Artemis User Guide

Information and Communications Technology

Sydney Informatics Hub

16th March, 2017

Table of Contents

Getting Started

Access to Artemis

For instructions to obtain an Artemis account, see the following page:
http://sydney.edu.au/research_support/hpc/access/index.shtml

Linux Command Line

Artemis is a Linux-based HPC with a command-line interface. If you have no background in Linux or the Linux command line, there are a number of excellent tutorials online that will help you get started. A recommended starting tutorial can be found here: http://www.ee.surrey.ac.uk/Teaching/Unix/ Once you finish tutorial 4, you will have enough command line knowledge to start use Artemis.

Linux Text Editors

It is possible to edit text files on Artemis using text editors such as **Nano**, **GEdit**, **Vim** or **Emacs**. If you have logged into Artemis with an X server connection (that is, with the -X or -Y options to the ssh login command and a connection to an X server running on your computer), then you can use gedit to edit text files. If you don't have an X server running, then the next simplest text editor is **nano**. If you find yourself editing text files frequently on Artemis and want to invest time to learn a more powerful text editor, then it is worth using either **emacs** or **vim**.

Projects and UniKeys

UniKey

Once you have been granted access to Artemis, you log in using your UniKey credentials. Throughout the guide, we use the example UniKey abcd1234 whenever we need to specify a UniKey. Remember to replace abcd1234 with your UniKey wherever needed.

Project

You need to know what your project is called. You need to specify a project for every job submitted to Artemis. There are two styles of project name you can use:

Your full project name has the form `RDS-FAC-Project-RW`, where `FAC` is your faculty code and `Project` is your abbreviated project name as specified in your RDMP. If you have access to Artemis, but don't know your project name, you could ask your Lead CI or type `groups` into Artemis after logging in:

```
[abcd1234@login4]$ groups
linuxusers HPC-ACL RDS-ICT-PANDORA-RW RDN-USERS RDN-ICT-TEST RDS-
ICT-TEST-RW
```

The projects you are a member of are any entries that start with RDS-. In the above case, abcd1234 is a member of the projects PANDORA and TEST. For most applications on Artemis, you can use your project name without the RDS-FAC-…-RW parts. However, some applications require the full project name. If you have difficulties using your abbreviated project name, try using your full project name instead.

Throughout this guide, we will use the project PANDORA in the examples provided. Remember to replace PANDORA with your project name when you use Artemis.

Opening GUI windows on Artemis

It is possible to open programs with a graphical user interface (GUI) on Artemis if you have an X server running on your computer and ssh into Artemis using either the -X or -Y options. Programs on Artemis with GUIs include gedit (a text editor), Matlab and Gnuplot.

Linux

If you are on a Linux computer, such as Ubuntu or Linux Mint, then you are already running an X server. Therefore, you can open programs with GUIs if you log into Artemis using the -X or -Y options.

```
ssh -X abcd1234@hpc.sydney.edu.au
```

MacOS

On MacOS, you need to install **XQuartz**. After installing XQuartz, you can then open a terminal window using MacOS's built-in terminal program, then SSH to Artemis using the -X or -Y options, as was the case for Linux.

Windows

On Windows, there are a number of X servers to choose from. The University of Sydney has a site license for Starnet X-Win32. Follow the instructions to download and install X-Win32 here: **http://staff.ask.sydney.edu.au/app/answers/detail/a_id/316/session/L2F2LzEvdGltZS8xNDg2MzM0MTM5L3NpZC9vcE5MRnZhbg%3D%3D**. Then use the new connection wizard to set up an Artemis connection:

- Select SSH and give your connection a name. For example, "John's Artemis connection" then click next.
- The host to connect to is hpc.sydney.edu.au
- The login name and password are your UniKey credentials
- The command to run on host will be filled in correctly if you click the Linux option.
- Click Finish

Your connection will appear as one of the options in the list. To start a command line session on Artemis, click the connection you just made, then click Launch. A command line terminal connection to an Artemis login node will open. If everything was done correctly, you will be able to launch programs such as gedit from the command line.

Software

There are many pre-installed programs on Artemis, including various **compilers**, scripting languages (for example, Python, Perl and R), libraries and other specialised software. For full details, see the **Software** section.

Artemis compute nodes

Artemis is a 4264-core HPC system comprised of the following nodes:

	Artemis Phase 1 Nodes			Artemis Phase 2 Nodes	
	Standard Memory	High Memory	GPU	Standard Memory	High Memory
Number of Nodes	56	2	5	80	3
Cores per node	2 x 12	2 x 12	2 x 12 CPUs 2 GPUs	2 x 16	4 x 16
RAM per node	128 GB	512 GB	128 GB	128 GB	6 TB

Core allocations per Artemis queue

Some nodes of Artemis are restricted to certain groups:

- Civil Engineering have ownership of 13 nodes (416 cores); they have exclusive access to these nodes.
- Some researchers were granted strategic allocations on Artemis. These nodes are reserved for these projects. As of February 2017, 936 cores are reserved for strategic allocations.

