Intermediate HPC

Automating analysis with job arrays on Artemis

Nathaniel Butterworth and Hayim Dar
Sydney Informatics Hub

sih.training@sydney.edu.au

Course Notes:

https://informatics.sydney.edu.au/services/coursedocs/

THE UNIVERSITY OF SYDNEY

Image: thecultureconcept.com

SIH Services

- Nathaniel Butterworth and Tracy Chew
  - Sydney Informatics Hub: Research Technical Officers in Modelling, Simulation, Visualisation, and Bioinformatics.

- Under the Research Portfolio, the Sydney Informatics Hub supports access to high-end research infrastructure and services: promoting quality research outcomes for USYD researchers
  http://sydney.edu.au/research/support/facilities.shtml

  - Supports usage of Artemis HPC and other platforms
  - Data management, analytics, and computational consulting
  - Translational data services and research engineering
  - Training courses, advice, face-to-face support

Overview of today's course

- Today we will learn four methods of automating job submission on the Artemis HPC

- Hopefully this will teach you the concepts so you can modify these methods to suit your own data

- We will learn how to submit:
  1. Array of jobs using the array index as a single input parameter
  2. Array of jobs using the array index to read a more complex set of parameters from a config file
  3. Single job that can be resubmitted on different datasets at different times by using the job name as an input variable
  4. Array of jobs submitted using a 'for loop' instead of a job array

Artemis review: The Data Transfer Queue

- Data transfer queue (dtq)
  - Schedule data transfers in a PBS job
  - /rds mounted on these nodes
  - Can set up ssh keys to enable password-less data transfer (handy for scripted transfer)
  - No fair share cost to use the queue

- https://sydneyuni.atlassian.net/wiki/spaces/RC/ for detailed info

Getting input data for today

1. Connect to Artemis via a terminal
2. Change into your project directory
3. Start an interactive data transfer job:

   ```
   qsub -I -P <project> -q dtq
   ```

4. Use wget to download data:

   ```
   cd $PBS_O_WORKDIR
   wget https://www.dropbox.com/s/b0m31e4cj9wudx9/Automation.tar.gz
   ```

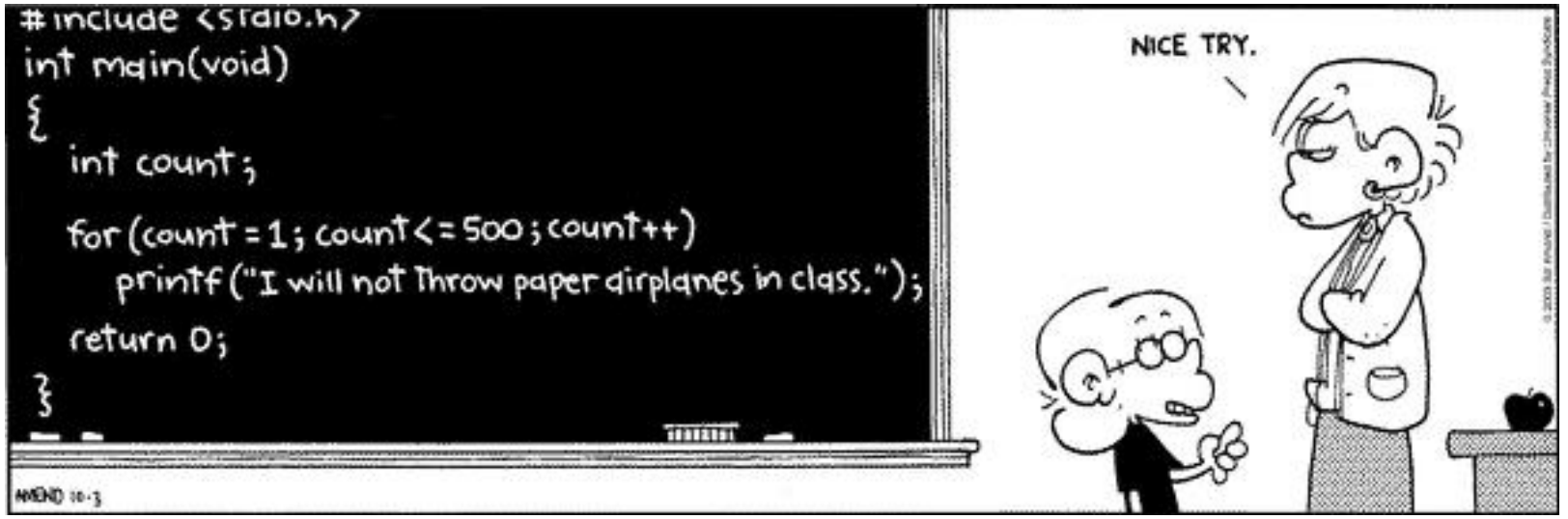
5. Unpack (`tar -zxvf Automation.tar.gz`)
6. Delete the .tar.gz if desired
7. Exit interactive data transfer job:

   ```
   exit
   ```

8. Change into the newly extracted 'Automation' directory

Job arrays

- Often you want to run the same analysis many times at once, varying some piece of input or parameter between jobs
- You could do this by modifying the script and submitting with ‘qsub’ after each saved change, but this is not efficient
- Job arrays enable you to submit multiple replicate jobs simultaneously with a single ‘qsub’ command
  - Use the $PBS_ARRAY_INDEX environment variable and other bash variables to read in the appropriate inputs for each job in the array



Bash variables 101

- Using variables within your jobs scripts makes them portable (easy to reuse across multiple runs/datasets/projects)
- If you are familiar with any other programming language, the concept of a variable in bash is the same
- A variable holds some value – the value can be a number, a word, a string, a list of things…
- A variable must have a name without whitespace or metacharacters (eg '$', '/')
- You assign the value to the variable with the '=' sign, eg

```
hpc_name=Artemis
```

creates a variable called 'hpc_name' and it's value is 'Artemis'

- To get at the contents of a variable, use the '$' symbol, eg

```
echo USYD HPC is named $hpc_name
```

would print 'USYD HPC is named Artemis' to standard output

Bash variables 101

- You can use multiple variables in combination, eg

```
sample_ID=502
input_directory=/project/BioServ/Data
output_file=$input_directory/$sample_ID.out
echo Writing output to $output_file
```

would print 'Writing output to /project/BioServ/Data/502.out'

- You can use squiggly brackets to improve visual clarity of variables in combination, eg

```
output_file=${input_directory}/${sample_ID}.out
```

PBS environment variables

- PBS has predetermined **environment variables** that are assigned values when you submit a job
- The most commonly used is PBS_O_WORKDIR. It's value is the full pathname of the directory from which you submitted the job. You can use environment variables just like you would use bash variables, eg

```
cd $PBS_O_WORKDIR
```

  - This is important because **PBS considers your HOME directory to be your working directory**, so you must use 'cd' (or full pathnames) to point to your data within the job script

- **Essential for job arrays** is the environment variable PBS_ARRAY_INDEX
  - When you submit a job array, each job within that array is assigned a unique index (integer)
    - You have control over the integers that are used
  - The value of the integer can be used in the script by calling on $PBS_ARRAY_INDEX

Job arrays

- To run a job array, include the PBS directive '-J'
- The number of jobs in the array and the integers used are controlled by you
  - Specify a numeric range
  - Specify an optional 'step' value

Examples:

```
#PBS -J 1-100
```

- Run 100 jobs, with PBS_ARRAY_INDEX values of 1,2,3, 4,5…100

```
#PBS -J 1-22:3
```

- Run 8 jobs, with PBS_ARRAY_INDEX values 1,4,7,10,13…22

```
#PBS -J 50-500:50
```

