Batch Scripting for Parallel Systems

Author
Timothy H. Kaiser, Ph.D.
tkaiser@mines.edu
Purpose:

- To give you some ideas about what is possible
- To give you some examples to follow to help you write your own script
- Be a reference, where we can point people
- Document responses to questions
To Cover...

- Our test codes
- Bash useful concepts
- Basic Scripts
- Using PBS Variables in Scripts
- Redirecting Output, getting output before a job finishes
- Getting Notifications
- Keeping a record of what you did
- Creating directories on the fly for each job
- Using local disk space
To Cover...

• Multiple jobs on a node
• Sequential
• Multiple scripts - one node
• One Script - different MPI jobs on different cores
To Cover...

- Mapping tasks to nodes
- Less than N tasks per node
- Different executables working together
- Hybrid MPI/OpenMP jobs (MPI and Threading)
- Running on heterogeneous nodes using all cores
- Job dependencies
- Chaining jobs
- Jobs submitting new jobs
The Source and Scripts

These programs vary from a glorified "Hello World" to being very complex.

We also include copies of all our scripts

http://geco.mines.edu/guide/scripts

[joeuser@mio tests]$ wget http://geco.mines.edu/scripts/morescripts.tgz
[joeuser@mio tests]$ tar -xzf morescripts.tgz
[joeuser@mio tests]$ cd morescripts

[joeuser@mio somescripts]$ make 2> /dev/null
mpicc -o c_ex00 c_ex00.c
mpif90 -o f_ex00 f_ex00.f
rm -rf fmpi.mod
icc info.c -o info_c
ifort info.f90 -o info_f
cp info.py info_p
chmod 700 info_p
ifort -O3 -mkl -openmp pointer.f90 -o fillmem
od -vAn -d -N1048576 < /dev/urandom > segment
tar -czf data.tgz segment
rm -rf segment*
mpicc -DDO_LOCAL_FILE_TEST -c sinkfile.c
mpif90 sinkf.f90 sinkfile.o -o sinkf
mpicc -DDO_LOCAL_FILE_TEST -DDO_C_TEST sinkfile.c -o sinkfile
rm *o *mod
chmod 700 nodes

These programs vary from a glorified “Hello World” to being very complex.

We also include copies of all our scripts

http://geco.mines.edu/guide/scripts
What we have

• [c_ex00.c, c_ex00]
  • hello world in C and MPI

• [f_ex00.f, f_ex00]
  • hello world in Fortran and MPI

• [info.c, info_c] [info.f90, info_f] [info.py]
  • Serial programs in C, Fortran and Python that print the node name and process id. Creates a node name process id
```c
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>

main() {
    char name[128], fname[128];
    pid_t mypid;
    FILE *f;
    char aline[128];

    /* get the process id */
    mypid = getpid();
    /* get the host name */
    gethostname(name, 128);
    /* make a file name based on these two */
    sprintf(fname, "%s_%8.8d", name, (int)mypid);
    /* open and write to the file */
    f = fopen(fname, "w");
    fprintf(f, "C says hello from %d on %s\n", (int)mypid, name);
}
```
program info  
  USE IFPOSIX ! needed by PXFGETPID  
  implicit none  
  integer ierr,mypid  
  character(len=128) :: name,fname  
  ! get the process id  
  CALL PXFGETPID (mypid, ierr)  
  ! get the node name  
  call mynode(name)  
  ! make a filename based on the two  
  write(fname,'(a,"_",i8.8)')trim(name),mypid  
  ! open and write to the file  
  open(12,file=fname)  
  write(12,*)"Fortran says hello from",mypid," on ",trim(name)  
end program

subroutine mynode(name)  
  ! Intel Fortran subroutine to return  
  ! the name of a node on which you are  
  ! running  
  USE IFPOSIX  
  implicit none  
  integer jhandle  
  integer ierr,len  
  character(len=128) :: name  
  CALL PXFSTRUCTCREATE ("utsname", jhandle, ierr)  
  CALL PXFUNAME (jhandle, ierr)  
  call PXFSTRGET(jhandle,"nodename",name,len,ierr)  
end subroutine
#!/usr/bin/env python
import os
# get the process id
mypid=os.getpid()
# get the node name
name=os.uname()[1]
# make a filename based on the two
fname="%s_%8.8d" % (name,mypid)
# open and write to the file
f=open(fname,"w")
f.write("Python says hello from %d on %s\n" %(mypid,name))
Example Output From the Serial Programs

[joeuser@mio cwp]$ ./info_c
[joeuser@mio cwp]$ ls -lt mio*
-rw-rw-r-- 1 joeuser joeuser 41 Jan 11 13:47 mio.mines.edu_00050205
[joeuser@mio cwp]$ cat mio.mines.edu_00050205
C says hello from 50205 on mio.mines.edu
[joeuser@mio cwp]$
C MPI example

```c
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include <math.h>

/***************************************************************************/
/* This is a simple hello world program. Each processor prints out */
/* it's rank and the size of the current MPI run (Total number of */
/* processors). */
/***************************************************************************/
int main(argc,argv)
int argc;
char *argv[];
{
    int myid, numprocs,mylen;
    char myname[MPI_MAX_PROCESSOR_NAME];
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Get_processor_name(myname,&mylen);

    /* print out my rank and this run's PE size*/
    printf("Hello from \%d of \%d on \%s\n",myid,numprocs,myname);

    MPI_Finalize();
}
Fortran MPI example

!****************************************************************
!
program hello
!
include "mpif.h"
!
character (len=MPI_MAX_PROCESSOR_NAME):: myname
!
call MPI_INIT( ierr )
!
call MPI_COMM_RANK( MPI_COMM_WORLD, myid, ierr )
!
call MPI_COMM_SIZE( MPI_COMM_WORLD, numprocs, ierr )
!
call MPI_Get_processor_name(myname,mylen,ierr)
!
write(*,*)"Hello from ",myid," of ",numprocs," on ",trim(myname)
!
call MPI_FINALIZE(ierr)
!
stop
end
Fortran Matrix Inversion Example

- Fills a number of arrays with random data
- Does a matrix inversion
- Used to test the performance of individual cores of a processor
- Can also be used to test threading