The remaining nodes (2912 cores in total) are available to all users of the system.

Data Storage on Artemis

Artemis uses a **Lustre** file system that is accessible from any compute node and provides excellent I/O performance (especially for large files). The Lustre file system contains the following storage spaces:

- Home (`/home/abcd1234`)
- Project (`/project/PANDORA`)
- Scratch (`/scratch/PANDORA`)

Remember to replace abcd1234 with your UniKey and PANDORA with your abbreviated project name.

Data will not be backed up so we cannot offer any ability to retrieve lost data. For long term storage please store all data in the **Research Data Store** (RDS).

/home

Each researcher will be allocated their own home directory within Artemis. This can be used to store program code, batch scripts and other files. Please note that the home directory has limited space and is intended for storing code and configuration information. Data on `/home` is backed up in case of system failure (that is, if the Lustre filesystem fails), but files cannot be recovered if they are accidentally deleted. Important data should be stored in the **research data store**.

To check your /home quota usage, run the following command:

```
pquota
```

/project

The default allocation is 1 TB of project storage, shared between all members of the project. Please note that no additional space will be provided. If you require more disk space for job output, use /scratch instead. Since /project (or /scratch or /home) is not backed up, please back up all important data to the **Research Data Store** regularly.

To check your quota and usage of your allocated storage on /project, use the pquota command.

/scratch

/scratch is a 204 TB pool of shared storage that is intended for data that needs to be saved during a job, but can be deleted once the job completes. **As this directory is shared by all users, please do not leave your data in this area for longer than absolutely necessary. Transfer important files to /project if you need them after your job finishes.** /scratch is not backed up, so please back up important data to the **Research Data Store**.

To see how much /scratch space is available, use the following command:

```
df -h /scratch
```

To see how much space your project is using:

```
lfs quota -hg RDS-ICT-PANDORA-RW /scratch
```

Remember to replace RDS-ICT-PANDORA-RW with your project name, as described in the **Projects and UniKeys** section.

For details about disk usage by specific subdirectories, use this command:

```
du -sh /scratch/PANDORA/directory
```

where the last argument is the directory you want to know the size of.

Software/Programs/Applications

Artemis has many pre-installed programs available. Your batch jobs can be set up to use these programs by loading "modules". When loaded, these modules quickly set up your batch job environment with paths to executables and other important settings so you don't have to manually configure your batch job environment or remember where programs are installed. A full list of all software on Artemis can be found on the **software** page.

Modules (or how to load installed software)

To see all available modules on Artemis, run the following command:

```
module avail
```

this will return a (long) list of modules. To narrow down the list to programs you're interested in using, try using grep. The following example returns any modules with the word matlab in it:

```
module avail |& grep -i matlab
```

To load a module, for example, Matlab, use the following command:

```
module load matlab
```

this will add Matlab to your PATH and load any required dependencies. It will, by default, load the most commonly used version of Matlab. If you require a different version, specify the version as follows:

```
module load matlab/R2014b
```

Having multiple versions of the same program open can cause conflicts. For example, if Matlab 2014 and Matlab 2015 were simultaneously loaded, then the last version loaded will be used. For this reason, it is good practice to unload unnecessary modules:

```
module unload matlab/R2014b
```

Occasionally, some modules confict with others and create strange and difficult to diagnose errors. To unload all modules with one command, use module purge:

```
module purge
```

Installing software

If the software you require isn't installed on Artemis, you can submit a **High Performance Computing request** to install software via the **ICT Self-Service portal**. To access the form using the links provided here, click the ICT Self-Service portal link, log in with your unikey and password, then click the High Performance Computing request link. Alternatively, you can compile your own programs on Artemis. For information, see the **Compiling** section.

Submitting Jobs to Artemis

Artemis uses a modified version of PBS Pro to manage and queue jobs. Some minimal example scripts are provided below.

Serial jobs (1 core)

```
#!/bin/bash
#PBS -P PANDORA                      # your project name
#PBS -l select=1:ncpus=1:mem=4GB     # select one chunk with 1 CPU and 4 GB memory
#PBS -l walltime=10:00:00            # actual time your job will run for

cd $PBS_O_WORKDIR
my_program > results.out
```

Save this script to a file (for example `MyScript.pbs`), then then submit it to the PBS scheduler using the following command on the command line:

```
qsub MyScript.pbs
```

there are other #PBS directives that you may optionally use. For a full list, see the **PBS Professional user manual**.

Parallel jobs using OpenMP

OpenMP is a shared memory parallelism model, so OpenMP parallelised programs can only be run on one chunk (a "chunk" is a virtual node) and either 24, 32 or 64 cores, depending on the node. An example script to run an OpenMP parallelised program is:

```
#!/bin/bash
#PBS -P PANDORA                      # your project name
#PBS -l select=1:ncpus=4:mem=4GB     # select one chunk with 4 CPUs and 4 GB memory
#PBS -l walltime=10:00:00            # actual time your job will run for

cd $PBS_O_WORKDIR
my_program > results.out
```

Note: OMP_NUM_THREADS is automatically set to the number of cores per chunk on Artemis, so you don't have to manually set the value of OMP_NUM_THREADS in your PBS script.