- Run 10 jobs, with PBS_ARRAY_INDEX values 50,100,150, 200, 250…500

Job arrays

- When you submit the job array PBS script via qsub, each job in the array is submitted to the scheduler at the same time
  - If your '`-J`' directive includes 10 jobs, you will have submitted 10 jobs
  - Some may run right away and some may enter the queue
  - The resources you have requested (cores, RAM, walltime) are granted to EACH job, not divided amongst the jobs
- The array index for each job is stored in the environment variable PBS_ARRAY_INDEX
  - This number can itself be used as an input parameter to the job
  - This number can be used to retrieve information for each specific job within the array from a configuration file
- The whole job array can be queried or deleted with empty square brackets or sub-jobs can be specified with the array index inside square brackets
  - Each job in the array has the same job ID, but a different index

```
qstat 1578913[]     qstat 1578913[1]     qstat 1578913[2]
qdel 1578913[]      qdel 1578913[1]      qdel 1578913[2]
```

Job arrays

Practice case 1a:

1. Change into the 'Demo-a' directory
2. Open the script 'array_demo.pbs'

$PBS_O_WORKDIR is your job submission directory

```
#Demonstrate the behaviour of job arrays

#PBS -P BioServ
#PBS -N CaliDemo
#PBS -l select=1:ncpus=1:mem=1gb
#PBS -l walltime=00:01:00
#PBS -J 1-10
#PBS -q small-express

sleep 60

#Make a  new directory for this job in array inside our current directory
mkdir $PBS_O_WORKDIR/Demo$PBS_ARRAY_INDEX

#Change into it
cd $PBS_O_WORKDIR/Demo$PBS_ARRAY_INDEX

#Send some output to a file
printf "This is job number $PBS_ARRAY_INDEX in the array\n" > job$PBS_ARRAY_INDEX.log
```

Array of 10 jobs, with values 1,2,3,4,5,6,7,8,9,10

Using the variable $PBS_ARRAY_INDEX within the script

Job arrays

Practice case 1a:

3. Modify the project at **-P**
4. Modify the job name at **-N**
5. Submit the job array

```
qsub array_demo.pbs
```

6. Use qstat to query the whole array of jobs

```
qstat -xt jobID[]
```

7. Use qstat to query individual jobs within the array

```
qstat -x jobID[1]
qstat -x jobID[2]
```

```
[skol2049@login4 Demo]$ qsub array_demo.pbs
1398913[].pbsserver
[skol2049@login4 Demo]$ qstat -t 1398913[]
Job id            Name             User             Time Use S Queue
----------------  ---------------- ---------------- -------- - -----
1398913[].pbsserv Demo             skol2049               0 B small-express
1398913[1].pbsser Demo             skol2049        00:00:00 R small-express
1398913[2].pbsser Demo             skol2049        00:00:00 R small-express
1398913[3].pbsser Demo             skol2049        00:00:00 R small-express
1398913[4].pbsser Demo             skol2049        00:00:00 R small-express
1398913[5].pbsser Demo             skol2049        00:00:00 R small-express
1398913[6].pbsser Demo             skol2049        00:00:00 R small-express
1398913[7].pbsser Demo             skol2049        00:00:00 R small-express
1398913[8].pbsser Demo             skol2049        00:00:00 R small-express
1398913[9].pbsser Demo             skol2049        00:00:00 R small-express
1398913[10].pbsse Demo             skol2049        00:00:00 R small-express
[skol2049@login4 Demo]$ qstat -x 1398913[1]
Job id            Name             User             Time Use S Queue
----------------  ---------------- ---------------- -------- - -----
1398913[1].pbsser Demo             skol2049        00:00:00 R small-express
[skol2049@login4 Demo]$ qstat -x 1398913[10]
Job id            Name             User             Time Use S Queue
----------------  ---------------- ---------------- -------- - -----
1398913[10].pbsse Demo             skol2049        00:00:00 R small-express
[skol2049@login4 Demo]$ qstat -u skol2049

pbsserver:
                                                                  Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
1398913[].pbsse skol2049 small-ex Demo          --    1   1    1gb 00:01 B   --
[skol2049@login4 Demo]$
```

If you leave off the square brackets when querying the job, you will receive an 'Unknown Job ID' error

Job arrays

Practice case 1a:

8. Check the output with 'ls', 'grep', and 'cat' - The array indexes 1 through 10 were used as variables to create directories and output for each job

```
grep -L "Exit Status: 0" *_usage
```
(returns nothing, remove –L for the opposite)

