# Exercise 1: Logging in

These practical exercises introduce new users to the HPC facility by demonstrating how to connect, log on and off the system; and to become familiar with simple UNIX commands.

1. Download MobaXterm.
2. Configure your session on the cluster.
3. Enable the “Keepalive” option.
4. Log in with your account (you can do double click on the created session or use the terminal).
5. Get familiar with basic commands: cd, ls, pwd, passwd.
6. Create a text file in your home directory with a text editor.
7. Log off.

# Exercise 1: NOTES

A Windows machine requires several pieces of software to be present in order to interface fully with the cluster. These are:

* a secure shell client, e.g. MobaXTerm, PuTTY or a Terminal client if you are a Mac or Linux user. This component allows to establish a secure connection between our local machine and the cluster. We will use this connection to send commands to the cluster.
* a secure shell file transfer client e.g CyberDuck, FileZilla or WinSCP. This component is required to transfer files between the cluster and our local machine.
* an X Windows server, e.g. Xming, X11 or Cygwin-X. This is built in to the MobaXTerm client. This component allows us to visualize windows and graphic user interfaces in our local machine from applications that are running on the cluster.

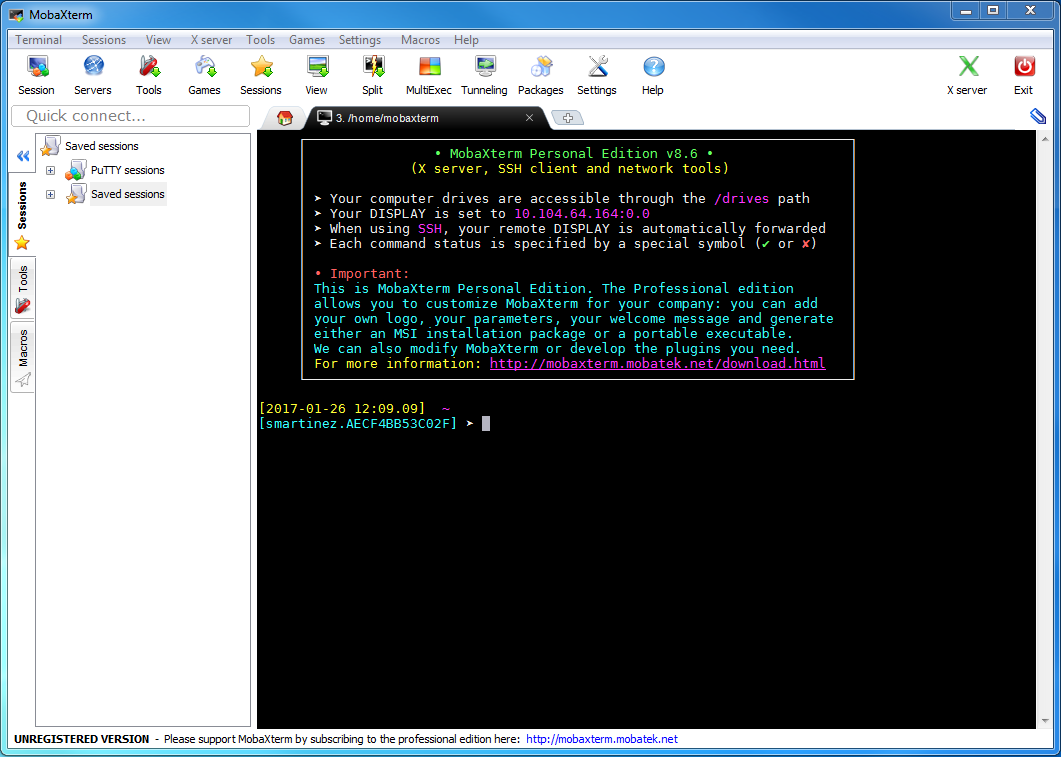
A single application, [MobaXterm](http://mobaxterm.mobatek.net/), combines all of these elements into a single software package.

Linux and MacOS machines generally have the required software already installed. For Linux and MacOS machines you simply need to launch the “Terminal” application. On MacOS this is located in the “Utilities” folder under the “Applications” folder.

