DUG McCloud User Manual

dugref CEUR-101

Last updated 17 February 2021

www.dug.com

Table of Contents

Introduction

Welcome to DUG McCloud.

This document introduces DUG McCloud and includes essential information on most features.

Please refer to our ***Connecting to DUG McCloud*** document available here.

General details: *(example Imperial user001, for this document)*

- Data centre: **perth.mccloud.dug.com**
- Group: **mcc_iclmsc**
- Username: **iclmsc_user001**
- SSH command:

  ssh -J iclmsc_user001@perth.mccloud.dug.com iclmsc_user001@mcc_iclmsc

Support

Send messages in your DUG Slack Channel

Your dedicated Slack channel is: **mcc_iclmsc** To join, click here or contact your Imperial Instructor.

Post your questions and comments on the channel, our IT and HPC Experts will assist.

Email DUG Support Team

DUG provides 24x365 support for high priority issues affecting DUG McCloud users. Support for lower priority issues is provided 24x5, from 9:00 AM Western Australia Standard Time (UTC+8) on Monday through 5:00 PM Central Time (UTC-5 / UTC-6) on Friday. Support during weekend hours is provided at DUG's discretion.

Contact DUG Support via at email: support@dug.com

Environment

HARDWARE

Head node

DUG provides a head node (login node) for Imperial users. The head node is your entry point to DUG McCloud and is suitable for small tasks such as viewing logs, editing scripts, file management, and submitting jobs to the cluster.

Tasks that require significant cpu or memory resources must not be run on the head node. They should be submitted to the cluster.

Compute nodes

DUG maintains various different types of nodes for Imperial users to work with including: Intel Xeon Phi KNLs, Intel Xeon CPUs, Dual Cascade Lake CPUs and GPU nodes. Specific node configurations are appropriate for different workloads.

Intel Xeon Phi KNLs	Most nodes in DUG McCloud contain single-socket, Intel Xeon Phi processors (Knights Landing, aka “KNL”). These x86 processors have 64 cores with four-way hyper-threading, 16GB of high-bandwidth memory and 128 AVX-512 vector units. Each node has at least 128 GB of RAM. Some KNL nodes at DUG have extra processor cores (68 instead of 64). For some workloads, one KNL node can run 256 (4 * 64) threads efficiently.
Xeon CPUs	Intel Xeon E5-2640 nodes are also available for this course. Each CPU has 10 cores, with 2 threads per core.
Dual Cascade Lake CPUs	A small number of dual Intel Xeon Cascade Lake machines have been included for this course. Each node contains 2 Cascade Lake SP processors providing a total of 2 * 20 cores * 2 threads per core.
GPUs	GPU machines are equipped with dual Intel Cascade Lake CPUs and 4 x NVIDIA V100 cards with 32GB RAM on each GPU.

SOFTWARE

Headnode and compute nodes run CentOS 7.

Installed software is managed using environment modules (see official documentation). Many packages are available, with multiple versions of each.

To list available packages, use “module avail”. Include additional text to filter the output. For example, to show available versions of OpenMPI:

```
[user@host ~]$ module avail openmp

--------------------------------------------------
/d/sw/Modules/modulefiles
--------------------------------------------------
openmpi/1.4.4          openmpi/2.0.1          openmpi/2.1.0-mt
openmpi/3.0.0-mt       openmpi/4.0.3-mlnx
openmpi/1.6.5(default) openmpi/2.0.1-debug    openmpi/2.1.0-mt-debug
```

```
openmpi/4.0.0          openmpi/latest
openmpi/1.6.5_el5      openmpi/2.0.2-mt       openmpi/2.1.1-mt
openmpi/4.0.1
openmpi/1.7a1r26501    openmpi/2.0.2rc4       openmpi/2.1.2-mt
openmpi/4.0.3-el6
```

To make a package visible in your current environment, use “module add” and the package name / version:

```
[user@host ~]$ module add openmpi/latest
```

To show all currently loaded models and their versions:

```
[user@host ~]$ module list
Currently Loaded Modulefiles:
  1) dugeo-base          3) slurm/latest          5) java64/14.0.2
  2) gcc/9.2.0           4) iclmsc-base           6) openmpi/4.0.5-mlnx-gcc
```

To see the path to the loaded version:

```
[user@host ~]$ which mpirun
/d/sw/openmpi/latest/bin/mpirun
```

To remove a package from your current environment, use “module remove”:

```
[user@host ~]$ module remove openmpi
```

To load packages automatically, edit your shell profile (~/.bashrc) and include the “module add” commands. Be careful editing your shell profile (eg. ~/.bashrc), as it includes settings essential for logging in and using the system. It is recommended to open a second login after making any profile changes but *before* logging out of the first, to ensure you don’t accidentally lock yourself out of the system.

If you need assistance or require additional software, contact support@dug.com or send messages in your DUG Slack channel. We can download, compile and install packages from the web or from your storage area.

Data Storage

HPC storage at DUG is hosted on a high-performance flash-based storage system.

USER HOME DIRECTORIES

Every Imperial user is allocated a home directory (/d/home/**mcc_iclmsc**/**iclmsc_user**).

Use the home directory to store small files and data that does not require sharing with other users. Home directory space is limited, so keep large files and datasets on the storage cluster.

Do not keep job files or scripts in your home directory. All jobs should be run from the cluster storage area.

IMPERIAL CLUSTER STORAGE

Your personal storage area is **/p9/mcc_iclmsc/user**.

Most of your files and jobs should live here. This directory is on our high-performance server and can be reached by the cluster nodes. The structure and organisation under this area is up to you.

INDIVIDUAL STORAGE QUOTA

Each Imperial user has been allocated **10 GB** of data storage for the duration of the course. Contact Prof. Gerard Gorman to request changes to your quota.