9. Note the '.o' and '.e' log files

- Each job in the array has its own set of log files
- They are named in the usual way, plus the array index:

```
JobName.oJobID.Index
JobName.eJobID.Index
JobName.oJobID.Index_usage
```

```
[cwil2281@login2 Demo]$ ls
array_demo.pbs              CaliDemo.o534089.5        Demo10
CaliDemo.o534089.1          CaliDemo.o534089.5_usage  Demo2
CaliDemo.o534089.10         CaliDemo.o534089.6        Demo3
CaliDemo.o534089.10_usage   CaliDemo.o534089.6_usage  Demo4
CaliDemo.o534089.1_usage    CaliDemo.o534089.7        Demo5
CaliDemo.o534089.2          CaliDemo.o534089.7_usage  Demo6
CaliDemo.o534089.2_usage    CaliDemo.o534089.8        Demo7
CaliDemo.o534089.3          CaliDemo.o534089.8_usage  Demo8
CaliDemo.o534089.3_usage    CaliDemo.o534089.9        Demo9
CaliDemo.o534089.4          CaliDemo.o534089.9_usage
CaliDemo.o534089.4_usage    Demo1
[cwil2281@login2 Demo]$ cat Demo10/job10.log
This is job number 10 in the array
[cwil2281@login2 Demo]$
```

Job arrays

Practice case 1b:

1. Change into the 'Demo-b' directory
2. Open the script 'array_demo.pbs'

.o and .e files can be individually named and managed using ^array_index^ in #PBS directives

.o and .e files are copied into the Demo^array_index^ directory and renamed to stdout and stderr

```
#PBS -P BioServ
#PBS -N Demo
#PBS -l select=1:ncpus=1:mem=1gb
#PBS -l walltime=00:01:00
#PBS -J 1-10
#PBS -o Demo^array_index^/stdout
#PBS -e Demo^array_index^/stderr
#PBS -q small-express

sleep 60

#Make a  new directory for this job in array inside our current directory
mkdir $PBS_O_WORKDIR/Demo$PBS_ARRAY_INDEX

#Change into it
cd $PBS_O_WORKDIR/Demo$PBS_ARRAY_INDEX

#Send some output to a file
printf "This is job number $PBS_ARRAY_INDEX in the array\n"
```

Job arrays

Practice case 1b:

3. Modify the project at –P
4. Modify the job name at –N
5. Submit the job array

   `qsub array_demo.pbs`

6. Check the output with 'ls' and 'cat'
   - The array indexes 1 through 10 were used as variables to create directories
   - `cat Demo*/stdout`
7. The '.o' and '.e' log files have been moved to each directory and renamed to 'stdout' and 'stderr'

```
[skol2049@login4 Demo-b]$ ls
array_demo.pbs  Demo1  Demo10  Demo2  Demo3  Demo4  Demo5  Demo6  Demo7
[skol2049@login4 Demo-b]$ ls Demo*
Demo1:
stderr  stdout  stdout_usage

Demo10:
stderr  stdout  stdout_usage

Demo2:
stderr  stdout  stdout_usage

Demo3:
stderr  stdout  stdout_usage

Demo4:
stderr  stdout  stdout_usage

Demo5:
stderr  stdout  stdout_usage

Demo6:
stderr  stdout  stdout_usage

Demo7:
stderr  stdout  stdout_usage

Demo8:
stderr  stdout  stdout_usage

Demo9:
stderr  stdout  stdout_usage
```

Job arrays: summary so far

- Now you know that:
  - Arrays are specified to PBS with '–J' directive
  - Each job in the array is assigned an index number which is stored in PBS_ARRAY_INDEX
  - Value of index can be **controlled** by you **with *range* and *step***
  - Value of index can be directly **used** within the job ***as a variable***
  - Value of index is appended as suffix to job logs, with separate logs for each job in array
  - Value of index can be used to query or delete specific jobs within the array inside square brackets
  - The whole array can be queried or deleted with empty square brackets
  - Use `^array_index^` to manage .o and .e files in #PBS -o and #PBS -e directives

- Next, we will use PBS_ARRAY_INDEX in a more meaningful way

Four (4) methods for automating job submission

1 Job arrays: using the array index as input parameter

- This approach is useful for analyses that require a single integer value to change between runs, eg modelling and simulation

- Scenario: you want to assemble a genome but you don't know which k-mer value is best for your data. You need to run the analysis for multiple k-mer values and then choose which has produced the best output. Understandably, you don't like the idea of repeatedly modifying the job script and re-submitting the analysis manually.
- Solution: prepare a PBS script to run a job array, using the PBS array index as the k-mer value for each sub-job.

