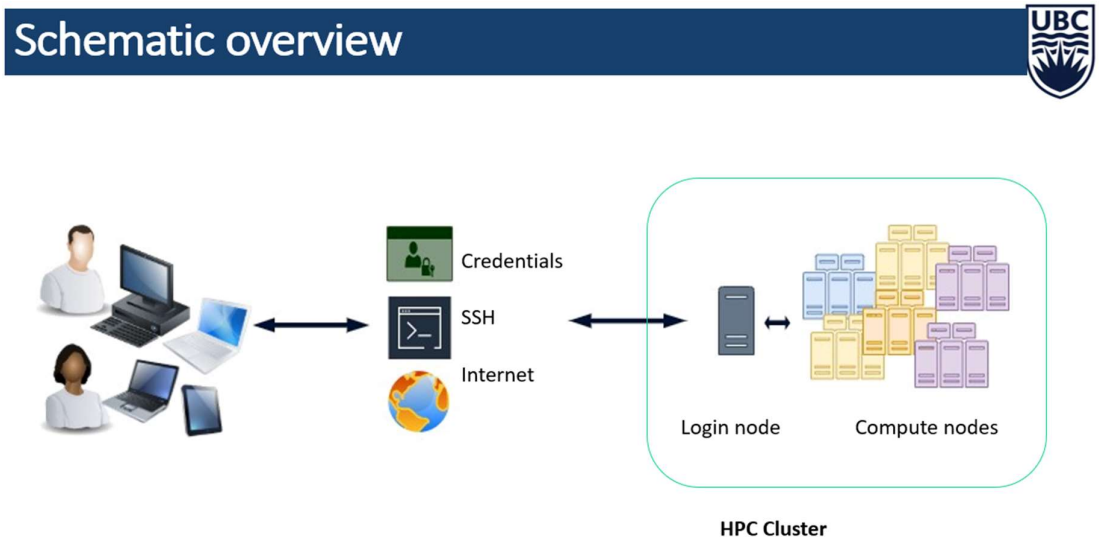
Day 03 Workshop Instruction Manual

Disclaimer: The primary goal of this workshop is to explore how to use a high-performance computing (HPC) environment. We will use various bioinformatics tools as examples to understand the cluster environment. Please be sure to consult the application manual for each tool, and choose the options and flags that are most appropriate for your own research question.

Table of Contents

Recap of Day 1 & 2



We have successfully used the bioinformatics tools for data analysis
How to speed up the computation?

Interactive compute resource request

```
[user01@login1 ~]$ salloc --mem=10G --time=1:00:00 \
 --cpus-per-task=1
salloc: Granted job allocation 28
salloc: Waiting for resource configuration
salloc: Nodes node1 are ready for job
[user01@node1 ~]$
```

Interactive sessions are great for testing, but for reproducibility and batch workflows.

Non-Interactive Job Submissions

On a HPC cluster, we use non-interactive jobs: prepare everything up-front, submit it, then monitor it.

- *Prepare a job script:* Write a file with all the instructions
- *Submit the job script*
- *Monitor the job:* Once submitted, we can check the status of our job

i. Prepare a job script

Yesterday we tried '`salloc`' command:

```
[user01@login1 ~]$ salloc --mem=10G --time=1:00:00 \
 --cpus-per-task=1
salloc: Granted job allocation 28
salloc: Waiting for resource configuration
salloc: Nodes node1 are ready for job


[user01@node1 ~]$ module load fastqc/0.11.9

[user01@node1 ~]$ cd \
/home/user01/scratch/omics_workshop/preprocessed_data/D2_set01

[user01@node1 D2_set01]$ ls -l *.fastq.gz

[user01@node1 D2_set01]$ fastqc --threads 1 subsampled_R1.fastq.gz
```

Here is how we can convert it into a script:

```
[user01@node1 preprocessed_data]$ mkdir D3_set01
[user01@node1 preprocessed_data]$ cd D3_set01/
[user01@node1 D3_set01]$ vi fastqc_job.sh
```

```
#!/bin/bash

#SBATCH --cpus-per-task=1
#SBATCH --mem=10G
#SBATCH --time=01:00:00

module load fastqc/0.11.9

fastqc --threads 1 \
~/scratch/omics_workshop/preprocessed_data/D2_set01/subsampled_R1.fastq.gz
```

Save this file for example, as `fastqc_job.sh`.