To find out your current usage, use "teamdf *mount point*".

```
teamdf /p9/mcc_iclmsc
```

This shows:

```
Group        size   quota limit grace files quota limit grace
mcc_iclmsc   744G -         10GB  -          256  -          -      -
```

To determine the disk usage of a directory, use "du -sh *directory*".

```
[user@host ~]$ du -sh .
6.0G  .
```

Data Transfer

DUG recommends the following methods to transfer data to DUG McCloud:

For macOS and Linux users:

- **RSYNC is** a powerful tool available on most Linux and macOS systems. By default, it will only transfer files that do not already exist in the destination and it will automatically resume interrupted transfers.
- **SCP** is a simpler program that may be available on more systems

For Windows users:

- **MobaXterm** (Windows program) includes tools for transferring and working with remote files. Please refer to Connecting to DUG McCloud for instructions on how to install & set up MobaXTerm.

RSYNC

Rsync copies files and folders from a source to a destination.

General Usage

One file:

rsync *sourceFile destinationPath*

- rsync file1.txt /p9/mcc_iclmsc/user001/

Multiple files:

rsync *sourceFile(s) destinationPath*

- *rsync file1.txt file2.txt /p9/*mcc_iclmsc*/user001/*

One folder and its contents, preserve modification times (-a):

rsync -a *sourceFolder destinationPath*

- *rsync -a myProject /p9/*mcc_iclmsc*/user001/*

One folder (-a), show progress (-P), remote destination (not DUG):

rsync -aP *sourceFolder* user@host:*destinationPath*

- *rsync -aP myProject user@remoteHost.com:/files/user001/*

Rsync to DUG McCloud

DUG McCloud provides an extra layer of security via our "jumpbox". Configuring a simplified connection makes connecting much easier (see details in Connecting to DUG McCloud document).

If you have a simplified connection named "dug":

```
rsync -aP localFile dug:/p9/mcc_iclmsc/user001/
```

To transfer data without using the simplified connection, extra parameters are required:

-e "ssh -J user@datacentre"

```
rsync -aP -e "ssh -J iclmsc_user001@perth.mccloud.dug.com"
localfile  iclmsc_user001@mcc_iclmsc:/p9/mcc_iclmsc/douglasm/
```

To simplify multiple rsync copies, set the RSYNC_RSH environment variable first:

export RSYNC_RSH="ssh -J user@datacentre "

```
export RSYNC_RSH="ssh -J iclmsc_user001@perth.mccloud.dug.com"
rsync -aP localfolder1 iclmsc_user001@mcc_iclmsc:/p9/mcc_iclmsc/douglasm/
rsync -aP localfolder2 iclmsc_user001@mcc_iclmsc:/p9/mcc_iclmsc/douglasm/
```

Alternatively, connect to DUG first and “pull” the data across. E.g.

Connect to DUG:

```
ssh -J iclmsc_user001@perth.mccloud.dug.com iclmsc_user001@mmcc_iclmsc
```

Then pull the files from the remote host:

```
rsync -aP douglas@remoteHost.com:/files/project /p9/mcc_iclmsc/user001
```

SCP

SCP can be a useful tool when rsync is not available. The syntax is very similar to rsync.

scp *source(s) destination*

If you have a simplified connection named “dug”:

scp -rp localfile dug:/mount-point

```
scp -rp localFile dug:/p9/mcc_iclmsc/user001
```

To transfer data without using the simplified connection, extra parameters are required:

-J user@datacentre

```
scp -rp -J iclmsc_user001@perth.mccloud.dug.com
localfile  iclmsc_user001@mcc_iclmsc:/p9/iclmsc/douglasm/
```

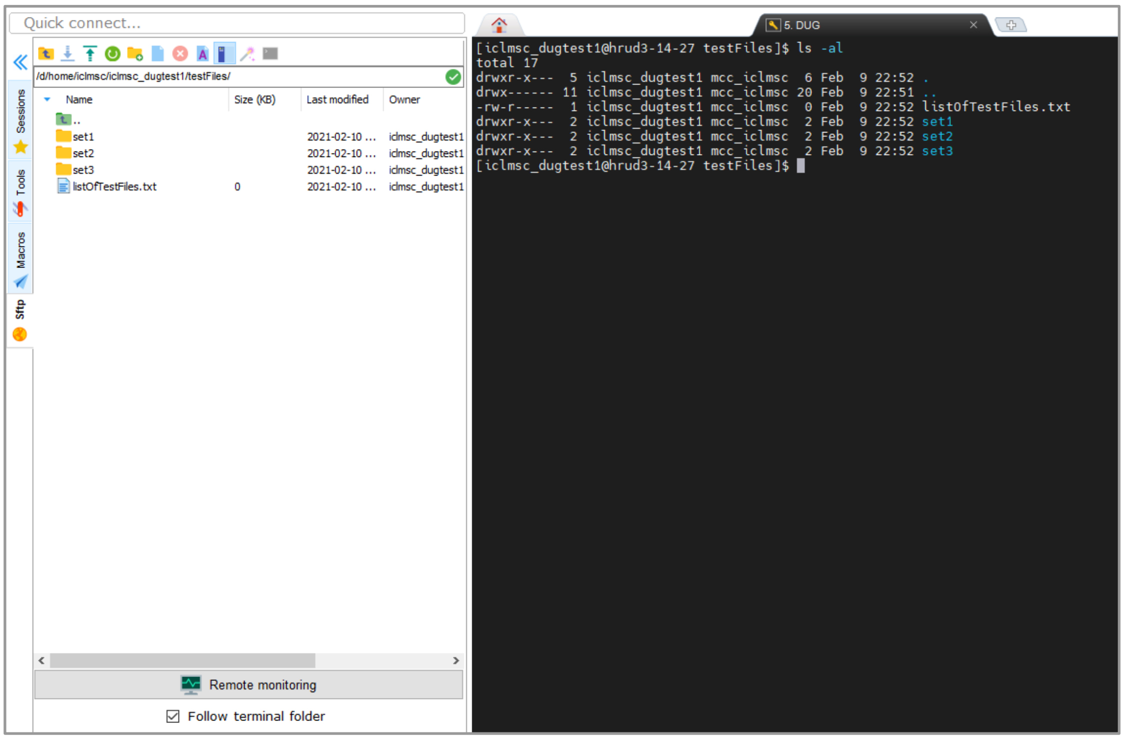
MOBAXTERM

MobaXterm includes tools for transferring and working with remote files. This is a very brief overview of the file transfer features.