1 Job arrays: using the array index as input parameter

Practice case 2:

1. Change into the 'Assembly' directory – the 2 'fastq.gz' files are your raw data
2. Open the assembly.pbs script

```
#PBS -P BioServ
#PBS -N NameVelvet
#PBS -l select=1:ncpus=2:mem=6gb
#PBS -l walltime=00:10:00
#PBS -J 31-51:10
#PBS -q small-express

# Load velvet
module load velvet

# Limit velvet to 2 cores
export OMP_THREAD_LIMIT=2
export OMP_NUM_THREADS=2

cd $PBS_O_WORKDIR

# Run velvet hashing program using array index value as input parameter for kmer size
# First argument to velveth is output directory - this is created by velvet. 2nd argument is k-mer size
velveth kmer$PBS_ARRAY_INDEX $PBS_ARRAY_INDEX -fastq.gz -shortPaired -separate $PBS_O_WORKDIR/134_R1.fastq.gz $PBS_O_WORKDIR/134_R2.fastq.gz

# Run velvet graph program using hash tables for this value of array index
velvetg kmer$PBS_ARRAY_INDEX -exp_cov 9
```

Array of 3 jobs, with index values 31, 41, 51

PBS environment variables used to specify directories

Array index used as value for k-mer parameter

1 Job arrays: using the array index as input parameter

Practice case 2:

3. Modify the project at –P
4. Modify the job name at –N
5. Submit the job array

```
qsub assembly.pbs
```

6. Use 'qstat' to query the whole array of jobs

```
qstat -x jobID[]
```

7. Use qstat to query individual jobs within the array

```
qstat -x jobID[31]
qstat -x jobID[41]
```

```
[cwil2281@login2 Assembly]$ qsub assembly.pbs
577943[].mgmt1
[cwil2281@login2 Assembly]$ qstat -x 577943[]
Job id            Name             User             Time Use S Queue
----------------  ---------------- ---------------  -------- - -----
577943[].mgmt1    CaliVelvet       cwil2281                0 Q training_test
[cwil2281@login2 Assembly]$ qstat -x 577943[]
Job id            Name             User             Time Use S Queue
----------------  ---------------- ---------------  -------- - -----
577943[].mgmt1    CaliVelvet       cwil2281                0 B training_test
[cwil2281@login2 Assembly]$ qstat -x 577943[21]
Job id            Name             User             Time Use S Queue
----------------  ---------------- ---------------  -------- - -----
577943[21].mgmt1  CaliVelvet       cwil2281         00:00:10 R training_test
[cwil2281@login2 Assembly]$ qstat -x 577943[31]
Job id            Name             User             Time Use S Queue
----------------  ---------------- ---------------  -------- - -----
577943[31].mgmt1  CaliVelvet       cwil2281         00:00:11 R training_test
[cwil2281@login2 Assembly]$ qstat -x 577943[41]
Job id            Name             User             Time Use S Queue
----------------  ---------------- ---------------  -------- - -----
577943[41].mgmt1  CaliVelvet       cwil2281         00:00:31 R training_test
```

Job status when querying whole array with [ ]

Q: all jobs in the array are in queue

B: at least one job in the array has left queue

F: all jobs in the array have finished

1 Job arrays: using the array index as input parameter

Practice case 2:

- Your completed job should produce new output directories using the array index values in the directory name, as well as log files
- Inside each directory is the output for each sub-job
  - Check that you have the expected output (eg with 'ls')
  - Also check your error log

```
grep -L "Exit Status: 0" <jobName>.*usage | xargs cat
```

- You could then run a script over this output to determine which value of k was best for your data and proceed with the most favourable output dataset

2 Job arrays: using the array index to read a config file

- This approach is useful when you have a pipeline that requires a number of different input parameters for each run of the job

- Scenario: You have 4 samples that have been DNA sequenced. You want to run the same sequence analysis pipeline over each but with a few different inputs, without having to modify and submit the script 4 times.
- Solution: set up a PBS job array script and use bash variables for the different parameters. Script the job to read the relevant parameter information for each sample from a config file.

