

Tutorial: Getting Started on the Opportunity

This tutorial is designed for researchers who are new to the Opportunity Linux cluster. It covers basic information about the cluster, as well as how to create and submit batch jobs using the Lava and Sun Grid Engine (SGE). It also contains sample job command files that can be used as templates for running jobs under Lava/SGE.

The Opportunity Linux Cluster at Northeastern University

The Opportunity Linux Cluster is a 65-node, distributed memory multi-processor system. Each node of the cluster contains two Xeon EMT 64 processors with 2048KB of cache (per cpu) and 4 GB of RAM (per node). The nodes are interconnected with Gigabit Ethernet (60-110 Mbytes/sec bandwidth, 50-200 µsecs latency).

The Opportunity Linux cluster uses [ROCKS](#) as its operating systems and Lava/Sun Grid Engine to distribute the computational workload across the nodes. Lava/SGE is a batch job scheduling application that provides the facility for building, submitting and processing batch jobs on the cluster.

Jobs are submitted to the cluster by creating a Lava/SGE job command file that specifies certain attributes of the job, such as how long the job is expected to run and how many nodes of the cluster are needed (e.g. for parallel programs). LAVA/SGE then schedules when the job is to start running on the cluster (based in part on those attributes), runs and monitors the job at the scheduled time, and returns any output to the user once the job completes.

System/Account Policies

Accounts are available to Faculty, Staff and Sponsored students. Faculty/Staff accounts will remain active while the account has shown activity (logged into within the last 6 months) and the user remains employed by the University. Sponsored accounts will be subjected to a renewal process every September/October.

- Each user will be provided with home directory that can grow in size to 2gb.
- Each user will be provided with access to a scratch directory. ***Note: Files left in Scratch longer than one month maybe deleted without notice.***
- Users can submit requests to the ARCUG (Academic Research Computing User Group) for project related disk storage. Provided that space is available, space will be granted to the project for a period of time.
- No user shall create a process (or processes) that will consume more than 16 processors without prior permission from ARCUG (Academic Research Computing User Group)

Is there internet accessible e-mail on the system?

No.

How do I check the status of the Cluster

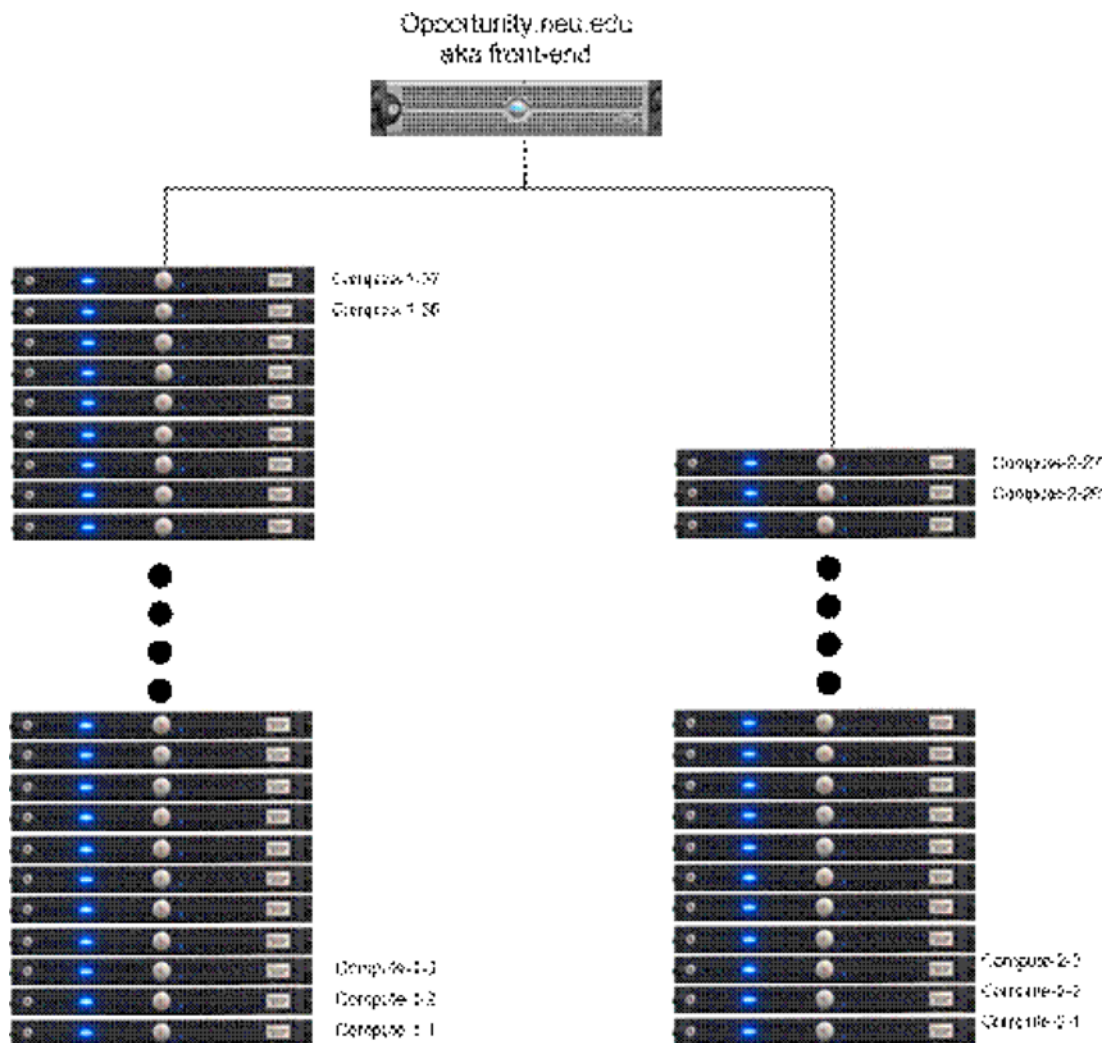
The Cluster has a web monitoring tool. To check the status of the cluster just point your web browser to the following HTTP address:

<http://opportunity.neu.edu/ganglia>

If you would like to check the status of the Lava job queue

<http://opportunity.neu.edu/clumon>

What does the Cluster logically look like



Logging on to the Cluster

Logins to the Linux cluster can be done through the machine **Opportunity.neu.edu** by ssh. This places

you on the head node of the cluster which acts as the control console for any interactive work such as **source code editing, compilation, and submitting jobs through LAVA/SGE**. Using applications such as Matlab or Mathematica interactively should not be done on Opportunity.neu.edu, but rather on other machines.

Important notice for Windows users: do not use a standard Windows editor such as Notepad to edit files that will be used on the Linux or other Unix systems. The two systems use different sequences of control characters to mark the end of line (EOL). If you are using the clusters from a Windows system, there are a number of options:

- Log on to the cluster via SSH or equivalent and use one of the many command-oriented text editors available, such as vi, emacs, or nano.
- If you must use a Windows-only editor, transfer the file and then apply `dos2unix` on the cluster to convert it to the correct Unix format. Type `man dos2unix` for more information about this command.

Getting around on the Compute Nodes

When you ssh into Opportunity.neu.edu you are accessing the front-end node of the cluster. This one node is your gateway to the other 64 computers. All the other computers are named using the following convention:

Compute-***R***-***N*** (where ***R*** is the Rack number and ***N*** is the computer #)

There are 37 computers in rack one. So the names of those compute nodes are

compute-1-1, compute -1-2, compute-1-3..... compute-1-37

There are 27 computers in rack two. The names of those compute nodes are

compute-2-1, compute-2-2, compute-2-3..... compute-2-27

To interactively log into a compute node. First log into opportunity.neu.edu (using ssh) and then ssh into the compute node of choice.

Applications and Tools available on Opportunity

The Opportunity cluster has the following applications/Tools installed:

Application	Where is	How to	How to
-------------	----------	--------	--------

	application installed	launch/Location (Command line)	launch with Xwindows
Java	All nodes	java	no difference
JavaC	All nodes	javac	no difference
Maple	Compute Nodes	maple10	xmaple10
Matlab	Compute Nodes	matlab	matlab
Mathematica	Compute Nodes	math	mathematica
Mpi	All nodes	(Programming library)	
MPICH	All nodes	(Programming library)	
MPICH2	All nodes	(Programming library)	
Sas	Compute Nodes	Not available	sas
Stata	Compute Nodes	stata9s3	xstata9-se
Intel C compiler	All nodes	icc	no difference
Intel C++ Compiler	All nodes		no difference
Intel Fortran Compilers	All nodes	ifort	no difference
Gnu compilers	All nodes	gcc, g77	no difference
Sun Grid Engine	Front End	qsub, qstat, qdel,etc	no difference
Lava	Front End	bsub, bjobs, bkill,etc	no difference
Globus	Front End	See Globus manual	
Neuron	Compute Nodes	/usr/local/neuron/nrn	

Backups

The home and project directories are the only directories that are backed up on the Opportunity Cluster. (eg /home/USERNAME or /projects/). Please do not store critical data on this system. Please copy critical/achieved data over to myFiles.