Parallel jobs using MPI

MPI (Message Passing Interface) allows jobs to communicate across several nodes. A basic MPI job script is shown below.

```
#!/bin/bash
#PBS -P PANDORA                                  # your project name
#PBS -l select=1:ncpus=4:mpiprocs=4:mem=4GB      # select 1 chunk, 4 MPI cores, 4 GB memory

#PBS -l walltime=10:00:00                        # actual time your job will run for
cd $PBS_O_WORKDIR                                # change to current working directory

module load siesta                               # Also automatically loads the Intel MPI library

mpirun -np 4 siesta < h2o.fdf > h2o.out
```

The above file will choose one chunk with 4 cores and 4 GB of memory. All 4 cores have been made available to MPI as specified by the mpiprocs=4 option. It is recommended to set ncpus and mpiprocs to the same value, unless you know you can set them differently.

Note: load the correct MPI implementation for your program. Most modules will automatically load the appropriate MPI implementation. However, you can load your own MPI implementation if you are compiling your own MPI programs or if you know you can use an alternative MPI implementation.

With MPI, you can select more than one chunk:

```
#!/bin/bash
#PBS -P PANDORA                                  # your project name
#PBS -l select=5:ncpus=8:mpiprocs=8:mem=4GB      # select 5 chunks, 8 MPI cores per chunk, 4 GB memory per chunk
#PBS -l walltime=10:00:00                        # actual time your job will run for

cd $PBS_O_WORKDIR                                # change to current working directory
module load siesta                               # Also automatically loads the Intel MPI library

mpirun -np 40 siesta < h2o.fdf > h2o.out
```

This script requests 5 chunks with 8 cores each (all cores are available to MPI) and 4 GB of memory *per chunk*, meaning your job will be allocated 40 cores and 20 GB of memory in total.

Interactive jobs

You can request an interactive job using the `qsub -I` directive from the command line. For example:

```
qsub -I -P PANDORA -l select=1:ncpus=1:mem=4GB,walltime=1:00:00
```

Remember to replace `PANDORA` with your project name.

Note: You cannot use a PBS script to start an interactive job. Only `qsub -I` works.

Running programs with a GUI in an interactive job

You can open a GUI on Artemis in an interactive session by firstly logging in as described in **getting started**, then requesting an interactive job using the `qsub -I -X` command. An example qsub command is (all on one line):

```
qsub -I -X -P PANDORA -l
select=1:ncpus=4:mem=10GB,walltime=1:00:00
```

Users with strategic allocations can run interactive jobs in their allocated area by specifying a queue in the interactive qsub command. For example, members of the project `porous` can request an interactive session in their allocation with the following command (all on one line):

```
qsub -I -X -P porous -q condo-civil -l
select=1:ncpus=4:mem=10GB,walltime=1:00:00
```

Array jobs

PBS array jobs are a convenient way of running many computations with a single script. They are particularly suitable for jobs which need to be run many times, but with different parameters or on different input files. To submit an array job, you need to include this PBS directive in your PBS script:

```
#PBS -J start_index-end_index:increment
```

For example:

```
#PBS -J 1-10
```

will start an array job with 10 elements. Each job will be assigned a single value of `$PBS_ARRAY_INDEX` from 1 to 10. The array indices must be integers. Another example is:

```
#PBS -J 1-10:2
```

This directive will start an array job containing 5 elements, with values of `$PBS_ARRAY_INDEX` of 1, 3, 5, 7, and 9.

Two common uses of array jobs are to run a program multiple times with different input parameters or different input files. Some example array job PBS scripts are shown below.

Single Parameter Change

This script will start an array job with 10 elements, each element will be assigned a different value of `$PBS_ARRAY_INDEX` from 1 to 10. In this case, `$PBS_ARRAY_INDEX` is used directly as a command line argument to the program called `my_program`.

```
#!/bin/bash
#PBS -P PANDORA
#PBS -l select=1:ncpus=1:mem=4GB
#PBS -l walltime=10:20:10
#PBS -J 1-10

cd $PBS_O_WORKDIR
my_program $PBS_ARRAY_INDEX > output.$PBS_ARRAY_INDEX
```

Multiple Parameter Changes

This script will run the program `my_program` with two parameters. `$PBS_ARRAY_INDEX` is only a single parameter; therefore, you will need a dictionary or lookup table to convert `$PBS_ARRAY_INDEX` indices to multiple parameters.

In the following example, all parameters required for the array job are stored in a file called `job_params`. The array job will read the line corresponding to `$PBS_ARRAY_INDEX`, and then pass these parameters to `my_program`.