```fortran
include 'mkl_vsl.fi'
...
program testinvert
use numz
...
call my_clock(cnt1(i))
CALL DGESV( N, NRHS, twod, LDA, IPIVs(:,i), Bs(:,i), LDB, INFOs(i) )
call my_clock(cnt2(i))
write(*,'(i5,i5,3(f12.3))')i,infos(i),cnt2(i),cnt1(i),real(cnt2(i)-cnt1(i),b8)
...
```
sinkfile.c

- A somewhat complicated example
- Does a parallel file copy
  - Copies a file seen by MPI task 0 to
  - Each nodes (not task) in an MPI program that does not share the node used by task 0
- Used in a situation where MPI tasks might not share file systems
Batch Scripts

• Batch scripts are just that - scripts
• Run with some “shell”, bash, csh, tsh, python
• Most of the things you can do in a normal script can be done in a batch script
• Some lines in the script are comments to the shell
• Comments can have meaning to the parallel environment
• The parallel environment can define/use variables
Bash

- Default shell on CSM machines
- Used to interact with the machine, run commands
- Bash commands can be run interactively or put in a script file
- A script file is really a “simple”
  - Program
  - List of commands
- We will use bash in our examples but other shells and scripting languages have similar capabilities
- First we discuss some features of bash

Notes on Commands

- `>` is used to sent output to a file (\texttt{date > mylisting})
- `>>` append output to a file (\texttt{ls >> mylisting})
- `>&` send output and error output to a file
- The `;` can be used to combine multiline commands on a single line. Thus the following are equivalent

\begin{align*}
\text{date} \\
\text{date } \text{; echo \textquote{line 2} } \text{; uptime} \\
\text{echo \textquote{line 2}} \\
\text{date}
\end{align*}
Notes on Commands

- Putting commands in ` ` returns the output of a command into a variable
- Can be use create a list with other commands such as “for loops”

```bash
myf90=`ls *f90`
echo $myf90
doint.f90 fourd.f90 tintel.f90 tp.f90 vect.f90
```

```bash
np=`expr 3 + 4`
np=`expr $PBS_NUM_NODES \* 4`
np=`expr $PBS_NUM_NODES / 4`
```

The command `expr` with “`” can be used to do integer math
For loops

```bash
myf90=`ls *f90`
for f in $myf90 ; do file $f ; done
```

doint.f90: ASCII program text
fourd.f90: ASCII program text
tintel.f90: ASCII program text
tp.f90: ASCII program text
vect.f90: ASCII program text

```bash
for (( c=1; c<=5; c++ )); do echo "Welcome $c times..."; done
Welcome 1 times...
Welcome 2 times...
Welcome 3 times...
Welcome 4 times...
Welcome 5 times...
```

```bash
for c in `seq 1 2 6`; do echo "Welcome $c times..."; date; done
```

Welcome 1 times...
Tue Jul 31 12:17:11 MDT 2012
Welcome 3 times...
Tue Jul 31 12:17:11 MDT 2012
Welcome 5 times...
Tue Jul 31 12:17:11 MDT 2012

```bash
myf90=`ls *f90`
for f in $myf90
    do file $f
done
```

```bash
for c in 1 2 3 4 5; do echo "Welcome $c times..."; done
```

Welcome 1 times...
Welcome 2 times...
Welcome 3 times...
Welcome 4 times...
Welcome 5 times...

```bash
for c in `seq 1 2 6`
do echo "Welcome $c times..."
done
```

Welcome 1 times...
Welcome 2 times...
Welcome 3 times...
Welcome 4 times...
Welcome 5 times...

Welcome 1 times...
Tue Jul 31 12:17:11 MDT 2012
# Combing Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ! EXPR ]</td>
<td>True if EXPR is false.</td>
</tr>
<tr>
<td>[ ( EXPR ) ]</td>
<td>Returns the value of EXPR. This may be used to override the normal precedence of operators.</td>
</tr>
<tr>
<td>[ EXPR1 -a EXPR2 ]</td>
<td>True if both EXPR1 and EXPR2 are true.</td>
</tr>
<tr>
<td>[ EXPR1 -o EXPR2 ]</td>
<td>True if either EXPR1 or EXPR2 is true.</td>
</tr>
</tbody>
</table>
Test Variable Being Set and “if”

We do this loop 3 times.
(1)“var” not set
(2)“var” set but empty
(3)var set and not empty

```bash
for i in 1 2 3 ; do
  echo "i=" $i
  if [ $i == 1 ] ; then unset var ; fi
  if [ $i == 2 ] ; then var="" ; fi
  if [ $i == 3 ] ; then var="abcd" ; fi

  if [ -z "$var" ] ; then echo "var is unset or empty A"; fi
  if [ ! -n "$var" ] ; then echo "var is unset or empty A2"; fi
  if [ -z "${var-x}" ] ; then echo "var is set but empty B"; fi
  if [ -n "$var" ] ; then echo "var is set and not empty C"; fi
  echo done
```

i= 1
var is unset or empty A
var is unset or empty A2

i= 2
var is unset or empty A
var is unset or empty A2
var is set but empty B

i= 3
var is set and not empty C
String Tests

if test "abc" = "def" ;then echo "abc = def" ; else echo "nope 1" ; fi

if test "abc" != "def" ;then echo "abc != def" ; else echo "nope 2" ; fi

if [ "abc" \< "def" ];then echo "abc < def" ; else echo "nope 3" ; fi

if [ "abc" \> "def" ]; then echo "abc > def" ; else echo "nope 4" ; fi

if [ "abc" \> "abc" ]; then echo "abc > abc" ; else echo "nope 5" ; fi

nope 1
abc != def
abc < def
nope 4
nope 5
## File Tests

<table>
<thead>
<tr>
<th>Test</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-a FILE</code></td>
<td>True if <code>FILE</code> exists.</td>
</tr>
<tr>
<td><code>-b FILE</code></td>
<td>True if <code>FILE</code> exists and is a block-special file.</td>
</tr>
<tr>
<td><code>-c FILE</code></td>
<td>True if <code>FILE</code> exists and is a character-special file.</td>
</tr>
<tr>
<td><code>-d FILE</code></td>
<td>True if <code>FILE</code> exists and is a directory.</td>
</tr>
<tr>
<td><code>-e FILE</code></td>
<td>True if <code>FILE</code> exists.</td>
</tr>
<tr>
<td><code>-f FILE</code></td>
<td>True if <code>FILE</code> exists and is a regular file.</td>
</tr>
<tr>
<td><code>-g FILE</code></td>
<td>True if <code>FILE</code> exists and its SGID bit is set.</td>
</tr>
<tr>
<td><code>-h FILE</code></td>
<td>True if <code>FILE</code> exists and is a symbolic link.</td>
</tr>
<tr>
<td><code>-k FILE</code></td>
<td>True if <code>FILE</code> exists and its sticky bit is set.</td>
</tr>
<tr>
<td><code>-p FILE</code></td>
<td>True if <code>FILE</code> exists and is a named pipe (FIFO).</td>
</tr>
<tr>
<td><code>-r FILE</code></td>
<td>True if <code>FILE</code> exists and is readable.</td>
</tr>
<tr>
<td><code>-s FILE</code></td>
<td>True if <code>FILE</code> exists and has a size greater than zero.</td>
</tr>
<tr>
<td><code>-t FD</code></td>
<td>True if file descriptor <code>FD</code> is open and refers to a terminal.</td>
</tr>
<tr>
<td><code>-u FILE</code></td>
<td>True if <code>FILE</code> exists and its SUID (set user ID) bit is set.</td>
</tr>
<tr>
<td><code>-w FILE</code></td>
<td>True if <code>FILE</code> exists and is writable.</td>
</tr>
<tr>
<td><code>-x FILE</code></td>
<td>True if <code>FILE</code> exists and is executable.</td>
</tr>
<tr>
<td><code>-o FILE</code></td>
<td>True if <code>FILE</code> exists and is owned by the effective user ID.</td>
</tr>
<tr>
<td><code>-g FILE</code></td>
<td>True if <code>FILE</code> exists and is owned by the effective group ID.</td>
</tr>
<tr>
<td><code>-l FILE</code></td>
<td>True if <code>FILE</code> exists and is a symbolic link.</td>
</tr>
<tr>
<td><code>-n FILE</code></td>
<td>True if <code>FILE</code> exists and has been modified since it was last read.</td>
</tr>
<tr>
<td><code>-s FILE</code></td>
<td>True if <code>FILE</code> exists and is a socket.</td>
</tr>
<tr>
<td><code>-S FILE</code></td>
<td>True if <code>FILE</code> exists and is a socket.</td>
</tr>
<tr>
<td><code>-t FILE1 -nt FILE2</code></td>
<td>True if <code>FILE1</code> has been changed more recently than <code>FILE2</code>, or <code>FILE1</code> exists and <code>FILE2</code> does not.</td>
</tr>
<tr>
<td><code>-t FILE1 -ot FILE2</code></td>
<td>True if <code>FILE1</code> is older than <code>FILE2</code>, or <code>FILE2</code> exists and <code>FILE1</code> does not.</td>
</tr>
<tr>
<td><code>-t FILE1 -ef FILE2</code></td>
<td>True if <code>FILE1</code> and <code>FILE2</code> refer to the same device and inode numbers.</td>
</tr>
</tbody>
</table>
Testing Return Code & /dev/null

• Commands return an exit code
  • 0 = success
  • not 0 = failure
• The exit code from the previous command is stored in `?`
• `?` can be echoed or tested
• This is often used with piping output into /dev/null “the bit bucket” when you only want to know if a command was successful