Or copy the script:

```
$ cp ~/projects/def-sponsor00/Day03/fastqc_job.sh .
```

ii. Submit a job script

We would submit the job script using **`sbatch`**:

```
[user01@node1 D3_set01]$  sbatch fastqc_job.sh
Submitted batch job 15
```

Here, `15` is the `JOBID`. We will use it to monitor the job's status, check logs, etc.

iii. Job monitoring

Checking Job Status with **`squeue`**:

```
[user01@node1 D3_set01]$ squeue
             JOBID     USER      ACCOUNT           NAME  ST  TIME_LEFT
NODES CPUS TRES_PER_N MIN_MEM NODELIST (REASON)
                28   user01 def-sponsor0    interactive   R      13:48     1
1        N/A     10G node1 (None)
                15   user01 def-sponsor0  fastqc_job.sh   R      59:57     1
1        N/A     10G node2 (None)
```

`seff` summarizes how efficiently a completed job used CPU and memory resources on a SLURM cluster.

```
[user01@node1 D3_set01]$ seff 15
Job ID: 15
Cluster: omics
User/Group: user01/user01
State: COMPLETED (exit code 0)
Cores: 1
CPU Utilized: 00:00:46
CPU Efficiency: 92.00% of 00:00:50 core-walltime
Job Wall-clock time: 00:00:50
Memory Utilized: 279.21 MB
Memory Efficiency: 2.73% of 10.00 GB
```

Another useful job monitoring command is `sacct`.
`sacct` (SLURM Accounting) is a command used to check the history and details of jobs on the cluster.

For example: Let us view the details for JOB ID 30

```
[user01@node1 D3_set01]$ sacct -j 15
    Account      User JobID                    Start                 End
AllocCPUS    Elapsed                  AllocTRES    CPUTime     AveRSS
MaxRSS MaxRSSTask MaxRSSNode        NodeList ExitCode                State
---------- --------- ------------ ------------------- ------------------
---------- ---------- ------------------------------ ---------- ----------
---------- ---------- ---------- --------------- -------- ----------------
----
def-spons+    user01 15           2025-11-07T13:39:59 2025-11-07T13:40:49
1   00:00:50 billing=1,cpu=1,mem=10G,node=1   00:00:50
node1      0:0            COMPLETED
def-spons+           15.batch     2025-11-07T13:39:59 2025-11-07T13:40:49
1   00:00:50           cpu=1,mem=10G,node=1   00:00:50    285912K
285912K          0      node1           node1      0:0
COMPLETED
def-spons+           15.extern    2025-11-07T13:39:59 2025-11-07T13:40:49
1   00:00:50 billing=1,cpu=1,mem=10G,node=1   00:00:50       188K
188K          0      node1           node1      0:0            COMPLETED
```

For example, we can filter results by date or time to see only recent activity:
Jobs from today:

```
$ sacct --starttime today
```

Jobs from the last 3 days:

```
$ sacct --starttime $(date -d '3 days ago' +%Y-%m-%d)
```

Jobs for a specific user (e.g., user01):

```
$ sacct -u user01 --starttime today
```

How does a job scheduler work?

Improving Job efficiency

If we want to speed up, we can increase `--cpus-per-task` and pass `--threads` accordingly:

```
#SBATCH --cpus-per-task=2
#SBATCH --mem=4G
```

We will use **Trimmomatic**, a tool for quality trimming of sequencing reads, as our example.

a. Input & Output Setup

We have the following directory structure:

```
/home/user01/scratch/omics_workshop/
├── preprocessed_data/
│   ├── D2_set01/      ← Input data here
│   └── D3_set01/      ← We will launch our SLURM script here
```

Input files are here:

```
/home/user01/scratch/omics_workshop/preprocessed_data/D2_set01
/
```

We will launch and run the script from:

```
/home/user01/scratch/omics_workshop/preprocessed_data/D3_set01
/
```

b. Define the Variables

We will define a variable for the input directory, so the script is reusable:

```
INPUT_DIR=/home/user01/scratch/omics_workshop/preprocessed_dat
a/D2_set01
```

c. Prepare the SLURM Script

Save the following script as trimmomatic_job.sh:

```
cp  ~/projects/def-sponsor00/Day03/trimmomatic_job.sh .
```

```
#!/bin/bash
#SBATCH --time=1:00:00
#SBATCH --mem=10G
#SBATCH --cpus-per-task=2

# ------------------------------
# Load required modules
# ------------------------------
module load trimmomatic

# ------------------------------
# Define paths and variables
# ------------------------------
INPUT_DIR=/home/user01/scratch/omics_workshop/preprocessed_data/D2_set01
OUTPUT_DIR=/home/user01/scratch/omics_workshop/preprocessed_data/D3_set01

# Move to output directory
cd $OUTPUT_DIR

# ------------------------------
# Run Trimmomatic
# ------------------------------
java -jar $EBROOTTRIMMOMATIC/trimmomatic-0.39.jar PE -threads 2 \
    $INPUT_DIR/subsampled_R1.fastq.gz $INPUT_DIR/subsampled_R2.fastq.gz
\
    subsampled_R1.trim.fastq.gz R1_un.fastq.gz \
    subsampled_R2.trim.fastq.gz R2_un.fastq.gz \
    SLIDINGWINDOW:3:20 MINLEN:50
```

d. Submit the Job

We can launch this job from the D3_set01 directory:

```
$ cd \
/home/user01/scratch/omics_workshop/preprocessed_data/D3_set01

$ sbatch trimmomatic_job.sh
```

