Contents

1 Logging in to Compute Canada (CC)

1.1 Windows

For Windows, I suggest downloading MobaXterm for two reasons: (i) it is color-coded, and (ii) there is a Secure File Transfer Protocol (SFTP) browser on the left when you use a Secure Shell (SSH) connection. You can download it from here: `https://mobaxterm.mobatek.net/`.

To enter CC, you have two options:

1. Log in from terminal (click on + `Start local terminal`)

   `ssh username@cluster.computecanada.ca`

2. Log in from user session (click on `Session` → `SSH`)

   Remote host*: `cluster.computecanada.ca`
   ✓ "Specify username": `username`

   where `username` is your CC's username (e.g., `abcd123`) and `cluster`, the cluster that you want to use (`Cedar`, `Niagara`, `Graham`, or `Beluga`).

By experience, you will mostly use Cedar, sometimes Niagara, or a combination of both. This should be discussed with Dr. Aris-Brosou (as to efficiently use your resources). While Cedar is generally used for general-purpose computing, Niagara is designed for large parallel jobs.

1.2 MacOS

For MacOS, I use a combination of (1) Terminal, already installed on your computer, and (2) Cyberduck (`https://cyberduck.io/`). With Cyberduck, there is also a SFTP browser that allows you to upload/download files easily and to directly edit files with your preferred text editor. I currently use TextMate, which is simple and works fine for me: `https://macromates.com/`.

To enter CC, you have to use the command line (see section 1.1 "Windows"):

`ssh username@cluster.computecanada.ca`

1.3 SSH keys

To log into Niagara only (as of now), you need to generate and use SSH keys.

On your machine (and not within a CC server), generate your keys:

`ssh-keygen -b 4096 -t rsa`

Give a name to your SSH keys when asked "`Enter file in which to save the key (/Users/admin/.ssh/id_rsa):` "

If you do not want a passphrase, which I recommend, simply press enter when asked "`Enter passphrase (empty for no passphrase):` "

Move your SSH keys, `SSH_key` and `SSH_key.pub` being the private and public key, respectively, to the hidden `.ssh` folder: `mv SSH_key* .ssh/`

1. Log in from terminal

   `ssh -i ~/.ssh/SSH_key username@cluster.computecanada.ca`

2. Log in from user session (with MobaXTerm)

   Remote host*: `cluster.computecanada.ca`
   ✓ "Specify username": `username`
   ✓ "Use private key": locate your private key on your computer (mine was in Documents → MobaXTerm → home → .ssh)

2 Specifically related to CC

2.1 Storage systems

On each cluster, you will have:

1. `Home` directory - should be used to store software or Python/Conda environments. Has a limited quota (volume space) per user.

2. `Scratch` directory - has a large file quota per user, is ideal to store databases and data, BUT is not backed up (i.e., if you delete a file from `Scratch`, it will be deleted forever). **Purges inactive data** (60-day-old files): you will receive an email stating which files have to be deleted (in `/scratch/to_delete/` for Cedar or `/scratch/t/to_delete/` for Niagara). You can simply copy and rename files that you want to keep.

3. `Project` directory - has a limited quota per **group**. Honestly, I have never used it as my group's `Project` was always full, but this could be useful for this lab (if used efficiently). This would avoid you the task of constantly copying your data in `Scratch`, like I do. This directory is backed up.

To see your usage: `diskusage_report`

2.2 Modules

Multiple software that you will be using are available under CC.

To see all the modules available under CC: `module avail` or `module av`

To find all possible versions, extensions, or modules to load first: `module spider software_name`

To load the module: `module load software_name`

To see what are the modules currently loaded in: `module list`

For `Niagara` only, you need to load the CC's environment **before** loading any modules:

```
module load CCEnv arch/avx512 StdEnv
```

All available software can be found here: `https://docs.computecanada.ca/wiki/Available_software`.

3 Slurm

3.1 Batch scripts

To submit a script in shell, the first line is **always** `#!/bin/bash`. Here is an example for batch script `script1.sh`:

```
#!/bin/bash
module load gcc/9.3.0 blast+

blastp \
-num_threads 24 -evalue 10 -use_sw_tback -max_target_seqs 1 \
-outfmt '6 qseqid sacc bitscore pident length stitle sscinames' \
-db /home/abcd123/scratch/nr_db/nr \
-query genes.fasta -out gene.txt
```

If executed, this script would run the `blastp` command. It is also important to load the modules in the script before running your commands (`module load gcc/9.3.0 blast+`).

You can write a script in any source code editor - for Windows, I suggest Notepad++ (because it is color-coded): `https://notepad-plus-plus.org/`. Alternatively, there is also Visual Studio Code (`https://code.visualstudio.com/`).

