these credentials are not the same as the Sharcnet or Calcul Québec login credentials). When you are finished using a system through ssh, the command is *logout*. This command will return you to the local system.

```
username@localpc:~$ ssh username@graham.computecanada.ca
username@graham.computecanada.ca's password:

Welcome to the ComputeCanada/SHARCNET cluster Graham.

Please refer to the following page:
https://docs.computecanada.ca/wiki/Graham
```

```
Email support@computecanada.ca for assistance.

***

[username@gra-login3 ~]$ logout
Connection to graham.computecanada.ca closed.
username@localpc:~$
```

Provided that users have a single project on Compute Canada, it is recommended that the ~/.bashrc file be altered (using **nano** or some other basic text editor). The addition (shown below) fulfills the requirement that all jobs specify the group name, without having to write in the account for every job submission.

```
export SLURM_ACCOUNT=def-someuser
export SBATCH_ACCOUNT=$SLURM_ACCOUNT
export SALLOC_ACCOUNT=$SLURM_ACCOUNT
```

Where someuser should be replaced by the group name. The group name can be found by logging onto the Compute Canada website and listing the project details. Alternatively, you can determine the group name by using the *groups* command while logged on to Graham.

## 4.2 Storage Space and Directory Structure

Graham has three main storage spaces for use: the scratch space, the home space, and the project space. The default allocations for these spaces are: 20TB per user, 50GB per user, and 1TB per group, respectively. These can be increased to maximums of 100TB per user for scratch and 10TB per group for project by request. There are also limits on the number of files that a user can have: 500,000 per user (home), 1,000,000 per user (scratch), and 5,000,000 per group (project). More information on these spaces is here: https://www.computecanada.ca/research-portal/accessing-resources/rapid-access-service/.

When a user first logs in to Graham, they are located in the home directory ($HOME, /~/, /home/username). From the home directory, the user can access scratch (/scratch/username OR /home/username/scratch) and project (/home/username/project/username). If a user is part of multiple projects, they can be accessed using the projects directory from the home directory (/home/username/projects/defgroupname/username).

It is usually convention to run jobs from the scratch directory, to save data being analysed in the project directory, and to keep small and important scripts and datafiles in the home directory. Files cannot be stored long-term in the scratch directory, because latent files will be removed after a certain time period. Jobs may receive performance boosts when they are run from the scratch directory (usually because the read/write capabilities are faster than other storage spaces). The home space is usually backed-up, whereas the other spaces are not. For groups that require more storage space, they can enter the Resource Allocation Competition (RAC).

## 4.3 Submitting Basic Jobs

To submit a job to Graham, the command *sbatch* is used. More information can be found on the Compute Canada help wiki: https://docs.computecanada.ca/wiki/Running_jobs.

Some basic arguments for the sbatch command include:

```
-t # time to complete the job
-J # job name
--mem=4G # (for example) for requesting memory (G=gigabytes,
    M=megabytes, K=kilobytes, etc., where the default unit is
    megabytes)
-N # number of nodes
-n # number of tasks (which can be thought of as number of
    CPUs)
--gres=gpu:2 # (for example) for generic resource, such as
    GPU
-o # output file name
-e # error output file name (by default the error file is
    included in the output file)
```

A full list of arguments can be found here: https://slurm.schedmd.com/sbatch.html. Note that most arguments have an explicit form (example --time=00:60:00) or a short form (example -t 00:60:00).

For example, to submit a python script called my-script.py to run on Graham for 20 minutes and writing to a file called output.txt, the command would be:

```
sbatch -t 00:20:00 -o output.txt python my-script.py
```

If the user has not specified the job name as an environment variable in ~/.bashrc, it will be necessary to add the account argument with every submission.

```
sbatch -t 00:20:00 -o output.txt --account=def-someuser
    python my-script.py
```

As with other servers, some jobs may require loading modules prior to submission. The same commands can be used on Graham as with other servers (i.e. *module (un)load*, *module list*, and *module avail* all work). For example, the default version of python on Graham is python2.7. To load another version of python, use:

```
[username@gra-login3 ~]$ module load python/3.5.2
[username@gra-login3 ~]$ module list

Currently Loaded Modules:
  1) nixpkgs/16.09 (S)    4) ifort/.2016.4.258 (H)   7) openmpi/2.1.1 (m)
  2) icc/.2016.4.258 (H)  5) intel/2016.4      (t)   8) StdEnv/2016.4 (S)
  3) gcccore/.5.4.0  (H)  6) imkl/11.3.4.258 (math)  9) python/3.5.2  (t)
```

