CCAST User Guide

Version: **2019.01**. Last updated: 01/22/2019

1. Introduction, Context, and Qualifications

The Center for Computationally Assisted Science and Technology (CCAST; pronounced "*c-cast*") provides high-performance computing (HPC) resources, training, and consulting to NDSU faculty, students, and staff and their collaborators. We use Linux primarily. The basic level of services is free of charge to NDSU researchers and their external collaborators. Additional services are available for a fee.

For more information, see *www.ccast.ndsu.edu*.

1.1 Acknowledging CCAST

Users are required to use the following statement (or a close variant) to acknowledge CCAST in all research outputs that have used CCAST resources: "*This work made use of computing resources at the Center for Computationally Assisted Science and Technology (CCAST), North Dakota State University*."

1.2 Reporting requirements

Users, usually through their Principal Investigators (PIs) or group leaders, will be requested to report any research output and activities that have been enabled by the use of CCAST resources. Reporting items often include publications, presentations, grant applications, patents, theses, etc.

1.3 CCAST usage policies

Users are required to carefully read and comply with CCAST Usage Policies. A full copy of the Usage Policies can be found on the CCAST website: *https://www.ccast.ndsu.edu/users/ccast-usage-policies*

1.4 How you can get help?

Please read this User Guide carefully and check the CCAST website before contacting us. If you still cannot find answers to your questions, send an e-mail to ***support@ccast.ndsu.edu***. In the e-mail, describe the issues, clearly state your questions, and provide a copy of the error messages and PBS job scripts, the IDs of your failed jobs, and any other info that may help debug the issues. Please do not directly contact CCAST individual staff for technical support as this bypasses our tracking system to avoid dropped calls.

1.5 About this document

This document will be updated often. A PDF copy of the latest version of the CCAST User Guide can always be found in the following directory on Thunder cluster: */gpfs1/projects/ccastest/training/tutorials*

2. Getting Started

2.1 Applying for an account

To be able to use CCAST's "Thunder"–an HPC cluster–you need to have an account with us. Please apply for a CCAST account if you have not already done so. Check out user eligibility requirements on the CCAST website and fill out an online form: *https://www.ccast.ndsu.edu/users/account-application/*

2.2 Connecting to the Thunder cluster

From a Windows computer: PuTTY, a free SSH and telnet client, should be used. Download (from here: *https://www.putty.org*) and install it, then double-click to open the application. In the "Host Name (or IP address)" field, enter the hostname: *thunder.ccast.ndsu.edu* Select (or leave) *22* for "Port" and *SSH* for "Connection type". Click "Open", you will be asked to enter your username and password.

From a Mac/Linux computer: Open a terminal and then execute the following line to access Thunder: *ssh thunder.ccast.ndsu.edu -l username* You will be prompted to enter your username and password.

2.3 Transferring files

Between a Windows computer and Thunder: WinSCP client should be used. Download for free (from here: *https://winscp.net*) and install it, then open the application. In the "WinSCP Login" window, enter the hostname *thunder.ccast.ndsu.edu* as well as your username and password, then click on "Login".

Once logged in, you will see a screen with two panels: the left shows files on your computer and the right shows your files on Thunder (usually your ***home*** directory, but you can double-click on the address bar and change the location). You can then easily drag and drop files between your computer and Thunder.

Between a Mac/Linux computer and Thunder: To transfer files from Thunder to your computer:

scp [[username@hostname]:[source-file]] [[destination]].

Example: *scp username@thunder.ccast.ndsu.edu:/gpfs1/home/username/myfile.txt /home/mycomputer/myfile.txt*

To transfer files from your computer to Thunder:

scp [[source-file]] [[username@hostname]:[destination]]

Example: *scp myfile.txt username@thunder.ccast.ndsu.edu:/gpfs1/home/username*

2.4 Learning Unix/Linux and HPC

Users are strongly recommended to attend Advanced Research Computing Training sessions, offered by CCAST every semester, as well as CCAST User Group Meetings and other special local training events. Specialized training for individual researchers/research groups is also available. Contact us for more info.