The latest versions of Mac OS do not have an X11 client installed so this will need to be downloaded and installed separately. X11 for Mac can be obtained from the [XQuartz](http://www.xquartz.org/index.html). This is a dmg file so will prompt you with the correct instructions for installation.

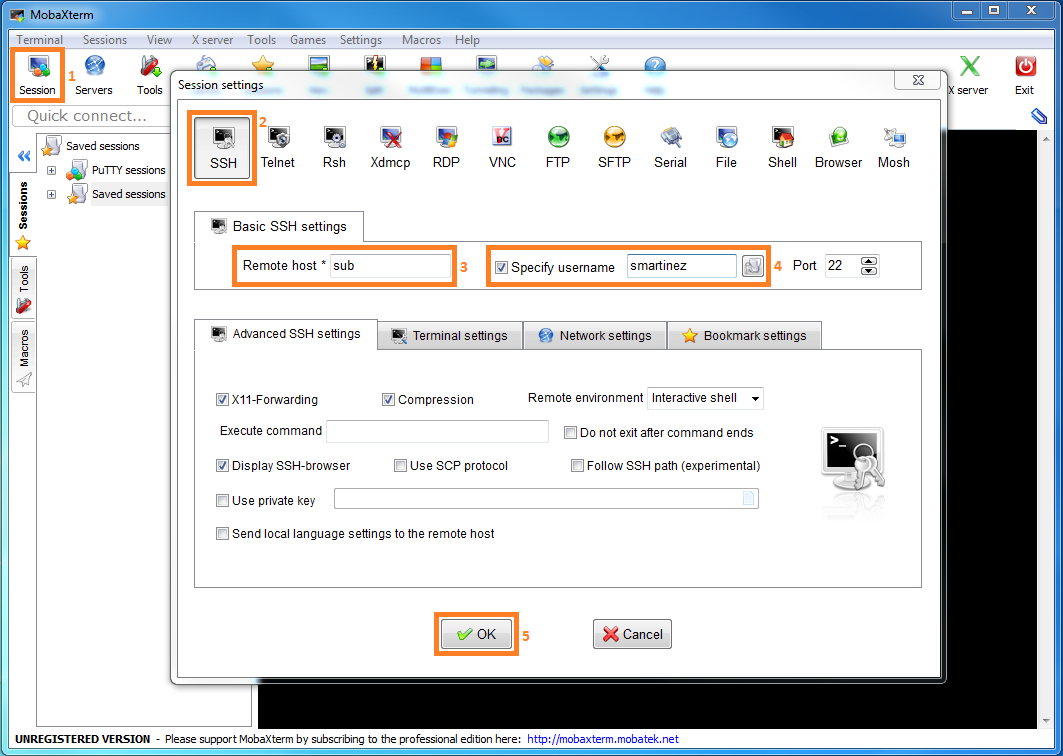
## MobaXterm (Windows)

Download and install MobaXterm from this [link](http://mobaxterm.mobatek.net/download-home-edition.html). No admin rights are needed.



Multiple sessions can be saved in MobaXterm in order to make it easier to connect with different machines. Here, you will need to set up a new session to connect with the MI HPC, via on of the submit nodes, named “sub”.

1. Click on Session button
2. Choose SSH (Secure Shell) connection type.
3. In “Remote host” type sub.
4. Mark “Specify username” box and type your username.
5. Click OK