## 4.4 Checking Job Status

To see what jobs are currently running, use the *squeue* command. You can also add the argument *-u $USER* to only see jobs for a given user. For example:

```
[username@gra-login2 ~]$ squeue -u $USER
   JOBID  USER  ACCOUNT  NAME  ST START_TIME TIME_LEFT NODES CPUS   GRES MIN_MEM NODELIST (REASON)
  433759  user  group    job1a PD N/A          2:00:00  1     32   gpu:2  2G            (Priority)
  433765  user  group    job1b PD N/A          2:00:00  1     32   gpu:2  2G          (Dependency)
  433766  user  group    jobA  PD N/A          0:30:00  1     32   gpu:1  1G           (Resources)
```

Here we have three jobs listed in the queue. The "REASON" column explains the state (ST) of the job (where PD is pending, R is running). Priority means that other users have higher priority than your group and are in line next to use resources. Dependency means that another job has to start somehow before its dependent job can run (i.e. job1a must finish before job1b can run). Resources means that the scheduler is waiting on resources in use before the job can start.

## 4.5 Cancelling a Job

Use *scancel jobid* to cancel a job, where jobid should be replaced by the number for the job you want to cancel. Similarly, to cancel all jobs for a user, use *scancel -u $USER*. To cancel all pending jobs, use *scancel -u $USER -t PENDING*. For more information on the scancel command, follow this link https://slurm.schedmd.com/scancel.html.

## 4.6 Submitting a job using a script

When a user has a job that is more complicated, usually they will write a script to submit that job. A script can be generated in a basic text editor on Graham. Scripts can also be written on the local system and subsequently uploaded to Graham. For information on how to transfer files between servers and personal computers, see Sections 1.6.2, 1.6.3, or 3.

An example of a submission script can be seen below. To submit this script to Graham, use the command: *sbatch script*, where script should be replaced with the actual script file name.

```
1  #!/bin/bash
2  #SBATCH --mem 2G
3  #SBATCH -n 32
4  #SBATCH -N 1
5  #SBATCH --gres=gpu:2
6  #SBATCH -t 05:00:00
7  #SBATCH -J production
8  #SBATCH -e error_%x_%j
9  #SBATCH -o step7.1_production.log
10 module load cuda/8.0.44
11 module load namd-multicore/2.12
12 namd2 +p32 +idlepoll step7.1_production.inp
```

The first line is indicating that the commands are in bash. The lines containing #SBATCH (lines 2-9) must preceed any other commands to be run by the server. These lines allocate resources. The minimum resources that must be specified are the time limit (−t or −−t=) and the group name (−−account=def-someuser). All other allocations are optional. Line 8, which specifies the error file, uses the percent sign to indicate job name and job id (%x and %j respectively). The actual error file would be similiar to: error_production_1203403.

## 4.7 Benchmarking and Running NAMD Jobs on Graham

NAMD is used for Molecular Dynamics (MD) jobs. Trajectory (dcd) files are generated. Combining these trajectory files with protein structure (psf) and protein database (pdb) structure files in VMD will generate a simulation that can be viewed and analysed. The Compute Canada webpage for NAMD is here: https://docs.computecanada.ca/wiki/NAMD.

NAMD can be run using a combination of CPUs and GPUs on the Graham system. To use GPUs, modules must be compiled with CUDA support. The CUDA module must also be loaded. In utilising GPUs and one node, load the following modules:

```
module load cuda/8.0.44
module load namd-multicore/2.12
```

In the executable line for a NAMD simulation, make sure to specify the number of CPU cores being utilised (+p32 for 32 CPU cores designation):

```
namd2 +p32 +idlepoll step7.1_production.inp
```

The actual name of the executable is namd2, and the file containing the instructions is called step7.1_production.inp. If you only want to run a simulation using CPUs (no GPUs), do not load the CUDA module. If you do not specify a number of GPU resources and try to load the CUDA module, the program will crash.

When running a NAMD simulation, you can request multiple nodes to complete the job. To use multiple nodes, this requires a different module than the standard NAMD multicore version. For Graham, the verbs module is more effient than the MPI (message passing interface) module. However, the scripts for these are non-trivial, and so the user should consult the Compute Canada NAMD wiki page for instructions on how to use NAMD on multiple nodes.

When running a NAMD job for the first time, it is a good idea to run some benchmarking tests. NAMD jobs perform better if they use full nodes. On Graham, a full node has 32 CPUs ("tasks"). NAMD jobs also fail more often if using multiple nodes (from personal experience). So there is a trade-off between stability in using fewer nodes and shorter time-to-completion in using more nodes. Adding a GPU usually helps to shorten the time-to-completion of a job.

To run a benchmark, submit multiple instances of the same job to Graham, each with a short time limit (about 5 minutes). Each test should use a different

quantity of resources. The jobs will not complete, but the program output should provide some information as to an estimate of resources required. Benchmarking is also a good way to make sure the NAMD job is using the number of CPUs that were requested.