Internally, MobaXterm includes a version of Cygwyn to provide SSH connections and file related features. See the MobaXterm documentation for more information if you prefer working from the command line.

To access files using MobaXterm:

1. Connect to DUG using a MobaXterm session (refer to Connecting to DUG McCloud to set up a MobaXterm session)
2. Display the Sftp (files) tab. This panel shows the files on the system you are connected to. Use this panel to transfer files between DUG and your local computer



```
/d/home/iclmsc/iclmsc_dugtest1/testFiles/
```

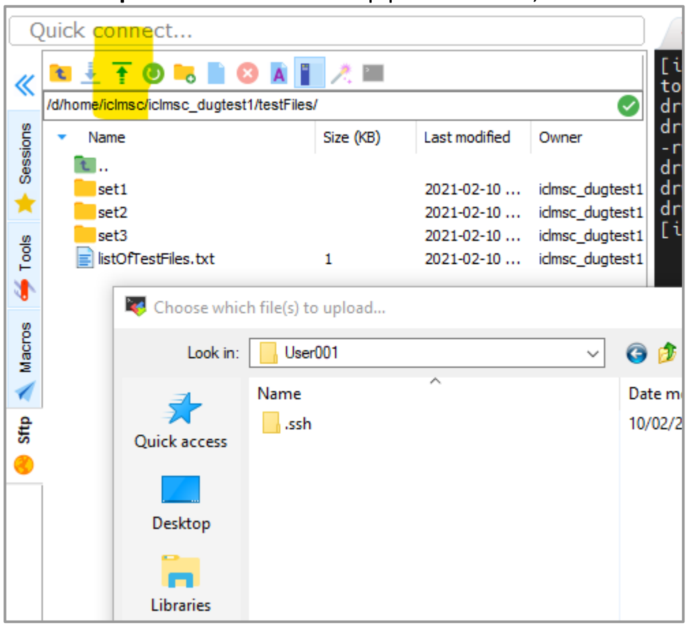
Name	Size (KB)	Last modified	Owner
..			
set1		2021-02-10 ...	iclmsc_dugtest1
set2		2021-02-10 ...	iclmsc_dugtest1
set3		2021-02-10 ...	iclmsc_dugtest1
listOfTestFiles.txt	0	2021-02-10 ...	iclmsc_dugtest1

```
[iclmsc_dugtest1@hrud3-14-27 testFiles]$ ls -al
total 17
drwxr-x---  5 iclmsc_dugtest1 mcc_iclmsc  6 Feb  9 22:52 .
drwx------ 11 iclmsc_dugtest1 mcc_iclmsc 20 Feb  9 22:51 ..
-rw-r-----  1 iclmsc_dugtest1 mcc_iclmsc  0 Feb  9 22:52 listOfTestFiles.txt
drwxr-x---  2 iclmsc_dugtest1 mcc_iclmsc  2 Feb  9 22:52 set1
drwxr-x---  2 iclmsc_dugtest1 mcc_iclmsc  2 Feb  9 22:52 set2
drwxr-x---  2 iclmsc_dugtest1 mcc_iclmsc  2 Feb  9 22:52 set3
[iclmsc_dugtest1@hrud3-14-27 testFiles]$
```

When the **Follow terminal folder** option is enabled, the files tab (on the left) will track the current path in the terminal window (on the right).

3. To upload files to the displayed folder:

   a. Click the **Upload button** in the Sftp panel toolbar, OR



```
/d/home/iclmsc/iclmsc_dugtest1/testFiles/
```

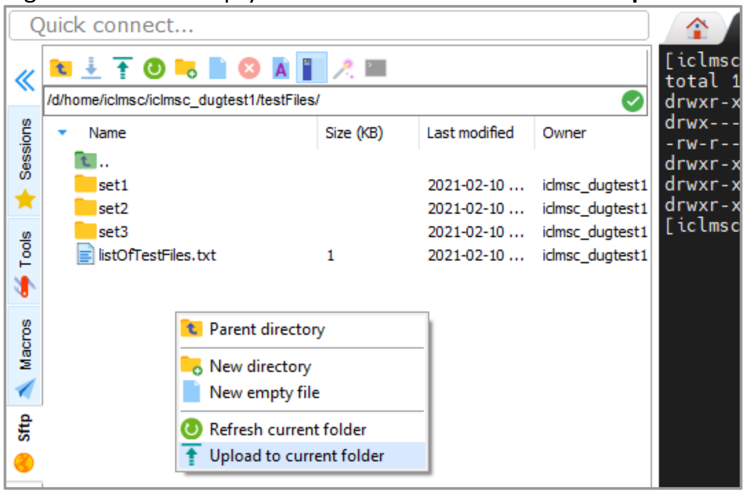
Name	Size (KB)	Last modified	Owner
..			
set1		2021-02-10 ...	iclmsc_dugtest1
set2		2021-02-10 ...	iclmsc_dugtest1
set3		2021-02-10 ...	iclmsc_dugtest1
listOfTestFiles.txt	1	2021-02-10 ...	iclmsc_dugtest1

Choose which file(s) to upload...