You usually execute a batch script without submitting to the job scheduler (`Slurm`) if you are in an interactive job mode (see next section). Otherwise, if you are using too many resources or your script is taking too long and it is affecting the other users, CC will automatically contact you and kill your script.

To execute a batch file, simply do: `./name_of_your_script`.

Why is my script not executing?

The two common reasons are:

1. You imported a `.sh` from MobaXterm and it needs to be converted to Unix. To convert your file, simply do:

```
dos2unix name_of_your_script
```

2. You do not have the permissions to execute the file. To be able to execute the file, simply do:

```
chmod +x name_of_your_script
```

`chmod +x` gives **everyone** the permissions to execute, write, and read your file. To learn more about permissions (and allow access to other users or to your group), see: `https://chmodcommand.com/`.

3.2 Submitting a job to `Slurm`

All jobs should be scheduled through `Slurm` (except for small quick jobs that do not require much memory/many CPUs). To do this, you have to add some parameters in your batch script, which differ from Cedar/Graham to Niagara. Here are some examples:

3.2.1 Cedar/Graham

```
#!/bin/bash
#SBATCH -c 4 # Number of CPUs requested
#SBATCH --mem=80G # Memory in Mb (M) or Gb (G)
#SBATCH -t 6-23:59:0 # How long will your job run for?
#SBATCH -J blast_job # Name of job

# Load your modules
module load gcc/9.3.0 blast+

blastp \
-num_threads 24 -evalue 10 -use_sw_tback -max_target_seqs 1 \
-outfmt '6 qseqid sacc bitscore pident length stitle sscinames' \
-db /home/abcd123/scratch/nr_db/nr \
-query genes.fasta -out gene.txt
```

`-c X`: the number of CPUs requested. If omitted, the default value is 1 CPU.

`--mem=X`: the memory requested in Mb (M) or in Gb (G).

`-t 0-00:00:0`: the syntax is days-hours:min:sec. The maximum time your job can run for is 14 days. If the time requested is not sufficient, your script could abort prematurely. If omitted, the default value is 3:00:0 (3 hours)

`-j X`: the name of the job. It is important to come up with a unique name for each of your jobs: if you ever need to kill it, it will easily be identified with `sq`.

Be mindful that the more resources (memory, CPUs, and time) you request, the more you will also wait in line (in the queue).

3.2.2 Niagara

```
#!/bin/bash
#SBATCH --nodes-1
#SBATCH --ntasks=40
#SBATCH -t 0-23:59:0 # How long will your job run for?
#SBATCH -J blast_job # Name of job

# Load your modules
module load CCEnv arch/avx512 StdEnv # Load CC environment
module load gcc/9.3.0 blast+

blastp \
-num_threads 24 -evalue 10 -use_sw_tback -max_target_seqs 1 \
-outfmt '6 qseqid sacc bitscore pident length stitle sscinames' \
-db /home/abcd123/scratch/nr_db/nr \
-query genes.fasta -out gene.txt
```

`--nodes=N`: the number of whole nodes requested.

`--ntasks=X`: the number of cores per node. The number has to be a multiple of 40. Each node has a fixed 202 Gb of RAM (memory), so you do not specify this parameter (like in Cedar).

`-t 0-00:00:0`: the syntax is days-hours:min:sec. The maximum time your job can run for is 24 hours in Niagara. If the time requested is not sufficient, your script could abort prematurely.

`-j X`: the name of the job. It is important to come up with a unique name for each of your jobs: if you ever need to kill it, it will easily be identified with `sq`.

Be mindful that the more resources (memory, CPUs, and time) you request, the more you will also wait in line (in the queue). However, Niagara is usually faster than Cedar.

Please also make sure that you use your resources efficiently (and do not always ask for more memory than needed, for example): otherwise, you could lower your priority and be stuck waiting days or weeks for your scripts to be scheduled. Do not hesitate asking Dr. Aris-Brosou or I for more help. See also the next section.

The amount of resources necessary will vary from script to script. There is no formula or right answer to determine this, you will find the best parameters through trial and error.

3.3 Interactive jobs

You should always test your scripts before sending them to Slurm, just to make sure that it will not stop in the first few minutes, and to be able to modify your resources accordingly.

On **Niagara**, you can do:

```
debugjob N
```

The number of nodes (N) can be up to 4, and the time allocated depends on the N requested: 1 hour for $N = 1$ to 22 minutes and 30 seconds for $N = 4$.

