Spartan Introduction - Coding Club

David Wilkinson

21 March 2017

Contents

What is Spartan?

Spartan is the University of Melbourne's general purpose hybrid traditional high performance computing system (HPC) with cloud instances from the NeCTAR Research Cloud and attached Research Data Storage Services (RDSS).

It is designed to suit the needs of researchers whose desktop/laptop is not up to the particular task. Models running slow, datasets are too big, not enough cores, application licensing issues, etc.

Spartan consists of:

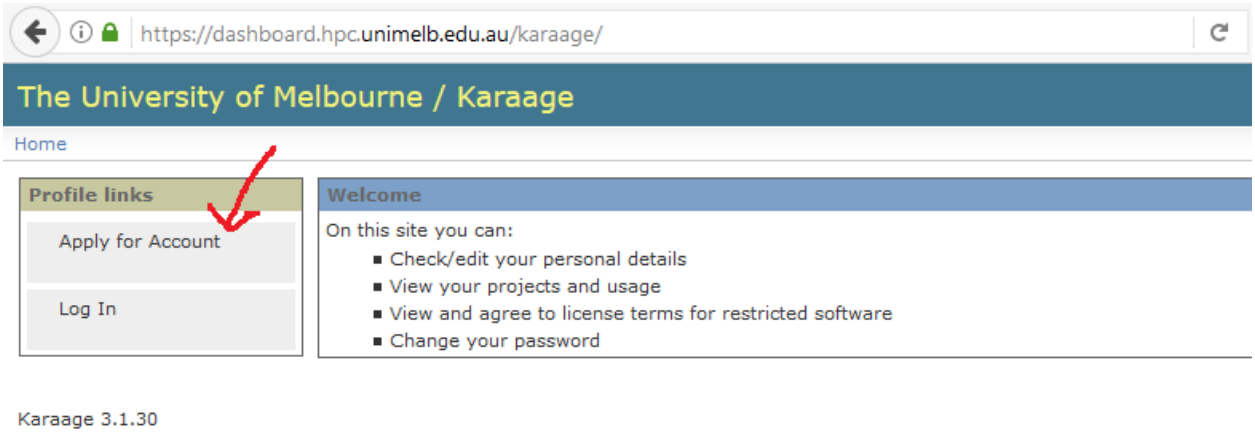
- a management node for system administrators,
- a log in node for users to connect to the system and submit jobs,
- a small number of 'bare metal' compute nodes for multinode tasks,
- any 'bare metal' user-procured hardware (e.g., departmental nodes),
- vHPC cloud compute nodes for overflow and GPGPU tasks, and
- general cloud compute nodes.

Accessing Spartan

Getting an Account

To gain access to Spartan you need to create an account.

- Via Karaage at link
- Need to create a project
- Need a project leader (you), and you can invite collaborators for joint projects
- Need a project title/description to demonstrate research goals and/or research support
- Takes ~2 days for approval



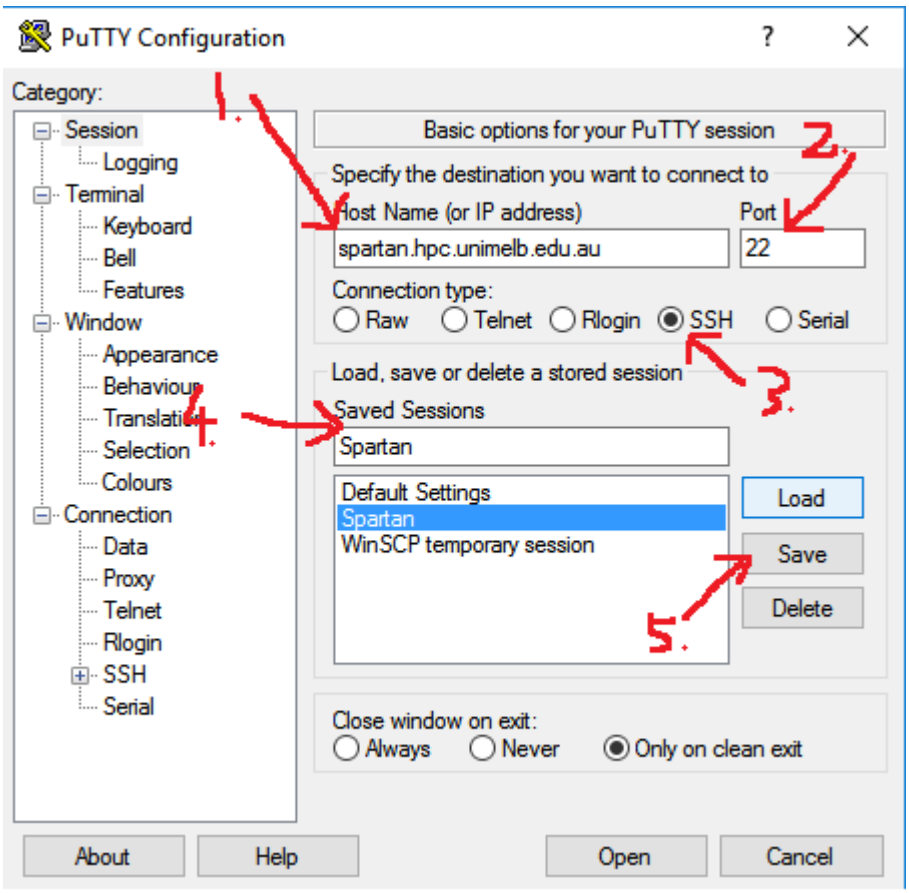
Required Programs

To connect to Spartan you will need a Secure Shell (SSH) and a Secure File Transfer Protocol (SFTP) client. The SSH client is your interface with Spartan while the SFTP client is used to transfer files from your local computer to your Spartan home directory.

Windows Users

Use PuTTY as your SSH client. This is an easy set-up with the following five steps:

1. Set your host name: spartan.hpc.unimelb.edu.au
2. Set your port number: leave as default (whereas Boab users need a defined port)
3. Set your connection type: SSH
4. Name your session to make it easy for future log-ins: Whatever you like i.e. Spartan
5. Save your session details