```bash
ls a_dummy_file >& /dev/null

if [ $? -eq 0 ] ; then
  echo "ls of a_dummy_file successful"
fi
```
rm -f a_dummy_file
while true ; do
    ls a_dummy_file >& /dev/null
    if [ $? -eq 0 ] ; then
        echo "ls of a_dummy_file successful"
    else
        echo "ls of a_dummy_file failed"
    fi
    if [ -a a_dummy_file ] ; then
        echo "a_dummy_file exists, breaking"
        break
    else
        echo "a_dummy_file does not exist"
    fi
    echo ; echo "bottom of while loop" ; echo
touch a_dummy_file
done

ls of a_dummy_file failed
a_dummy_file does not exist
bottom of while loop

ls of a_dummy_file successful
a_dummy_file exists, breaking
Running Batch Scripts

- A batch script is submitted to a scheduler
  - pbs/torque/moab, sge, lsf, poe
- Commands to submit scripts
  - qsub, msub, bsub, poe
- The scheduler decides where and when to run your script
  - Wait for nodes to become available
  - Wait for other jobs to finish
  - Jobs are given a name so that you can track them in the system
## Related Commands pbs/torque/moab

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qsub</td>
<td>submit jobs</td>
</tr>
<tr>
<td>canceljob</td>
<td>cancel job</td>
</tr>
<tr>
<td>qdel</td>
<td>delete/cancel batch jobs</td>
</tr>
<tr>
<td>checkjob</td>
<td>provide detailed status report for specified job</td>
</tr>
<tr>
<td>checkjob -v</td>
<td>show why a job will not run on specific nodes</td>
</tr>
<tr>
<td>releasehold</td>
<td>release job defers and holds</td>
</tr>
<tr>
<td>releaseres</td>
<td>release reservations</td>
</tr>
<tr>
<td>sethold</td>
<td>set job holds</td>
</tr>
<tr>
<td>showq</td>
<td>show queued jobs</td>
</tr>
<tr>
<td>showres</td>
<td>show existing reservations</td>
</tr>
<tr>
<td>showstart</td>
<td>show estimates of when job can/will start</td>
</tr>
<tr>
<td>showstate</td>
<td>show current state of resources</td>
</tr>
<tr>
<td>pbsnodes</td>
<td>view/modify batch status of compute nodes</td>
</tr>
<tr>
<td>qhold</td>
<td>hold batch jobs</td>
</tr>
<tr>
<td>qrls</td>
<td>release batch job holds</td>
</tr>
<tr>
<td>qsig</td>
<td>send a signal to a batch job</td>
</tr>
<tr>
<td>qstat</td>
<td>view queues and jobs</td>
</tr>
</tbody>
</table>
A Simple PBS Script for a MPI job

#!/bin/bash

#PBS -l nodes=1:ppn=8
#PBS -l walltime=00:02:00
#PBS -N testIO
#PBS -o stdout
#PBS -e stderr
#PBS -V
#PBS -m abe
#PBS -M joeuser@mines.edu

#----------------------

cd /home/joeuser/examples

mpiexec -n 8 ./c_ex00

Scripts contain comments designated with a # that are interpreted by PBS and normal shell commands.

We go to this directory
Run this MPI program on 8 cores.
A Simple PBS Script for a MPI job

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#!/bin/bash</code></td>
<td>This is a bash script</td>
</tr>
<tr>
<td><code>#PBS -l nodes=1:ppn=8</code></td>
<td>We want 1 node and expect 8 cores</td>
</tr>
<tr>
<td><code>#PBS -l walltime=00:02:00</code></td>
<td>We want the node for 2 minutes</td>
</tr>
<tr>
<td><code>#PBS -N testIO</code></td>
<td>The job name is “testIO”</td>
</tr>
<tr>
<td><code>#PBS -o stdout</code></td>
<td>Output will go to a file “stdout”</td>
</tr>
<tr>
<td><code>#PBS -e stderr</code></td>
<td>Errors will go to a file “stdout”</td>
</tr>
<tr>
<td><code>#PBS -V</code></td>
<td>Pass current environment to nodes</td>
</tr>
<tr>
<td><code>#PBS -m abe</code></td>
<td>Send email on abort, begin, end</td>
</tr>
<tr>
<td><code>#PBS -M joeuser@mines.edu</code></td>
<td>Address for email</td>
</tr>
<tr>
<td><code>cd /home/joeuser/examples</code></td>
<td>Go to this directory first</td>
</tr>
<tr>
<td><code>mpiexec -n 8 ./c_ex00</code></td>
<td>Run c_ex00 on 8 cores</td>
</tr>
</tbody>
</table>
What happens when you run a script?

- You are given a collection of nodes
- You are logged on to one of the nodes, the primary compute node
- Any “normal” script command only run on the primary compute node
- Extra effort must be taken to run on all nodes
  - mpiexec
    - Also Run only on the primary compute node
- Makes the effort to launch MPI jobs on all nodes
Variables in Scripts
## PBS “script” Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
<th>Typical Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBS_O_WORKDIR</td>
<td>Directory for the script</td>
<td>/panfs/storage/scratch/joeuser</td>
</tr>
<tr>
<td>PBS_O_HOME</td>
<td>Users Home Directory</td>
<td>/home/joeuser</td>
</tr>
<tr>
<td>PBS_O_LOGNAME</td>
<td>User</td>
<td>joeuser</td>
</tr>
<tr>
<td>PBS_NUM_NODES</td>
<td># nodes for the job</td>
<td>1</td>
</tr>
<tr>
<td>PBS_JOBID</td>
<td>Job ID</td>
<td>269924.mio.mines.edu</td>
</tr>
<tr>
<td>PBS_NODEFILE</td>
<td>File for the list of nodes</td>
<td>/var/spool/torque/aux/269924.mio.mines.edu</td>
</tr>
<tr>
<td>PBS_O_HOST</td>
<td>Host used to launch job</td>
<td>mio.mines.edu</td>
</tr>
</tbody>
</table>

You can also use variables you define before you submit your script and variables defined in your environment.
A Simple PBS Script for a MPI job

