Advanced Scripting Using PBS Environment Variables

Your job submission script has a number of environment variables that can be used to help you write some more advanced scripts. These variables can make your code more portable and save you time.

Variables listed by functionality

The following list are some of the more useful environment variables available for you to use in your scripts:

General Useful Environment Variables

Variable Name	Description
USER	User Name (NetID). Useful if you would like to dynamically generate a directory on some scratch space.
HOSTNAME	Name of the computer currently running the script.This should be one of the nodes listed in the file $PBS_NODEFILE.
HOST	Same as $HOSTNAME.

Directories

Variable Name	Description
HOME	User Home directory (typically /mnt/home/$USER). You can also use the simple short cut '~' to refer to your home directory.
PWD	Current directory that the script is running in. The value of this environment variable will change as you change directories. When your jobs start, the current directory will be your $HOME directory.

PBS_O_WORKDIR	Directory where the qsub command was executed. Useful with the cd (change directory) command to change your current directory to your working directory.
TMPDIR	Local temporary disk storage (typically /mnt/local/$PBS_JOBID) unique to each node and each job. This directory is automatically created at the beginning of the job and deleted at the end of the job.

Queues

Variable Name	Description
PBS_O_QUEUE	Queue job was submitted to.
PBS_QUEUE	Queue job is running in (typically this is the same as PBS_O_QUEUE).

Job Information

Variable Name	Description
PBS_JOBID	Job ID number given to this job. This number is used by many of the job monitoring programs such as qstat, showstart, and dque.
PBS_JOBNAME	Name of the job. This can be set using the -N option in the PBS script (or from the command line). The default job name is the name of the PBS script.
PBS_NODEFILE	Name of the file that contains a list of the HOSTS provided for the job.
PBS_ARRAYID	Array ID numbers for jobs submitted with the -t flag. For example a job submitted with #PBS -t 1-8 will run eight identical copies of the shell script. The value of $PBS_ARRAYID will be an integer between 1 and 8.
PBS_VNODENUM	Used with pbsdsh to determine the task number of each processor. For more information see http://www.ep.ph.bham.ac.uk/general/support/torquepbsdsh.html.
PBS_O_PATH	Original PBS path. Used with pbsdsh.
PBS_NUM_PPN	Number of cores requested (per node)

Examples

Probably the most useful aspect of these variables is that you can make your jobs more portable and easier to run. This is very useful if you want to run different jobs from different directories, reuse code or give your code to other users to try out.

Example 1: Running from the PBS working directory

Most users run their jobs from a subdirectory that contains their code, input data and submission scripts. However, when your job runs, the starting directory is your HOME directory. So, typically one of the first lines in your submission script is to change the directory to the one with your code. For example,

```
#!/bin/bash -login
cd /mnt/home/username/workingdirectory/
myprogram
```

This should work fine. However, if you make a copy of your code and put it in workingdirectory2 you then need to change your submission script to match the new directory. This is fine, but annoying and you do not want to constantly keep track of what directory you are working from. Instead, you can replace your hard coded working directory with the **PBS_O_WORKDIR** environment variable. As long as your submission script (myscript.sh) is in your working directory you will be able to run correctly and you can make many copies of your submission script without having to change the directory.

myscript.sh

```
#!/bin/bash -login
cd $PBS_O_WORKDIR
myprogram
```

Example 2: Running multiple copies of the same job at the same time

For some experiments, it is important to run the same job more than once (this is especially true if a random number generator is used and you need to average the results). Consider the following example:

```
#!/bin/bash -login
cd $PBS_O_WORKDIR
myprogram 1> myprogram.out 2> myprogram.err
```

The problem with this code is that if you qsub the job more than once, results from one job overwrite the results from a previous job. The way PBS solves this problem is by using the unique job ID to label the standard output and standard error. You can use the **PBS_JOBID** variable to do the same thing in your script. This variable contains the Job ID that is displayed when you run the qsub command. Each job ID is unique so it is a very easy way to generate a unique file name or directory name. For example:

myscript.sh

```
#!/bin/bash -login
cd $PBS_O_WORKDIR
myprogram 1> $PBS_JOBID.out 2> $PBS_JOBID.err
```

Here is another example where we create a directory to place all of our programs output:

myscript2.sh

```
#!/bin/bash -login
cd $PBS_O_WORKDIR
mkdir $PBS_JOBID
myprogram 1> $PBS_JOBID/myprogram.out 2> $PBS_JOBID/myprogram.err
```

With either of these modifications, you should be able to run your job as many times as you want without the output overwriting itself.