Look in: User001

Name	Date m
.ssh	10/02/2

b. Right-click in the empty area below the files and choose **Upload to current folder**, OR



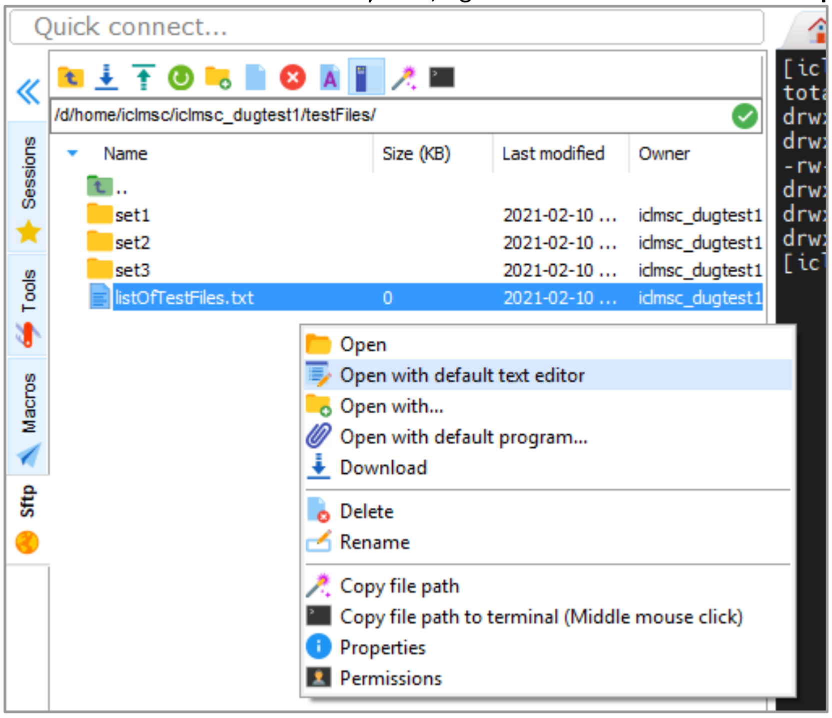
/d/home/iclmsc/iclmsc_dugtest1/testFiles/

Name	Size (KB)	Last modified	Owner
..			
set1		2021-02-10 ...	iclmsc_dugtest1
set2		2021-02-10 ...	iclmsc_dugtest1
set3		2021-02-10 ...	iclmsc_dugtest1
listOfTestFiles.txt	1	2021-02-10 ...	iclmsc_dugtest1

```
[iclmsc
total 1
drwxr-x
drwx---
-rw-r--
drwxr-x
drwxr-x
drwxr-x
[iclmsc
```

c. Drag files into the panel from a Windows explorer window.

4. To edit a text file on the remote system, right-click on the file and choose **Open with default text editor**



/d/home/iclmsc/iclmsc_dugtest1/testFiles/

Name	Size (KB)	Last modified	Owner
..			
set1		2021-02-10 ...	iclmsc_dugtest1
set2		2021-02-10 ...	iclmsc_dugtest1
set3		2021-02-10 ...	iclmsc_dugtest1
listOfTestFiles.txt	0	2021-02-10 ...	iclmsc_dugtest1

```
[ic
tota
drw
drw
-rw
drw
drw
drw
[ic
```

The file will open in an editor window. When you save the file, you will be asked whether the remote version should be updated.

Running jobs on DUG McCloud

INDIVIDUAL COMPUTE QUOTA

Each Imperial user has been allocated **10 machine hours** for the duration of the course. Contact your Imperial course instructor to request changes to your quota.

To find out your current usage, please go to the DUG McCloud Portal section.

INTRODUCTION TO SLURM AND RJS

SLURM is a workload manager. It manages the list of jobs submitted by users and schedules them to run on the compute nodes that are best suited for running them. Jobs are executed in a sequence determined by rules including priorities, dependencies, available resources and accounting.

DUG McCloud uses SLURM to schedule jobs on the available compute nodes. Rjs has been developed by DUG to simplify the job submission process. With rjs, the simplest job looks like a normal script:

```
#!/bin/bash
#rj queue=iclmsc
echo "Hello world!"
```

The line starting with #rj specifies options for the job. In this case, it includes only the name of your queue. Only nodes included in this queue will be eligible for running the job.

If this is saved to a file called **hello.job**, it can be submitted by entering:

```
$ rjs hello.job
34058333
```

The number *34058333* returned by rjs is the job's ID. Use this number to refer to the job in other commands.

By default rjs will create a logs/ directory and store the job's text output there.

Rjs also provides other quality of life features:

- timestamps are prepended to each line in the output
- jobs run from the script directory (i.e. the working directory is automatically set)
- network/io statistics are reported at the end of the job
- chained/dependent jobs are easy to configure

NOTE: THE EXAMPLE JOBS STUART MIDGLEY, DUG CIO, USED IN HIS LECTURE ARE AVAILABLE IN /P9/MCC_ICLMSC/DUG

TASK BEHAVIOR

Useful commands:

- To submit a script to the queue: `rjs script.job`
- To list the jobs in the queue: `squeue`
- To cancel a job (in any state): `scancel JOBID`
- To rerun a job (in the SE state): `scontrol release JOBID`

When submitted, jobs are queued in the pending state (PD) and wait for an available compute node. Once a job is allocated to a node, it enters the running state (R). Jobs that complete successfully (without error) will be removed from the queue.

If a job fails or an error occurs while it runs, the job will be set to the "Special Exit" state (SE). Jobs in the SE state should be investigated to understand the cause of the problem.

The full list of states can be found in the Task / Job States section.

TRANSITIONING TO RJS FROM SBATCH / SRUN

The following examples show how to submit more complex jobs using DUG rjs. Comparisons to SLURM sbatch and srun are provided for reference.

The first example is a parallel multi-task array job. The second is a multi-node MPI job.

Example: Array jobs

One approach to parallelisation is array jobs. In an array job, one script is run multiple times with each instance using different parameters. Each execution of the script is a distinct, isolated task and hundreds (or thousands) of tasks can be run simultaneously if enough nodes are available. There is no communication between the tasks.