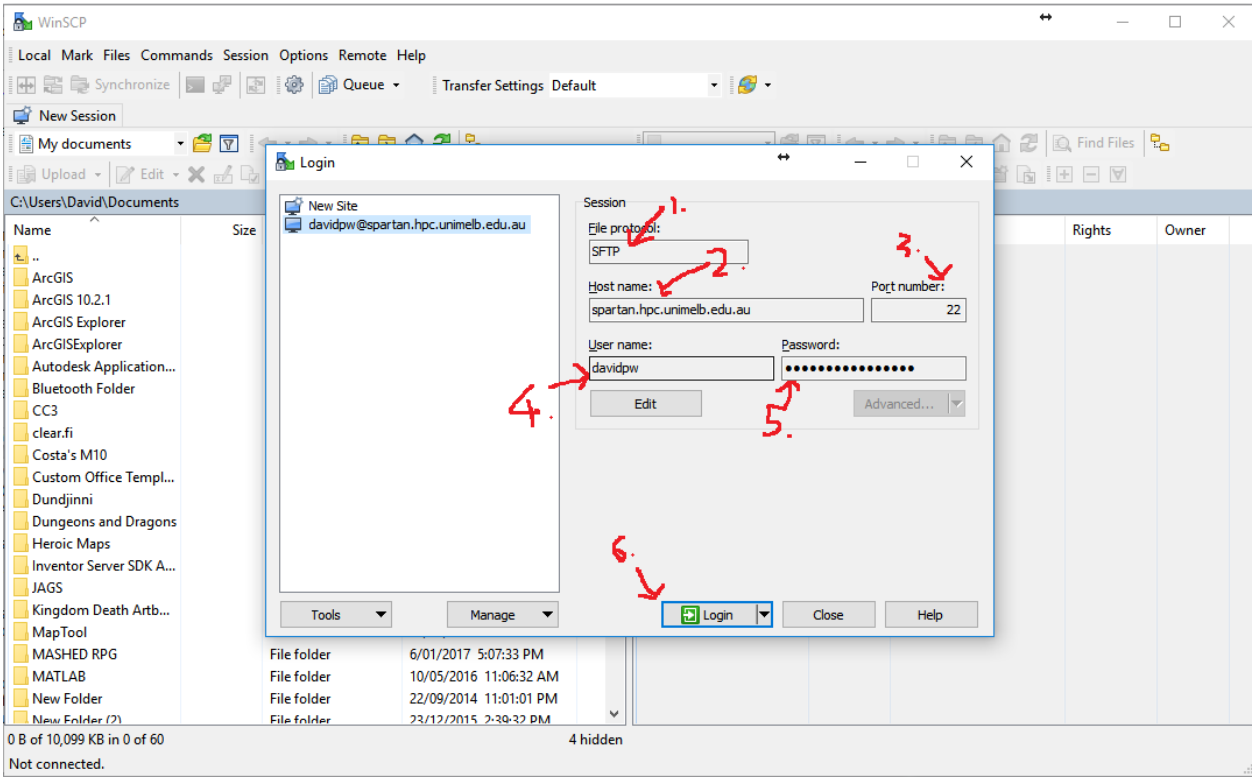


Your first log-in to Spartan via the SSH client creates your home directory on Spartan so it is important to do that before setting up your SFTP client.

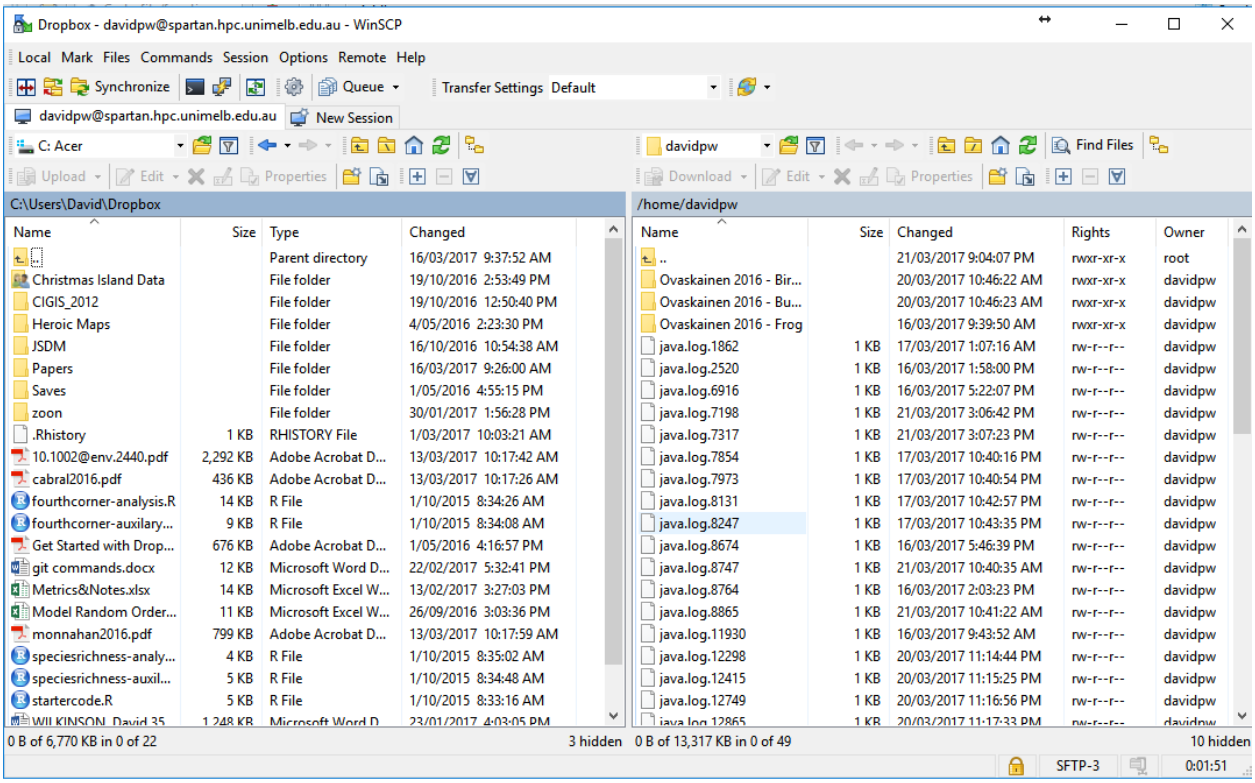
Use WinSCP as your SFTP client. This is an easy process with the following six steps:

1. Set your file protocol: SFTP
2. Set your host name: spartan.hpc.unimelb.edu.au
3. Set your port number: leave as default (whereas Boab users need a defined port)
4. Enter your username
5. Enter your password (note, this will show more characters than you entered when saved)

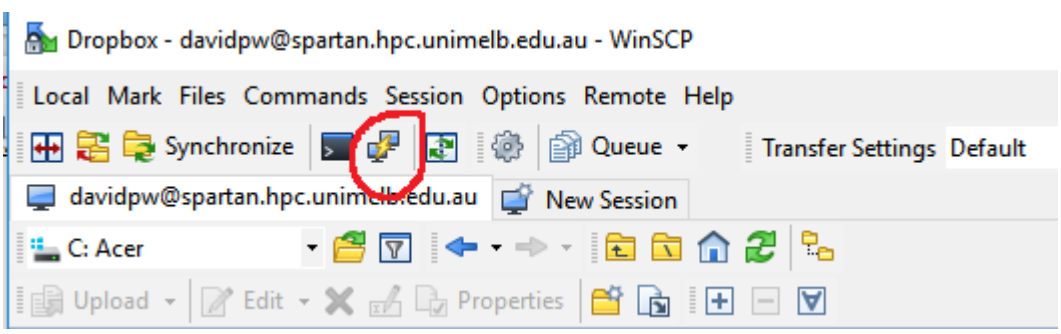
6. Log-in



Inside a WinSCP session you will have dual file explorer windows: your local machine (left) and your Spartan home directory (right). If you have not made your initial log-in to Spartan via your SSH client you will have a blank white screen in the right-hand window. Transferring files between the two directories is achieved with a simple drag-and-drop interface.