There are also lots of free training materials out there on the Internet. We recommend the following:

+ Unix/Linux Tutorial for Beginners: *http://www.ee.surrey.ac.uk/Teaching/Unix/*

+ HPC Training Materials at LLNL: *https://hpc.llnl.gov/training/tutorials*

See also the attached **CCAST Reference Card** for a list of the most useful Linux commands and tricks. Tutorials for certain applications on Thunder can be found in */gpfs1/projects/ccastest/training/tutorials*

3. Research Computing Resources

3.1 Hardware

CCAST's Thunder has over 100 nodes (>3,000 cores); most have 20 cores (63GB RAM) or 44 cores (100GB RAM) per node. There are also 3 big-memory (1TB RAM) nodes and 2 GPU nodes (12 cards).

To check which nodes are currently free or partially free on Thunder, execute the command: *freenodes* The information will help you make the right choice when you request computing resources for your jobs.

3.2 Software

There are many software programs installed on Thunder. Most are available to all CCAST users; some, e.g., ANSYS, Gaussian, VASP, etc., available only to those who have valid licenses and other authorized users. Software are usually organized as modules; to check available modules, execute: *module avail*

You can also install software for yourself. Contact us at *support@ccast.ndsu.edu* if you need help.

3.3 Storage space

Once logged in, you are in your ***home*** directory (*/gpfs1/home/username*). ***/home*** data is backed up to tape, so it is a reliable data storage area. Do not use your ***home*** directory for data or job input/output. Running jobs out of ***/home*** is not permitted as it affects the interactive use and other important jobs on the system.

Each research group usually has a ***projects*** directory; the full path is */gpfs1/projects/PI-username*, where *PI-username* is the username of the Principal Investigator (PI). This area has a larger storage space and is backed nightly to tape. All researchers working under the PI can store and share data in this project space.

Each regular user has a ***scratch*** directory (*/gpfs1/scratch/username*). It is designed as a place for working directories for jobs. Please submit your jobs from this directory. Note that ***scratch*** data is NOT backed up, and CCAST reserves the right to delete files as necessary (a **60-day** maximum is the current target).

More storage space (beyond the basic level) is available for a fee. Contact CCAST if you have questions.

3.4 Compute Condominium

Researchers can purchase condo nodes using equipment purchase funds from their grants or other available funds. These PI-owned compute nodes are attached to CCAST's Thunder cluster to take advantage of the existing infrastructure. Contact CCAST if you have questions regarding the condominium model.

4. Running Jobs

Once you logged in to CCAST's Thunder, you are on one of its login nodes. Login nodes have limited resources and are intended only for basic tasks such as transferring data, managing files, compiling software, editing scripts, and checking on or managing jobs. DO NOT run your jobs on the login nodes!

Jobs must be submitted to a queue system, which is monitored by a job scheduler, using a job script. The job scheduler currently used on the Thunder cluster is PBS Professional (PBS Pro). The scheduler handles job submission requests and assigns jobs to specific compute nodes available at the time.

To be able to run your jobs and run them efficiently, you need to have some basic knowledge of the application you are using. This includes whether the application is serial (i.e., runs on only one core) or parallel (i.e., can run on multiple cores). If it is parallel, what is the underlying parallel programming model: shared-memory (e.g., using OpenMP, Pthreads, etc.), distributed-memory (e.g., using MPI), or hybrid? You need such information to determine how you would like to request resources for your jobs.

4.1 Sample input files and job scripts

If you are new to running jobs on the Thunder cluster or if it has been a while since the last time you ran an application, it is highly recommended that you first run some sample jobs we provide before running your own jobs. On Thunder, users can copy sample input files and job scripts for various applications from */gpfs1/projects/ccastest/training/examples* More job examples for more applications will be added as they become available. Please check this directory frequently for the latest version of the job scripts.