2 Job arrays: using the array index to read a config file

- The dataset in more detail:
  - Your data comes from different species so sub-jobs need to use different reference files
  - They come from different breeds and were sequenced at different institutions and this information needs to be embedded within the output file
  - You have a different number of input files per sample, so you use bash scripting to save the file names in variables and loop over them
  - Apart from these differences, the pipeline you want to run over each sample is the same
  - You save the information for each sample in a tab-delimited text file – a 'config file' – which is read by the PBS script to ensure the right details are applied to the right sample

#ArrayIndex	SampleID	Breed	Reference	SeqCentre
1	USCF70	Dalmatian	canfam3_chr20.fasta	Ramaciotti
2	BD394	Boxer	canfam3_chr20.fasta	UCDavis
3	FM0238	FranchesM	equcab2_chr20.fasta	UBern
4	FM0570	FranchesM	equcab2_chr20.fasta	UBern

2 Job arrays: using the array index to read a config file

Practice case 3:

1. Change into the 'Alignment' directory
2. Open the align.pbs script – note how the array index is used to pull the data from the config file

```
cd $PBS_O_WORKDIR
config=samples.config #If config file not in same directory from where qsub is issued, must specify full pathname to config

# Get the sample ID from config file
sample=$(awk -v taskID=$PBS_ARRAY_INDEX '$1==taskID {print $2}' $config)
# Get the breed name from config file
breed=$(awk -v taskID=$PBS_ARRAY_INDEX '$1==taskID {print $3}' $config)
# Get the reference name config file
reference=$(awk -v taskID=$PBS_ARRAY_INDEX '$1==taskID {print $4}' $config)
# Get the sequencing centre from config file
centre=$(awk -v taskID=$PBS_ARRAY_INDEX '$1==taskID {print $5}' $config)
# Get the list of runs for this sample from config file
runs=$(awk -v taskID=$PBS_ARRAY_INDEX '$1==taskID {print $6}' $config | tr "," "\n")

echo Aligning $breed sample $sample to $reference - combining data from HiSeq runs $runs
```

Array index is first column in config file – use it with 'awk' to capture the various required information in variables

The 6th column could hold a comma-separated list, so split it on commas into an 'array variable'

2 Job arrays: using the array index to read a config file

Practice case 3:

3. This script uses more complex bash scripting – **peruse the script and please ask for clarification if you don't understand some syntax**. Having a small amount of bash scripting knowledge under your belt can really help set up pipelines on Artemis, even if you code in another language eg Python

3a. Before proceeding further, let's do "dry run". Comment out (prepend '#') to each line after the '*echo*' command so that the script doesn't actually execute the analysis programs.

3b. Run the script as a simple *bash* script, ie do not run with *qsub:*

```
bash align.pbs
```

3c. Inspect standard output – is it what we expected?

2 Job arrays: using the array index to read a config file

Practice case 3:

3. This script uses more complex bash scripting – **peruse the script and please ask for clarification if you don't understand some syntax**. Having a small amount of bash scripting knowledge under your belt can really help set up pipelines on Artemis, even if you code in another language eg Python

4. Modify the project at **-P**

5. Modify the job name at **-N**

6. Submit the job array

```
qsub align.pbs
```

7. Use 'qstat' to query the whole array of jobs or individual sub jobs

```
qstat -x jobID[]  qstat -x jobID[2]
```

8. View your output and log files to check the array ran correctly. You should have a new output directory for each sample containing some BAM and BAI files

- Your error log will not be empty: BWA prints update messages there. Run tail to check for fatal errors.