If the testing takes more than 1 hour, you can try an interactive job in both Cedar and Niagara:

`Cedar: salloc -c X, -t X, --mem=X`

`Niagara: salloc --nodes=N, -t X`

3.4 Slurm commands

To submit a job to slurm: `sbatch Name_of_the_script`

To see the status of your pending/running jobs: `sq`

To see the estimated date your job will start: `sq --start`

To cancel a job: `scancel -j Job_ID`

To see the status of your completed job: `sacct -j Job_ID`

To hold the execution of a job: `scontrol hold -j Job_ID`

Requeue a held job: `scontrol release -j Job_ID`

Once you submit a job with the `sbatch` command, you get a message from Slurm (e.g., "`Submitted batch job 14228735`"). In this case, `14228735` is the job ID.

4 R

You will most likely use R under Dr. Aris-Brosou's supervision. To use it on your personal computer, I suggest that you download R (`https://www.r-project.org/`) and use it inside of RStudio (`https://www.rstudio.com/`). You can use R alone, but I prefer RStudio to visualize my data.

I will let the R expert, Dr. Aris-Brosou, help you to learn R. Just a quick tip: to learn the basic commands, you can use the library `swirl`, which allows users to learn R inside of R. To try it, type these commands inside R or RStudio:

```
install.packages("swirl")
library(swirl)
swirl()
```

In Cedar or Niagara, you need to load the module first (`module load r/version` - make sure that you always keep the same version). To get the interactive mode, simply type:

```
R
```

Once you have entered the R interactive mode (either in CC, or on your personal computer with R and RStudio), you can install the libraries and load them into your R environment with these commands:

```
install.packages("name_of_library")
library(name_of_library)
```

To execute a R script (e.g., `script.R`), either in a batch script or on the command line:

```
module load r/version
R CMD BATCH name_of_script
```

To avoid re-installing packages, I suggest to (i) install all your R packages interactively (through Shell), and (ii) only use the `library` command inside of your R scripts, to load your librairies.

5 Miniconda

To download Python software, it is a good idea to get `Miniconda`. You can download it here for Unix (`https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh`) and follow the instructions for Miniconda here (`https://conda.io/projects/conda/en/latest/user-guide/install/linux.html`).

To create a new conda environment with a specific python version: `conda create --name env_name python=version`

To remove a conda environment: `conda remove --name env_name --all`

To activate a conda environment: `conda activate env_name`

To install conda packages to an activated environment: `conda install package_name`

To install Python packages: `pip install package_name`

For more information, refer to Conda documentation: `https://conda.io/projects/conda/en/latest/index.html`.

6 Transfer files

To transfer files with MobaXterm, you have three options:

1. Transfer files (either from your computer to the cluster, or from the cluster to your computer, or between clusters) using the command line. To do this, you have to be **disconnected from the cluster** (on local terminal).

   Uploading files (from computer to cluster), which I do not recommend:
   `sftp path_of_file_on_your_computer username@server.computecanada.ca:path_of_destination`

   Downloading files (from cluster to computer):
   `sftp username@server.computecanada.ca:path_of_file_on_the_server path_of_destination`

   Transferring files between clusters: `sftp username@server1.computecanada.ca:path_of_file_to_transfer username@server2.computecanada.ca:path_of_destination`

2. Transfer files (either from your computer to the cluster, or from the cluster to your computer) using MobaXterm's or Cyberduck's SFTP browser. Ideal for small or medium size files.

3. Transfer files (either from your computer to the cluster, or from the cluster to your computer, or between clusters) using Globus: `https://globus.computecanada.ca`. Select `Compute Canada` and log in using your CC credentials. To access your computer's files, you have to download Globus Connect Personal: `https://www.globus.org/globus-connect-personal`. Ideal for medium to large size files.

7 Unix

7.1 Tips

To write a comment (that will not be executed), use the `#`

Use the `tab` key to complete commands.

7.2 Directories

A directory can contain as many subdirectories as you like.

"In Linux and other Unix-like operating systems, a **forward slash** is used to represent the *root directory*, which is the directory that is at the top of the directory hierarchy and that contains all other directories and files on the system. Thus every absolute path, which is the address of a filesystem object (e.g., file or directory) relative to the root directory, begins with a forward slash." (source: `http://www.linfo.org/forward_slash.html`) Forward slashes (`/`) are also used as separators for directories.

If I have these folders on **Cedar** (my username being `abcd123`):

/home
  /scratch
    /ete2021
    /taxonomy_db
      /viruses
        db.txt
  /project

The **absolute** path of directory `ete2021` is `/home/abcd123/scratch/ete2021` **or** `~/scratch/ete2021`.

The absolute path of your home directory is: `/home/username` in Cedar and Graham. For Niagara, it will be: `/home/s/sarisbr0/username`. In both cases, ~ replaces `/home/username` (or `/home/s/sarisbr0/username` in Niagara).