Your two clients are now set up and everything is ready to access Spartan. After set-up you don't need to deal directly with PuTTY as a separate program anymore as there is a button in the WinSCP toolbar to open a session.

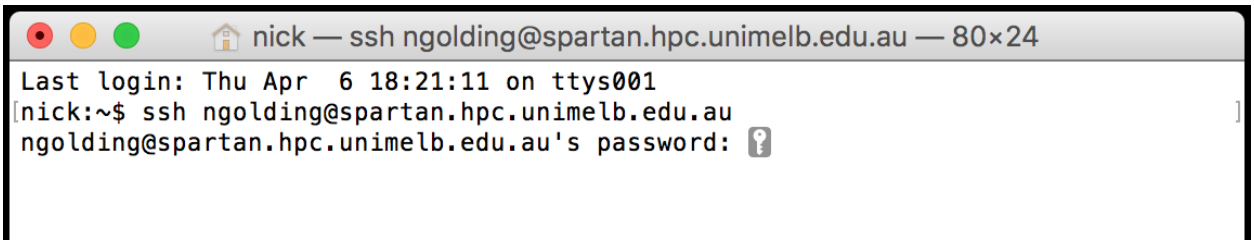


Mac/Linux Users

Unlike for Windows, OSX and (most) linux operating systems already have SSH installed and have a fully functional terminal built in. So to SSH into the Spartan log-in node, you just need to open up a terminal (the `Terminal` application on a mac), and issue the command:

```
ssh myusername@spartan.hpc.unimelb.edu.au
```

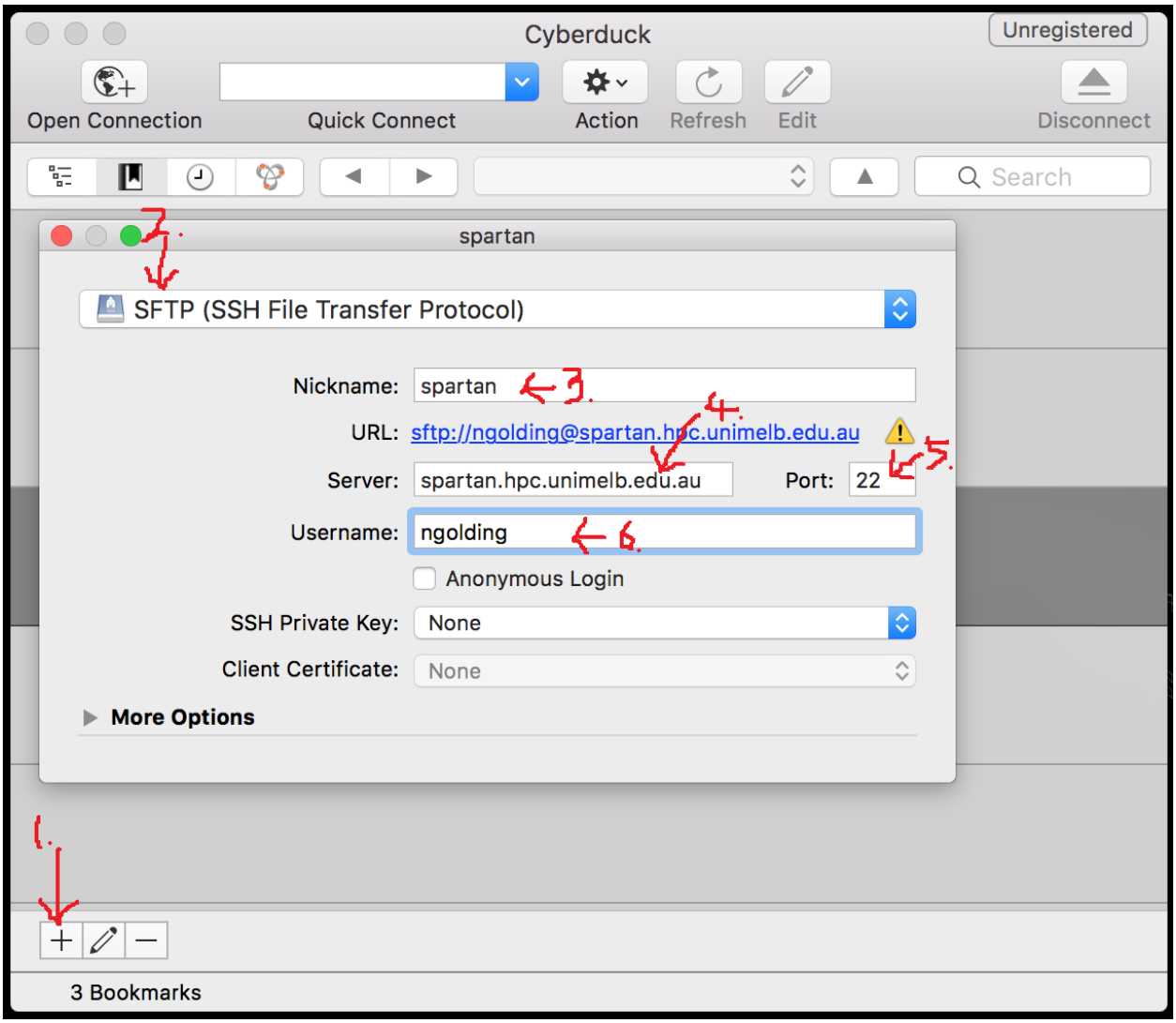
Replacing `myusername` with your username. Then enter your password (no characters will show, that's normal) and hit return. That should look something like this:



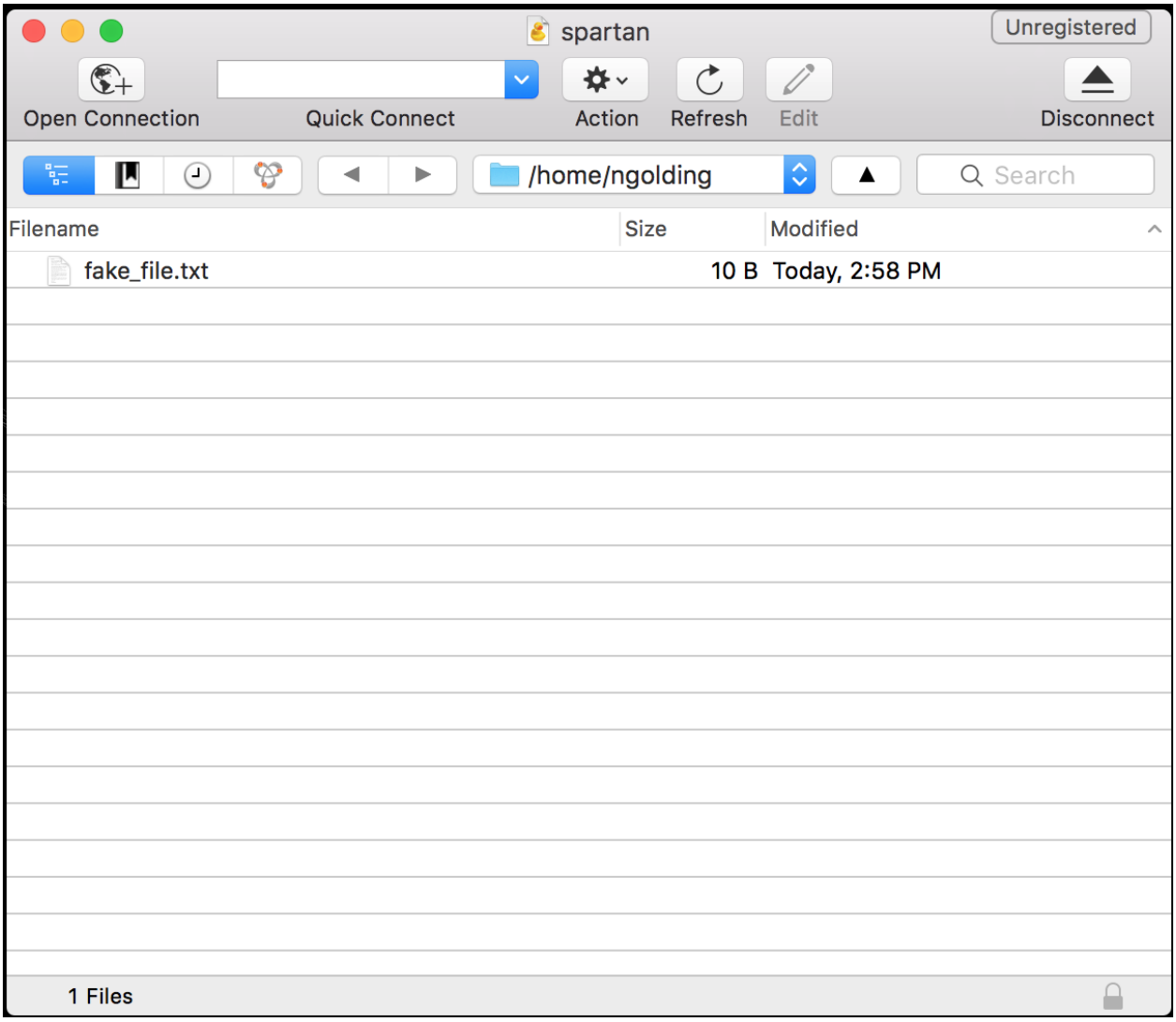
As for Windows, your first log-in to Spartan via the SSH client creates your home directory on Spartan so it is important to do that before setting up your SFTP client.

SFTP is also installed on most OSX and linux versions, so you could transfer files to Spartan directly from the terminal (do `man sftp` in the terminal if you're interested in that). However it's normally easier to use a SFTP client with a graphical user interface. One nice option for OSX (that also works on Windows) is Cyberduck. Once installed, you can add a new connection to Spartan with the following steps:

1. Clicking on the `+` in the lower-left corner
2. Set the connection type to SFTP
3. Give the connection a name
4. Set the server name: `spartan.hpc.unimelb.edu.au`
5. Set the port number to 22
6. Enter your username

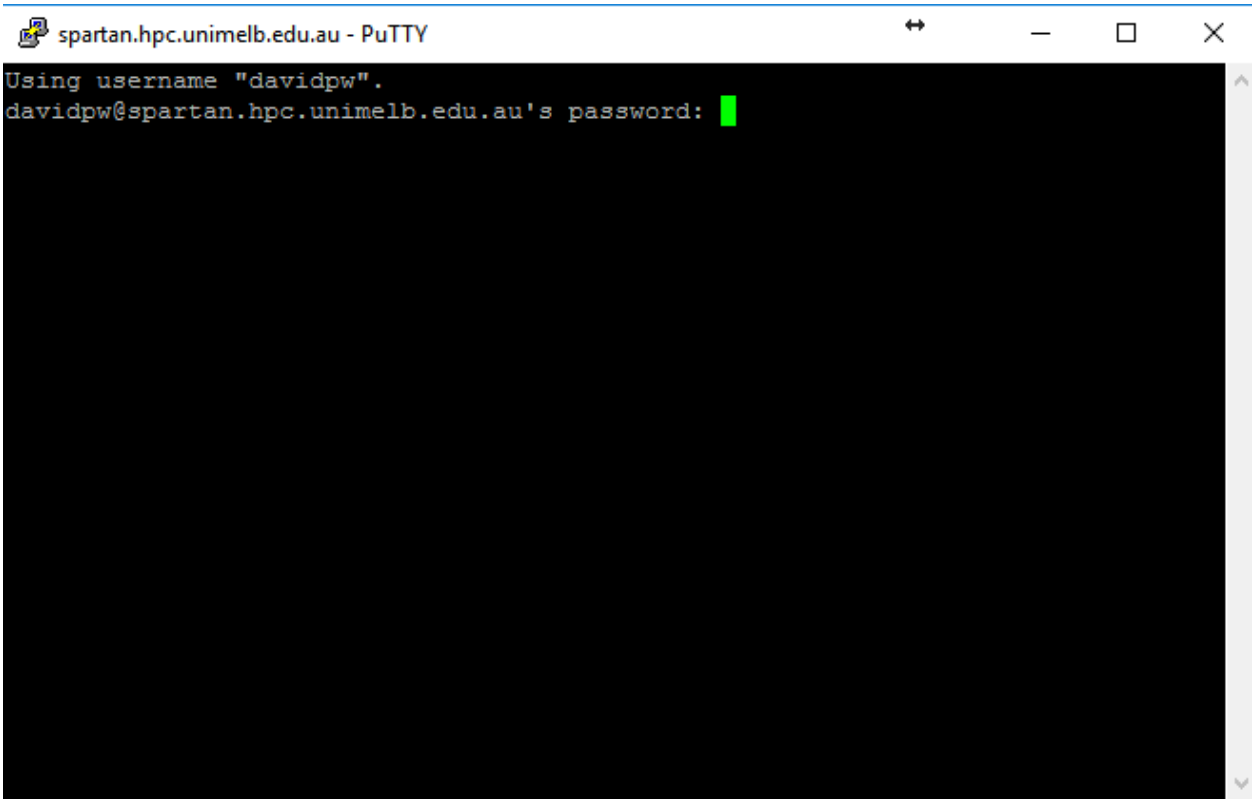


You can then close the connection details box, and double-click on the new connection to initiate it. This will pop up a file explorer like the image below, listing the files in your working directory on Spartan. You can drag and drop files between there and your computer.