A job script (also referred to as a "PBS job script") to run a serial job is given below as an example:

```
#!/bin/bash
#PBS -q default
#PBS -N test
#PBS -l select=1:ncpus=1
#PBS -l walltime=08:00:00
#PBS -W group_list=x-ccast-prj-prjname
cd $PBS_O_WORKDIR
./my-serial-program
```

Note: You need to replace `prjname` with the actual project group name of your PI. If you do not know your PI's `prjname`, on Thunder, execute the command *id* and look for the group name *x-ccast-prj-...*

A PBS job script is simply a text file in your working directory. The easiest way to create the file is to copy an appropriate sample PBS job script from */gpfs1/projects/ccastest/training/examples* on Thunder and then modify it as needed using some text editor such as `nano` (for novice Linux users), `emacs`, or `vi` (for more experienced users). See also the **PBS Pro Cheat Sheet** attached to this CCAST User Guide.

4.2 Queue policies on Thunder

Different types of queues are given below. Users can also find info about the queues by executing *qstat -q*

Route Queue	Execution Queue	Walltime (hours)	Authorized Group
`default`	`def-short`	24	All users
	`def-medium`	72	
	`def-long`	168	
`def-devel`		8	
`preemptible`		--	
`bigmem`	`bm-short`	24	
	`bm-long`	168	
`condo01`, `condo02`, etc.		--	Condo owners

If a route queue is given in the job script (e.g., `default`), the job will automatically be assigned to an appropriate execution queue based on the requested walltime (e.g., `def-short` in the earlier example).

4.3 Launching and monitoring jobs

After preparing a suitable job script (with the filename *job.pbs*, for instance), see Sec. 4.1, you can submit the job by typing: *qsub job.pbs*. This will assign your job to the queue. Depending on the available resources, it may or may not start immediately. To check the status of your job(s), type: *qstat -u $USER*. If you want to kill the job, use the command *qdel <jobid>*, where *<jobid>* is the ID of the job you want to kill. For more useful PBS Pro commands and options, see the attached **PBS Pro Cheat Sheet**.

4.4 How to get your work done faster?

If you use software packages developed by others, be mindful of the parameters used in your input files. A small tuning of the parameters can significantly improve computational efficiency. If you write and run your own code, see if it can be optimized to make it run faster or parallelize it if it is not yet parallel.

When running parallel jobs, a question arises: How many cores/nodes should you request for the jobs? Note: the requested resources in the sample PBS job scripts we provide are not optimized for your jobs! Also note that, if you want to get your jobs done faster, simply adding a lot more cores/nodes is rarely the answer! You should do some scaling tests to identify the optimal number of cores/nodes for your jobs.

When you have many similar parallel jobs, we recommend that you run a first few jobs with different numbers of cores/nodes. By looking the computing time needed to finish the jobs vs. the number of cores/nodes, you'll have a pretty good idea of how many cores/nodes you should choose for the remaining jobs.

Contact CCAST for help with improving your job efficiency and speeding up your research process.

5. Utilization Monitoring

We use XDMoD for data collection and monitoring of HPC resource utilization. The tool allows CCAST staff, PIs, and users to view data about their CCAST usage. It includes metrics like total CPU hours, number of jobs submitted, average walltime per job, and much more. Information is updated daily for all jobs completed at the time of update. The link to this service is *https://xdmod.ccast.ndsu.edu*

NDSU CENTER FOR COMPUTATIONALLY ASSISTED SCIENCE AND TECHNOLOGY

CCAST Reference Card

https://www.ccast.ndsu.edu
support@ccast.ndsu.edu

Logging In

ssh secure shell
options include:
-X enables X11 forwarding
example:
ssh *user*@thunder.ccast.ndsu.edu

Transferring Files

scp secure copy
options include:
-r recursively copy entire directories
examples:
scp *myfile.txt user@hostname*:/gpfs1/home/*user*
scp -r *user@hostname*:/gpfs1/home/*user/mydir* .

winscp scp/sftp GUI for windows

Checking Resources

freenodes list currently free/partially free compute nodes

Configuring Shell Environment

module interface to modules package
options include:
avail list all available modulefiles
load load modulefile into shell environment
unload remove modulefile from shell environment
list list loaded modulefiles
display display the modulefile information
purge unload all previously loaded modulefiles
examples:
module avail
module display *intel*
module load *intel*
module list