```bash
#!/bin/bash
#PBS -l nodes=1:ppn=8
#PBS -l walltime=00:02:00
#PBS -N testIO
#PBS -o stdout
#PBS -e stderr
#PBS -V
#PBS -m abe
#PBS -M joeuser@mines.edu
#----------------------

cd $PBS_O_WORKDIR
mpiexec -n 8 ./c_ex00
```

We go to “starting” directory

Run this MPI program on 8 cores
A Simple PBS Script for a MPI job

#!/bin/bash

#PBS -l nodes=1:ppn=8
#PBS -l walltime=00:02:00
#PBS -N testIO
#PBS -o out.$PBS_JOBID
#PBS -e err.$PBS_JOBID
#PBS -V
#PBS -m abe
#PBS -M joeuser@mines.edu

#----------------------

cd $PBS_O_WORKDIR

mpiexec -n 8 ./c_ex00

Our output and error go to files with the job id appended

The output is from the script and the program(s) we are running

stdout
Problem and Solution

- Using `#PBS -o out.$PBS_JOBID`
  - Useful to give you a unique output for each run
- Problem: you don’t see “stdout” and “stderr” until the job completes
- Solution
  - You can get output as the job runs by “>” redirecting on the program run line
  - You can still use `$PBS_JOBID` to redirect to a unique file for each run
A Simple PBS Script for a MPI job

#!/bin/bash

#PBS -l nodes=1:ppn=8
#PBS -l walltime=00:02:00
#PBS -N testIO
#PBS -o out.$PBS_JOBID
#PBS -e err.$PBS_JOBID
#PBS -V
#PBS -m abe
#PBS -M joeuser@mines.edu

#----------------------

cd $PBS_O_WORKDIR

mpiexec -n 8 ./c_ex00 > myout.$PBS_JOBID

The output from the script

Gives the program output as it runs with each run having a unique output file
Shorten JOBID

- $PBS_JOBID is of the form
  - 45682.mio.mines.edu

- How can we shorten this to just a number?
  - `sed -e 's/\..*//'`

  - strips everything past the first period

$ echo $PBS_JOBID
201665.mio.mines.edu

$ MY_JOBID=`echo $PBS_JOBID | sed -e 's/\..*//'`

$ echo $MY_JOBID
201665
A Simple PBS Script for a MPI job

#!/bin/bash
#PBS -l nodes=1:ppn=8
#PBS -l walltime=00:02:00
#PBS -N testIO
#PBS -o out.$PBS_JOBID
#PBS -e err.$PBS_JOBID
#PBS -V
#PBS -m abe
#PBS -M joeuser@mines.edu
#----------------------
MY_JOBID=`echo $PBS_JOBID|sed -e 's/\..*//`
MY_JOBID=`echo $PBS_JOBID|sed -e 's/\..*//`
cd $PBS_O_WORKDIR
mpiexec -n 8 ./c_ex00 > myout.$MY_JOBID

The output from the script

myout.#####
Keeping Records & Notifications
We want...

- To keep records of what scripts we have run
- To be notified when a script runs
  - `-M` with `-abe` will send a notice at start and stop
- We want more than the job number
- We want everything
#!/bin/bash
#PBS -l nodes=2:ppn=8
#PBS -l naccesspolicy=singleuser
#PBS -l walltime=00:05:00
#PBS -N testIO
#PBS -o out.$PBS_JOBID
#PBS -e err.$PBS_JOBID
#PBS -r n
#PBS -V
#PBS -m abe
#PBS -M joeminer@mines.edu
#-----------------------------------------------------
cd $PBS_O_WORKDIR
#save our list of nodes
sort -u $PBS_NODEFILE > mynodes.$PBS_JOBID
#save a copy of this sciprt (may strip comments)
cat $0 > runscript.$PBS_JOBID
#save our PBS environment, the path to our mpiexec command, and executable
printenv | grep PBS > env.$PBS_JOBID
which mpiexec >> env.$PBS_JOBID
which ./c_ex00 >> env.$PBS_JOBID
#mail us the environment and other "stuff"
### mail < env.$PBS_JOBID -s $PBS_JOBID $USER@mines.edu
ssh mio "mail < $PBS_O_WORKDIR/env.$PBS_JOBID -s $PBS_JOBID $USER@mines.edu"

mpiexec -np 16 ./c_ex00

How can I record what I did and where? How can I know when a particular script starts and exactly what is running?
Lots of records...

[joeuser@ra quick]$ ls -l *126833*
-rw-rw-r-- 1 joeuser joeuser 1688 Mar 4 13:18 env.126833.ra.local
-rw------- 1 joeuser joeuser  0 Mar 4 13:18 err.126833.ra.local
-rw-rw-r-- 1 joeuser joeuser  38 Mar 4 13:18 mynodes.126833.ra.local
-rw------- 1 joeuser joeuser 1018 Mar 4 13:18 out.126833.ra.local
-rw-rw-r-- 1 joeuser joeuser  716 Mar 4 13:18 runscript.126833.ra.local
[joeuser@ra quick]$
More on variables and a few other details
#!/bin/bash

##PBS -l nodes=1:ppn=8
#PBS -l nodes=1
#
#PBS -W x=NACCESSPOLICY:SINGLEUSER
#PBS -l walltime=24:00:00
#PBS -N testIO
#PBS -o outx8.$PBS_JOBID
#PBS -e errx8.$PBS_JOBID
#PBS -r n
#PBS -V
#PBS -m abe
#PBS -M joeuser@mines.edu
#
#-----------------------------------------------------
cd $PBS_O_WORKDIR

sort -u $PBS_NODEFILE > mynodes.$PBS_JOBID

export INPUT=sinput
export APP=fillmemic

sort -u $PBS_NODEFILE > $APP.$INPUT.nodes.$PBS_JOBID

env1

Not an MPI job so we don't have mpiexec
#!/bin/bash

#PBS -l nodes=1:ppn=8
#PBS -l nodes=1
#
#PBS -W x=NACCESSPOLICY:SINGLEUSER
#PBS -l walltime=24:00:00
#PBS -N testIO
#PBS -o outx8.$PBS_JOBID
#PBS -e errx8.$PBS_JOBID
#PBS -r n
#PBS -V
#PBS -m abe
#PBS -M joeuser@mines.edu

#-----------------------------------------------------

cd $PBS_O_WORKDIR

sort -u $PBS_NODEFILE > mynodes.$PBS_JOBID

export INPUT=sinput
export APP=fillmem

sort -u $PBS_NODEFILE > $APP.$INPUT.nodes.$PBS_JOBID

cat $INPUT > $APP.$INPUT.input.$PBS_JOBID

./$APP < $INPUT >> $APP.$INPUT.output.$PBS_JOBID

Create a short list of the nodes used in my job and give it a unique name. We recommend people always do this.

Same as before but now we put our list in a file that has application name, file name, and input file as part of the file name

We are going to use variables for both our input file and application names

Save a copy of our input and put the output in its own file
What we get

[jkaiser@mio morescripts]$ ls -lt *edu
-rw-rw-- 1 tkaiser tkaiser 1755 Aug  3 12:47 fillmem.sinput.output.569804.mio.mines.edu