In this example, an array job outputs data (using several tasks) then follows up with a single job to merge the results.

This is the job array script. When this is run, each task will have different values for the variables: **TMPDIR**, **RJS_TASK**, **var**, and **desc** . Values for TMPDIR and RJS_TASK are provided by SLURM / rjs. Values for **var** and **desc** come from the schema file.

```
[user@host 200jobs]$ cat array.job
#!/bin/bash
#rj queue=iclmsc priority=550 schema=test.schema logdir=logs
# The schema=test.schema parameter tells rjs this is an array job

# Helpful options to catch scripting errors:
# -e: abort if any command fails
# -u: abort if an undefined variable is referenced
# -o pipefail: raise error if any component of a pipeline fails
set -euo pipefail

echo "My current working directory is $PWD"

# $TMPDIR should only be used for small amounts of data. It may be on
# local disk or a network disk or a ramdisk depending on how the
# compute node is configured. $TMPDIR is automatically deleted after
# the job exits.
#
# Intermediate and final data should be stored in your area on /p9, which
# is yours to keep clean (or not). It's good practice to create separate
# directories for separate jobs so they're easy to manage.
echo "For this job, TMPDIR=$TMPDIR"

echo "This is task number $RJS_TASK running on $(hostname)"

outDir=work
echo "Making output directory"
mkdir -p "$outDir"

outFile=$outDir/task${RJS_TASK}.txt
# note the definition of ${var} and ${desc} comes from the schema file
output="For task ${RJS_TASK}, variable var equals ${var} (aka ${desc})"
echo "Writing data to ${outFile}"
echo $output > $outFile
echo "Completed writing data"
```

Schema Files

A schema file is a text file containing lines defining array tasks.

The first field on each line is the task number, available as **RJS_TASK** in the script. Subsequent fields are separated by spaces and multi-word strings must be wrapped in quotes.

```
[user@host 200jobs]$ cat test.schema
1 var=1 desc=one
2 var=4 desc="two plus two"
3 var=9 desc="three squared"
```

This schema defines three array tasks. Each job task will have two variables set: **var** and **desc** using the values from the schema lines.

In the job script line starting with **#rj**, there are parameters to control the job scheduling:

```
#!/bin/bash
#rj queue=iclmsc priority=550 schema=test.schema logdir=logs
```

The job is:

- submitted to the **iclmsc** queue (queue=**iclmsc**)
- at medium priority (priority=**550**)
- using values for each task defined in **test.schema** (schema=**test.schema**)
- saving logs to the **logs** directory (logdir=**logs**)

See the RJS Options Table for a complete description of **#rj** parameters. For a quick reference, run "**rjs**" (with no arguments) in a terminal.

Next, the job writes some log messages and text to an output file.

```
echo "My current working directory is $PWD"
echo "For this job, TMPDIR=$TMPDIR"

echo "This is task number $RJS_TASK running on $(hostname)"

export outDir=work
echo "Making output directory"
mkdir -p "$outDir"

export outFile=$outDir/task${RJS_TASK}.txt
output="For task ${RJS_TASK}, variable var equals ${var} (aka ${desc})"
echo "Writing data to ${outFile}"
echo $output > $outFile
echo "Completed writing data"
```

The job:

- Writes a few messages (echo "My current…")
- Creates an output directory (mkdir -p …)
- Defines a task-unique file for its output (export outFile=$outDir/task${RJS_TASK}.txt)
- Outputs some text to that file

Anything directed to STDOUT or STDERR will appear in the same log file. See the Job Output and Logging section for more details.

Note the task-unique file in the output directory. This is very important! Without separate output files, one task will overwrite the output from another. For more complicated jobs, consider having each array task write to a unique work directory.

After the array job completes, we combine the output from the separate tasks using a merge script.

```
[user@host 200jobs]$ cat merge.job
#!/bin/bash
#rj name=merge queue=iclmsc logdir=logs mem=110g priority=100

set -euo pipefail

finalOutputDir=final
echo "create final output directory: ${finalOutputDir}"
mkdir -p $finalOutputDir

tmpData=work
echo "processing files from temporary data location: ${tmpData}"

export tmpOut=$tmpData/_tmp_output_all.txt
export finalOut=final_output.txt
cat $tempData/task*.txt > ${tmpOut} && mv ${tmpOut}
${finalOutputDir}/${finalOut} && echo "Move in place successful"
```

In the **#rj** line, we have added a requirement: **mem=110g**. This job will only run on nodes with 110 GB of memory available.

The job:

- Creates an output directory (mkdir -p ...)
- Collect the task results (cat $tempData/task*.txt)
- And write to a temporary file (> $tmpOut) and if successful (&&)
- Renames / moves it to a final output file (mv $tmpOut ...) and if successful (&&)
- Outputs a success message

The job uses a temporary file ($tmpOut) to collect the output. This avoids partial output from appearing while the job is running or if the job is interrupted. The final output file is only created if the job completes successfully.

For this sequence of jobs, the merge job should run only after the array job is complete. We could wait until the first job is complete before submitting the merge job, or we can have SLURM do it for us, by using rjs dependencies.

```
[user@host 200jobs]$ rjs array.job
34022407
[user@host 200jobs]$ rjs merge.job dep=34022407
34022411
```

In this example, the array.job is submitted using rjs and it tells us the JOBID (34022407). When submitting the merge job, we add the rjs “dep” parameter with that job id. The merge.job will wait in the pending state (PD) until all the array tasks are finished.

Example: Array Job with SBATCH / SRUN

Using sbatch to achieve the same goal is more complicated. sbatch requires one #SBATCH line for each parameter and there is no convenient way to define per-task variables. Logs will not include timestamps or runtime statistics.