Example 3: Utilizing scratch space on code given to other users

It can be difficult to share your code with other users and have it work properly. This is especially true if you are using the scratch file space and have hard coded the directory. Consider the following example that has a hypothetical program that can take an input argument called workingdir:

```
#!/bin/bash -login
cd $PBS_O_WORKDIR

mkdir /mnt/scratch/myprogram_scratch_space
myprogram -workingdir /mnt/lustre/myprogram_scratch_space 1> $PBS_JOBID/myprogram.out 2> $PBS_JOBID
/myprogram.err
```

This will result in the similar problem as hard coding your output file names, even though the outputs are directed to different files by the **PBS_JOBID** variable. If two users use the above code at the same time they will be using the same scratch space. You could solve the problem using the **PBS_JOBID** again. For example

myscript.sh

```
#!/bin/bash -login
cd $PBS_O_WORKDIR

mkdir /mnt/scratch/myprogram_scratch_space
mkdir /mnt/scratch/myprogram_scratch_space/$PBS_JOBID
myprogram -workingdir /mnt/scratch/myprogram_scratch_space/$PBS_JOBID 1> $PBS_JOBID.out 2> $PBS_JOBID.err
```

This will work by putting all of the jobs in subdirectories under the scratch space. However, now you have the problem of remembering which users go with which job id. A second solution uses the **USER** name to make user specific scratch spaces. For example:

myscript2.sh

```
#!/bin/bash -login
cd $PBS_O_WORKDIR

mkdir /mnt/scratch/myprogram_scratch_space
mkdir /mnt/scratch/myprogram_scratch_space/$USER
myprogram -workingdir /mnt/scratch/myprogram_scratch_space/$USER 1> $PBS_JOBID/myprogram.out 2> $PBS_JOBID
/myprogram.err
```

You could clean this code up even further by defining your own working variables. For example:

myscript3.sh

```
#!/bin/bash -login
cd $PBS_O_WORKDIR

SCRATCH=/mnt/scratch/myprogram_scratch_space
mkdir $SCRATCH

mkdir $SCRATCH/$USER
myprogram -workingdir $SCRATCH/$USER 1> $PBS_JOBID/myprogram.out 2> $PBS_JOBID/myprogram.err
```

This same technique can also be used if your code is taking advantage of /mnt/local disk space on the compute nodes.

Example 4: Running the same code using different inputs.

It is common for users to want to run copies of their jobs over different input files. Consider the example where you have the following inputfiles:

```
data1.in
data2.in
data3.in
data4.in
data5.in
.
.
.
data110.in
```

And you have a program that takes each of these inputs and generates an output file using the following command:

```
#!/bin/bash -login
cd $PBS_O_WORKDIR
myprogram < data1.in 1> data1.out 2> data1.err
```

Making a new copy of the script and then submitting each one for every input data file is time consuming. An alternative is to make a job array using the -t option in your submission script. The -t option allows many copies of the same script to be queued all at once. You can use the **PBS_ARRAYID** to differenciate between the different jobs in the array.

arrayscript.sh

```
#!/bin/bash -login
#PBS -t 1-8
cd $PBS_O_WORKDIR
myprogram < data${PBS_ARRAYID}.in 1> data${PBS_ARRAYID}.out 2> data${PBS_ARRAYID}.err
```

Other variables

the **env** command will list all of the environment variables currently set on your system. The following job script will who all of the variables available within a job (by selecting only those with PBS in the name):

environment.sh

```
#!/bin/bash -login
#PBS -l nodes=1:ppn=1,walltime=00:00:10
env | grep PBS
```

Submitting the above job script using qsub results in output similar to the following. Some of these variables may or may not be useful to you:

```
PBS_VERSION=TORQUE-2.3.2
PBS_JOBNAME=environment.sh
PBS_ENVIRONMENT=PBS_BATCH
PBS_O_WORKDIR=/mnt/home/colbrydi/Testing/NewTests
PBS_TASKNUM=1
PBS_O_HOME=/mnt/home/colbrydi
PBS_MOMPORT=15003
PBS_O_QUEUE=main
PBS_O_LOGNAME=colbrydi
PBS_O_LANG=en_US.UTF-8
PBS_JOBCOOKIE=EB20CE58D95293D4A3DD4EBE4D91C280
PBS_NODENUM=0
PBS_O_SHELL=/bin/bash
PBS_SERVER=dev-intel07-00.i
PBS_JOBID=3335811.cmgr01
PBS_O_HOST=dev-intel07-00.i
PBS_VNODENUM=0
PBS_QUEUE=main
PBS_O_MAIL=/var/mail/colbrydi
PBS_NODEFILE=/var/spool/PBS/aux//3335811.cmgr01
PBS_O_PATH=/opt/hpc/svn/1.5.0/bin:/opt/torque/sbin:/opt/torque/bin:/opt/moab/sbin:/opt/moab/bin:/opt
/totalview/bin:/opt/hpc/mpiexec/mpiexec-0.82p1/bin:/opt/h
pc/mvapich/0.9.9-intel/bin:/opt/abaqus/Commands:/opt/hpc/Fluent.Inc/bin:/opt/matlab/bin:/opt/intel/fce/10.
0.025/bin:/opt/intel/cce/10.0.025/bin:/usr/local/bi
n:/usr/bin:/usr/X11R6/bin:/bin:/usr/games:/opt/gnome/bin:/opt/kde3/bin:/usr/lib/mit/bin:/usr/lib/mit/sbin:
/usr/local/ofed/bin:/usr/local/ofed/sbin:.:/opt/sca
li/bin:/opt/scali/sbin:/usr/pbs/bin:/mnt/home/colbrydi/Internal/
```

More information about these variables can be found at http://www.clusterresources.com/torquedocs21/users/2.1jobsubmission.shtml