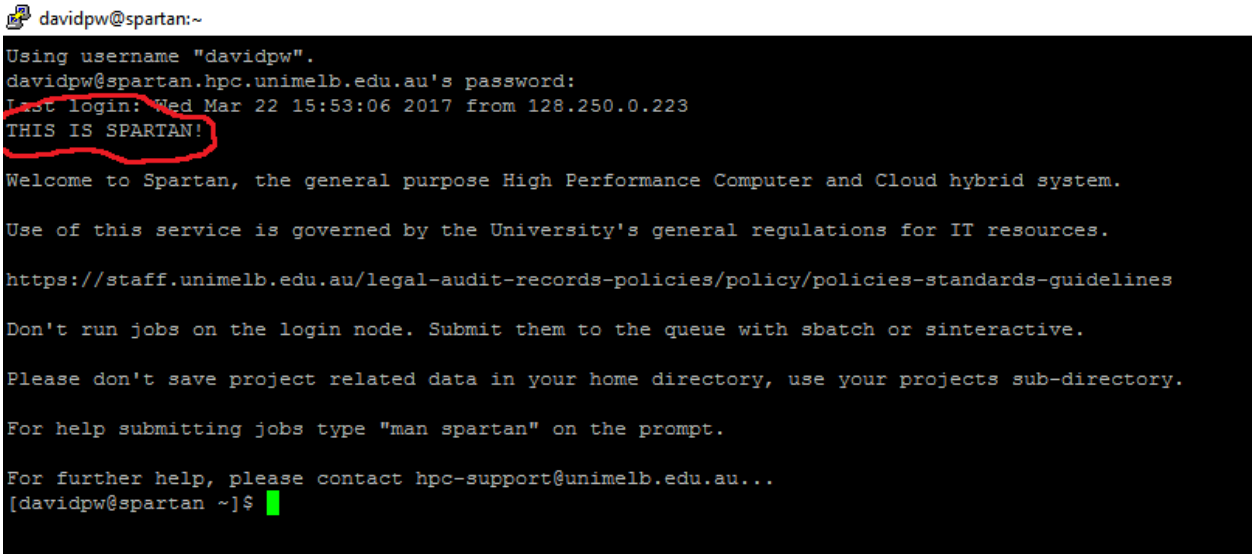


Log in to Spartan

Now that you have everything in place to access Spartan, open an SSH session (in PuTTY or Terminal). as before, this wont display characters as you type your password.



The text that loads at the beginning gives you the usual university IT policy spiel, a couple of getting started prompts, a warning about the log-in node, and the obligatory 300 reference.



Help

Typing `man Spartan` loads the university's Spartan FAQ. Rather than loading the entire document it loads it a screen at a time and you have to navigate by keyboard commands. For now, you will only need these:

- `h`: help, shows you the keyboard shortcuts
- `q`: quit the document and go back to the Spartan interface
- `^` is used in place of the control/command button, so `^V` is Ctrl+V
- `Enter/Return` will let you advance by one line
- `^Y` will let you go back by one line
- `^V` will let you advance by one window
- `^B` will let you go back by one window

---

davidpw@spartan:~

```
                    SUMMARY OF LESS COMMANDS

      Commands marked with * may be preceded by a number, N.
      Notes in parentheses indicate the behavior if N is given.
      A key preceded by a caret indicates the Ctrl key; thus ^K is ctrl-K.

  h  H                 Display this help.
  q  :q  Q  :Q  ZZ     Exit.
---------------------------------------------------------------------------

                           MOVING

  e  ^E  j  ^N  CR  *  Forward  one line   (or N lines).
  y  ^Y  k  ^K  ^P  *  Backward one line   (or N lines).
  f  ^F  ^V  SPACE  *  Forward  one window (or N lines).
  b  ^B  ESC-v      *  Backward one window (or N lines).
  z                 *  Forward  one window (and set window to N).
  w                 *  Backward one window (and set window to N).
  ESC-SPACE         *  Forward  one window, but don't stop at end-of-file.
  d  ^D             *  Forward  one half-window (and set half-window to N).
  u  ^U             *  Backward one half-window (and set half-window to N).
  ESC-)  RightArrow *  Left  one half screen width (or N positions).
  ESC-(  LeftArrow  *  Right one half screen width (or N positions).
  F                    Forward forever; like "tail -f".
  r  ^R  ^L            Repaint screen.
  R                    Repaint screen, discarding buffered input.
        ---------------------------------------------------
        Default "window" is the screen height.
        Default "half-window" is half of the screen height.
---------------------------------------------------------------------------
```

If you want help with a particular function in Spartan you can type `man <function name>` (without < >). This works like `?` in R.

Using Spartan

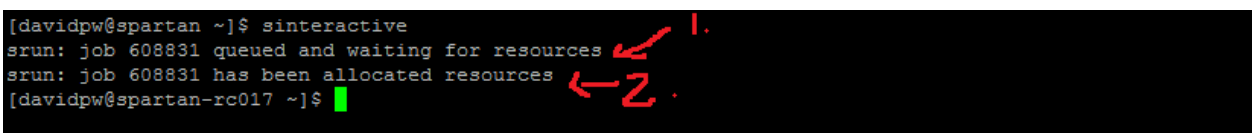
The log-in node on Spartan is a shared resource between all users and is only allocated the memory to handle log-ins and job submission. Do not run jobs in the log-in node or the admins will get upset and kill the job. There are two ways to get out of the log-in node and into dedicated compute nodes: `sinteractive` and `sbatch`.

sinteractive

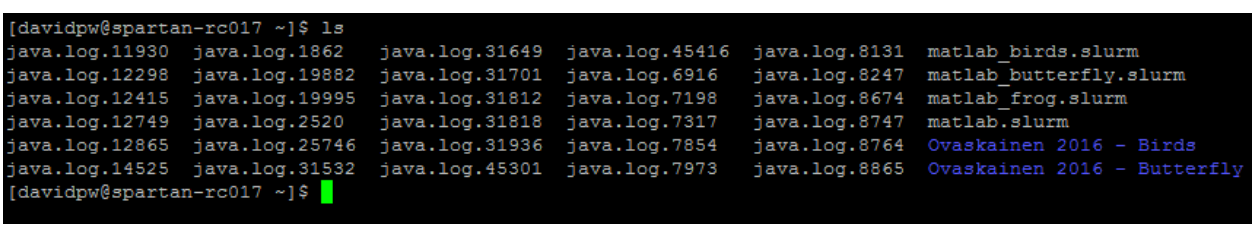
The `sinteractive` command will give you access to a compute node (as soon as available) where you can work interactively with your job. While you can submit an entire job in this method, that is better saved for `sbatch` and `sinteractive` used for testing/debugging. There are default settings for `sinteractive` which should be enough for most uses, but they can be modified if needed like so:

```
sinteractive --time=00:10:00 --nodes=1 --ntasks=1 --cpus-per-task=2
```

This will give you access to one node to perform one task with two processors for ten minutes (default values I think). Lets open a default node and explore some basic commands. Type `sinteractive`. This submits a request for a default compute node (1), and you have to wait for the resource to become available (2) before you can continue.