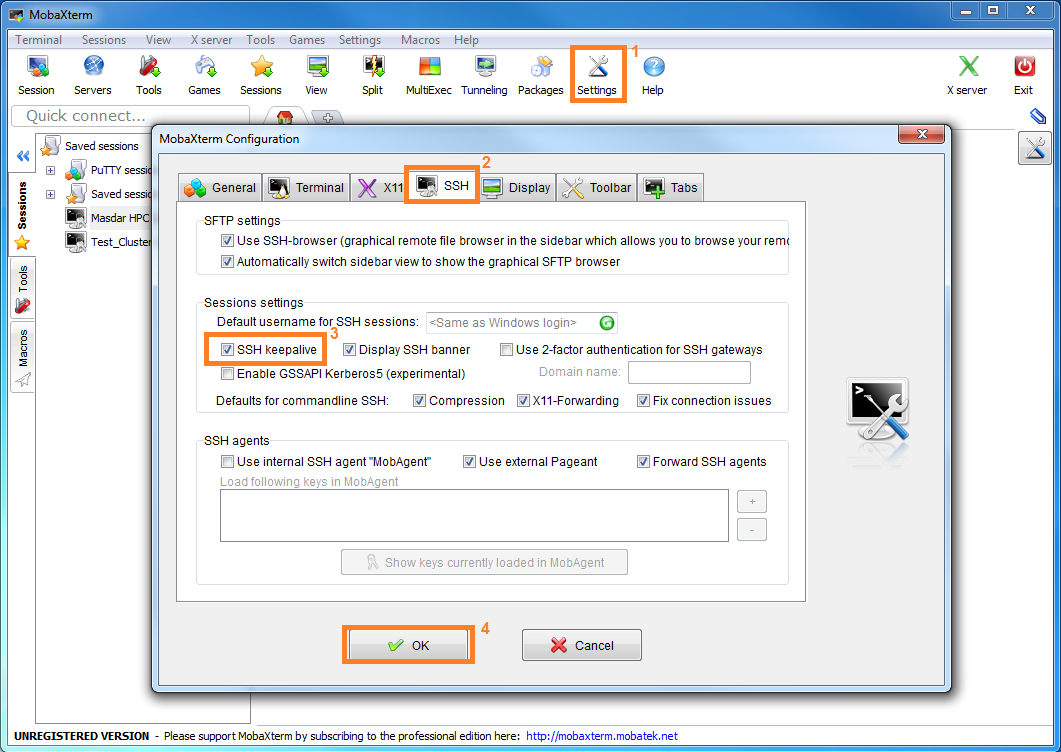
Now you can connect with the HPC cluster by double clicking on the session just created. You will be prompted to enter your password. You will then asked whether you would like to save the password (not recommended).

It is a good habit to configure the client to send a “Keepalive” message to the server once in a while. This will prevent your session from being closed due to inactivity.

In Linux, you can edit the file /etc/ssh/ssh\_config and add a new line with the text ServerAliveInterval 180

In MobaXterm:

1. Click on "Settings".
2. Select "SSH" tab.
3. Enable "SSH keepalive" feature.
4. Click "OK".



## Command line: Linux terminal or MobaXterm

To connect to Masdar HPC cluster, on the Linux terminal or on MobaXterm command window issue the command:

$ ssh -Y <username>@sub

NOTE: throughout this document a $ sign will be used to denote the command prompt, which usually takes the form [user@machine ~]$ , and so you do not need to enter it at the command line.

Where <username> is your training session id. You will be prompted to enter your password. MobaXterm will then ask whether you would like to save the password (not recommended).

## File editors

There are several editors available on HPC cluster, such vi, nano or sublime. Alternatively, MobaXTerm has its own built-in text editor. To use this, simply right-click the file you want to edit in the left hand file navigator pane within MobaXTerm and choose 'Open with default text editor'.

## Logging off

To log off the system enter the following at the command prompt:

$ logout

Once you have logged out, the file transfer window will close and you will be presented with a prompt on the local PC.

# Exercise 2: File transfer

This exercise demonstrates to new HPC users how files can be transferred between different systems:

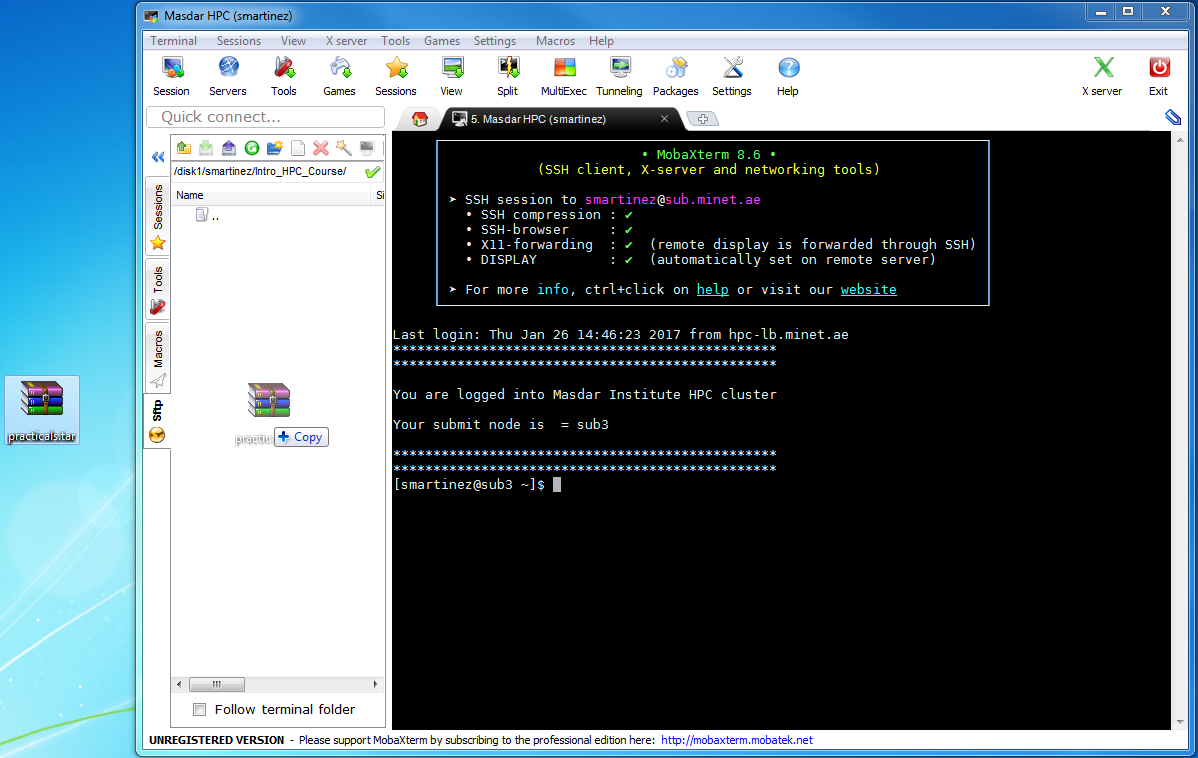
1. Download the practical exercise files to your local machine, on your Desktop for example: [Link](https://midrive.masdar.ac.ae/no-shib/index.php/s/BomPDrCb2UwwZbc).
2. Transfer the files to your home directory on the cluster (You might need to log in first)
3. Unpack the files

Try to transfer the files using different methods: MobaXterm inbuilt file transfer capability, scp, rsync.

# Exercise 2: Notes

## MobaXterm (Windows)

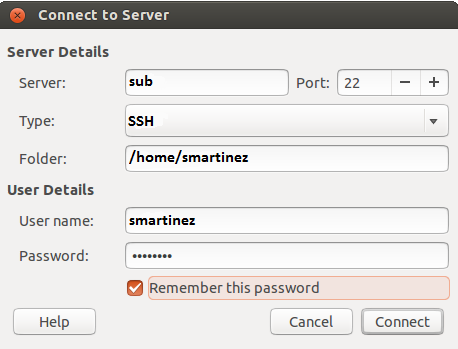
MobaXterm has an inbuilt file transfer capability. You should be able to the drag the file from your Desktop into the window labelled "SFTP" on MobaXterm. This window will list all the files that are on your HPC home directory. Once you have dragged the file across, this window should show the file practicals.tar.gz.



## Ubuntu GUI

You can use Ubuntu to transfer the files using a graphic interface too.

* Open your folder
* File → Connect to server…
* Fill with your credentials



Now, by moving files to this folder, they will be transferred to the HPC cluster.