An example `job_params` file:

```
1      0.1
2.4524 0.2
3.45   0.3
4.4    0.4
```

and an example `my_array_script` PBS script:

```
#!/bin/bash
#PBS -P PANDORA
#PBS -l select=1:ncpus=1:mem=4GB
#PBS -l walltime=15:20:10
#PBS -J 1-4

cd $PBS_O_WORKDIR

params=`sed "${PBS_ARRAY_INDEX}q;d" job_params`
param_array=( $params )
my_program ${param_array[0]} ${param_array[1]}
```

Multiple input files

A third common usage of array jobs is to run the same program on multiple input files. The example here will run computations on all files in a directory. This script first saves a list of files in the directory to a file called `file_list`. The number of files in this list is then counted and saved to a variable called `num_files`. A PBS array script called `run_array_job`, which runs `my_program` with each file in the directory as input, is written to disk. At the end of the script, the script submits `run_array_job` to the scheduler using the `qsub` command.

```
#!/bin/bash

rm file_list
find . -type f -maxdepth 1 | sed 's/.\///' > file_list
num_files=`cat file_list | wc -l | tr -d ' '`

cat > run_array_job << EOF
#!/bin/bash
#PBS -P PANDORA
#PBS -l select=1:ncpus=1:mem=4GB
#PBS -l walltime=15:20:10
#PBS -J 1-${num_files}

cd \$PBS_O_WORKDIR
filename=\`sed "\${PBS_ARRAY_INDEX}q;d" file_list\`
my_program < \$filename > \${filename}.output
EOF

qsub run_array_job
```

Job monitoring/management

There are a number of commands available to view the staus of jobs on the system. A brief set of useful commands is shown below. For more commands, see the **PBS Professional user manual**.

Command	Description
qstat -u abcd1234	show status of abcd1234's jobs
qdel 1234567	delete job 1234567 from queue
qstat	show status of all jobs
qstat -f 1234567	show detailed stats for job 1234567
qstat -xf 1234567	show detailed stats for job 1234567, even after it has finished

When jobs finish, they produce three output files. One for standard output, one for standard error and a resource usage file. The file formats are as follows:

```
<JobName>.o<JobID>        - Standard output file
<JobName>.e<JobID>        - Standard error file
<JobName>.o<JobID>_usage - Resource usage file
```

If you don't redirect standard output or standard error to a file, they will be printed in the .o or the .e files and only appear after your jobs finish. These files may contain useful information about why your job terminated before it finished.

The resource usage file contains details about how long your job ran for and also the memory used by your job. You can use the information in the resource usage file to optimise your walltime and memory requests for future jobs. An example resource usage file is shown below:

```
Job Id: 1050977.pbsserver for user abcd1234 in queue small
Job Name: TestJob
Project: RDS-ICT-PANDORA-RW
Exit Status: 0
Walltime requested:   00:03:00 :      Walltime used:   00:01:36
    Cpus requested:         48 :
          Cpu Time:   00:36:38 :        Cpu percent:       3102
     Mem requested:        8gb :           Mem used:  2342348kb
    VMem requested:       None :          VMem used:  2342348kb
    PMem requested:       None :          PMem used:       None
```

Job Scheduling Priority: "Fair Share"

"Fair Share" assigns priority to jobs based on each project's recent usage of the system. If a project has recently used a lot of CPU time, then the priority of their future jobs, relative to other projects, will be reduced. Once a job runs, it is allowed to complete and is unaffected by fair share.

Fair share only has an impact when there is contention for resources. Fair Share is calculated at a project level, so if one member of a project uses a lot of CPU time, future jobs submitted by that project will have lower priority.

Different queues (see the **Job Queues** section for a description of each queue) have different fair share weightings. The small, normal and large queues have a fair share weight of 10, which is considered to be the "standard" fair share weighting. The high memory and GPU queues, however, have a fair share weight of 50. If you request excessive resources (for example, too much memory), your job may be placed in a queue with a higher fair share weighting.

In addition to the above accumulation, fair share also decays with a "half-life" of 2 weeks. If you were to stop or reduce your use of Artemis, your fair share would decrease and the priority of your future jobs would increase.

Overall, it is best not to worry about fair share. The sooner you submit your jobs, the sooner they will run.

Job Queues

The `qstat -Q` command will show a range of queues on the system. Not all of these are directly available to Artemis users. Some queues are reserved for users with strategic allocations of compute resources. Your job may be routed to other "internal" queues after submission depending on the size of your job.