Accessing myFiles.neu.edu

(File Transfer to and from myFiles and the Cluster)

Cadaver is a WebDAV client which looks a lot like an ftp client. Cadaver is available on most Unix platforms and is already be installed on your machine. The executable is named `cadaver`.

To connect to a WebDAV server using cadaver do the following:

```
>./cadaver https://myfiles.neu.edu/USERNAME/

[root@opportunity root]# cadaver https://myfiles.neu.edu/l.hill
Authentication required for myfiles.neu.edu on server `myfiles.neu.edu':
Username: l.hill
Password: *****
```

Most commonly used commands:

`mkdir`, `cd`, `ls`, `get`, `put`, `mget`, `mput`, `lcd`, `lls`, `lpwd`, `pwd`

All available commands:

```
dav:/l.hill/> help
Available commands:
ls          cd          pwd          put          get          mget         mput
edit        less        mkcol        cat          delete       rmcol        copy
move        lock        unlock       discover     steal        showlocks    version
checkin     checkout   uncheckout   history      label        propnames    chexec
propget     propdel    propset      search       set          open         close
echo        quit        unset        lcd          lls         lpwd         logout
help        describe   about
Aliases: rm=delete, mkdir=mkcol, mv=move, cp=copy, more=less, quit=exit=bye
```

Use `ls` to list the files and directories.

Use `cd` changes to a subdirectory. In cadaver, directories are called Collections, or 'Coll'.

```
dav:/l.hill/deed/> cd MyDev
dav:/l.hill/deed/>ls
Coll: System                                0 Oct 5 2001
```

move', 'copy' and 'rm' work the same and for other commands see 'help'.

Compilers

Programs for which the user has written the source code must first be compiled on Opportunity.neu.edu to run on the cluster. Currently, two sets of compilers are supported on Opportunity: the Intel, and the Gnu compilers. Intel offers C, C++, and Fortran 95 compilers; the Gnu compilers include C, C++, and Fortran 77.

The Intel compilers are licensed by ITC to run on Linux platforms at the University. The Intel compilers available on the Opportunity Cluster are:

```
icc [options] file.c                (C)
icc [options] file.C file.cc file.cpp (C++)
ifc [options] file.f                (Fortran 77)
ifc [options] file.f90              (Fortran 90/95)
```

For a complete list of options consult the relevant compiler man page, e.g. `man ifc` on Opportunity. More detailed information about the Intel compilers can be found in the documentation on the [Fortran](#) and [C/C++](#) Web pages.

To compile parallel programs, the open source MPI (Message Passing Interface) libraries [MPICH](#) have been provided.

The Intel version of the MPI tools has been installed into directory

```
/opt/intel_mpi_10
```

The tools included:

```
mpicc [options] file.c           (C)
mpiicpc [options] file.C         (C++)
mpif77 [options] file.f          (Fortran 77)
mpiifort [options] file.f        (Fortran 90)
```

Local and Global Disk Scratch locations

****** Please note these areas are NOT backed up ********

The system has been configured to have both local and global scratch disk locations for the temporary storage of non-critical data. This global scratch space “/scratch_global” is an NSF mounted file share from the front end to the rest of the cluster. The local scratch space “/scratch_local” is unique disk space residing on the individual compute node (hence the term local)

Policies regarding the size and longevity of files on these file systems is currently being addressed by the User advisory committee and this area will be updated accordingly.