## Command line: Linux Terminal or MobaXterm

Alternatively, you can transfer files using the command line:

### File transfer via scp:

From local machine to upload a file:

$ scp <file> <username>@sub:<directory>

$ scp –r <directory> <username>@sub:<directory>

From HPC to transfer a file back:

$ scp <file> <localhostusername@<localmachine>:<directory>

$ scp –r <directory> <localhostusername@<localmachine>:<directory>

### File transfer via rsync:

From local machine to upload a file:

$ rsync –v –e ssh <file> <username>@sub:<directory>

$ rsync –v –e ssh –r <directory> <username>@sub:<directory>

From HPC to transfer a file back:

$ rsync –v –e ssh <file> <localhostusername@<localmachine>:<directory>

$ rsync –v –e ssh –r <directory> <localhostusername@<localmachine>:<directory>

## Unpacking the practical exercises

You will notice that the practical exercises file you have transferred to HPC cluster has two file extensions: **tar** and **gz**. This means that it is a compressed (the gz bit) archive (the tar bit- tar is short for 'tape archiving' as all the files have been combined into a single archive).

To unpack the practical files enter the following into the command line:

$ tar -zxvf practicals.tar.gz

You will note that the tar instruction has been passed four parameters: zxvf

* z= unzip
* x= extract archive
* v= verbose output (so we know what is happening)
* f= unpack the filename that follows

The practical exercises should be unpacked into the directory $HOME/, where $HOME is your home directory on the cluster.

# Exercise 3: Modular Environment

This exercise introduces the modular environment to the new HPC users.

1. Check which modules are currently loaded in your environment.
2. Try to open Matlab without loading its module. Use the command which matlab to see where Matlab executable is located.
3. Check the list of modules available in the system.
4. Load the latest version of Matlab available in the system.
5. Check that the module was loaded in your environment. Use also which matlab to see which version of Matlab you are using and its path.
6. Switch Matlab module to a different version.
7. Check the modules loaded in your environment.

# Exercise 3: Notes

On a complex computer system, on which it is necessary to make available a wide choice of software packages in multiple versions, it can be quite difficult to set up the user environment so as to always find the required executables and libraries. (This is particularly true where different implementations or versions use the same names for files). Environment modules provide a way to selectively activate and deactivate modifications to the user environment which allow particular packages and versions to be found. The most commonly used options are:

|  |  |  |
| --- | --- | --- |
| Command | Example | Description |
| module list | module list | Lists your currently loaded modules. |
| module avail | module avail | Lists all available modules in the system. |
| module load <module\_name> | module load matlab/R2015b | Loads the module <module\_name>. |
| module switch <old\_mod> <new\_mod> | module switch matlab/R2015a matlab/R2015b | Switches similar/conflicting modules. |

# Exercise 4: Platform LSF

This exercise introduces to new HPC users to the cluster scheduler (also called workload manager) practicing some important commands to check the status of the cluster.

Before we start submitting jobs to the cluster it is a good habit to check the status of it. That is, to check how many nodes we have available and how the different queues are being utilized. Connect with the cluster and perform the next operations using some of the above commands.

1. Check how many nodes has the cluster available to execute jobs on.
2. Check how many different queues are implemented in the system.
3. Check the list of nodes that can be used for each queue.
4. Check the list of jobs submitted to the system.

# Exercise 4: NOTES

Platform Load Sharing Facility (or simply LSF) is a workload management platform, [job scheduler](https://en.wikipedia.org/wiki/Job_scheduler), for distributed [HPC](https://en.wikipedia.org/wiki/High_Performance_Computing) environments. It is based on a queues system and it can be used to execute batch and interactive jobs on networked Unix and Windows systems on many different architectures. LSF provides some commands to submit jobs to the queues, monitor the status of the jobs and queues, and cancel/kill the jobs. Below are listed some important commands along with a brief description:

|  |  |  |
| --- | --- | --- |
| Command | Example | Description |
| bsub | bsub < myscript.sh | Submits a job to the queue. |
| bhosts | bhosts -l | Displays hosts and their static and dynamic resources. |
| bqueues | bqueues -l | Displays information about the queues. |
| bjobs | bjobs –u all | Displays information about jobs. |
| bkill | bkill 10254 | Sends a signal to a job (kill the job). |
| bhist | bhist –l 10254 | Displays historical information about jobs. |

# Exercise 5: Job submission methods

A batch job is a program that runs in the background and without any user intervention. This type of jobs are the most commonly used in a high performance computer environment. The user submits them and waits for the results.