The following script is similar but not equivalent to the rjs version!

```
#!/bin/bash
#SBATCH --job-name=array_test
#SBATCH --array=1-3

#SBATCH --partition=iclmsc
#SBATCH --priority=550
```

```
#Setting STDERR and STDOUT to the same place - this is default behaviour
for SLURM
#SBATCH --output=./logs/%x_%A.%a
#SBATCH --error=./logs/%x_%A.%a

set -euo pipefail

echo "My current working directory is $PWD"
echo "For this job, TMPDIR=$TMPDIR"
echo "This is task number ${SLURM_ARRAY_TASK_ID} running on $(hostname)"

mkdir -p work
OUTFILE="work/out_${SLURM_JOB_NAME}_${SLURM_ARRAY_TASK_ID}"
((RESULT=SLURM_ARRAY_TASK_ID*SLURM_ARRAY_TASK_ID))
echo "$SLURM_ARRAY_TASK_ID squared = $RESULT" > "$OUTFILE"
```

Submit *sbatch* scripts using the command `sbatch slurm_array.job`.

Example: Multi-node MPI job with rjs

Another approach to parallelisation is MPI. Using MPI, many nodes work concurrently on a single job, sharing data and results and communicating status. There are many variations on how to do this efficiently.

Here is a simple rjs script to run a three-node MPI application:

```
[user@host 200jobs]$ cat mpi_test.job
#!/bin/bash
#rj nodes=3 taskspernode=1 queue=iclmsc priority=100 features=
logdir=logs/mpi_rjs name=rjs_mpi_example
set -euo pipefail

echo "Starting MPI example job"
#add an openmpi library to the environment
module add openmpi/4.0.5-mlnx-gcc

#note: you must supply the MPI binary for this example
binary=./bin/mpi_hello_world

echo "Using these nodes:"
echo ${SLURM_JOB_NODELIST}

#Running the binary
mpirun ${binary}
```

Example: Multi-node MPI job with SBATCH

Here is a roughly equivalent sbatch version:

```
#!/bin/bash
#
#SBATCH --job-name=sbatch_mpi_example
#SBATCH --nodes=3
#SBATCH --ntasks-per-node=1
#SBATCH --partition=iclmsc
#SBATCH --priority=990
```

```
#Setting STDERR and STDOUT to the same place - this is default behaviour
for SLURM
#SBATCH --output=./logs/mpi_sbatch/%x_%A.%a
#SBATCH --error=./logs/mpi_sbatch/%x_%A.%a

set -euo pipefail

echo "Starting MPI example job"
#add an openmpi library to the environment
module add openmpi/4.0.5-mlnx-gcc

#note: you must supply the MPI binary for this example
binary=./bin/mpi_hello_world

echo "Using these nodes:"
echo ${SLURM_JOB_NODELIST}

#Running the binary
mpirun ${binary}
```

TMPDIR AND TEMPORARY OUTPUTS

When a job runs, SLURM sets TMPDIR to the local disk (if available) or to a temporary directory on the file server. TMPDIR is always set to a valid location.

At the end of the job, TMPDIR and its contents are automatically removed. To keep files or data, move them to another location at the end of the script.

TASK RUNTIMES

Note: Jobs for the Imperial College queue have a runtime limit of 5 minutes. Contact your Imperial course instructor if you want to change this runtime limit.

We recommend job run times of 30 minutes to six hours. The DUG internal target is roughly 4 hours. We understand that this is not always practical and depends greatly on the data and problem design.

The 30 minute to 6 hour target is for a variety of reasons.

- Very short tasks have higher overhead. More time can be spent starting up and shutting down the job than doing useful work.
- Very long tasks have a greater risk of failing. There are ways to reduce the risk, e.g. checkpointing, but these approaches affect performance and add complexity.
- With moderate run times, the user can review the results of the first completed tasks to determine if the job is configured correctly, allowing for quality-control before the remainder of the job has run.
- The job scheduler can handle job priorities better. When jobs complete, SLURM will choose which job to run next based on priority. Very long tasks result in fewer opportunities for prioritisation to occur.

Please contact DUG support before running tasks with expected run times longer than 24 hours.

INTERACTING WITH SLURM

See the manual page of each command for full details, e.g.:

```
[user@host ~]$ man squeue
```

Frequently used tools:

Command name	Description
squeue	Show queued or running jobs running Limit to a given user or partition using: -u user_name -p partition_name
scontrol	Modify job priority, dependency, partition
scancel	Cancel a job: scancel JOBID
jless	Show the output of a running or failed job jless JOBID
jcd	Show the working directory for a job jcd JOBID

Each task has a job ID and state (running, cancelled, errored etc). See Task / Job States for details..

Show jobs:

```
[user@host 200jobs]$ squeue -u username
PARTITION PRIORITY NAME USER ST TIME TIME_LEFT NODES NODELIST(REASON JOBID
client_part 100 mpi_example username PD 0:00 1:00:00 3 (Resources) 34022647
```

Change job priority:

```
[user@host 200jobs]$ rjs mpi_test.job
34058333
[user@host 200jobs]$ scontrol update jobid=34058333 priority=700
```

Cancel a job:

```
[user@host 200jobs]$ rjs mpi_test.job
34058367
[user@host 200jobs]$ scancel 34058367
```

Show the log output (STDERR and STDOUT) for a job:

```
jless JOBID
```

Show the working directory for a job:

```
jcd JOBID
```

JOB OUTPUT AND LOGGING

By default, the job log contains the STDERR and STDOUT from the job script. When a job starts, rjs prepends the log “PROLOG”, and at completion appends an “EPILOG”.