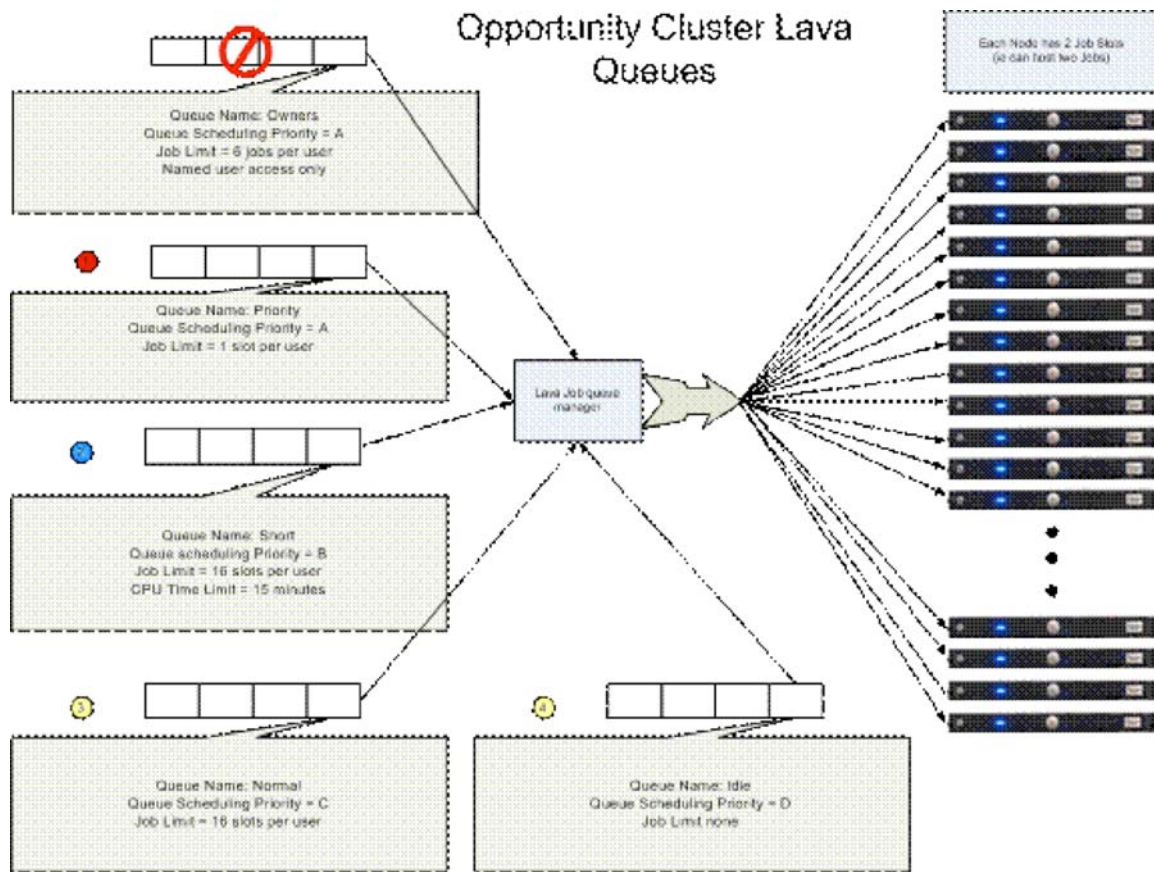
Quick overview of Lava Job Queue

Platform Lava is an entry-level workload management solution embedded in Platform Rocks. Platform Lava is a critical component for the day to day management of the whole cluster and provides commercial grade job execution, management and accounting in a simple easy to use package.

Lava's workload management capabilities improve cluster resource utilization by active monitoring of CPU, memory, bus and network resources.

Platform Lava's interface is compatible with Platform LSF® and Platform LSF® HPC, providing customers with a clear migration path from cluster workload management to enterprise workload management and enterprise grid deployments.

The logical configuration of the queues is depicted below.



The Lava Queues are as follows:

Queue Name	Description	Status	# of Jobs per user	Scheduling priority
Owner	High Priority queue for node owners	Not Available		
Priority	General High Priority Queue	Operational	1	A
short	For jobs using less than 15 minutes cpu time	Operational	16	B
normal	Default Queue	Operational	16	C
idle	Queue for submitting lots of jobs	Operational	Unlimited	D

(extracts from the Lava user's manual)

(ftp://ftp.platform.com/distrib/3.3.0-1.1/enterprise/lava_using_6.1.pdf)

How to submit a job

Any command or script you can execute from a shell prompt can be submitted to Lava for batch execution.

To submit a script to Lava:

1 Create a script. For example, create the following script and save it as `myscript`.

```
#!/bin/csh
#BSUB -q normal
#BSUB -o outfile
myjob arg1 arg2
#BSUB -J myjob
^D
```

2 Make the script executable. For example:

```
$ chmod u+x myscript
```

3 Submit the script to Lava:

```
$ bsub < myscript
```

Job <1234> is submitted to queue <normal>.

Submitting a Matlab job

We want to run a Matlab program in a job. The Matlab commands are in the file 'matlab.in', and we estimate it will take about 50000 CPU-seconds (834 minutes) for the program to complete. As it is not too urgent, we will tell LSF to start the job next Friday evening (2nd of next month in our case) at 10 PM. The job queue will be XL (extra large) as our CPU time is too high for any other queue. We will put our job instructions in the file `matlab.job`. **Important:** if you run a matlab program in background or a job, **add a quit command at the end of your matlab instructions**; otherwise your matlab program will start looping until a time-limit is exceeded.