-rw------- 1 tkaiser tkaiser   45 Aug  3 12:47 errx8.569804.mio.mines.edu
-rw-rw-- 1 tkaiser tkaiser  10 Aug  3 12:47 fillmem.sinput.input.569804.mio.mines.edu
-rw-rw-- 1 tkaiser tkaiser    5 Aug  3 12:47 fillmem.sinput.nodes.569804.mio.mines.edu
-rw-rw-- 1 tkaiser tkaiser    5 Aug  3 12:47 mynodes.569804.mio.mines.edu
-rw------- 1 tkaiser tkaiser    0 Aug  3 12:47 outx8.569804.mio.mines.edu

[joeuser@mio somescripts]$ cat fillmem.sinput.nodes.569804.mio.mines.edu
n101

[joeuser@mio somescripts]$ cat fillmem.sinput.input.569804.mio.mines.edu
n53

[joeuser@mio somescripts]$ head outx8.529478.mio.mines.edu
matrix size 1024
copies 32
bytes 268435456 gbytes 0.250
...
Multiple Executables Tricks

• Case 1: Multiple jobs running on the same node at the same time
  • Independent
  • Launched from different scripts

• Case 2: Multiple executables running on the same node at the same time
  • Independent
  • Launched from a single script

• Either case could be serial or MPI

• Case 3: Using mpiexec to launch several serial programs
Case 1: Multiple Scripts

Or the same script several times
#!/bin/bash
##PBS -l nodes=1:ppn=8
PBS -l nodes=1
#PBS -W x=NACCESSPOLICY:SINGLEUSER
PBS -l walltime=24:00:00
PBS -N testIO
PBS -o outx8.$PBS_JOBID
PBS -e errx8.$PBS_JOBID
PBS -V
PBS -m abe
PBS -M joeuser@mines.edu
#-----------------------------------------------------
cd $PBS_O_WORKDIR

export INPUT=sinput
export APP=fillmemc

sort -u $PBS_NODEFILE > mynodes.$PBS_JOBID

sort -u $PBS_NODEFILE > $APP.$INPUT.$PBS_JOBID

./$APP < $INPUT >> $APP.$INPUT.$PBS_JOBID

We have our application name and input file set as a variable.

Save a list of nodes, first in just a mynodes.* file and then a file that contains the input file name.

This is not an MPI job so we don't have mpiexec.
#!/bin/bash

#PBS -l nodes=1:ppn=8
#PBS -l nodes=1
#PBS -W x=NACCESSPOLICY:SINGLEUSER
#PBS -l walltime=24:00:00
#PBS -N testIO
#PBS -o outx8.$PBS_JOBID
#PBS -e errx8.$PBS_JOBID
#PBS -r n
#PBS -V
#PBS -m abe
#PBS -M joeuser@mines.edu

#-----------------------------

cd $PBS_O_WORKDIR

sort -u $PBS_NODEFILE > mynodes.$PBS_JOBID

#export INPUT=sinput
export APP=fillmemc

sort -u $PBS_NODEFILE > $APP.$INPUT.$PBS_JOBID

./$APP < $INPUT >> $APP.$INPUT.$PBS_JOBID

We have commented out the line that sets the input file name. We can (must) specify the input before running the script.
Assume we have 4 data sets and we are willing to run on any one node.

Which node are we using?
It is in the mynodes* file

Force the rest of our jobs to the same node

We have specified the input file here. This is picked up by the script

(If you have a reserved node you can specify it for the first run also.)
Our output files:

[joeuser@mio test]$ ls -l fillmemc.sinput*

-rw-rw-r-- 1 joeuser joeuser 3395 Feb 15 11:31 fillmemc.sinput1.267335.mio.mines.edu
-rw-rw-r-- 1 joeuser joeuser 5035 Feb 15 11:32 fillmemc.sinput2.267336.mio.mines.edu
-rw-rw-r-- 1 joeuser joeuser 6675 Feb 15 11:32 fillmemc.sinput3.267337.mio.mines.edu
-rw-rw-r-- 1 joeuser joeuser 1541 Feb 15 11:29 fillmemc.sinput4.267334.mio.mines.edu
-rw-rw-r-- 1 joeuser joeuser 1755 Feb 15 11:31 fillmemc.sinput4.267338.mio.mines.edu

[joeuser@mio test]$  

Note different job numbers

The input file name becomes part of the output file name

Our execution line

./$APP  < $INPUT >> $APP.$INPUT.$PBS_JOBID
Case 2: Multiple Executables Same Script
#!/bin/bash
#PBS -l nodes=1:ppn=8
#PBS -W x=NACCESSPOLICY:SINGLEUSER
#PBS -l walltime=24:00:00
#PBS -N testIO
#PBS -o outx8.$PBS_JOBID
#PBS -e errx8.$PBS_JOBID
#PBS -r n
#PBS -V
#PBS -m abe
#PBS -M joeuser@mines.edu
#-----------------------------------------------------

cd $PBS_O_WORKDIR

sort -u $PBS_NODEFILE > mynodes.$PBS_JOBID

export APP=fillmem

for INPUT in sinput1 sinput2 sinput3 sinput4 ; do
  ./$APP  < $INPUT >> $APP.$INPUT.$PBS_JOBID &
done

wait

We have our application name and input file set as a variable.

We launch the application over a list of input files.

The wait command “holds” the node until all of your applications are done.

Forces the job into the background so we can launch the next multiwait.
Our output files:

Note common job numbers with different input files

-rw-rw-r-- 1 joeuser joeuser  6671 Feb 15 12:16 fillmemc.sinput3.267355.mio.mines.edu
-rw-rw-r-- 1 joeuser joeuser  5031 Feb 15 12:14 fillmemc.sinput2.267355.mio.mines.edu
-rw-rw-r-- 1 joeuser joeuser  3391 Feb 15 12:13 fillmemc.sinput1.267355.mio.mines.edu
-rw-rw-r-- 1 joeuser joeuser  1751 Feb 15 12:11 fillmemc.sinput4.267355.mio.mines.edu

./$APP  < $INPUT >> $APP.$INPUT.$PBS_JOBID &
mpiexec and serial applications

- Some versions of mpiexec and mpirun will work with nonMPI programs
- Creates specified number of copies of the program, all independent
Our Batch file, batch1

#!/bin/bash
#PBS -l nodes=2:ppn=8
#PBS -l walltime=08:00:00
#PBS -N test_1
#PBS -o outx.$PBS_JOBID
#PBS -e errx.$PBS_JOBID
#PBS -r n
#PBS -V
##PBS -m abe
##PBS -M joeuser@mines.edu
#-----------------------------------------------------
cd $PBS_O_WORKDIR

# get a short and full list of all of my nodes
sort -u $PBS_NODEFILE > mynodes.$PBS_JOBID
sort $PBS_NODEFILE > allmynodes.$PBS_JOBID

export MYPROGRAM=info_p

info_p is a python program that creates a file based on node name and process id

echo "running" $PBS_O_WORKDIR/$MYPROGRAM
mpiexec -np 16 $PBS_O_WORKDIR/$MYPROGRAM
Running a serial program with mpiexec