After the jobs have finished their allocated time, use *grep* to search for the keyword **Benchmark**. Remember to specify the name (and path, if applicable) of the text file to search. The *grep* command is, by default, case sensitive. Use the argument **-i** to make a search case insensitive. Examples are below.

The age of the computing resource is also important for time-to-completion. For example, running 1ns of a membrane simulation on the old system Saw would take more than 10 hours with 64CPUs, whereas running the same simulation on Graham would take less than 2 hours with 32CPUs and 2GPUs:

```
[user@gra-login4 namd]$ grep 'Benchmark' step7.1_production.log
Info: Benchmark time: 64 CPUs 0.0801173 s/step 0.463642 days/ns 917.887 MB memory
Info: Benchmark time: 64 CPUs 0.0803184 s/step 0.464806 days/ns 925.812 MB memory
Info: Benchmark time: 64 CPUs 0.0804023 s/step 0.465291 days/ns 925.812 MB memory
Info: Benchmark time: 64 CPUs 0.0814741 s/step 0.471493 days/ns 925.812 MB memory
Info: Benchmark time: 64 CPUs 0.0832847 s/step 0.481972 days/ns 925.812 MB memory
Info: Benchmark time: 64 CPUs 0.0821933 s/step 0.475656 days/ns 925.812 MB memory
[user@gra-login4 namd]$ grep -i 'benchmark' step7.77_production.log
Info: Benchmark time: 32 CPUs 0.0109838 s/step 0.0635635 days/ns 1114.56 MB memory
Info: Benchmark time: 32 CPUs 0.0110412 s/step 0.0638959 days/ns 1121.52 MB memory
Info: Benchmark time: 32 CPUs 0.0111505 s/step 0.0645286 days/ns 1126.04 MB memory
Info: Benchmark time: 32 CPUs 0.0110386 s/step 0.0638808 days/ns 1127.25 MB memory
Info: Benchmark time: 32 CPUs 0.0110486 s/step 0.0639384 days/ns 1127.71 MB memory
Info: Benchmark time: 32 CPUs 0.0110241 s/step 0.063797 days/ns 1128.59 MB memory
```

The output of the benchmarking search should show how long it takes to complete 1ns of simulation. The days/ns figure can be multiplied by 24hours/day to get an estimate in hours (which is usually more useful). The last column is the amount of RAM needed (specified in the script by −−mem) in MB. Recall: there are approximately 1024MB in one GB.

Note that some steps (i.e. equilibration steps) are smaller than 1ns long, and so you should consult the input files (inp) for the actual running time. Use *grep* to search for "steps =" and "run" in the input file. The 'run' value is how many steps are run, and the comments should indicate the timescale for each step. In the example below, this equilibration file would run 25,000 steps, where 500 steps = 1ps, meaning that the equilibration step is 50ps long (0.05ns).

```
[user@gra-login2 namd]$ grep 'steps =' step6.1_equilibration.inp
restartfreq          500;                # 500 steps = every 1ps
outputEnergies       125;                # 125 steps = every 0.25ps
[user@gra-login2 namd]$ grep 'run' step6.1_equilibration.inp
outputName           step6.1_equilibration; # base name for output from
    this run
firsttimestep          0;                # last step of previous run
run 25000
```

To ensure a job has completed successfully (that is, after the benchmarking has been done), check the end of the log file using the *tail* command. On Graham, it

should appear similar to this:

```
[user@gra-login2 namd]$ tail step7.77_production.log
FINISHED WRITING RESTART VELOCITIES
WRITING EXTENDED SYSTEM TO OUTPUT FILE AT STEP 500000
CLOSING EXTENDED SYSTEM TRAJECTORY FILE
WRITING COORDINATES TO OUTPUT FILE AT STEP 500000
CLOSING COORDINATE DCD FILE step7.77_production.dcd
WRITING VELOCITIES TO OUTPUT FILE AT STEP 500000
=============================================================

WallClock: 5575.174316  CPUTime: 5560.857422  Memory: 1184.992188 MB
[Partition 0][Node 0] End of program
```

# 5 Calcul Québec Systems

Systems currently in use:

- Briaree
- Guillimin

Another system on Calcul Québec is Helios - a GPU system. It may be used for future work if needed. Information with regards to submitting jobs on Calcul Québec systems can be found here: https://wiki.calculquebec.ca/w/Ex%C3%A9cuter_une_t%C3%A2che/en#tab=tab4.

## 5.1 Briarée

### 5.1.1 Briarée Resources

Briarée is a CPU system on Calcul Québec. It has a total of 672 nodes, with 12 cores per node. The RAM available depends on the node; nodes can have 24, 48, or 96GB of RAM, or 2, 4, or 8GB per core respectively.