Queues are requested using the following PBS directive:

```
#PBS -q Queue
```

Where `Queue` is the name of the queue you want to place your job in. For example, if you wanted to place your job into the queue called "defaultQ", you would write:

```
#PBS -q defaultQ
```

Queues available to all users

The queues available to all Artemis users are listed below:

defaultQ

If you don't specify a queue, your job will be placed in this queue. Once placed in this queue, your job will be routed to one of the following internal queues based on the resources requested in your PBS script. The sub-queues of this queue are:

- Small
- Normal

- Large
- Highmem (High Memory)
- GPU

Jobs are routed to these queues based on their resource requests. For details, see the **queue resource limits** table.

small-express

An "express" queue for small jobs. These jobs will typically start quickly, but will increase your fair share weight 5 times faster than standard memory jobs in **defaultQ** This queue is useful for quickly testing your jobs before submitting a longer job to **defaultQ**.

scavenger

A low priority queue that attempts to run jobs on idle resources if and when they are available. If your job completes it will be "free", in that it will not increase your fair share weight at all. The downside is there is no assurance that your job will start. If the system is heavily utilised it may never run. Additionally, if a job in any other queue requires resources that a scavenger job is using, the scavenger job will be suspended. If the scavenger job cannot be resumed within a given period (approximately half the expected run time) it will be terminated and you will lose all unsaved progress.

dtq

This queue grants access to nodes dedicated to input/output (I/O) jobs and can be used to copy data between Artemis, the Research Data Store and other external locations. It can also be used for other I/O intensive tasks, like compressing and archiving data, or simply for moving or copying data around Artemis. Jobs run in this queue will not increase your fair share weight, but we reserve the right to terminate compute jobs running in this queue. For more information about using this queue, see the **Data Transfer Queue** section.

interactive

This queue is for running interactive (command line) sessions on a compute node. Interactive sessions can only be requested using the qsub –I command from the command line. You cannot request an interactive session from a job script. This queue attracts a fair share weighting 10 times higher than standard memory jobs in **defaultQ**.

gpu

This queue is for performing computations with programs that use GPUs (Graphical Processing Units). To use GPUs, request gpus in the select directive, and submit your job to **defaultQ**:

```
#PBS -l select=1:ncpus=1:ngpus=1
```

`ngpus` should be 1 or 2 per chunk as GPU nodes have a maximum of 2 GPUs.

Note: The fair share weighting of this queue is 5 times higher than **defaultQ**.

Queue resource limits

Queue	Max Walltime	Max Cores Per Job	Max Cores per User	Memory per node (GB)	Memory per core (GB)	Fair Share Weight
Small	1 day	24	96	< 123	< 20	10
Normal	7 days	96	96	< 123	< 20	10
Large	21 days	288	288	< 123	< 20	10
High Memory	7 days	192	192	123 to 6144	> 20	50
GPU	7 days	120	120	< 123	N/A	50
small-express	12 hours	4	40	< 123	N/A	50
scavenger	2 days	288	288	< 123	N/A	0
dtq	10 days	2	16	< 16	N/A	0
Interactive	4 hours	4	4	< 123	N/A	100

- The Small, Normal, Large, High Memory and GPU queues are all accessed via the defaultQ queue. You cannot directly request these queues.
- The **interactive queue** is requested using the `qsub -I` command via the command line. You cannot request interactive access with `#PBS -q interactive`.
- The maximum number of jobs a user can have queued is:
  - 200 in defaultQ,
  - 10 in small-express.
- The maximum number of cores one user can simultaneously use is 600.
- PBS array jobs are limited to 1000 elements.
- N/A = Not Applicable.

Strategic Allocations

Some researchers have access to dedicated compute resources on Artemis. Only members of the Lead Chief Investigator's projects may submit jobs to these queues.

Queue	Lead Chief Investigator
condo-civil	Associate Professor Luming Shen
alloc-dh	Professor David Hensher
alloc-jr	Professor John Rasko
alloc-nw	Dr. Nicholas Williamson
alloc-am	Dr. Alejandro Montoya
alloc-md	Associate Professor Meredith Jordan
alloc-fs	Dr. Fatemeh Salehi

Note: Users with strategic allocations are also free to request their jobs be run in all other publically accessible queues in addition to their strategic allocation.

Data Transfer Queue (dtq)

Artemis has a queue, called dtq, that has nodes with access to /rds (which is the **RCOS** version of the **Research Data Store**) and the internet. This queue is dedicated to running data moving jobs and other input/output (I/O) intensive work, such as compressing and archiving files, copying data from Artemis to the **Research Data Store**, or copying data to and from remote locations. These nodes have dedicated ethernet and infiniband bandwidth to maximise file transfer speeds, so they deliver significantly better data transfer and I/O performance than the login nodes. No other compute nodes on Artemis have access to /rds or the internet. The login nodes can still be used for very small I/O work, but heavy copy or I/O work should be submitted to dtq instead.

An example dtq job, which moves the `mydata` directory from Artemis to RCOS, using an example project called PANDORA, is shown below:

```
#!/bin/bash
#PBS -P PANDORA
#PBS -q dtq
#PBS -l select=1:ncpus=1:mem=4gb,walltime=1:00:00

rsync -avr /project/PANDORA/mydata /rds/PRJ-PANDORA/
```

To run this script. You can save it to a file (for example my-copy-job.pbs) and submit it to the scheduler with the qsub command:

```
qsub my-copy-job.pbs
```

Alternatively, the data mover nodes can be used interactively. To gain interactive access to a data mover node, type the following command:

```
qsub -I -P PANDORA -q dtq
```

remembering to replace PANDORA with your abbreviated project name.