```
export MYPROGRAM=info_p
```

```
[joeuser@mio cwp]$ qsub serial
3590.mio.mines.edu
[joeuser@mio cwp]$
```

```
[joeuser@mio cwp]$ ls -lt
```

```
total 1792
-rw------- 1 joeuser joeuser 0 Jan 11 14:13 errx.3590.mio.mines.edu
-rw------- 1 joeuser joeuser 44 Jan 11 14:13 outx.3590.mio.mines.edu
-rw------- 1 joeuser joeuser 64 Jan 11 14:13 allmynodes.3590.mio.mines.edu
-rw------- 1 joeuser joeuser 8 Jan 11 14:13 mynodes.3590.mio.mines.edu
-rw------- 1 joeuser joeuser 36 Jan 11 14:13 n46_00051792
-rw------- 1 joeuser joeuser 36 Jan 11 14:13 n46_00051793
-rw------- 1 joeuser joeuser 36 Jan 11 14:13 n46_00051794
-rw------- 1 joeuser joeuser 36 Jan 11 14:13 n46_00051795
-rw------- 1 joeuser joeuser 36 Jan 11 14:13 n46_00051796
-rw------- 1 joeuser joeuser 36 Jan 11 14:13 n46_00051797
-rw------- 1 joeuser joeuser 36 Jan 11 14:13 n46_00051798
-rw------- 1 joeuser joeuser 36 Jan 11 14:13 n46_00051799
-rw------- 1 joeuser joeuser 36 Jan 11 14:13 n47_00051784
-rw------- 1 joeuser joeuser 36 Jan 11 14:13 n47_00051785
-rw------- 1 joeuser joeuser 36 Jan 11 14:13 n47_00051786
-rw------- 1 joeuser joeuser 36 Jan 11 14:13 n47_00051787
-rw------- 1 joeuser joeuser 36 Jan 11 14:13 n47_00051788
-rw------- 1 joeuser joeuser 36 Jan 11 14:13 n47_00051789
-rw------- 1 joeuser joeuser 36 Jan 11 14:13 n47_00051790
-rw------- 1 joeuser joeuser 36 Jan 11 14:13 n47_00051791
```

```
[joeuser@mio cwp]$ cat n46_00051792
Python says hello from 51792 on n46
```
Mapping Tasks to Nodes
Need better than default mappings...

- Want to use less than all of the nodes on a node
- Large memory/task
- Hybrid MPI/OpenMPI
- Different executables on various cores (MPMD)
- Heterogeneous environment with different core counts
Method for OpenMPI and MVAPICH2

- Same method works for both versions of MPI
- Create a description of your job on the fly from inside your script
- The description is a mapping of programs to cores
- Tell mpiexec/mpirun to use your description to launch your job
- We created a utility script to make it easy
Alternate syntax for mpiexec

- Normally you specify the number of MPI tasks on the mpiexec line
- The alternate syntax is to provide an “appfile”
  - `mpiexec -app appfile`
  - The appfile is a mapping of executables to nodes
Appfile format

- Collection of lines of the form
  - `host <host name> -np <number of copies to run on host> <program name>`
- Specify different application names in your appfile for MPMD
- You can specify a node or program more than once
Examples

These two are equivalent

Appfile Example 1

- host compute-1-1 -np 1 myprogram
- host compute-1-1 -np 1 myprogram
- host compute-1-1 -np 1 myprogram
- host compute-1-1 -np 1 myprogram

Appfile Example 2

- host compute-1-1 -np 4 myprogram

These two are not equivalent

Appfile Example 3

- host compute-1-1 -np 2 aya.out
- host compute-2-3 -np 2 bee.out

Appfile Example 3

- host compute-1-1 -np 1 aya.out
- host compute-2-3 -np 1 aya.out
- host compute-1-1 -np 2 bee.out
- host compute-2-3 -np 2 bee.out

Note: You should specify the full path to your program
Difficulty and Solution

- Problem:
  - Names of the nodes that you are using are not known until after the job is submitted
  - You need to create the appfile on the fly from within your PBS script
Difficultly and Solution

- Solution:
  - The PBS variable $PBS_NODEFILE contains the name of a file that has the list of nodes on which your job will run.
  - We have created a script "match" which takes a list of nodes and a list of applications to run on those nodes and creates an appfile
  - Located on Ra at /lustre/home/apps/utility/match
  - Located on Mio at /opt/utility/match
Solution

Given your $PBS_NODEFILE and a list of programs in a file app_list the simplest usage of match is:

```bash
match $PBS_NODEFILE app_list > appfile
mpiexec --app appfile
```

For mvapich2 replace --app with --configfile
Match notes

- Number of applications that get launched
  - Is equal to the length of the longer of the two lists, the node file list, or the application list
  - If the lists are not the same length then multiple copies will be launched

- Match also takes an optional replication count, the number copies of an application to run on a node

- Feel free to copy and modify the match script for your own needs
Examples

#get a copy of all of our nodes, each node will be
#listed 8 times (16 on AuN and newer Mio Nodes)
cat $PBS_NODEFILE > fulllist

#save a nicely sorted short list of nodes, each node only
#listed one time
sort -u $PBS_NODEFILE > shortlist

fulllist
compute-8-15.local
compute-8-15.local
compute-8-15.local
compute-8-15.local
compute-8-15.local
compute-8-15.local
compute-8-15.local
compute-8-15.local

shortlist
compute-8-15.local
compute-8-13.local

/lustre/home/apps/utility/nsort $PBS_NODEFILE

Also works to give a sorted list
Examples

- We have two programs we are going to play with f_ex00 and c_ex00
- We have two program lists that we are going to use
  - oneprogram
    - c_ex00
    - twoprograms
      - c_ex00
      - f_ex00
match fulllist twoprograms > appfile1

-host compute-8-15.local -np 1 c_ex00
-host compute-8-15.local -np 1 f_ex00
-host compute-8-15.local -np 1 c_ex00
-host compute-8-15.local -np 1 f_ex00
-host compute-8-15.local -np 1 c_ex00
-host compute-8-15.local -np 1 f_ex00
-host compute-8-15.local -np 1 c_ex00
-host compute-8-15.local -np 1 f_ex00
-host compute-8-13.local -np 1 c_ex00
-host compute-8-13.local -np 1 f_ex00
-host compute-8-13.local -np 1 c_ex00
-host compute-8-13.local -np 1 f_ex00
-host compute-8-13.local -np 1 c_ex00
-host compute-8-13.local -np 1 f_ex00
-host compute-8-13.local -np 1 c_ex00
-host compute-8-13.local -np 1 f_ex00
match shortlist twoprograms > appfile2

-host compute-8-13.local -np 1 c_ex00
-host compute-8-15.local -np 1 f_ex00
match shortlist twoprograms 2 > appfile3

-host compute-8-13.local -np 2 c_ex00
-host compute-8-15.local -np 2 f_ex00
match shortlist oneprogram 2 > appfile4

-host compute-8-13.local -np 2 c_ex00
-host compute-8-15.local -np 2 c_ex00

This will be useful for hybrid MPI OpenMP programs
Can take names from command line

match <node list file> -p"list of programs" [<number of copies per node>]

match shortlist -p"c_ex01 f_ex01" 1 8

Run 1 copy of c_ex01 on the first node in shortlist and 8 copies of f_ex01 on the second node

If you don’t specify the number of copies then do 1 per core
Running on Heterogeneous Nodes

- Mixed numbers of cores (8, 12, 16)
- Want to use all of the cores
- The number of cores expected on a node is ppn=N
- Could use match with a fixed core count but this might leave some open or over subscribed

match shortlist -p "c_ex01" 8 8

- If you don’t specify the number of copies then you will be given 1 per core

match shortlist -p "c_ex01"
Creating Directories on the fly
Using Local Disk
A Directory for Each Run

```bash
#!/bin/bash
PBS -l nodes=4:ppn=12
PBS -l walltime=15:00:00
PBS -N ben_abby
PBS -o stdout.$PBS_JOBID
PBS -e stderr.$PBS_JOBID
PBS -V
PBS -M joeuser@mines.edu
PBS -m abe

cd $PBS_O_WORKDIR

# This line would give us a NEW_DIR based on a
time stamp, the year, month, day, hour, minute,
and second.
export NEW_DIR=`date +%y%m%d%H%M%S`
#
export NEW_DIR=`echo $PBS_JOBID | sed -e 's/..*//'`

mkdir $NEW_DIR
Go to it
save its name (we will use this later)

cp $PBS_O_WORKDIR/data.tgz .
"Copy" our data to
the new directory
from our starting
directory

mpiexec -np 48 $PBS_O_WORKDIR/sinkfile
```

This line would give us a NEW_DIR based on a
time stamp. Note here we use sed to strip off
the machine name part of the $PBS_JOBID

Or

```
Create the new directory
```
Local Disk Space

- Most parallel machines have some disk space that is local
- Can only be seen by tasks on the same nodes
- Can’t be seen from the primary compute node
- Might be faster than shared space
- Size? Location?
  - Usually a bad idea to use /tmp
  - On “Rocks” it’s /state/partition1
- Usage is up to local policy
Using Local disk

• Figure out where it is
• Create directories
• Copy to the new directory
• Compute
• Copy what you want to shared storage
• Clean up after yourself
Here it is...

- Figure out where local disk is
- Create a shared directory where all of the results will be copied in the end
- Get a list of nodes
- Use ssh to create a directory on each node
- Do all the “normal” saves done in other examples
- Go to the new directory (This only happens on master node.)
- Use “match” to create an appfile
- Run the application (All tasks will get launched in the same named directory)
- Use scp to copy the files to shared storage
- Clean up
#!/bin/bash
#PBS -l nodes=4:ppn=12
#PBS -l walltime=15:00:00
#PBS -N ben_abby
#PBS -o stdout.$PBS_JOBID
#PBS -e stderr.$PBS_JOBID
#PBS -V
#PBS -M joeuser@mines.edu
#PBS -m abe

if [ -n "$JOBTMP" ] ; then
  echo using $JOBTMP from environment
else
  export JOBTMP=/scratch
fi

cd $PBS_O_WORKDIR

# create a directory for this run
#export NEW_DIR=`date +%y%m%d%H%M%S`
export NEW_DIR=`echo $PBS_JOBID | sed -e 's/\..\..//g'`
mkdir $NEW_DIR
cd $NEW_DIR
export ODIR=`pwd`

cp $PBS_O_WORKDIR/data.tgz .
tar -xzf data.tgz
Set Up 2

# get a list of nodes...
export nlist=`sort -u $PBS_NODEFILE`

# For each node...
for i in $nlist
do
# Create my temporary directory in /scratch on each node
  ssh $i mkdir $JOBTMP/$NEW_DIR
# Copy my data
  echo $USER@$i:$JOBTMP/$NEW_DIR
  scp * $USER@$i:$JOBTMP/$NEW_DIR
done

# save a copy of our nodes
sort -u $PBS_NODEFILE > nlist.$PBS_JOBID
cat $PBS_NODEFILE > flist.$PBS_JOBID

# save a copy of this script
cat $0 > runscript.$PBS_JOBID

# save our environment
printenv >> env.$PBS_JOBID
lst -lt >> env.$PBS_JOBID

# Go to our working directory
cd $JOBTMP/$NEW_DIR
Run

\begin{verbatim}
export APP= $PBS_O_WORKDIR/sinkfile
match $ODIR/flist.$PBS_JOBID -p"$APP" > appfile
mpiexec -app appfile >& screen.$PBS_JOBID
\end{verbatim}
Clean Up