Jobs on Briarée can spend a maximum of 7 days running (not including time spent in the queue). There is a default limit of 4 nodes per job, meaning that the maximum number of cores that can be requested is 48. To get access to more nodes, users must provide scaling figures (i.e. benchmarking numbers) to the support staff at Calcul Québec. These figures must display that the code will benefit from more resources (scales well) when using the appropriate software. The support staff for Briarée can be emailed at: briaree@calculquebec.ca.

The storage space on Briarée is divided into two main sections: home (located at /RQusagers/username) and scratch (located at /RQexec/username). The home space is backed up, and the scratch space is not backed up. These spaces do not have a specified quota, but the limits should be similar to those for Guillimin (10GB home, 1TB scratch). The scratch space is not cleared for Briarée as it is for Guillimin, but there is no designated project space for Briarée. The storage space used by both Briarée and Guillimin is called GPFS, which is a high-performance storage space that allows for good throughput for parallel jobs.

Module information for Briarée can be found at this website: https://wiki.calculquebec.ca/w/Modules_disponibles_sur_Briar%C3%A9e?setlang=en

### 5.1.2 MWE for Briarée Submission Script

To run this script, open a terminal and type: *bash scriptname.bash*.

```
1 #!/bin/bash
2 #PBS -r n
3 #PBS -A nmf-334-aa
4 #PBS -N job-name
```

```
5  #PBS -l mem=2GB
6  #PBS -l nodes=2:ppn=12
7  #PBS -l walltime=01:00:00
8  #PBS -j oe
9  module purge
10 module load NAMD/2.9
11 module load python/2.7.1
12 cd $PBS_O_WORKDIR
13 mpiexec -n 24 namd2 input-file.inp > output-file.log
```

By line number, here is an explanation for this script:

1. execute this script using bash commands
2. job should not be automatically restarted if there are issues
3. RAP ID for the project
4. job name (should be short)
5. how much RAM to allocate for the job
6. how many nodes and processors per node (ppn) to allocate for the job
7. how long the job will take to complete (set for 1 hour)
8. join the output and error files to one file
9. remove all currently loaded modules
10. load the NAMD module
11. load the python module
12. change directory to the working directory
13. run the input-file.inp using the namd2 exectuable, with 24 cores for the job (2x12), on the mpiexec queue, and write the ouput to the output-file.log file

## 5.2 Guillimin

### 5.2.1 Guillimin Resources

Guillimin is a CPU system on Calcul Québec. It has a total of 1582 nodes, where 1200 of these nodes have 12 cores per node and the remaining 382 nodes have 16 cores per node. The RAM available depends on the node; 12-core nodes can have 24, 36, or 72GB of RAM, or 2, 3, or 6GB per core respectively. The 16-core nodes can have 64, 128, 256, 384, 512, or 1024GB of RAM, or 4, 8, 16, 24, 32, or 64GB per core respectively.

Jobs on Guillimin can spend a maximum of 30 days running (not including time spent in the queue). The support staff for Guillimin can be emailed at: guillimin@calculquebec.ca.

The file structure for Guillimin is unique compared to other Calcul Québec systems. The user has access to two main locations for their files: the home directory (located at /home/username) and the project directory (located at /gs/project/nmf-334-aa, where nmf-334-aa is the project or RAP ID). There is also the scratch directory (located at /gs/scratch/username) that can be used to store temporary files. Files stored on the scratch directory will be deleted if they are not updated after 45 days. The cleanup occurs on the 15th day of each month, with users being warned on the 8th day of each month if there are files that are going to be deleted. More information about file storage on Guillimin can be found on this website: http://www.hpc.mcgill.ca/index.php/starthere/81-doc-pages/87-disk-space.

To determine how much disk space is being used on Guillimin, the commands *myquota* and *prquota* can be used for the home/scratch and project spaces respectively. The home directory is backed up on a daily basis, and has a quota size of 10GB. This location is used for important files that are used regularly (such as submission scripts). The project space is not backed up, but can be used to store more files. It has a quota of 1TB across all project members. Lastly, the scratch space has a quota of 1TB per user, and it should be used to store temporary files and/or files that are being run.

Module information for Guillimin can be found at this website: https://wiki.calculquebec.ca/w/Modules_disponibles_sur_Guillimin?setlang=en

### 5.2.2 MWE for Guillimin Submission Script

```
1  #!/bin/bash
2  #PBS -A nmf-334-aa
3  #PBS -l walltime=10:00:00
4  #PBS -l nodes=3:ppn=12
5  #PBS -l pmem=7700m
6  module purge
7  module load openmpi/1.6.3-gcc
8  module load NAMD/2.9-openmpi-gcc
9  module load python/2.7.9
10 cd $PBS_O_WORKDIR
11 mpiexec -n 36 namd2 input-file.inp > output-file.log
```

Note that assigning RAM for Guillimin jobs is slightly different than for Briarée. Otherwise, these are the minimum number of instructions that should be given for a job submission. The -n argument specifies to use 36 cores for the job (3x12).