With this exercise, we will practice different methods for batch job submission:

Change the directory to “practicals/command\_line\_submission”:

1. Submit the circle\_area.py program through the command line. Specify at least the number of cores to use, the queue name, a name for the job, output filename and error filename.
2. Check the status of the job.
3. Check output and error files when finished.
4. Repeat the process changing the radius value, the name of the job and the output filenames.

Change the directory to “practicals/config\_script\_submission”:

1. Edit config.sh script to submit circle\_area.py program as a job. Include at least the number of cores to use, the queue name, a name for the job, output filename and error filename.
2. Submit the script with bsub < config.sh.
3. Check the status of the job.
4. Check output and error files when finished.
5. Repeat the process changing the radius value, the name of the job and the output filenames.

# Exercise 5: Notes

All the computational task (simulations, programs, applications, scripts) have to be submitted to the cluster using the command bsub. This allows the scheduler to optimize the resource usage and monitor them at every moment. By this way, the scheduler will only allow a job to be executed only if there are enough resources available in the system for it. In this sense, the scheduler acts as a load balancer avoiding over use of some resources while other are not being used.

In order to specify the resources that our job needs there are several options that can be added to the bsub command. The next options are the more commonly used:

|  |  |
| --- | --- |
| Option | Description |
| -n <cores> | Requests a number of CPU cores. |
| -q <queue-name> | Submits job to the specified queue. By default this is “normal". |
| -o <file\_name> | Writes output to <file name>, "%J" can be used to include the JOBID, e.g. “result.J%.out“. |
| -e <file\_name> | Writes errors to <file\_name>, "%J" can be used to include the JOBID, e.g. “result.J%.err“. |
| -J <job\_name> | Assigns the specified name "<job\_name>" to the job. |
| -W <HH:MM> | Requests wall time limit of HH hours and MM minutes. |
| -R "rusage[mem=XXX]" | Requests a specific amount of memory for your job. XXX is the memory size in MB. |
| -R "span[ptile=m]" | Requests m CPU cores on each node. |
| -R "span[hosts=1]" | Indicates that all the processors allocated to this job must be on the same host. |
| -I | Submits an interactive job. Use -XF with -I to submit an interactive job using SSH X11 forwarding. |
| -x | Puts the host running your job into exclusive execution mode. |

A job can be submitted on the command-line or via a configuration script file (spooling).

Before we explain them, let’s have a program to submit. As this is not a programming course, we will not go for a complicate code or programming language. Let’s have a code written in python that takes the radius of a circle as an input parameter and computes its circumference and its area. It also includes a delay of 60 seconds that will give us some time to use control job commands. Don’t worry if you don’t know python, It is very easy to understand.

|  |  |  |
| --- | --- | --- |
| Line no. | Code: circle\_area.py | Comments |
| 1  2  3  4  5  6  7  8  9  10  11 | **import** sys, time  **from** math **import** pi  time**.**sleep**(**60**)**  **print** "Radius is = %s" **%** sys**.**argv**[**1**]**  radius**=**float**(**sys**.**argv**[**1**])**  circumference **=** 2**\***pi**\***radius  **print** "Circumference is = %.5f" **%** circumference  area **=** pi **\*** pow**(**radius**,**2**)**  **print** "Area is = %.5f" **%** area | #imports **sys and time** libraries which allows us to use input parameters and sleep.  #imports **Pi** constant from the library **math**.  #sleeps for 60 seconds  #prints on screen the input parameter.  #transform the parameter from text to single precision number.  #computes the length of the circumference.  #prints on screen the value of the circumference.  #computes the area of the circle.  #print on screen the value of the area. |

As you can see, this is not a parallel code and, therefore, it will not need more than one core to be executed. But this is a good example that will allow us to practice with the job submission techniques. We will use the same code in all our exercises. The script can be found inside every folder of the practices that you have uncompressed before, with the name circle\_area.py. If we wanted to execute the program in our laptop or workstation, the way we called it would be: python circle\_area.py r, where r is the radius we want to use. Things are different when we want to use the scheduler in a queue system, additional information is required in order to submit this program as a job to the HPC cluster. There are two ways to perform this:

## Command Line

In its simplest form, a job can be submitted using a command such:

$ bsub -n 1 -q training -J circle\_area -o circle\_area.%J.out -e circle\_area.%J.err "python circle\_area.py 5"

This will run the specified command and write standard output and error to a file with the job number %J in the filename. Note that the command "python circle\_area.py 5" contains spaces, it is a good habit to enclose commands with double quotes.