Example log (from mpi example job):

```
[user@host slurm_mpi]$ cat slurm_mpi_example.o34058333
2020-04-10 03:31:33 -0500 hnod1-3-43: PROLOG TMPDIR=/tmp/.tmpdir.34058333 (linked to
/h4/000scratch/slurm_tmpdir/20200410_job_34058333.R5UI)
2020-04-10 03:31:33 -0500 hnod1-3-43: PROLOG TMPDIR_SHM=/tmp/.tmpdir_shm.34058333 (linked to
/dev/shm/slurm_tmpdir/20200410_job_34058333.hGsR)
2020-04-10 03:31:33 -0500: == JOB START  NAME=slurm_mpi_example QUEUE=msi HOST=hnod1-3-43 DATE=Fri Apr 10 03:31:33 CDT
2020 JOBID=34058333 ARRAY_JOBID= TASK= DEPENDENCIES= ==
2020-04-10 03:31:33 -0500: Starting MPI example job
2020-04-10 03:31:33 -0500: Using these nodes:
2020-04-10 03:31:33 -0500: hnod1-3-43,hnod1-11-[17,40]
2020-04-10 03:31:36 -0500: Hello world from processor hnod1-3-43, rank 0 out of 3 processors
2020-04-10 03:31:36 -0500: Hello world from processor hnod1-11-40, rank 2 out of 3 processors
2020-04-10 03:31:36 -0500: Hello world from processor hnod1-11-17, rank 1 out of 3 processors
2020-04-10 03:31:37 -0500: == JOB END  STATUS=0 HOST=hnod1-3-43 DATE=Fri Apr 10 03:31:37 CDT 2020 RUNTIME=5 ==
2020-04-10 03:31:37 -0500: EPILOG deleting /h4/000scratch/slurm_tmpdir/20200410_job_34058333.R5UI
2020-04-10 03:31:38 -0500: EPILOG deleting /dev/shm/slurm_tmpdir/20200410_job_34058333.hGsR
```

The PROLOG includes summary information such as the temporary directory for the job and the name, queue, node, etc..

The EPILOG includes the runtime (in seconds) and confirmation of the deletion of the temporary files.

We recommended using the default log files to capture information rather than logging elsewhere.

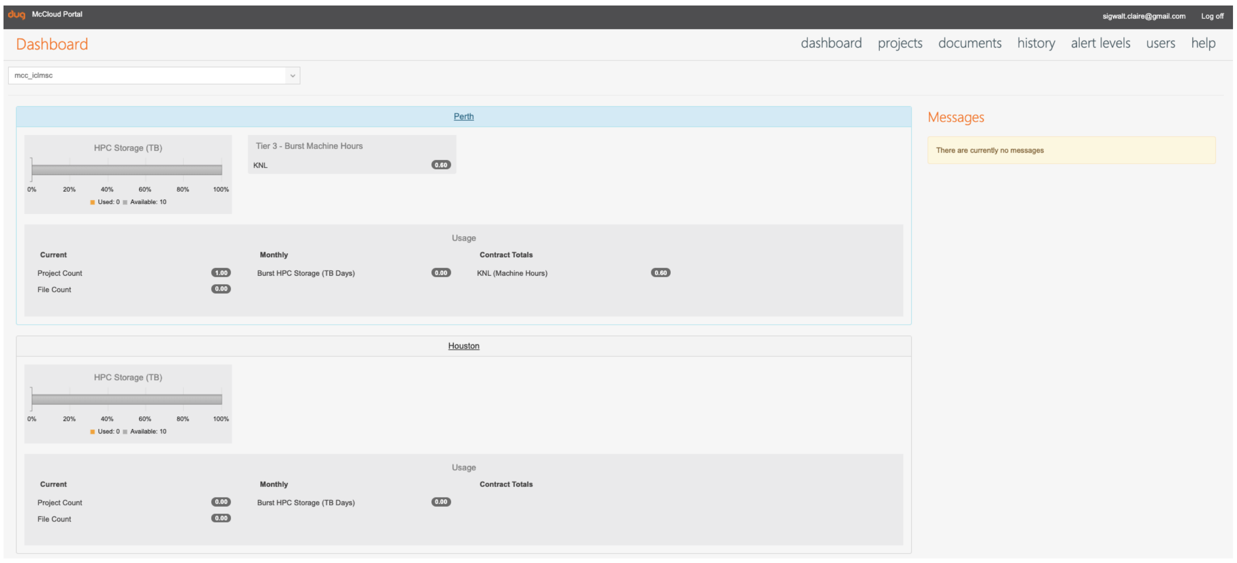
NAMING CONVENTIONS

Log file names default to:

Multi task array jobs	job_name.o[job_id].[array_jobid]
MPI jobs	job_name.o[job_id]

DUG McCloud Portal

The DUG McCloud Portal allows users to see the McCloud resources that have been consumed. It also provides the ability to set usage alerts based on pre-set thresholds and defined allocations. Portal users can be added and removed by administrators on the Portal.



Perth

HPC Storage (TB)
Used: 0 Available: 10

Tier 3 - Burst Machine Hours
KNL 0.60

Usage

Current		Monthly		Contract Totals	
Project Count	1.00	Burst HPC Storage (TB Days)	0.00	KNL (Machine Hours)	0.60
File Count	0.00				

Houston

HPC Storage (TB)
Used: 0 Available: 10

Usage

Current		Monthly		Contract Totals
Project Count	0.00	Burst HPC Storage (TB Days)	0.00	
File Count	0.00			

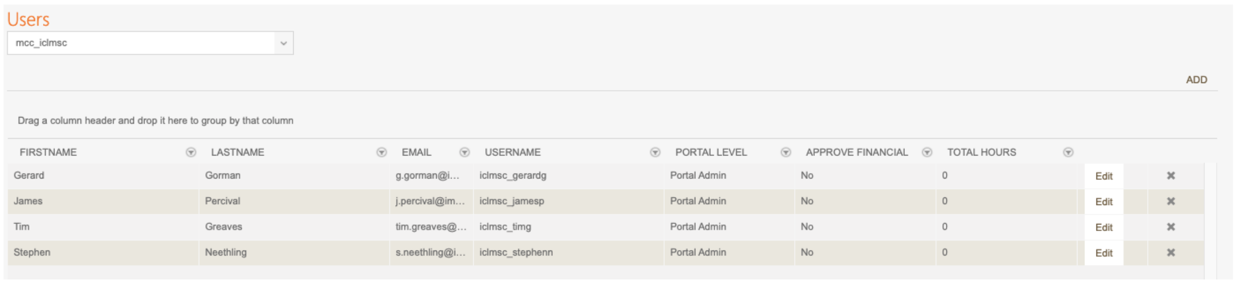
Messages

There are currently no messages

Portal Users

Imperial instructors have been given ‘Portal Administrator’ access and should have already received an invitation email to access the portal. Please message us on Slack to get assistance on how to access the portal.