The `ls` command will show you everything in your current directory. Folders show up in blue.



The `cd` command will let you change your current directory. As Spartan is Linux based it uses / not for directories (unlike Windows). The character is used to say that the preceding term is a special character. Lets change into a sub-folder and see what is inside. Spartan lets you pre-fill file paths/names using the tab key like R, but it will pre-fill everything until a choice needs to be made, and won't provide options at that stage, but will complete everything once no more divergences exist.

```
[davidpw@spartan ~]$ ls
java.log.11930  java.log.1862   java.log.31649  java.log.45416  java.log.8131  matlab_birds.slurm          Ovaskainen 2016 - Frog
java.log.12298  java.log.19882  java.log.31701  java.log.6916   java.log.8247  matlab_butterfly.slurm      slurm-594363.out
java.log.12415  java.log.19995  java.log.31812  java.log.7198   java.log.8674  matlab_frog.slurm           slurm-594518.out
java.log.12749  java.log.2520   java.log.31818  java.log.7317   java.log.8747  matlab.slurm                slurm-594586.out
java.log.12865  java.log.25746  java.log.31936  java.log.7854   java.log.8764  Ovaskainen 2016 - Birds     slurm-594801.out
java.log.14525  java.log.31532  java.log.45301  java.log.7973   java.log.8865  Ovaskainen 2016 - Butterfly slurm-597789.out
[davidpw@spartan ~]$ cd Ovaskainen\ 2016\ -\ Birds/
[davidpw@spartan Ovaskainen 2016 - Birds]$ ls
examples  source code
[davidpw@spartan Ovaskainen 2016 - Birds]$
```

The programs available on Spartan as referred to as modules, and you can see a complete list using `module avail`.

---

davidpw@spartan-rc017:~

```
[davidpw@spartan-rc017 ~]$ module avail
Rebuilding cache, please wait ... (written to file) done.

------------------------------------------------------------------ /usr/local/easybuild/modules/all ----------------
   ABINIT/8.0.8b-intel-2016.u3                          NAMD/2.10-intel-2016.u3-mpi
   APR/1.5.2-GCC-4.9.2                                  NAMD/2.10-iomkl-2016.u3-mpi
   APR-util/1.5.4-GCC-4.9.2                             NAMD/2.10-iompi-2016.u3-mpi
   ARAGORN/1.2.36-GCC-4.9.2                             NAMD/2.12-intel-2016.u3-CUDA
   ATK/2.18.0-intel-2016.u3                             NAMD/2.12-intel-2016.u3-mpi-CUDA
   ATLAS/3.10.1-GCC-4.9.2-LAPACK-3.5.0                  NAMD/2.12-intel-2016.u3-mpi
   ATLAS/3.10.1-GCC-6.2.0-LAPACK-3.5.0                  NAMD/2.12-intel-2016.u3                                  (D)
   ATLAS/3.10.1-gompi-2015a-LAPACK-3.5.0                NASM/2.11.08-GCC-4.9.2
   ATLAS/3.10.1-goolf-2015a-LAPACK-3.5.0        (D)     NASM/2.11.08-intel-2016.u3
   AlignGraph/20160823-intel-2016.u3                    NASM/2.12.02-intel-2016.u3                               (D)
   Armadillo/7.600.2-intel-2016.u3-Python-2.7.9         NEURON/7.4-gompi-2015a
   Autoconf/2.69-GCC-4.6.4                              ORCA/3_0_2-linux_x86-64-OpenMPI-1.6.5
   Autoconf/2.69-GCC-4.9.2                              ORCA/3_0_3-linux_x86-64-OpenMPI-1.6.5
   Autoconf/2.69-GCC-5.4.0                              ORCA/4_0_0-linux_x86-64-OpenMPI-2.0.2                    (D)
   Autoconf/2.69-GCC-6.2.0                              Octave/3.8.2-goolf-2015a
   Autoconf/2.69-intel-2016.u3                          OpenBLAS/0.2.14-gompi-2015a-LAPACK-3.5.0
   Autoconf/2.69-intel-2017.u2                  (D)     OpenBLAS/0.2.15-GCC-4.9.2-LAPACK-3.5.0
   Automake/1.11-GCC-4.9.2                              OpenBLAS/0.2.15-GCC-6.2.0-LAPACK-3.5.0                   (D)
   Automake/1.14-GCC-4.9.2                              OpenBUGS/3.2.3
   Automake/1.15-GCC-4.6.4                              OpenCV/3.1.0-intel-2016.u3
   Automake/1.15-GCC-4.9.2                              OpenEXR/2.2.0-intel-2016.u3
   Automake/1.15-GCC-5.4.0                              OpenFOAM/2.1.1-gompi-GCC-4.6.4-OpenMPI-1.8.4
   Automake/1.15-GCC-6.2.0                              OpenFOAM/2.1.1-iompi-2016.u3
   Automake/1.15-intel-2016.u3                          OpenFOAM/2.3.1-intel-2016.u3
   Automake/1.15-intel-2017.u2                  (D)     OpenFOAM/2.3.1-iompi-2016.u3
   Autotools/20150119-GCC-4.9.2                         OpenFOAM/4.0-intel-2016.u3                               (D)
   Autotools/20150119-GCC-5.4.0                         OpenImageIO/1.6.17-intel-2016.u3
   Autotools/20150119-GCC-6.2.0                         OpenMPI/1.6.5-GCC-4.9.2
```

Loading a particular module requires two commands: `module load <module name>` (to load the module into the node's environment) and `<module name>` (to start the program). To load R for example:

```
[davidpw@spartan-rc017 ~]$ module load R
[davidpw@spartan-rc017 ~]$ R

R version 3.2.1 (2015-06-18) -- "World-Famous Astronaut"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-unknown-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> a <- 1
> b <- 2
> a + b
[1] 3
>
```

---

sbatch

The `sbatch` command is used for direct job submission to Spartan using the batch system called SLURM (Simple Linux Utility for Resource Management).