The data transfer queue is designed for running file manipulation commands. For example, you could use `wget` (for downloading data from the internet), `scp`, `tar`, `cp`, `mv`, `rm`, `rsync` plus more. This queue, however, is not intended for compute jobs. Compute jobs running in this queue will be terminated without notice. Since no real compute jobs are allowed to run in this queue, jobs run in dtq will not contribute to your project's **fair share**. However, your current fair share weight will impact the priority of jobs submitted to this queue.

Resource limits for dtq can be found in the **queue resource limits** table.

Automating data transfers

The data transfer queue can be used to schedule data transfers and compute jobs that run sequentially. For example, you could set up jobs that do the following:

1. Submit a dtq job to copy data to Artemis for processing.
2. Submit a processing job that automatically runs after the dtq job successfully completes.
3. Upon successful completion of the processing job:

a. Have the processing job copy the resultant data to another location on Artemis; or
b. If the result of processing is to be copied to a destination that is not accessible from the compute nodes, then submit another dtq job to copy this data to its remote location.

4. Optionally, delete the data on Artemis that was used for processing.

This entire workflow can be automated using three PBS scripts: `copy-in.pbs`, `process-data.pbs` and `copy-out.pbs`, shown below.

`copy-in.pbs`:

```
#!/bin/bash
#PBS -P PANDORA
#PBS -q dtq
#PBS -l select=1:ncpus=1:mem=4gb,walltime=00:10:00

rsync -avx /rds/PRJ-PANDORA/input_data /scratch/PANDORA/
```

`process-data.pbs`:

```
#!/bin/bash
#PBS -P PANDORA
#PBS -q defaultQ
#PBS -l select=1:ncpus=4:mem=10gb, walltime=20:00:00

cd /scratch/PANDORA/input_data
my-program < input.inp > output.out
```

`copy-out.pbs`:

```
#!/bin/bash
#PBS -P PANDORA
#PBS -q dtq
#PBS -l select=1:ncpus=1:mem=4gb,walltime=00:10:00

rsync -avx /scratch/PANDORA/input_data/ /rds/PRJ-
PANDORA/output_data
```

Then, you can submit these three scripts (using the `-W depend=afterok` option) to the scheduler as follows:

```
[abcd1234@login1]$ qsub copy-in.pbs
1260945.pbsserver
[abcd1234@login1]$ qsub -W depend=afterok:1260945 process-data.pbs
1260946.pbsserver
[abcd1234@login1]$ qsub -W depend=afterok:1260945:1260946 copy-out.pbs
```

If successful, your jobs should look similar to this if you type `qstat -u abcd1234`:

```
[abcd1234@login1]$ qstat -u abcd1234

pbsserver:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
1260945.pbsserv abcd1234 dtq      copy-in.pb    --    1   1 4096mb 01:00 R   --
1260946.pbsserv abcd1234 small    process-da    --    1   1    2gb 01:00 H   --
1260947.pbsserv abcd1234 dtq      copy-out.p    --    1   1 4096mb 01:00 H   --
```

Note that `process-data.pbs` and `copy-out.pbs` are both in the "H" state, which means they're being "held" by the scheduler until the previous jobs have terminated successfully.

Data transfer script: dt-script

A data transfer script, called `dt-script`, has been created to help simplify data transfer and submitting compute jobs on the transferred data. This tool is especially useful if you need to transfer large amounts of data from RCOS to Artemis before processing it.

The syntax of `dt-script` is:

```
dt-script -P PANDORA -f <from> -t <to> -j <job.pbs>
```

This script uses `rsync` to copy from the source directory `<from>` to the destination directory `<to>` and then submit the PBS job script contained in `job.pbs` that will start once the copy successfully completes.

The arguments this script are shown in the following table:

Short argument	Long argument	Description
`-f`	--from	The source of the data
`-t`	--to	The destination of the data
`-P <project>`	--project <project>	All pbs jobs require a project to run under. Specify it here.
`-notest`	--skip	Skip testing of readable source. Useful if called from a node and /rds is the source which is not available on the calling node
`-w <walltime>`	--walltime <walltime>	Wall Time Required. Default 24:00:00 (24 hours)
`-ncpus <ncpus>`	--ncpus <ncpus>	CPU cores required. Default 1
`-mem <mem>`	--mem <mem>	RAM Required. Default 4GB
`-n`	--name	Set the copy job name. (default "dt-script")
`-rf <rsync extra flags>`	--rflags <rsync extra flags>	Any extra rsync flags you may require
`-j <pbs job script>`	--job <pbs job script>	The pbs job script to run after the copy. if no job script is specified, then no subsequent job is run
`-jf <flags>`	--jobflags <flags>	Any extra 'qsub' flags that may be needed to run the pbs job script specified with --job
`-d <depend option>`	--depend <depend option>	The default depend option is "afterok". You may change this to "afterany" or "afternotok" with this option
`-l <logfile>`	--log <logfile>	Rather than wait for the PBS output files, you may specify a log of stdout and stderr from the rsync command here

The script returns the PBS job ID of the last job it submits as follows:

- If no PBS job script is specified, the PBS job ID of the dtq job is returned and may be used as a dependency of subsequent jobs.
- If a PBS job script is specified, the PBS job ID of that job is returned.