# 6 Sharcnet Systems

Systems currently in use:

- Monk
- Orca

Getting help for Sharcnet systems involves submitting a ticket, and there is a unified help email (unlike Calcul Québec, which has emails per system). Contact: help@sharcnet.ca. Sharcnet also has regular webinars that can be accessed via VidyoDesktop. These webinars are also uploaded to their youtube channel. The webinars are typically 1 hour long and run on Wednesdays at noon (12pm). They cover a variety of computing topics.

## 6.1 Sharcnet File Storage

For the most part, Sharcnet has a unified storage space across the systems mentioned in this document: Monk, Orca, and vdi-centos6. This means that any files generated by these systems will be accessible by the other systems. The details for the Sharcnet file storage can be found on this webpage: https://www.sharcnet.ca/my/systems/storage.

The home storage space (located at /home/username) has a quota of 10GB per user and is backed up. It should be used for important files such as source code, scripts, parameters, etc. The user cannot exeed the quota, which means that jobs will not be able to write data to the home space past 10GB.

The work directory (located at /work/username) has a quota of 1TB, but it is not backed-up. It should be used for large files. The scratch directory (located at /scratch/username) can be used for large file storage (no quota), but files must be deleted or moved from this space, as files will be removed after 62 days.

The freeezer directory (located at /freezer/username for orca login nodes) has a quota of 2TB; files stored here have a lifetime of 2 years. This space can only be accessed from login nodes (for example, you can only access the freezer directory from login not after ssh to a development node).

```
[username@orc-login1 ~]$ cd /freezer/username/
[username@orc-login1 username]$ ssh orc-dev2
[username@orc130 ~]$ cd /freezer/username
-bash: cd: /freezer/username: No such file or directory
```

The command *quota* will show the file usage for the work and home directories on Sharcnet. Alternatively, login to the Sharcnet web portal - https://www.sharcnet.ca/my/security/login - and scroll down to **Disk Usage**, which will show a plot of the files stored by age in the home and work directories. If the user exceeds their quota for either the work or freezer directories, the user has a 3-day grace period to

reduce their file count. After the grace period, if the user has not reduced their file count, he/she will no longer be able to submit jobs on Sharcnet. Quotas are updated at approximately midnight each day, but this update depends on the disk usage and the system.

## 6.2 Monk Resources

Monk is a GPU system on Sharcnet. It has a total of 54 nodes, with 8 cores, 2 GPUs, and 48GB of RAM per node. This system should only be used for jobs that can make use of GPU acceleration. It is tied-in to the file-system shared by Orca, Saw, and other Sharcnet systems.

## 6.3 Orca Resources

Orca is a CPU system on Sharcnet. It has a total of 392 nodes, split into 4 groups. Group 1 (nodes 1-320) has 24 cores and 32GB RAM per node. Group 2 (nodes 321-360) has 16 cores and 32GB RAM per node. Group 3 (nodes 361-388) has 16 cores and 64GB RAM per node. Lastly, group 4 (nodes 389-392) has 16 cores and 128GB RAM per node. Orca has 4 development nodes: orc-dev1, orc-dev2, orc-dev3, and orc-dev4. These development nodes should be accessed after logging in to the system (*ssh orc-dev1*). The login nodes are not supposed to support heavy CPU usage, and are designed as a stable access point to the system. Code can be run from these development nodes directly (without submitting to the queue) for testing purposes. These sorts of programs can be run for up to 3 CPU days (which is not equal to 3 days).