```bash
# for each node...
for i in $nlist
do
  # Copy files from our local space on each node back to
  # my working directory creating a subdirectory for each node.
  mkdir -p $ODIR/$i
  scp -r $USER@$i:$JOBTMP/$NEW_DIR/* $USER@mio.mines.edu:$ODIR/$i
  # or
  # ssh -r $USER@$i cp -r $JOBTMP/$NEW_DIR/* $PBS_O_WORKDIR/$i

  # Remove the temporary directory
  ssh $i rm -r $JOBTMP/$NEW_DIR
done
```
Chaining Jobs
Running jobs in sequence

- In theory a pbs script can submit another script
- One script creates a second then runs it
- Most systems don’t support submission from compute nodes
- Must run from primary compute node
  - `ssh mio.mines.edu “cd rundir ; qsub next_run”`
- In most cases it is better to use the qsub option
  - `-W depend=...`
Depend section of the qsub man page

-W depend=dependency_list

The dependency_list is in the form:
-type[:argument[:argument...]][,type:argument...].

after:jobid[:jobid...]  
This job may be scheduled for execution at any point after jobs jobid have started execution.

afterok:jobid[:jobid...]  
This job may be scheduled for execution only after jobs jobid have terminated with no errors.

before:jobid[:jobid...]  
When this job has begun execution, then jobs jobid... may begin.

beforeok:jobid[:jobid...]  
If this job terminates execution without errors, then jobs jobid... may begin.

beforenotok:jobid[:jobid...]  
If this job terminates execution with errors, then jobs jobid... may begin.

beforeany:jobid[:jobid...]  
When this job terminates execution, jobs jobid... may begin.

qsub -W depend=afterok:123.big.iron.com /tmp/script
Qsub - requesting specific nodes

- qsub normally gives you any node
- Can select nodes based on properties
- Can select nodes based on name

qsub -l nodes=1:lcarr
qsub -l nodes=1:lcarr:core12
qsub -l nodes=n3:ppn=8+n64:ppn=12

Mio Node Property sets

- properties = compute,x5500,core8,cmmaupin,m2933
- properties = compute,x5500,core8,dmunoz,m2933
- properties = compute,x5500,core8,kazemi,m2933
- properties = compute,x5500,core8,lcarr,m2933
- properties = compute,x5500,core8,mlusk,m2933
- properties = compute,x5500,core8,tkaiser,m2933
- properties = compute,x5500,core8,zhiwu,m2933
- properties = compute,x5600,core12,anewman,m2933
- properties = compute,x5600,core12,asum,m2933
- properties = compute,x5600,core12,asum,m3066
- properties = compute,x5600,core12,cciobanu,m2933
- properties = compute,x5600,core12,cciobanu,m3066
- properties = compute,x5600,core12,dmunoz,m3066
- properties = compute,x5600,core12,geco,m3066
- properties = compute,x5600,core12,lcarr,m3066
- properties = compute,x5600,core12,mganesh,m2933
- properties = compute,x5600,core12,mlusk,m3066
- properties = compute,x5600,core12,mooney,m2933
- properties = compute,x5600,core12,psava,m2933
- properties = compute,x5600,core12,psava,m3066
- properties = gpu,x5300,core8,cuda,gpu1
- properties = gpu,x5600,core12,cuda, gpu2
A Digression, Not a Batch Script...

This script prints your nodes if running in batch or all nodes if running on the front end nodes.

```
#!/bin/bash

# $PBS_NODEFILE defined? (running in batch)
if [ -n "$PBS_NODEFILE" ] ; then
    # yes defined, test if file exists
    if [ -a $PBS_NODEFILE ] ; then
        # yes exists, write it
        cat $PBS_NODEFILE
    else
        # $PBS_NODEFILE defined but does not exist, this should not happen
        echo $PBS_NODEFILE not found
    fi
else
    # $PBS_NODEFILE not defined
    # this implies we are not running in batch
    # list all the nodes
    pbsnodes | grep "^[a-zA-Z]"
fi
```
#!/bin/bash

#PBS -l nodes=1:ppn=8
#PBS -l walltime=00:02:00
#PBS -N testIO
#PBS -o out.$PBS_JOBID
#PBS -e err.$PBS_JOBID
#PBS -V
#PBS -m abe
#PBS -M tkaiser@mines.edu

#----------------------

cd $PBS_O_WORKDIR

# set up our redirects of stdout and stderr
# 1 and 2 are file descriptors for
# stdout and stderr
# 3 and 4 are descriptors to logfile
# we will use 3 for stdout 4 for stderr
exec 3>>logfile.\`date \"+%y%m%d%H%M%S\"\`
    # anything that goes to 4 will go to 3
    # which is our file we have created
exec 4>&3
exec 5>&1 6>&2              # save "pointers" to stdin and stdout
exec 1>&3 2>&4              # redirect stdin and stdout to file

# normal commands
# this line goes to stdout
echo this is a test from stdout
# this line goes to stderr
echo this is a test from stderr >&2
    # error message goes to stderr
ls file_that_does_not_exist
ls
mpiexec -n 8 ./c_ex00 > myout.$PBS_JOBID
mpiexec -n 8 ./c_ex00

# restore original stdin and stdout
exec 1>&5 2>&6              # close logfile descriptors
5>&- 6>&-                   # close saved stdin and stdout
exec 1>&& 2>&&
With that we say good day