Using the Queuing System

qsub submit job to queuing system
example:
qsub *jobscript*

qstat show status of batch jobs
options include:
-u $USER show only user's jobs
-n list nodes allocated to a job

qdel delete batch job with given job ID
example:
qdel *123456*

Useful Linux Commands

File/Directory Basics

ls list directory contents
examples:
ls -ltr long listing, most recently modified last
ls -h file sizes in readable format e.g. 1K, 21G

pwd print working directory

echo display a line of text
examples:
echo $HOME display user's home directory
echo $PATH display user's search path

cd change current directory
examples:
cd .. change to directory above
cd */path/to/dir* change to directory given in path
cd $HOME change to user's home directory
cd $SCRATCH change to user's scratch directory

cp copy files and directories
examples:
cp *file1 file2* create a copy of *file1* called *file2*
cp -r *dir1 dir2* recursively copy *dir1*

mv move (rename) files and directories
examples:
mv *file1 file2* rename *file1* as *file2*
mv *dir1 /new/path* move *dir1* to a new location

rm remove files or directories
examples:
rm -i *file1* prompt before deleting *file1*
rm -rf *dir1* recursively & forcefully remove *dir1*

mkdir make directories

rmdir remove empty directories

ln make links between files and directories
example:
ln -s */path/to/dir1 ./dir1* symbolically link to *dir1*

Viewing & Manipulating Text Files

head output the first part of files
example:
head -7 *file.txt* view first 7 lines of *file.txt*

tail output the last part of files
example:
tail -7 *file.txt* view last 7 lines of *file.txt*

cat concatenate files and print to stdout
example:
cat *file1.txt* >> *file2.txt* append *file2.txt* to *file1.txt*

wc *file.txt* print line, word and byte counts

diff *file1.txt file2.txt* compare files, line by line

cut print selected parts from each line of files
example:
cut -d',' -f1,2 *file.csv* print first two columns of file.csv

paste merge lines of files
example:
paste *file1.txt file2.txt* concatenate each line of *file1.txt* and *file2.txt*, in turn, and print

sort sort lines of files
examples:
sort -d *file1.txt* print contents of *file1.txt* in dictionary order
sort -nr *file1.txt* print contents in reversed (descending) numerical order

uniq report or omit repeated lines
example:
uniq *file1.txt* print only unique lines of *file1.txt*

sed stream editor for filtering and transforming
examples:
sed 's/*cat*/*bat*/g' *file1.txt* replace all instances of '*cat*' in *file1.txt* with '*bat*'
sed 's/*ed//g' *file1.txt* replace all words in *file1.txt* ending with '*ed*' with the empty string

awk pattern scanning and processing language
example:
awk '{print $2}' *file1.txt* print second column of *file1.txt*

nano text editor (for novice Linux users)

emacs text editor

vi text editor (highly recommended)

Redirection and Pipelines

> redirect stdout
example:
cat *file1 file2* > *file1-and-2*

< redirect stdin

>> redirect stdout and append
example:
cat *file1-and-2 file3* >> *file-1-and-2-and-3*

| pipe stdout from one cmd to stdin of another
example:
head -7 *file1* | tail -1 view 7th line of *file1*

Viewing Other Files

od dump files in octal and other (e.g. binary) formats

nm list symbols from object (& library) files
example:
nm *mylib.a* | less view symbols in *mylib.a*, 1 page at a time

ldd report shared library dependencies
example:
ldd *myprog.exe* view *myprog.exe*'s dependencies

File Properties

file determine file type

touch change file timestamps
example:
touch *file1* updates access and modification times of *file1* to the present time

chmod change file mode bits
example:
chmod a+r *file.txt* allow all to read *file.txt*

chown change file owner and group

md5sum compute/check MD5 message digest

du estimate file space usage
example:
du -sh . summarize (in readable format) total usage of file-tree rooted in current dir

df report file system disk space usage
example:
df -h . report usage (including available space) for file system holding current dir