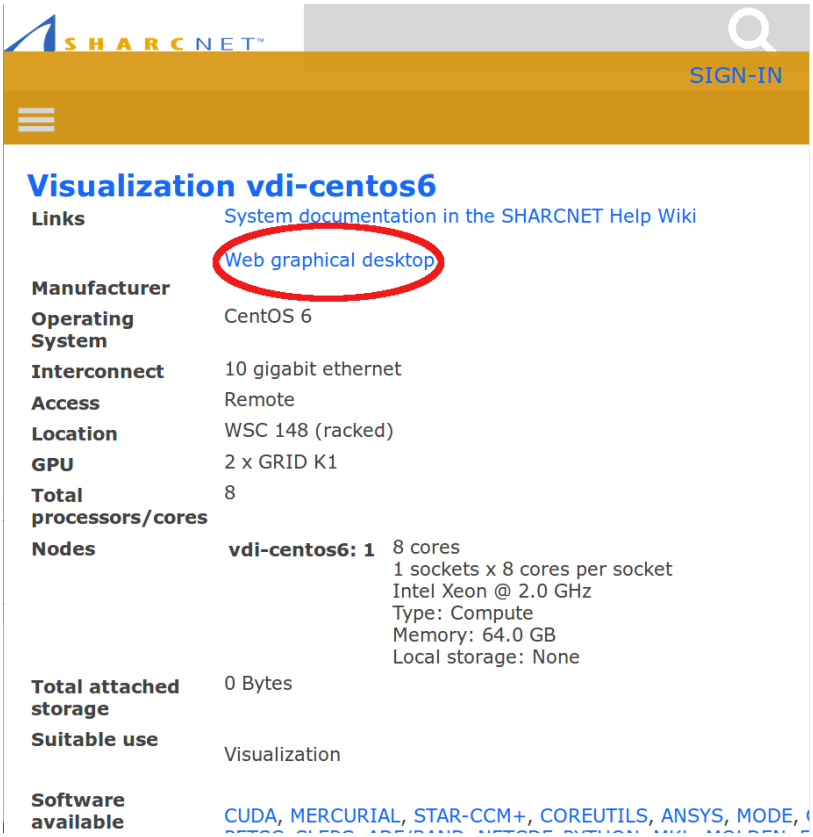
## 6.4 Sharcnet Visualisation Systems

Some systems on Sharcnet are used for visualisation purposes - that is, the user can use graphics programs and is not restricted to the command-line interface. The documentation for these systems can be found here: https://www.sharcnet.ca/help/index.php/Remote_Graphical_Connections. One such system is called centos6-vdi. The documentation for this system can be found here: https://www.sharcnet.ca/my/systems/show/104.

### 6.4.1 Accessing vdi-centos6

This system can be accessed from any computer using a web-browser. Open this url: https://www.sharcnet.ca/my/systems/show/104 and select the **Web graphical desktop** link - see Figure 10.

**Figure 10:** The vdi-centos6 system on Sharcnet can be accessed using a web-browser, such as Chrome or Firefox. The link is circled in the figure below.

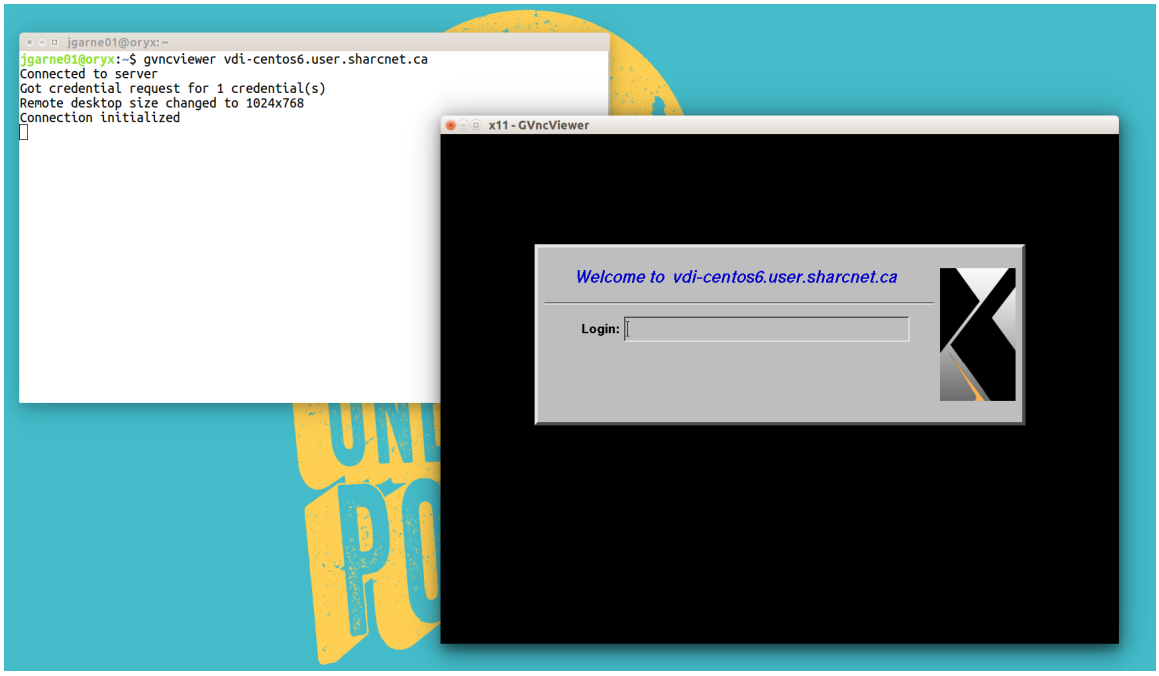



Another way to access vdi-centos6 is by using a VNC client. For Ubuntu, the recommended client is called gvncviewer. To install, configure, and use the gvncviewer program, follow these instructions:

```
1 apt-get install gvncviewer
2 mkdir -p ~/.pki/CA
3 ln -s /etc/ssl/certs/ca-certificates.crt ~/.pki/CA/cacert.pem
4 gvncviewer vdi-centos6.user.sharcnet.ca
```