There are many additional options that can be used to control the resources allocated to an individual job including the queue to submit to, the wall time limit (-W option), and any processor or memory allocation/limitations for your job. This method is not indicated for jobs with complex configurations.

## Configuration Script File (Spooling)

For more complex jobs, running a sequence of commands or requiring more complicated environment configuration, a configuration script can be constructed, such as:

|  |  |  |
| --- | --- | --- |
| Line no. | Code: config.sh | Comments |
| 1  2  3  4  5  6  7 | #BSUB -n 1  #BSUB –q training  #BSUB -J circle\_area  #BSUB -o circle\_area.%J.out  #BSUB -e circle\_area.%J.err  **python** circle\_area.py 5 | #requests 1 CPU core.  #uses training queue.  #sets the name of the job as “circle\_area”.  #writes standard output to “circle\_area.JobID.out”.  #writes error output to “circle\_area.JobID.out”.  #calls the program. |

This bash script contains a series of directives, prefixed with “#BSUB” which provides the same functionality as the command line method.

To submit a job using a job script file we redirect the configuration file to bsub by this way:

$ bsub < config.sh

# Exercise 6: interactive jobs

This exercise introduces to new PC user the interactive jobs submission. This kind of jobs can allow input and output from the terminal as well as open GUIs for some applications.

1. Submit an interactive jobs from the command line that opens the latest version of Matlab available in the system. You might need to load the Matlab module first.  
   Tip: If you want to load the Matlab GUI add the flag -desktop to matlab executable (matlab -desktop). Finish your submission line with the character & in order to get the prompt back and continue using the console.
2. Check the status of the job you just submitted and see whether the requested resources were assigned to the job correctly.
3. Close the Matlab user interface and check whether the job finished it execution.

# Exercise 6: Notes

It is possible to launch a program on the back end machines and interact with it in real time. This however will only be successful if the required resources are available at the time the job is submitted. This kind of jobs are called interactive. They allow input and output from the terminal or using a GUI (e.g. Matlab, Lumerical FDTD). They have the same defaults and constraints as any other job.

To indicate LSF that the job is interactive, we add -I to the list of bsub options and specify a queue that supports interactive jobs. Also, if our job requires X forwarding, we should indicate it with -XF option.

The next example shows how to submit an interactive job to open Lumerical FDTD from the command line to general queue:

$ bsub -I -q training -XF fdtd-solutions

# Exercise 7: Array jobs

Change the directory to “practicals/array\_submission”:

1. Edit config.sh script to submit circle\_area.py program as an array job with 10 elements. Include at least the number of cores to use, the name of the queue, a name for the job, output filename and error filename.
2. Submit the script with bsub < config.sh.
3. Check the status of the job.
4. Check output and error files when finished.
5. Repeat the process limiting the number of jobs that can run simultaneously to 3 and check the execution with bjobs.
6. Repeat the process using a no consecutive sequence of numbers. Check with bjobs.
7. Submit the last array of jobs again and kill only one of the jobs in the array with bkill.
8. Submit the last array of jobs again and kill the whole array with bkill.

# Exercise 7: Notes

Command line and spooling submission methods are suitable when the number of jobs to submit is reduced. Imagine that we want to perform a parameter sweep of our program with 100 different input combinations, then these methods can be hard to implement. The solution is to use array syntax. The next example shows how to configure an array of jobs, comparing it with the previous configuration method:

|  |  |  |
| --- | --- | --- |
| Line no. | config.sh | config\_array.sh |
| 1  2  3  4  5  6  7 | #BSUB -n 1  #BSUB -q training  #BSUB -J circle\_area\_3  #BSUB -o circle\_area\_3.%J.out  #BSUB -e circle\_area\_3.%J.err  **python** circle\_area.py 3 | #BSUB -n 1  #BSUB -q training  #BSUB -J circle\_area[1-6]  #BSUB -o circle\_area.%I.%J.out  #BSUB -e circle\_area.%I.%J.err  **python** circle\_area.py **$LSB\_JOBINDEX** |