[Contents of the batch file `matlab.job`](#)

```
#!/bin/csh
#BSUB -q normal           # Job queue
#BSUB -c 834             # Time limit in minutes
#BSUB -M 128000          # Memory limit in Kbytes
#BSUB -o matlab.out      # Output file for Matlab
#BSUB -e Matlab.err      # Output file for errors
#BSUB -J My-Matlab       # name of the job
#BSUB -b "2:22:00"       # start time(2nd of next month,@10PM)
```

```
# setting the graphical environment display
# this is OPTIONAL; do not use it if you do not want graphical output
# on some Xterminal
# setenv DISPLAY my-xterminal:0

# And run our Matlab program; we will also 'time' it
# output from the matlab instruction will be matlab.out
#
time matlab < matlab.in
```

Sending the job to LSF ...

We simply enter the command

```
bsub < matlab.job
```

When the job is launched correctly, the system will reply with something similar to

```
Job <100398> is submitted to queue <normal>
```

... and following its progress

To follow the progress of your job, you can use the command `bjobs`, like in [bjobs -u all](#).

Or you can use the command `bpeek` to examine the current output of a specific job. Once the job is finished, it will disappear from the RUN queue. You will find the result in the file *matlab.out*.

Submitting a Mathematica job

Purpose of the job

We will be running a Mathematica job, I want the results fast, I'm going to run it in short queue as it only is a "short" job and normally would have been run interactively. I'm putting the job instructions in the file *math.job*.

Contents of the batch file *math.job*

```
#!/bin/csh
#BSUB -q short                # Job queue
#BSUB -c 10                   # Time limit in minutes
#BSUB -o math.out             # save the little output we have in math.out
#BSUB -J Mathematica          # name of the job

# My instructions
# Changing to the directory where my program is
cd
cd math-dir

# setting the graphical environment display
# this is OPTIONAL; do not use it if you do
```

```
# not want graphical output on some Xterminal
# setenv DISPLAY my-xterminal:0

# And run our Mathematica program; output will be on stdout
math < math.infile
```

Sending the job to LSF ...

We simply enter the command

```
bsub < math.job
```

When the job is launched correctly, the system will reply with something similar to

```
Job <100398> is submitted to queue <short>
```

... and following its progress

To follow the progress of your job, you can use the command `bjobs`, like in [bjobs -u all](#). Or you can use the command `bpeek` to examine the current output of a specific job. Once the job is finished, it will disappear from the RUN queue. You will find the result in the file `matlab.out`.

Killing a job

The `bkill` command cancels pending batch jobs and sends signals to running jobs. By default, `bkill` sends the SIGKILL signal to running jobs. Before SIGKILL is sent, SIGINT and SIGTERM are sent to give the job a chance to catch the signals and clean up. The signals are forwarded from `mbatchd` to `sbatchd`, which waits for the job to exit before reporting the status. Because of these delays, for a short period of time after entering the `bkill` command, `bjobs` may still report that the job is running. (For descriptions of `mbatchd` and `sbatchd`, see *Inside Platform Lava*.)

Example To kill job 3421:

```
$ bkill 3421
```

```
Job <3421> is being terminated
```

Requeuing a job

You can use `brequeue` to kill a job and requeue it. When the job is requeued, it is assigned the PEND status and the job's new position in the queue is after other jobs of the same priority.

- You can only use `brequeue` on running (RUN), user-suspended (USUSP), or system-suspended (SSUSP) jobs.

- Users can only requeue their own jobs. Only root and Lava administrator can requeue jobs submitted by other users.
- You cannot use brequeue on interactive batch jobs
-

Examples:

```
$ brequeue 109
```

Lava kills the job with job ID 109, and requeues it in the PEND state. If job 109 has a priority of 4, it is placed after all the other jobs with the same priority.

```
$ brequeue -u user5 45 67 90
```

Lava kills and requeues three jobs belonging to User5. The jobs have the job IDs 45, 67, and 90.

Submitting a rerunnable job

To enable automatic job rerun at the job level, use `bsub -r`.

If the execution host fails, Lava dispatches the job to another host. You receive a mail message informing you of the host failure and the requeuing of the job.

If the Lava system fails, Lava requeues the job when the system restarts.