This system tracks resources throughout the cluster and builds a queue of jobs to run, and when resources are available the job scheduler will direct your job to a compute node to run.

```
[davidpw@spartan ~]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
            591761  physical   kSST74   rchiew CG 5-15:13:31      1 spartan-bm022
            608964  physical av1324_2 guttmann PD       0:00     12 (Resources)
            608965  physical av1324_2 guttmann PD       0:00     12 (Priority)
            493325 punim0095 LdBPK_24    bjpop PD       0:00      1 (launch failed requeued held)
            493328 punim0095 ENSG0000    bjpop PD       0:00      1 (launch failed requeued held)
            493341 punim0095 ENSG0000    bjpop PD       0:00      1 (launch failed requeued held)
            493343 punim0095 ENSG0000    bjpop PD       0:00      1 (launch failed requeued held)
            602302     cloud matlab_b  davidpw  R 1-23:40:57      1 spartan-rc047
            601462     cloud trimer.p    cgao2  R 2-10:59:19      1 spartan-rc003
            602222     cloud triazine    cgao2  R 2-01:16:14      1 spartan-rc025
            607520     cloud  Full.sh    nread  R   22:33:10      1 spartan-rc007
            609350  physical matlab_b  davidpw  R      19:56      1 spartan-bm021
            608772  physical Precompu    mitch  R   18:02:48      1 spartan-bm015
            597912     cloud  IC_1_12   zhuol4  R 5-09:11:04      1 spartan-rc009
            601852     cloud ITB_1_12   zhuol4  R 2-14:58:48      1 spartan-rc057
            594731  physical N2_O2_gn  kcatani  R 6-16:13:10      1 spartan-bm018
            601804  physical NO_NO_ra  kcatani  R 2-18:16:49      1 spartan-bm001
```

To submit a job using the `sbatch` command you need to write a slurm file that sets up your instance (amount of memory, number of processors, etc) and runs your model. Note for Windows users: write these in Notepad++ as Windows and Linux use different line break notation (/r/n vs /n) and standard text files written on Windows won't run on Spartan.

C:\Users\David\Dropbox\JSDM\JSDM Comparison Compiled\Boab model runs\matlab_butterfly.slurm - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

matlab_butterfly.slurm

```
#!/bin/bash
#SBATCH -p cloud
#SBATCH --time=21-00
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8
#SBATCH --mem 51200
#SBATCH --mail-user=davidpw@student.unimelb.edu.au
#SBATCH --mail-type=ALL
module load MATLAB
matlab
cd Ovaskainen\ 2016\ -\ Butterfly/source\ code/
matlab -r "run('A0_master_script.m')"
```

Slurm files can be as simple or as complex as required depending on the job you want to run. Slurm files always need to begin with `#!/bin/bash` on the first line. Why? It tells the shell what kind of interpretor to run (in this case bash). On subsequent lines you set up your instance first using `#SBATCH <your command here>` and then the commands to run your job. Some useful `#SBATCH` commands to consider are:

- `#SBATCH --job-name=<name>`: Lets you give your job a name alongside its job id. Useful if you're running lots of jobs at once. By default it sets it to your file name, or replace with your name of choice like this:

```
#SBATCH --job-name=MyJob
```

- `#SBATCH -p <partition>`: This is where you select which partition on Spartan your job will run. For most jobs you will set your partition as `cloud` which can run single node jobs of up to 8 CPUs and up to 50GB of memory. For multi-node jobs or >50GB of memory set it as `physical` (up to 200GB), and if you have access to a dedicated partition then use `your partition name`. In most cases you will use the following:

```
#SBATCH -p cloud
```

- `#SBATCH --time=<>`: As Spartan is a communal resource and jobs are allocated a share from a queue you need to specify a maximum amount of walltime that you want your instance to remain open. As you aren't likely to know how long your model will need to run for (outside of a rough guess) it is recommended that you give a conservative estimate. If necessary you can contact Spartan support and get your time extended. There are multiple formats for entering a time value depending on the scale of your job: "minutes", "minutes:seconds", "hours:minutes:seconds", "days-hours", "days-hours:minutes" and "days-hours:minutes:seconds". Many SLURM documentations will list setting `--time=0` as a way to set an indefinite walltime but this will automatically be rejected by Spartan. For example, a one hour instance could be called with the following:

```
#SBATCH --time=01:00:00   # hours:minutes:seconds format
```

- `#SBATCH --nodes=<number>`: You need to request an allocation of compute nodes. Most jobs will be single node jobs, but there is the ability to run jobs over multiple nodes that talk to each other. It is not recommended to try running multiple communicating nodes via the cloud partition, use the physical partition instead. To call a single node use the following:

```
#SBATCH --nodes=1
```

- `#SBATCH --ntasks=<number>`: This line informs the SLURM controller that job steps within the allocation will launch a maximum of *number* tasks and to provide sufficient resources. Most jobs will need to a perform a single task which can be set as follows:

```
#SBATCH --ntasks=1
```

- `#SBATCH --cpus-per-task=<number>`: This informs the SLURM controller that each task will need *number* of processors per task. Cloud nodes have access to eight cores each. To allocate four processors (like on a quad-core desktop without multi-threading) you would set:

```
#SBATCH --cpus-per-task=4
```

- `#SBATCH --mem=<number>`: This is where you nominate the maximum amount of memory required per node (in megabytes). Cloud nodes have access to up to 50GB of memory, physical nodes are used for large jobs of up to 200GB. To request 10GB of memory (remembering that 1GB = 1024MB) you would use:

```
#SBATCH --mem=10240
```

- `#SBATCH --mail`: The final group of useful `sbatch` commands that you would regularly use sets up email notifications of various events during job submission/running. There are two separate commands here: `--mail-user=<>` to set who gets notified, and `--mail-type=<>` to chooses what you get notified about. Some useful mail options include:
  - `BEGIN`: the model is out of the queue and started to run (includes start time and time in queue)
  - `END`: the model has completed (includes run-time)
  - `FAIL`: the model has failed (includes run-time)
  - `REQUEUE`: the model has been re-queued (i.e. someone with priority has had you booted off)
  - `ALL`: all of the above plus `STAGE_OUT`
  - `TIME_LIMIT_50`: reached 50% of your allocated time
  - `TIME_LIMIT_80`: reached 80% of your allocated time
  - `TIME_LIMIT_90`: reached 90% of your allocated time
- You can string multiple notifications types together by separating them with commas, so you could do something like the following:

```
#SBATCH --mail-user=<your email here>
#SBATCH --mail-type=ALL,TIME_LIMIT_90
```

After setting up your instance you then need to supply the instructions for what you want your job to do. This will often just be a change of directory, loading the module (program) you need, and calling the script you want to run. Calling the script you want to run follows a set format of commands: [options] [< infile] [> out-file].The first two of these are covered earlier, so lets look at complete example for R before looking at options in more details:

```
cd Coding\ Club/R/
module load R
R --vanilla < tutorial.R
```

As you can see this merges the second command in loading a module with loading a script. You can see the different options available for calling an R script using `man R` (after loading the module into the environment). Some useful options include:

- `--save`: Save workspace at the end of the session
- `--no-save`: Don't save workspace at the end of the session
- `--vanilla`: A wrapper around `--no-save` and a few other commands useful for starting R as a blank slate (no loading pre-saved objects, etc)

`< infile` is where you call the script you want to run e.g. `< myFile.R`, and `> outfile` presumably lets you set specific output files (it isn't compulsory, and I've never had reason to use it).