Searching for Things

grep print lines matching a pattern
examples:
grep -n '*foo*' *file.txt* print all lines (prefixing the line number) containing '*foo*' in *file.txt*
grep -i '*foo*' * print all lines containing '*foo*' (case insensitive) from all files in current

find search for files in a directory hierarchy
examples:
find . -name *test* find all *test* files in current and sub-directories
find /home -name *.*dat* find all .*dat* files in /home and its sub-directories
find . -name *test* -exec rm {} \; find and delete all *test* files in current and sub-directories

which locate a command
example:
which *gcc* report location of *gcc* compiler

whoami print effective userid

man an interface on on-line reference manuals

info read Info documents.

? wildcard: matches a single character

***** wildcard: matches any sequence of characters

Compressing and Combining

tar archiving utility
example:
tar -zcvf *archive*.tar.gz *file1 dir1 dir2* archive *file1*, *dir1*, and *dir2* into a single file and compress it
tar -xzf *archive*.tar.gz unpack compressed archive

gzip compress files
example:
gzip *file.txt* compress *file.txt*

gunzip expand files

Process Management

top display Linux tasks

kill send a signal to a process.

fg place a job in the foreground

bg place a job in the background

PBS Pro Cheat Sheet

User Commands

qsub	submit a job
qsub -I	submit an interactive job
qsub -IX	submit an interactive job with X forwarding
qstat <jobid>	job status
qstat -q	print queue information
qhold <jobid>	hold a job
qrls <jobid>	release a job
pbsnodes -a	print node information
qstat -B	cluster status
qdel	delete a job
qalter	alter a PBS job
tracejob <jobid>	print log information about a job
qselect	select PBS batch jobs

Job Monitoring

qstat -x	job history
qstat -f <jobid>	job status with all information
qstat -ans	job status with comments and vnode info

Deleting Jobs

qdel <jobid>	kill a job
qdel -Wforce <jobid>	force kill a job

Requesting Job Resources

-l select=2:ncpus=4	request 2 nodes with 4 cores each
-l select=1:ncpus=4:mem=1gb	1 node with 4 cores and 1GB RAM
-l walltime=01:00:00	request for 1 hour total wall time
-l cput=00:30:00	request for 30 minutes CPU time
-l place=pack:exclhost	request node to be exclusively allocated to the job

Job Submission Options (qsub)

-P project_name	specifying a project name
-q destination	specifying queue and/or server
-r value	marking a job as rerunnable or not
-W depend = list	specifying job dependencies
-W stagein=list stageout=list	input/output file staging
-W sandbox=<value>	staging and execution directory: user's home vs. job-specific
-a date_time	deferring execution
-c interval	specifying job checkpoint interval
-e path	specifying path for output and error files
-h	holding a job (delaying execution)
-J X-Y[:Z}	defining job array
-j join	merging output and error files
-k keep	retaining output and error files on execution host
-l resource_list	requesting job resources
-M user_list	setting email recipient list
-m MailOptions	specifying email notification
-N name	specifying a job name
-o path	specifying path for output and error files
-p priority	setting a job's priority

Environment Variables

PBS_JOBID	job identifier given by PBS when the job is submitted, created upon execution
PBS_JOBNAME	job name given by user, created upon execution.
PBS_NODEFILE	the filename containing a list of vnodes assigned to the job
PBS_O_WORKDIR	absolute path to directory where qsub is run, value taken from user’s submission environment.
TMPDIR	pathname of job’s scratch directory
NCPUS	number of threads, defaulting to number of CPUs, on the vnode
OMP_NUM_THREADS	number of threads, defaulting to number of CPUs, on the vnode
PBS_ARRAY_ID	identifier for job arrays, consists of sequence number
PBS_ARRAY_INDEX	index number of subjob in job array
PBS_JOBDIR	pathname of job’s staging and execution directory on the primary execution host

For a more comprehensive reference, see PBS Professional 14.2 User’s Guide
https://www.pbsworks.com/pdfs/PBSUserGuide14.2.pdf