* The elements in the array are indicated in the job name: #BSUB -J circle\_area[1-6].
* We can use the character %I to refer the job element: #BSUB -o circle\_area.%I.%J.out.
* When a job array is submitted, each array value is stored in the variable **$LSB\_JOBINDEX**, which can be used as an input to an executable, e.g.: **python** circle\_area.py **$LSB\_JOBINDEX**
* It can be submitted as usual: $ bsub < config\_array.sh

The output of bjobs command for this example after submission would be:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| JOBID | USER | STAT | QUEUE | FROM\_HOST | EXEC\_HOST | JOB\_NAME | SUBMIT\_TIME |
| 93795 | smartinez | RUN | training | sub4 | hpc070 | circle\_area[1] | Nov 29 10:26 |
| 93795 | smartinez | RUN | training | sub4 | hpc034 | circle\_area[2] | Nov 29 10:26 |
| 93795 | smartinez | RUN | training | sub4 | hpc056 | circle\_area[3] | Nov 29 10:26 |
| 93795 | smartinez | RUN | training | sub4 | hpc074 | circle\_area[4] | Nov 29 10:26 |
| 93795 | smartinez | RUN | training | sub4 | hpc027 | circle\_area[5] | Nov 29 10:26 |
| 93795 | smartinez | RUN | training | sub4 | hpc083 | circle\_area[6] | Nov 29 10:26 |

All elements have the same JOBID, but individual ones can be specified using brackets (e.g., 93795[5]).

There is no need to submit “continuous numbers”, we can specify different sequences for the array:

* #BSUB -J circle\_area[1-10] passes values of 1,2,3,4,5,6,7,8,9,10 to **$LSB\_JOBINDEX**
* #BSUB -J circle\_area[1,3-6,42,90] passes values of 1,3,4,5,6,42, and 90 to **$LSB\_JOBINDEX**
* #BSUB -J circle\_area[1,10:2] passes values of 1,3,5,7, and 9 to **$LSB\_JOBINDEX**

## Killing an array of jobs

To kill the whole array, you only need to specify the single job number:

$ bkill 123 kills all jobs in the array

You can also kill a single entry of an array:

$ bkill “123[5]” kills element 5 of the job array

## Limiting the number of jobs that can run

Sometimes it may be desirable to limit how many array members can run simultaneously. This can be limited by adding “%val” at the end of the job name directive.

|  |  |
| --- | --- |
| Line no. | config\_array.sh |
| 1  2  3  4  5  6  7 | #BSUB -n 1  #BSUB –q training  #BSUB -J circle\_area[1-6]%3  #BSUB -o circle\_area.%I.%J.out  #BSUB -e circle\_area.%I.%J.err  **python** circle\_area.py **$LSB\_JOBINDEX** |

In this example, a maximum of 3 jobs will run simultaneously, the remaining will be pending.

# Exercise 8: Array jobs ADVANCED

Change the directory to “practicals/array\_advanced\_submission”:

* Edit config.sh script to submit circle\_area.py program as an array job with 10 elements that reads the input values from a text file radius.txt. Include at least the number of cores to use, the name of the queue, a name for the job, output filename and error filename.
* Submit the script with bsub < config.sh.
* Check the status of the job.
* Check output and error files when finished.

# Exercise 8: Notes

The above example with array jobs works when the radius is an integer number, but what if the values that we want to use are not integers? What if they aren’t even numbers? A good solution is to write down the parameter values in a text file and make every element in the array to take one value. The next example shows how we can submit an array of jobs that read its inputs from a text file radius.txt:

|  |  |  |
| --- | --- | --- |
| Line no. | radius.txt | config\_array.sh |
| 1  2  3  4  5  6  7  8 | 3.7861  8.1158  5.3283  3.5073  9.3900  8.7594  5.5016  3.4997 | #BSUB -n 1  #BSUB -q training  #BSUB -J circle\_area[1-8]  #BSUB -o circle\_area.%I.%J.out  #BSUB -e circle\_area.%I.%J.err  radius**=`head -n$LSB\_JOBINDEX radius.txt | tail -n1`**  **python** circle\_area.py **$radius** |