The user only has to install (line 1) and configure (lines 2 and 3) the gvncviewer once. If these instructions have been used previously, the last command (line 4) is all that is needed. Note that *user* here does NOT mean username. See Figure 11 for an example of the VNC client.

**Figure 11:** The vdi-centos6 system on Sharcnet can be accessed on Ubuntu using a VNC client such as gvncviewer. When the command *gvncviewer vdi-centos6.user.sharcnet.ca* is used, the user will be prompted for their Sharcnet username and password.




To logout from vdi-centos6, simply close the web-browser or (if using gvncviewer) close the gvncviewer window. To close a terminal that has been opened on this system, the command is *exit* rather than *logout*. The *exit* command also applies for closing terminals open on the user's home computer.

### 6.4.2 Graphical VMD via vdi-centos6

This system is helpful for simulation work. It enbales the user to open, view, and analyse simulation data stored on the global filesystem of Sharcnet using VMD, without downloading all of the files to a home computer. To use the graphical version of VMD

**Figure 12:** The vdi-centos6 system can be used to open a graphical version of VMD. The user first needs to load the *vmd/tachyon/1.9.2* module. Note that the version number may change over time, and so use *module avail* to see the current version on the system.

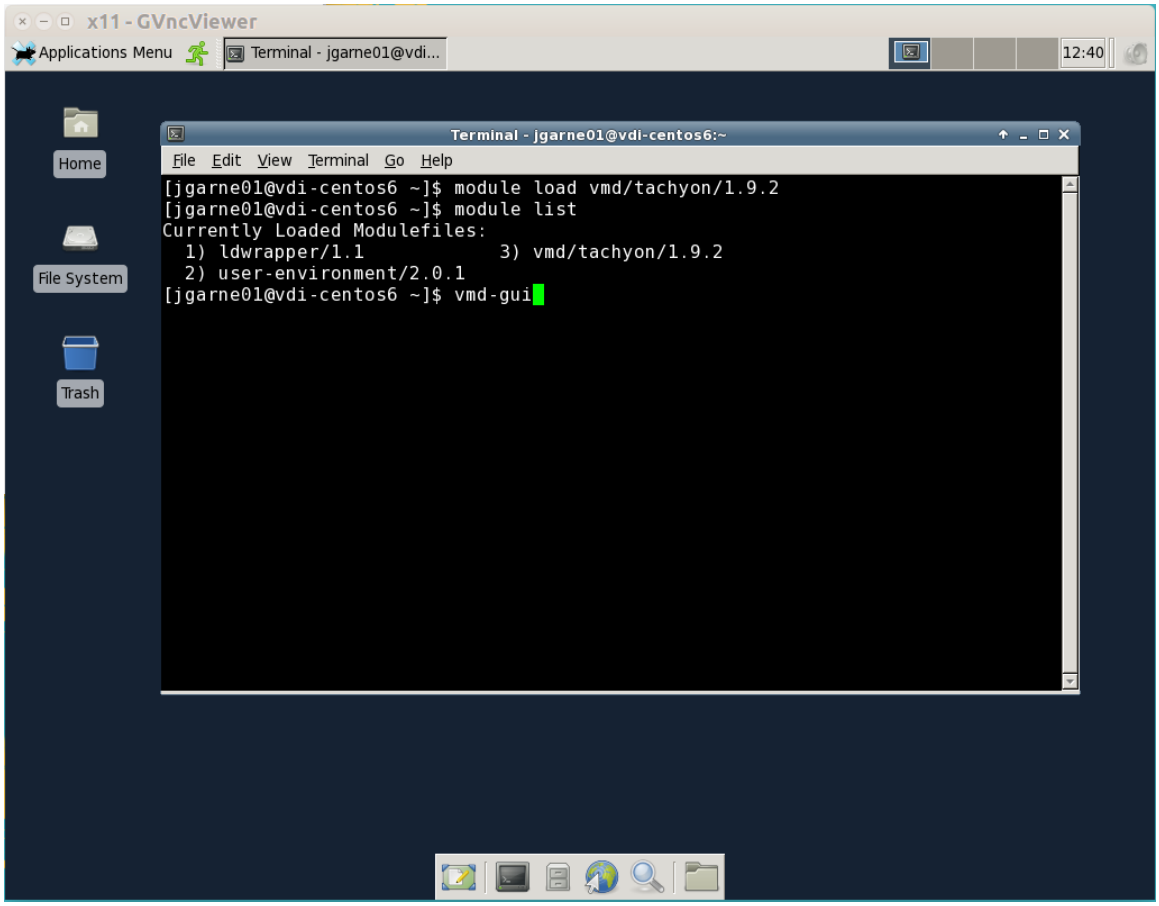
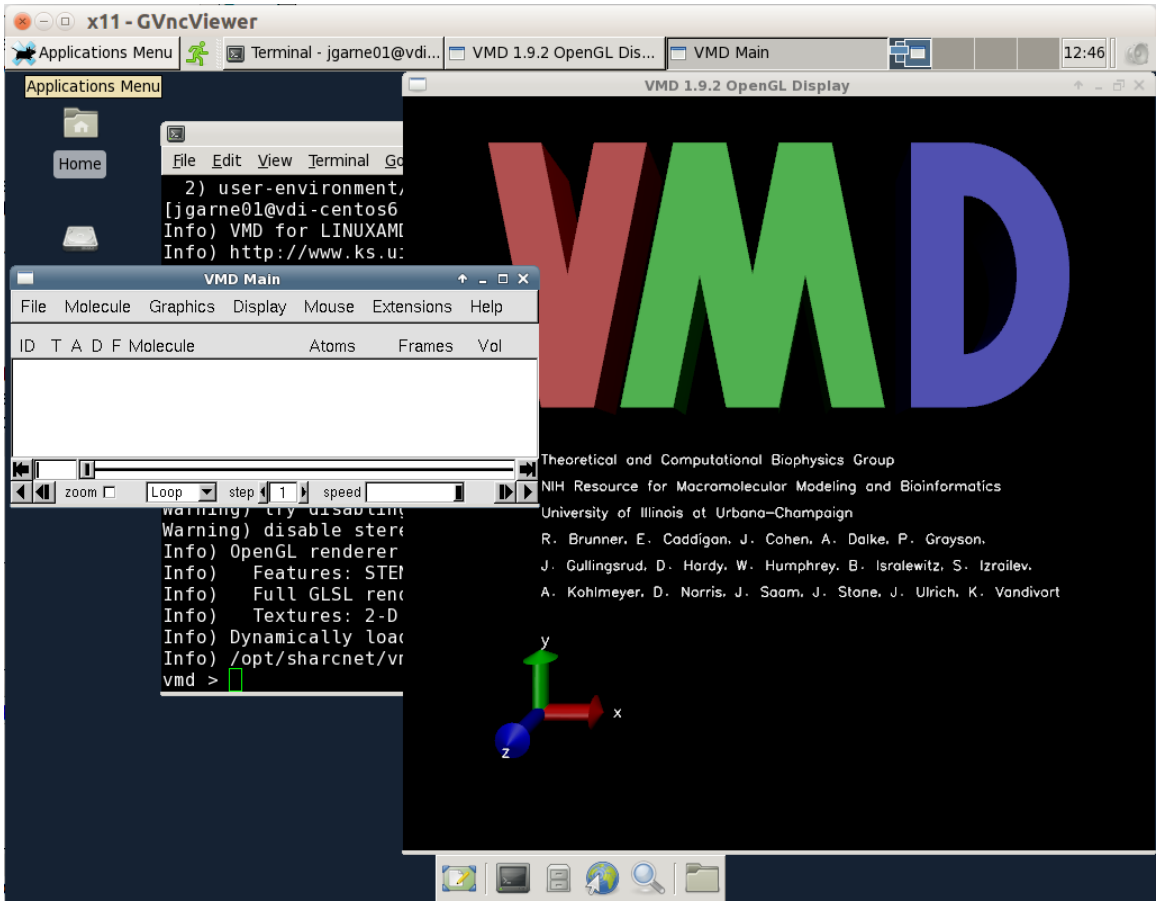


**Figure 13:** An example of the graphical version of VMD opened using the vdi-centos6 system.




# 7 Abbreviations

| Abbreviation | Full Name |
|---|---|
| RAM | Random Access Memory |
| MWE | Minimum Working Example |
| HPC | High Performance Computing |
| VNC | Virtual Network Computing |
| VMD | Visual Molecular Dynamics (Program) |
| GB | gigabyte ($10^9$ bytes) |
| MB | megabyte ($10^6$ bytes) |
| KB | kilobyte ($10^3$ bytes) |
| ppn | processors per node (for Calcul Québec) |