The source code of dt-script is available to all Artemis users. The path to the script is `/usr/local/bin/dt-script`. Feel free to make a copy of this script if you would like to modify it for your needs.

Example dt-script workflow

An example dt-script workflow, using the example project PANDORA, is shown below:

```
[abcd1234@login2]$ dt-script -P PANDORA \
--from /rds/PRJ-PANDORA/mydata \
--to /scratch/PANDORA/ \
--job /scratch/PANDORA/run-processing.pbs
1261577.pbsserver
[abcd1234@login2]$ qstat –u abcd1234

pbsserver:

                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
1261576.pbsserv abcd1234 dtq      dt-script     --    1   1    4gb 24:00 Q   --
1261577.pbsserv abcd1234 small    process-da    --    1   1    2gb 01:00 H   --
```

After verifying the processing job ran successfully, you can transfer data back to RCOS using another dt-script command.

```
[abcd1234@login2]$ dt-script -P PANDORA \
--from /scratch/PANDORA/mydata/ \
--to /rds/PRJ-PANDORA/mydata_output
1261588.pbsserver
[abcd1234@login2]$ qstat -u abcd1234

pbsserver:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
1261588.pbsserv abcd1234 dtq      dt-script     --    1   1    4gb 24:00 Q   --
```

Finally, you can remove any temporary data from Artemis after checking all data was successfully transferred to RCOS:

```
[abcd1234@login2]$ rm /scratch/PANDORA/mydata/*
[abcd1234@login2]$ rmdir /scratch/PANDORA/mydata
```

Compiling on Artemis

Researchers using the Artemis service have access to the following compiler suites to compile and debug their own code:

- GNU (i.e. GCC, G++, gfortran)
- Portland Group (PGI)
- Intel
- Java
- GPU

Small compiliations may be performed on the login nodes, but large compiliations should be done by submitting a job or by starting an **interactive** session. Some large compilations will fail if performed on the login nodes (especially Java compilation).

GNU

To use the GNU compilers, load the GNU compiler modules:

```
module load gcc
```

or

```
module load gcc openmpi-gcc
```

for OpenMPI programs.

There are many versions of the GNU compilers installed on Artemis. To see what versions are available, run the following command:

```
module avail |& grep gcc
```

The following table shows the various commands to use when compiling code, depending on the language and style of execution being used:

Language	Single CPU	Using MPI	Using OpenMP
Fortran	gfortran	mpif90	gfortran -fopenmp
C	gcc	mpicc	gcc -fopenmp
C++	g++	mpicxx	g++ -fopenmp

PGI

To use the PGI compiler, load the PGI modules with the following command:

```
module load pgi
```

or

```
module load openmpi-pgi
```

for OpenMPI programs.

The following table shows the various commands to use when compiling code, depending on the language and style of execution being used:

Language	Single CPU	Using MPI	Using OpenMP
Fortran	pgf90	mpif90	pgf90 -mp
C	pgcc	mpicc	pgcc -mp
C++	pgcc	mpicxx	pgCC -mp

Intel

To use the Intel compilers, load the Intel modules with the following command:

```
module load intel
```

or

```
module load intel-mpi
```

for Intel MPI programs.

The following table shows the various commands to use when compiling code, depending on the language and style of execution being used:

Language	Single CPU	Using MPI	Using OpenMP
Fortran	ifort	mpiifort	ifort -openmp
C	icc	mpiicc	icc -openmp
C++	icpc	mpiicpc	icpc -openmp

Java

To use Java compilers, load the java module with the following command:

```
module load java
```

The java compiler is called `javac`. Node that all Java compilations performed on the login nodes will fail. To compile Java code on Artemis, submit a Java compilation job to Artemis, or compile your Java program in an **interactive** session.

GPU

To take advantage of the GPU accelerated compute nodes, your code will need to be modified to run in the hybrid GPGPU environment. The compile chain will need to specify that the target is GPU via the target argument to the pgcc/pgfortran compiler. This include the compiler argument: -ta=nvidia. This compiler flag tells the PGI compiler to compile for the Nvidia GPU.

The preferred method of accessing GPUs is to use the NVIDIA CUDA toolkit:

```
module load cuda
```

which places the nvcc compiler in your path. See the **NVIDIA CUDA programming guide** for more details.

Research data store (RDS)

The University of Sydney provides a **research data storage service** (called RDS) to researchers at the University of Sydney to meet modern data handling requirements. Data in the RDS is regularly backed up, has built in redundancy, and is covered by the University's **Information Security Policy**.

There are two different RDS services: Classic RDS and Research Computing Optimised Storage (RCOS). Of these two, RCOS is the easiest to use with Artemis.