3 Alternative automation methods: job name as variable

- This approach is useful when you want to run the same job numerous times – **but not all at the same time like a job array** – and the job name is sufficient to provide the correct inputs

- Scenario: You are using a program called "POV-Ray" to render (create) images. The rendering is quite complex, so each image has a long script and auxiliary files that are stored in that image's directory. Each directory has a main script (eg castle.pov) in a directory with the same name that you can pass to POV-Ray to render the image. You don't want to use a separate job script for each image, yet you want the flexibility to render only a single image file, a subset of images or all of them at once.
- Solution: Create a job script for your analysis that reads in the image name (eg castle) and pass this as a variable to PBS to direct the script to the image you want to render.

3 Alternative automation methods: job name as variable

Practice case 4:

1. Change into the 'Povray' directory – the raw data are saved in folders with different image names (e.g. castle, escargot, fridge)
2. Open single_image.pbs
3. Modify the project name at –P

```
#!/bin/bash

#PBS -P BioServ
#PBS -l select=1:ncpus=1:mem=1gb
#PBS -l walltime=0:10:00
#PBS -q small-express

module load povray

cd $PBS_O_WORKDIR/$PBS_JOBNAME
povray res $PBS_JOBNAME.pov
```

Note that there is no '–N' job name directive in this script! The PBS environment variable $PBS_JOBNAME is taken from the command line

Each directory has a POV-Ray file with the same name as the directory.

3 Alternative automation methods: job name as variable

Practice case 4:

4. Submit the job, **passing the directory name as the job name** to PBS

```
qsub -N castle single_image.pbs
```

5. Check the status of the job in the usual way

```
qstat -x JobID
```

6. Once our job has finished, check it has run successfully
   - You should have an image file called castle.png in the 'castle' directory.
   - Open in firefox (firefox is installed on Artemis):
   - `firefox castle/castle.png`
7. To process another image file, pass a different image name to qsub.
8. Submit the same job for escargot:

```
qsub -N escargot single_image.pbs
```

9. Once again, check your output is as expected (open the image file escargot.png)

4 Alternative automation methods: for loop

❖ A bash 'for loop' can also be used to submit multiple runs of the same jobs from a single script

- In most cases where a for loop could be used to automate multiple job submission, **a job array is preferable**
  - Easier for the user to script, submit and monitor (single job ID)
  - More **efficient** for the scheduling software to manage, especially with very large numbers of sub-jobs

- Scenario: you want to submit the same analysis to run over multiple similarly-structured datasets, but the input parameter required to specify the data is not an integer (could be a letter, decimal, list of words…) and you couldn't be bothered (!!?) to set up a config file to read in the right inputs for each sub-job.
- Solution: use a 'for loop' to iterate over the range of letters, decimals or list of words and pipe to qsub

4 Alternative automation methods: for loop

Practice case 5:

1. Open the shell script loop.sh
2. Observe the varying syntax for looping over a range of values – integers, letters, decimals or a list
3. Run the shell script

   `bash loop.sh`

   - Check that the output corresponds to your understanding of the 'for loop' syntax
   - Two example for loops:

```
# Iterate over list of words/strings
words=(One Two Three '4 on the floor')
for i in "${words[@]}"
do
        echo The string is $i
done
printf "\n"
```

```
# Iterate over range of numbers
for i in {1..4}
do
   echo The number is $i
done
printf "\n"
```

4 Alternative automation methods: for loop

Practice case 5:

4. Now open povray.sh

- You have 6 images to render and you want to submit them all simultaneously.
- The names of all 6 images are stored in a list called 'images'. The for loop iterates over this list, submitting each image as a separate job using the single_image.pbs script.

```
#!/bin/bash

images=(castle escargot fridge glass plants snow)

for image in ${images[@]}
do
    qsub -N $image single_image.pbs
done
```

4 Alternative automation methods: for loop

Practice case 5:

5. Submit the job AS A BASH SCRIPT

```
bash povray.sh
```

6. Check the job status with 'qstat -u' instead of using separate job IDs to query

7. Check your output – every directory should have a corresponding png image file

```
find . -name "*.png" | xargs firefox
```

```
[skol2049@login2 Povray]$ bash povray.sh
1399546.pbsserver
1399547.pbsserver
1399548.pbsserver
1399549.pbsserver
1399550.pbsserver
1399551.pbsserver
[skol2049@login2 Povray]$ qstat -u skol2049

pbsserver:
                                                                  Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
1399546.pbsserv skol2049 small-ex castle        --    1   1    1gb 00:10 Q   --
1399547.pbsserv skol2049 small-ex escargot      --    1   1    1gb 00:10 Q   --
1399548.pbsserv skol2049 small-ex fridge        --    1   1    1gb 00:10 Q   --
1399549.pbsserv skol2049 small-ex glass         --    1   1    1gb 00:10 Q   --
1399550.pbsserv skol2049 small-ex plants        --    1   1    1gb 00:10 Q   --
1399551.pbsserv skol2049 small-ex snow          --    1   1    1gb 00:10 Q   --
```

Exercises

Exercise 1:

Set up the same analysis as in povray.sh, but submit the 6 sub-jobs **as a job array** rather than a 'for loop'. There are multiple ways of doing this, but for this exercise, try first creating a config file and using it for the array job.

Exercises

Exercise 1: A possible solution

ex1.pbs:

```
#!/bin/bash
#PBS -P SASTEST
#PBS -l select=1:ncpus=1:mem=4gb
#PBS -l walltime=00:20:00
#PBS -q small-express
#PBS -J 1-6

module load povray

cd $PBS_O_WORKDIR

config=ex1.config
image=$(awk -v taskID=$PBS_ARRAY_INDEX '$1==taskID {print $2}' $config)

cd $image
povray res ${image}.pov
```

Config file:

```
# Array ID, image
1           castle
2           escargot
3           fridge
4           glass
5           plants
6           snow
```

Exercises

Exercise 2:

Povray can render images at any desired resolution. We previously specified image resolution in a file called res.ini, but we could have also specified the resolution on the command line:

```
povray -W640 -H480 castle.pov
```

Update the config file to render the images at the following resolutions, and submit another array job that renders each image with the resolution specified in the config file.

Image	Width	Height
Castle	480	360
Escargot	480	360
Fridge	768	576
Glass	1024	768
Plants	480	360
Snow	320	120

Exercises

Exercise 2: A possible solution

ex2.pbs:

```
#!/bin/bash
#PBS -P SASTEST
#PBS -l select=1:ncpus=1:mem=2gb
#PBS -l walltime=00:20:00
#PBS -q small-express
#PBS -J 1-6

module load povray

cd $PBS_O_WORKDIR

config=ex2.config

image=$(awk -v taskID=$PBS_ARRAY_INDEX '$1==taskID {print $2}' $config)
width=$(awk -v taskID=$PBS_ARRAY_INDEX '$1==taskID {print $3}' $config)
height=$(awk -v taskID=$PBS_ARRAY_INDEX '$1==taskID {print $4}' $config)

cd $image
povray -W${width} -H${height} ${image}.pov
```

Config file:

# Array ID	image	width	height
1	castle	480	360
2	escargot	480	360
3	fridge	768	576
4	glass	1024	768
5	plants	480	360
6	snow	320	120

Adaptability

- Now you know how to submit job arrays with or without config files
- You also know how to use job names or 'for loops' to automate job submission as an alternative to PBS job arrays
- How you best apply these automation methods to your data depends on your unique data structure and analysis requirements
  - This may require creativity, and hopefully today's course has shown that just a little bit of bash scripting can go a long way to making your pipelines portable and easy to automate

Acknowledgements

- DNA sequence sample data (sub-sampled) was contributed by the Wade lab in Faculty of Veterinary Science
- The content on job name and 'for loop' automation was adapted from https://my.cqu.edu.au/web/eresearch/
- POV-Ray image source code came from: http://www.oyonale.com/modeles.php?format=POV&lang=en

Feedback

- We would be grateful to receive any feedback, comments or suggestions: sih.training@sydney.edu.au
- Thank you for your participation! ☺
- Check https://informatics.sydney.edu.au/ for more info, docs, and courses