Viewing Job Information

The `bjobs` command displays the status of jobs in the Lava system. For more details on these or other `bjobs` options, see the `bjobs` command in the *Platform Lava Man Pages*.

The `bjobs` command reports the status of Lava jobs.

When no options are specified, `bjobs` displays information about jobs in the PEND, RUN, USUSP, PSUSP, and SSUSP states for the current user.

For example:

```
$ bjobs
JOBID USER STAT QUEUE FROM_HOST EXEC_HOST JOB_NAME SUBMIT_TIME
3926 user1 RUN priority hostf hostc verilog Oct 22 13:51
605 user1 SSUSP idle hostq hostc Test4 Oct 17 18:07
1480 user1 PEND priority hostd generator Oct 19 18:13
7678 user1 PEND priority hostd verilog Oct 28 13:08
7679 user1 PEND priority hosta coreHunter Oct 28 13:12
7680 user1 PEND priority hostb myjob Oct 28 13:17
```

All jobs

`bjobs -a` displays the same information as `bjobs` and in addition displays information

about recently finished jobs (PEND, RUN, USUSP, PSUSP, SSUSP, DONE and EXIT statuses).

All your jobs that are still in the system and jobs that have recently finished are displayed.

Running jobs

bjobs -r displays information only for running jobs (RUN state).

All jobs for all users

Run `bjobs -u all` to display all jobs for all users. Job information is displayed in the following order:

- 1 Running jobs
- 2 Pending jobs in the order in which they will be scheduled
- 3 Jobs in high priority queues are listed before those in lower priority queues

For example:

```
$ bjobs -u all
JOBID USER STAT QUEUE FROM_HOST EXEC_HOST JOB_NAME SUBMIT_TIME
1004 user1 RUN short hostA hostA job0 Dec 16 09:23
1235 user3 PEND priority hostM job1 Dec 11 13:55
1234 user2 SSUSP normal hostD hostM job3 Dec 11 10:09
1250 user1 PEND short hostA job4 Dec 11 13:59
```

Other examples of submitting jobs. (from man page)

```
% bsub sleep 100
```

Submit the UNIX command `sleep` together with its argument `100` as a batch job.

```
% bsub -q short -o my_output_file "pwd; ls"
```

Submit the UNIX command `pwd` and `ls` as a batch job to the queue named `short` and store the job output in `my_output` file.

```
% bsub -m "host1 host3 host8 host9" my_program
```

Submit `my_program` to run on one of the candidate hosts: `host1`, `host3`, `host8` and `host9`.

```
% bsub -q "queue1 queue2 queue3" -c 5 my_program
```

Submit `my_program` to one of the candidate queues: `queue1`, `queue2`, and `queue3` which are selected according to the CPU time limit specified by `-c 5`.

```
% bsub -I ls
```

Submit a batch interactive job which displays the output of `ls` at the user's terminal.

```
% bsub -b 20:00 -J my_job_name my_program
```

Submit `my_program` to run after 8 p.m. and assign it the job name `my_job_name`.

```
% bsub my_script
```

Submit `my_script` as a batch job. Since `my_script` is specified as a command line argument, the `my_script` file is not spooled. Later changes to the `my_script` file before the job completes may affect this job.

```
% bsub < default_shell_script
```

where `default_shell_script` contains:

```
sim1.exe  
sim2.exe
```

The file `default_shell_script` is spooled, and the commands will be run under the Bourne shell since a shell specification is not given in the first line of the script.

```
% bsub < csh_script
```

where `csh_script` contains:

```
#!/bin/csh  
sim1.exe  
sim2.exe
```

`csh_script` is spooled and the commands will be run under `/bin/csh`.

```
% bsub -q night < my_script
```

where `csh_script` contains:

```
#!/bin/csh  
sim1.exe  
sim2.exe
```

csh_script is spooled and the commands will be run under /bin/csh.

```
% bsub -q night < my_script
```

where my_script contains:

```
#!/bin/csh
#BSUB -q normal
#BSUB -o outfile -e errfile # my default stdout, stderr files
#BSUB -m "host1 host2" # my default candidate hosts
#BSUB -f "input > tmp" -f "output << tmp"
#BSUB -D 200 -c 10/host1
#BSUB -t 13:00
#BSUB -k "dir 5"
sim1.exe
sim2.exe
```

The job is submitted to the night queue instead of test, because the command line overrides the script.

```
% bsub -b 20:00 -J my_job_name
```

```
bsub> sleep 1800
bsub> my_program
bsub> CTRL-D
```

The job commands are entered interactively.