RCOS

Transferring data between RCOS and Artemis

RCOS is a linux-based (NFS) file server, and can be accessed from Artemis login nodes and the **data transfer queue** in the directory:

```
/rds/PRJ-PANDORA
```

where `PANDORA` is an example abbreviated project name. If your abbreviated project name is TEST, then your RCOS storage is in the directory:

```
/rds/PRJ-TEST
```

If you are logged into Artemis, you can use standard linux shell commands, such as `cp`, `rsync` and `mv` to move data to and from Artemis and RCOS.

Note: Only the Artemis login nodes and **data transfer nodes** can access `/rds`. If you need to analyse data on Artemis, you must transfer your data from `/rds` to either **`/project`** or **`/scratch`** before submitting your job.

Transferring data between RCOS and your local computer

For instructions on transferring data to and from RCOS and your local computer, refer to the **RCOS guide** (**http://sydney.edu.au/dam/intranet/documents/working/it-phones/RCOS-Guide-November-2016.pdf**).

Note: RCOS data is stored in the directory starting with `/rds` only. Do not store your data in `/home` on RCOS because this directory is not visible on Artemis.

For further information about RCOS, see the **Ask Sydney page**.

Classic RDS

Transferring data between Classic RDS and Artemis using a GUI

The most straightforward way to transfer data between Artemis and Classic RDS is to mount Classic RDS as a network drive on your computer following the instructions here: http://staff.ask.sydney.edu.au/app/answers/detail/a_id/174/~/how-do-i-map-my-network-drive-so-i-can-access-my-network-storage%3F, then connect to Artemis using Cyberduck and drag-and-drop data between the Cyberduck window (which shows data on Artemis) and the Classic RDS network drive.

Transferring data between Classic RDS and Artemis using the command line

Alternatively, it is possible to transfer data from Artemis to Classic RDS via the Artemis command line using `smbclient`. To connect to Classic RDS from Artemis, type the following command:

```
smbclient //research-data.shared.sydney.edu.au/VolumeName -U
abcd1234 -W SHARED
```

where `abcd1234` is your unikey and `VolumeName` is the name of the volume where your data is stored, which is usually a faculty code, like FSC for the Faculty of Science, or FEI for the Faculty of Engineering and IT.

If you successfully logged in, your command prompt will change to this:

```
smb: \>
```

Once logged into `smbclient`, your command prompt will allow you to navigate the Classic RDS filesystem using commands like `cd` and `ls`. To move a file from Artemis to Classic RDS, use `put`. To move a file from Classic RDS to Artemis, use `get`. For example, to move a file called `my_data.out` from Artemis to Classic RDS, type:

```
smb: \> put my_data.out
```

To move a file called `other_data.txt` from Classic RDS to Artemis, type:

```
smb: \> get other_data.txt
```

To move multiple files using a single command, use the `mput` and `mget` commands (multiple put and multiple get). By default, smbclient will ask to confirm whether you want to transfer each individual file. To disable this functionality, type `prompt off` before using `mput` or `mget` commands:

```
smb: \> prompt off
smb: \> mput my_data*
```

To run commands on Artemis while logged into smbclient, place an exclamation mark before your command. For example, to view files in your current directory on Artemis, type:

```
smb: \> !ls
```

To transfer data recursively from Artemis to Classic RDS (that is, from the current directory and all sub-directories), use the `recurse` command:

```
smb: \> recurse on
smb: \> prompt off
smb: \> mput directory
```

where `directory` is the directory you want to copy to Classic RDS. The `recurse on` and `prompt off` commands toggle recursive mode and disable prompting for every file transfer, so these only need to be run once. To move data from Classic RDS to Artemis, replace `mput` with `mget`.

Glossary

Name	Definition
Chunk	A virtual computer. It is possible to have multiple virtual computers on one physical computer.
CI	Chief Investigator
Classic RDS	Windows-based file server that is backed up regularly
Core/CPU	Central Processing Unit. Technically these two are different, but they are used interchangeably in this document.
GNU	An extensive collection of open source software, including compilers.
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HPC	High Performance Computing
Intel	Intel Compiler Suite
Intersect	Intersect High Performance Computer
Lustre	A high performance filesystem
MPI	Message Passing Interface
NCI	National Computational Infrastructure
Node	A physical computer. Artemis is comprised of many nodes.
OpenMP	A shared memory parallelism application programming interface
OpenMPI	An open source implementation of Message Passing Interface
PBS	Portable Batch System
PGI	Portland Group Compiler Suite
PuTTY	A secure shell client for Windows
RAM	Random Access Memory
RCOS	Research Computing Optimised Storage - A linux-based file server that is backed up regularly
RDMP	Research Data Management Plan
RDS	Research Data Store
SFTP	Secure File Transfer Protocol
SSH	Secure Shell
Terminal	A command line interface to a computer
UniKey	Your Sydney University Unique ID
VPN	Virtual Private Network