The **relative** path is relative to your current emplacement. The relative directory before yours is "`..`".

If you are currently in directory `viruses` and you are trying to move to `ete2021`, you would have to do

With absolute paths: `cd /home/abcd123/scratch/ete2021`

With relative paths: `cd ../../ete2021`

Useful commands:

To create a directory: `mkdir directory_name`

To move to this directory: `cd directory_name`

To rename the `original_name` directory to `new_name`: `mv original_name new_name`

To get the absolute path of your current directory: `pwd`

To move to your home directory: `cd`

To back up one directory: `cd ..`

7.3 Other useful commands under Unix

To print on the screen "Hello World": `echo "Hello World"`

To cope the file `file1.txt` and name it `copy.txt`: `cp file1.txt copy.txt`

To move a file: `mv file_name path`

To rename a file: `mv file_name new_name`

To list all the files in a directory: `ls`

To see the size of the files in your directory: `ls -sh`

To count the number of lines of your file: `wc -l`

To find the word "blast" in your file (or the line numbers): `grep "blast" file_name` (or `grep "blast" file_name -n`)

To show the first lines of a file (or to show a specific number of lines, e.g., 18): `head file_name` (or `head -n 18 file_name`)

To show the last lines of a file (or to show a specific number of lines, e.g., 18): `tail file_name` (or `tail -n 18 file_name`)

To create a file called `file1.txt`: `touch file1.txt`

To view the content of a file: `less file_name`. Press the `q` key to exit.

To concatenate (combine) `file1.txt` and `file2.txt` in `combinedfile.txt`: `cat file1.txt file2.txt > combinedfile.txt`

To get help for a command (e.g., to see the possible options/parameters): `command_name -h`, or sometimes `command_name --h`

To get the manual page for a command: `man command_name` (and to quit the manual page, press the `q` key)

7.4 `rm` command

The `rm` command is used to remove a file (`rm file_name`) or a directory (`rm -r directory_name`). In Niagara, you have to accept to remove a file or a directory (they ask you `Y/N`). To "force" the command, you can use add the option `-f`. **HOWEVER**, be aware that removing a file from `Scratch` is irreversible - thus, be absolutely certain that you do not need these files when using the `rm` command.

7.5 Pipelines

`Command 1 | Command 2`

Use the output of `Command 1` as an input for `Command 2`.

For example:

`ls | wc -l` gives the number of files in the directory. `ls` prints the names of all the files in the directory, and is used as an input for `wc -l`, who counts the number of lines (thus, the number of files).

`ls file* | wc -l` gives the number of files starting with "file" in the directory. `ls` prints the names of all the files starting with "file" in the directory, and is used as an input for `wc -l`, who counts the number of lines (thus, the number of files).

`ls *.txt | wc -l` gives the number of files ending with ".txt" in the directory. `ls` prints the names of all the files ending with ".txt" in the directory, and is used as an input for `wc -l`, who counts the number of lines (thus, the number of files).

`grep "blast" file1.txt -n | head -n 5` gives the first 5 occurrences of finding the word "blast" in the file `file1.txt`, with the line numbers. `grep "blast" file1.txt -n` prints the line numbers of all the findings of the word "blast", and is used as an input for `head -n 5`, who only outputs the 5 first occurrences.

You can use more than one pipe (|) to create long pipelines with multiple commands, if you want.

7.6 `vim` or `vi`

`Vim file_name` allows the user to directly edit a file from the command line.

Press the `i` key to enter "writing mode". To exit "writing mode", press the `Esc` key

Move lines with the arrows (on your keyboard)

To delete `X` number of lines: exit "writing mode" (if not already done) → type the number of lines you wish to delete → press the `d` key twice. For example, if I wish to delete 14 lines, I would do: `Esc` → `14` → `dd`

Write `:q!` to quit without any changes

Write `:wq!` to save and quit

8 To go further

If you get stuck:

Pause job: `ctrl + Z`

List processes: `ps`

List processes: `kill -9 process_ID`

Further information can be found in these links below. All the information related to CC was retrieved from its wiki page.

Compute Canada wiki - `https://docs.computecanada.ca/wiki/Compute_Canada_Documentation`

Running jobs - `https://docs.computecanada.ca/wiki/Running_jobs`

Niagara Quickstart - `https://docs.computecanada.ca/wiki/Niagara_Quickstart`

Compute Canada's modules - `https://docs.computecanada.ca/wiki/Available_software`

Compute Canada clusters' status - `https://status.computecanada.ca/`

Conda documentation - `https://conda.io/projects/conda/en/latest/index.html`

Chmod command - `https://chmodcommand.com/`

(Free!) R course - `https://www.edx.org/course/statistics-and-r`