Now we can put all of this together to create our SLURM file:

```
#!/bin/bash

#SBATCH --job-name=Coding_Club_Example
#SBATCH -p cloud

#SBATCH --time=1:00:00

#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1

#SBATCH --mem=10240

#SBATCH --mail-user="davidpw@student.unimelb.edu.au"
#SBATCH --mail-type=ALL

module load R

R --vanilla < tutorial.R
```

Assuming this Slurm file is named "MyFile.slurm", you can submit your job to the queue from the log-in node using the following command

```
sbatch MyFile.slurm
```

```
[davidpw@spartan ~]$ sbatch matlab_birds.slurm
Submitted batch job 613336
[davidpw@spartan ~]$ sbatch matlab_butterfly.slurm
Submitted batch job 613337
[davidpw@spartan ~]$ squeue -u davidpw
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
            613337  physical Butterfl  davidpw PD       0:00      1 (Resources)
            613336  physical Bird300G  davidpw  R       0:13      1 spartan-bm023
```

Useful Auxiliary Commands

Now that we know how to submit jobs to Spartan either through `sinteractive` or `sbatch`, lets look at some other useful commands in Spartan. Spartan uses a Unix-based operating system so a lot of Unix xommands work here. These commands range from help functions, in built text-editors, checking job status, or job control. Some of these may have been used earlier in this document but are important enough for another mention:

- `man <function name>`: This is Spartan's help function. For example, `man Spartan` will call the Spartan FAQ, `man sbatch` will call the help file for the `sbatch` command, and `man R` will call the help file for running R in Spartan.

```
[davidpw@spartan ~]$ man spartan
Spartan(8)                                                          Spartan Manual

NAME
       Spartan - A High Performance Computer and Cloud hybrid system

DESCRIPTION
       Spartan is high performance computing (HPC) and research cloud (Melbourne Research Cloud, MRC), with

WHAT IS SPARTAN?
       Spartan consists of

       (a) a management node for system administrators,

       (b) a login node for users to connect to the system and submit jobs,

       (c) a small number of 'bare metal' compute nodes for multinode tasks,

       (d) any 'bare metal' user-procured hardware (e.g., departmental nodes),

       (e) vHPC cloud compute nodes for overflow and GPGPU tasks, and

       (f) general cloud compute nodes.

       The  aim of the University is to provide a more unified experience for researchers to access compute
       cialised processing for graphics and imaging (General Purpose Graphic Processing Units).

WHY DO I NEED IT?
       There are a number of common reasons why a researcher may find that a standard user computer (desktop
       tasks.   They  may  find that the tasks they're running are taking too long, or there's not enough co
       too difficult to install, or that it's inefficient to purchase licenses for each user.
```

- `pwd`: Print working directory
- `ls`: List all files/directories in current directory
- `cd <file path>`: Change current directory. `cd` sets you to your home directory, `cd -` goes to previous directory, `cd ..` goes back one step towards root
- `rm <filename>`: Delete a file
- `mkdir <folder name>`: Create a new folder in current directory
- `rmdir <folder name>`: Delete a folder (must be empty)
- `nano <filename>`: This is Spartan's in-built text editor. This can be used for on the fly alterations to a file (i.e. increase memory limit), but I use it most often for reading the slurm output files for checking why a model run failed.

```
  GNU nano 2.3.1            File: matlab_birds.slurm

#!/bin/bash

#SBATCH --job-name=Bird300GB
#SBATCH -p physical

#SBATCH --time=28-00
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8
#SBATCH --mem 204800

#SBATCH --mail-user=davidpw@student.unimelb.edu.au
#SBATCH --mail-type=ALL,TIME_LIMIT_50,TIME_LIMIT_80,TIME_LIMIT_90

cd Ovaskainen\ 2016\ -\ Birds/source\ code/

module load MATLAB

matlab -nodesktop -nodisplay -nosplash< A0_master_script.m



                              [ Read 19 lines ]
^G Get Help   ^O WriteOut   ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit       ^J Justify    ^W Where Is   ^V Next Page  ^U UnCut Text ^T To Spell
```

- `head <filename>` and `tail <filename>`: Print the first/last ten lines of a file
- `history`: List commands you've used previously. You can navigate previous commands in your current session using the up/down keys like in R, but this lists previous commands over previous sessions as well
- `echo <text>`: Prints text. Useful for debugging
- `squeue`: This command is used for checking on job status.
  - `squeue`: This base command will show all jobs in the queue for all users
  - `squeue -u <username>`: This will show all jobs for a particular user
  - `squeue -u <username> -t RUNNING`: This will show all running jobs for a particular user (can also use `PENDING`)

```
[davidpw@spartan ~]$ squeue -u davidpw
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
            613336  physical Bird300G  davidpw  R      45:02      1 spartan-bm023
            613337  physical Butterfl  davidpw  R      25:30      1 spartan-bm016
[davidpw@spartan ~]$
```

- `scancel`: This command is used for cancelling jobs in the queue. For example:

  - `scancel -u <username>`: cancels all jobs for that username
  - `scancel -u <jobid>`: cancels the job called by
  - `scancel --name <JobName>`: cancels jobs by name
  - `scancel -t PENDING -u <username>`: This cancels all pending jobs for a particular user

- `sstat`: This command is used to show memory information of running jobs:
  - For non-admin users there is no need to specify a username as you are restricted to your own jobs only by default
  - You can use the `--format` option to specify the details you want to see. You can list multiple comma-separated fields to view more details
  - `sstat --helpformat` will list the available fields to view
  - Browsing `man sstat` will let you see descriptions for each field
  - For example: `sstat -j <jobid> --format JobID,NTasks,Nodelist,MaxRSS,MaxVMSize,AveRSS,AveVMSize`

- `sacct`: This command is used to show memory information of completed jobs:
  - For non-admin users there is no need to specify a username as you are restricted to your own jobs only by default
  - You can use the `--format` option to specify the details you want to see. You can list multiple comma-separated fields to view more details
  - `sacct --helpformat` will list the available fields to view (many more options that `sstat`)
  - Browsing `man sacct` will let you see descriptions for each field
  - For example: `sacct -j <jobid> --format JobID,jobname,NTasks,nodelist,MaxRSS,MaxVMSize,AveRSS,Ave`

Additional information and links

- There are simple example slurm files for the most common modules at `/usr/local/common/` (including R, MATLAB, and Python)
- Online Spartan FAQ. Same as using `man spartan`
- Online SLURM FAQ
- `srun` and `salloc` functions are used for parallel jobs
- Support for multi-core/multi-threaded architecture
- Running R in parallel using the rslurm package
