Working remotely

Do not work locally

- Do not work locally; you will not have enough disk space, memory, CPU, or GPU, you will spend time waiting with your computer on, and you will have an idiosyncratic software environment (e.g., Windows) that you will waste time trying to get set up right
- The only thing I do locally is analysis of data (i.e., making graphs and running stats on experimental or modelling results), because, even though the university has an Rstudio server, in most cases (but not all) there is no advantage resource-wise
- Occasionally I used to set up resource-intensive jobs on my own machine because they used some specific software that I didn't feel like setting up on the cluster - this is not a good reason; a good reason to work locally would be if you lack internet access or if you are a spy

Logging in

- You all have accounts on Sci-Net, which gives you access to a number of inter-linked machines; we will work on *Graham*
- You will need to have an SSH client, and I assume you already do, via a terminal in your operating system; you will technically be able to log in with a password, but this becomes impractical and so you will need an SSH key, which comes as a* private-public key pair*
- Think of a combination lock – your private key is the combination that opens only one specific lock; the "lock" is called the *public* key, because you can put it on any server that you have an account on, just like you can bring your combination lock to any pool and safely leave it out in the open (in public)
- Because there are steps involved, you created an SSH key and uploaded it to Graham in advance of the workshop, following https://docs.alliancecan.ca/wiki/SSH ; **now, log in**

Cluster computing

- The machine you log in to is not the machine where you run things; it's a *login node* which allows you to send jobs to be run on other computers called *compute nodes*
- The compute and login nodes all share a common filesystem, connected over the network
- This setup is called a *compute cluster,* and it's used because it's arbitrarily extensible, whereas a single computer is not
- Do not run anything on the login node! from the Alliance wiki: "*exceptions are made for compilation and other tasks not expected to consume more than about 10 CPU-minutes and about 4 gigabytes of RAM*"

Scheduler

- To access compute, use the *scheduler* (https://docs.alliancecan.ca/wiki/What_is_a_scheduler%3F, https://docs.alliancecan.ca/wiki/Running_jobs)
- The scheduler we use is called *Slurm;* the usual way to submit jobs is using *sbatch*; another thing you can do is create an *interactive session* on one of the nodes, in case you want to do something like exploratory data analysis or debugging
- In either case, you will need to declare how much *time*, how many *CPU cores*, how much *RAM*, and how many and what type of *GPUs*; in general, the more you ask for, the lower you will be on the queue (and there is no way to ask for infinity of any of these resources); if you exceed what you ask for, your job will be killed; interactive jobs are also lower priority
- Before proceeding, consider the "number of CPU cores" somewhat carefully: some of the libraries you use daily may by default be using *parallelism,* that is, dividing a large task into several subprocesses and running each on a different CPU core – make sure you know
- Note also that if you wanted to use more CPU cores than one single compute node has (mostly 32 for Graham: see here https://docs.alliancecan.ca/wiki/Graham), you would need to use a more sophisticated kind of parallelism called MPI; you can also do (single-node) parallelism by splitting into threads ("light processes"), rather than processes, often using something called OpenMP; we won't cover any of these things but you see them on the wiki
- Using your favourite command-line editor (vi or emacs), write a bash script called test.sh that writes something to a file, sleep for 30 seconds using the sleep command (so we can watch it run), and then afterwards print something else to the screen
- The wiki now tells you to run your job using a *job script* which would imply either adding some special magic comments into the beginning of test.sh, or writing a second bash script with magic comments in order to

run test.sh; the magic comments pass arguments to the scheduler, and this can be convenient but for running one-off jobs it is easier to pass these arguments to *sbatch* on the command line – only the time doesn't have a default:

```
sbatch --time=00:01:00 test.sh
```

- This will run your (30 second) job with a time allocation of one minute; now quick, type

```
squeue -u <your username>
```

- Currently (when I wrote this) my job is **PD** or pending (not running yet); after a while it will change to **R**
- When the script starts executing, I should see my output file being created; 30 seconds later I should see a file called *slurm-<jobid>.out* created, which contains the stdout of my job

Storage and file management

- The cluster has a filesystem layout which is quite typical of shared compute servers (https://docs.alliancecan.ca/wiki/Storage_and_file_management): a relatively small **home directory;** a **scratch folder;** and access to our team **project folders**
- From the wiki: *"While your home directory may seem like the logical place to ... do all your work ... this isn't the case; your home ... has a relatively small quota and doesn't have ... good performance for writing and reading large amounts of data. ... your home directory is typically [for] source code, small parameter files and job submission scripts."*
- On the other hand, **scratch,** which is much bigger and faster, and thus made for work, is regularly purged; so, given that you should **not ever** use git repositories to store binary files larger than about 1MB (e.g., data, models), once you know what artefacts you need to keep, you need to think about where you are going to store these files - it isn't scratch, and it isn't home
- Move these files into a team project folder, with the approval of those working on the project
- On another topic, large files can be moved on and off the cluster using *rsync* (via *scp*) for your local machine, *wget* or *curl* for off the web, and Globus for other institutional machines: https://docs.alliancecan.ca/wiki/Transferring_data
- Do this on the data mover node rather than the login node

Workflow

- Some people forever do all their work in command line editors like vim, but I sympathize with my CSC207 prof who, when referring to such people, did that L thing on his forehead
- If you don't already have one, install an IDE with an integrated debugger like VS Code; once you start doing this, you will realize you can't write your code directly on the command line on the remote machine (the IDE only runs locally)
- This suggests the following: (1) *start a git repository immediately, before you start working on anything, and find a remote such as Github* (remember that repo and remote are different things; we don't have time for a Git tutorial today, see https://docs.alliancecan.ca/wiki/Git)
- (2) *whenever you need to run anything on the cluster, commit, push, and pull*
- (3) If this is annoying for initial exploration or debugging, use an interactive Slurm job
- Lo and behold: if you do this, as a side effect, your code will always be backed up!
- If you want to pull from a private Github repository you own, or push to any repository, you will need a way to authenticate to Github; you can create a fresh SSH key on the cluster (Github>Account icon>Settings>SSH and GPG keys): *ssh-keygen*
- **Do this now: in your scratch, create a folder containg a "workspace" repo for this boot camp connected to a remote on Github**
- Given that you are principally using the remote machine to run batch (i.e., non-interactive) jobs, and only rarely to run interactive jobs, or write code or do other tasks directly on the login server, you can usually safely disconnect: jobs you have on the queue keep running
- However, if at any time you do happen to need to do any of these things, you will find that the jobs are killed when you get disconnected (e.g., if you close your laptop)
- For this reason, you should learn to use *tmux*, for which there is a very cursory document on the wiki: https://docs-dev.alliancecan.ca/wiki/Tmux