To add new users, select the “Users” section and click add.


Users
mcc_iclmsc
ADD
Drag a column header and drop it here to group by that column

FIRSTNAME	LASTNAME	EMAIL	USERNAME	PORTAL LEVEL	APPROVE FINANCIAL	TOTAL HOURS		
Gerard	Gorman	g.gorman@i...	iclmsc_gerardg	Portal Admin	No	0	Edit	✖
James	Percival	j.percival@im...	iclmsc_jamesp	Portal Admin	No	0	Edit	✖
Tim	Greaves	tim.greaves@...	iclmsc_timg	Portal Admin	No	0	Edit	✖
Stephen	Neethling	s.neethling@i...	iclmsc_stephenn	Portal Admin	No	0	Edit	✖

Monitoring HPC resource usage

The DUG McCloud Portal provides two level of usage monitoring:

- Group level : Available on the homepage of the dashboard
- User level : Available on the ‘Users” section

Final Remarks

Thank you for taking the time to read this documentation, whilst hopefully useful it is not exhaustive.

If at any point you feel something has been missed or require assistance, please do not hesitate to contact DUG Support at support@dug.com.

Appendix

RJS OPTIONS TABLE

Option	Description	Default	Required
array	Select a subset of tasks to submit (see "ranges" below). Without this parameter, the entire task array (as defined by the schema, if provided) is submitted.		
partitio n	Partition to submit to (the default * will use the name of your primary Unix group).		
priority	Job priority, between 1 and 1,000.		
dep	List of job IDs to serve as dependencies. This newly-submitted job will not run until the dependent job is no longer in the queue (i.e. it has run successfully to completion *or been canceled by the user*).		
hold	Set to 1 to submit job in a "held" state (i.e. it will not be scheduled until released).		
name	Job name, which otherwise defaults to the name of the script.		
feature s	Request specific features from the queueing system. You should typically do this only after consulting with support.		
nodes[1]	Number of nodes to use for multi-node (e.g. MPI) jobs.		
taskspe rnode	Number of proces on each node to use for multi-node jobs.		
mem	Instruct SLURM to run this script only on nodes with at least this much memory (use K, M, and G suffices for kilo-, mega-, giga-byte). Note that it is *your responsibility* to supply an appropriate memory request. If you attempt to use more memory than your request, your task will most-likely fail.		
localdis k	Run this job only on nodes with a local disk. localdisk=1 requests local disk (any size) localdisk=# requests local disk of #GB Note that most nodes lack local disks, and the disks may be small. <u>Please discuss with support before submitting jobs that require local storage.</u>		
io	set to 1 for I/O-bound jobs, which prevents a huge number of concurrent tasks that overwhelm the network file system. As a general rule of thumb, set io=1 if you expect tasks to		

[1] Please note that a SLURM will allocate an entire bare metal node. The user who submitted the job (and that user only) will have access to all cores available on the node.

	consistently use more than 50 MB/s.		
logdir	Log directory.		
schema	Schema file (see example above).		
export	Comma-separated list of environment variables (from the shell where you're running rjs) that you want to be present in your script execution environment.		
runtim e	Number of hours that each task is expected to run. This is an initial estimate; for job arrays rjs will collect statistics as tasks complete to update the estimate.		

TASK / JOB STATES

The following table documents the various states a task can exhibit on the cluster.

Jobs typically pass through several states in the course of their execution. The typical states are PENDING, RUNNING, SUSPENDED, COMPLETING, and COMPLETED. An explanation of each state follows.

Job code	Job code long name	Job code description
BF	Boot fail	Job terminated due to launch failure, typically due to a hardware failure (e.g. unable to boot the node or block and the job cannot be requeued).
CA	Cancelled	Job was explicitly cancelled by the user or system administrator. The job may or may not have been initiated.
CD	Completed	Job has terminated all processes on all nodes with an exit code of zero.
CF	Configuring	Job has been allocated resources, but are waiting for them to become ready for use (e.g. booting).
CG	Completing	Job is in the process of completing. Some processes on some nodes may still be active.
DL	Deadline	Job terminated on deadline.
F	Failed	Job terminated with non-zero exit code or other failure condition
NF	Node fail	Job terminated due to failure of one or more allocated nodes.

OOM	Out of memory	Job experienced out of memory error.
PD	Pending	Job is awaiting resource allocation
PR	Pre-empted	Job terminated due to pre-emption.
R	Running	Job currently has an allocation.
RD	RESV_DEL_HOLD	Job is held.
RF	REQUEUE_FED	Job is being requeued by a federation.
RH	REQUEUE_HOLD	Held job is being requeued.
RQ	Requeued	Completing job is being requeued.
RS	Resizing	Job is about to change size.
RV	Revoked	Sibling was removed from the cluster due to other clusters starting the job.
SI	Signalling	Job is being signalled.
SE	SPECIAL_EXIT	The job was requeued in a special state. This state can be set by users, typically in EpilogSLURMctld, if the job has terminated with a particular exit value.
SO	STAGE_OUT	Job is staging out files.
ST	Stopped	Job has an allocation, but execution has been stopped with SIGSTOP signal. CPUS have been retained by this job.
S	Suspended	Job has an allocation, but execution has been suspended and CPUs have been released for other jobs.
TO	Timeout	Job terminated upon reaching its time limit.