We will get output files as:

1. `subsampled_R1.trim.fastq.gz`
2. `subsampled_R2.trim.fastq.gz`
3. `R1_un.fastq.gz`
4. `R2_un.fastq.gz`

Let's examine the job using SLURM commands:

1. squeue reports the state of jobs or job steps

```
[user01@login1 D3_set01]$ squeue
             JOBID     USER      ACCOUNT           NAME  ST  TIME_LEFT NODES CPUS
TRES_PER_N MIN_MEM NODELIST (REASON)
              32   user01 def-sponsor0 trimmomatic_jo   R      59:57     1    2
N/A     10G node1 (None)
```

2. sacct is used to report job or job step accounting information about active or completed jobs

```
[user01@login1 D3_set01]$ sacct -j 32
    Account      User JobID                         Start                 End
AllocCPUS    Elapsed                       AllocTRES    CPUTime     AveRSS
MaxRSS MaxRSSTask MaxRSSNode        NodeList ExitCode                State
---------- --------- ------------ ------------------- -------------------
---------- ---------- ------------------------------ ---------- ----------
---------- ---------- ---------- --------------- -------- ----------------
----
def-spons+    user01 32           2025-11-01T17:35:56             Unknown
2   00:00:59  billing=2,cpu=2,mem=4G,node=1   00:01:58
node1      0:0              RUNNING
def-spons+           32.batch     2025-11-01T17:35:56             Unknown
2   00:00:59            cpu=2,mem=4G,node=1   00:01:58
node1      0:0              RUNNING
def-spons+           32.extern    2025-11-01T17:35:56             Unknown
2   00:00:59  billing=2,cpu=2,mem=4G,node=1   00:01:58
node1      0:0              RUNNING
```

e. Understanding CPU Cores and Job Efficiency

After our job finishes, we can check efficiency with:

```
[user01@login1 D3_set01]$ seff 32
```

We will see something like:

```
Job ID: 32
Cluster: omics
User/Group: user01/user01
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 2
CPU Utilized: 00:03:01
CPU Efficiency: 91.41% of 00:03:18 core-walltime
Job Wall-clock time: 00:01:39
Memory Utilized: 556.32 MB
Memory Efficiency: 13.58% of 4.00 GB
```

Processing for Multiple Files

If we have multiple FASTQ files to process (say four samples), we can write a job script that loops over them. Example:

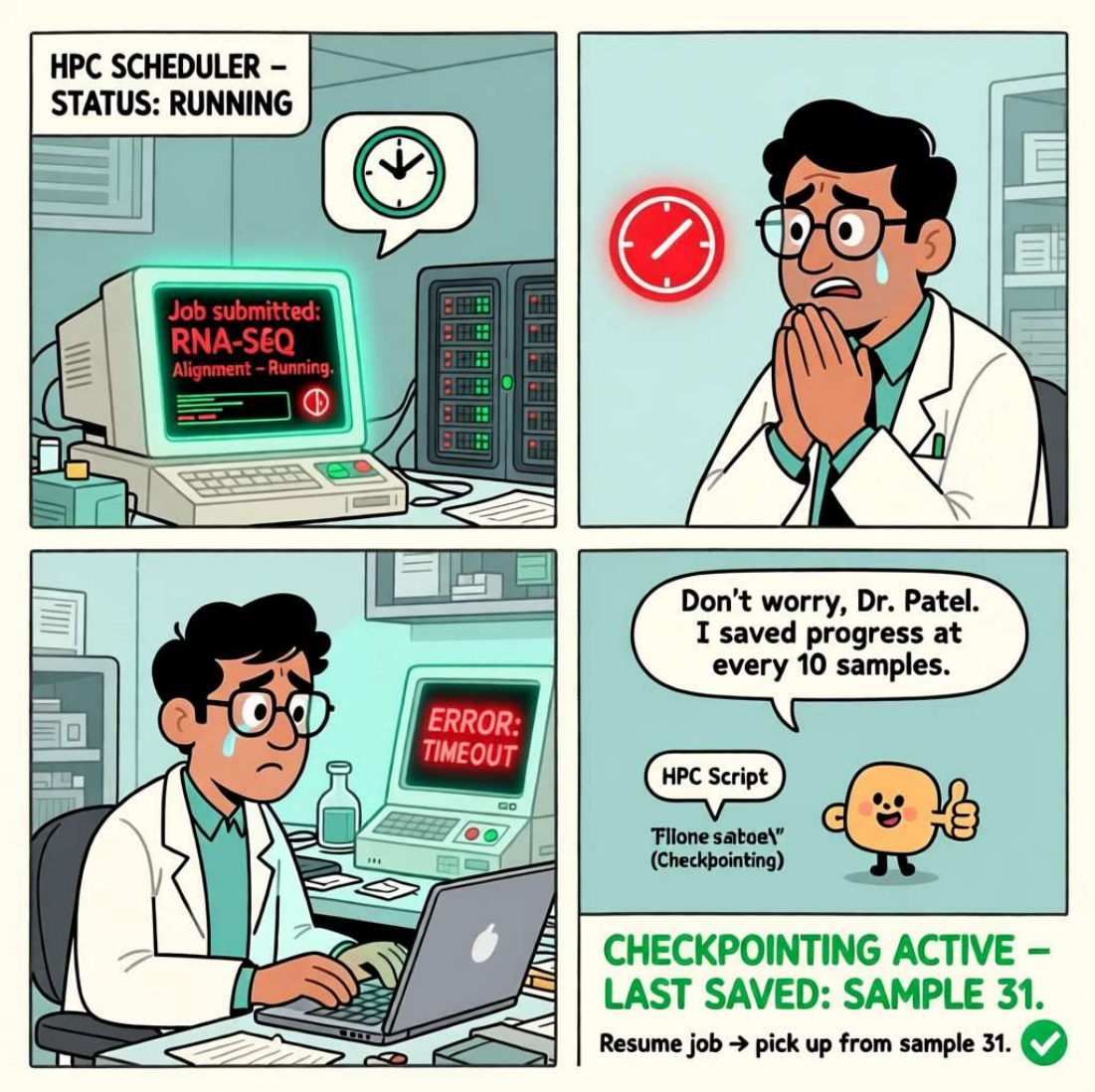
```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --cpus-per-task=2
#SBATCH --mem=8G
#SBATCH --time=01:00:00

module load fastqc

for fq in subsampled_R1.trim.fastq.gz subsampled_R2.trim.fastq.gz
R1_un.fastq.gz R2_un.fastq.gz;
do
  fastqc --threads 2 $fq
done
```

Let us practice the job submission and job monitoring for this script available at `~/projects/def-sponsor00/Day03/multi_fastq.sh`

Implement Checkpointing for Reliable Computation



On HPC systems, jobs often run with time limits (e.g., 7 days or 1 day).
If a long computation stops or is preempted, how to handle it? Checkpointing saves progress periodically so you can resume later.

Checkpointing captures the partial results and writes them to a file (calc_checkpoint.pkl).

If the program stops unexpectedly, it will load the last saved state and continue from there.

For example, here is python script

```
$ cd /home/user01/scratch/omics_workshop/scripts
$ ls -l cp_demo.py
-rwxr-x---. 1 user01 user01 1357 Nov  7 17:32 cp_demo.py
```

Let's copy this python script 'cp_demo.py' to our script directory and see how it works.

```
$ cp  ~/projects/def-sponsor00/Day03/cp_demo.py \
~/scratch/omics_workshop/scripts
```

Next, we are going to write a SLURM script with checkpointing demo so we can test it end-to-end on the cluster.

```
$ cd ~/scratch/omics_workshop/output_data/
$ mkdir checkpointing
$ cd checkpointing
$ cp  ~/projects/def-sponsor00/Day03/cp_job.sh .
```

For example, **cp_job.sh**

```
#!/bin/bash
#SBATCH --job-name=checkpoint_demo
#SBATCH --output=cp_demo_%j.out
#SBATCH --time=00:10:00
#SBATCH --mem=1G
#SBATCH --cpus-per-task=1


module load python/3.11


python /home/user01/scratch/omics_workshop/scripts/cp_demo.py
```

After copying the Python script cp_demo.py and creating the SLURM job script cp_job.sh, we are ready to test the checkpointing

Summary:

- The python script saves progress to `calc_checkpoint.pkl` (or another checkpoint file) periodically.
- If the job is cancelled, interrupted, or hits a time limit, the checkpoint file remains in the directory.
- When we resubmit the job, the script's `load_checkpoint()` function detects the file and resumes from the last saved `x`.

References

1. Data Carpentry: https://datacarpentry.org/
2. European Nucleotide Archive: https://www.ebi.ac.uk/ena/browser/home
3. National Center for Biotechnology Information: https://www.ncbi.nlm.nih.gov/
4. Digital Research Alliance of Canada: https://www.alliancecan.ca/en
5. O'Leary NA, Cox E, Holmes JB, Anderson WR, Falk R, Hem V, Tsuchiya MTN, Schuler GD, Zhang X, Torcivia J, Ketter A, Breen L, Cothran J, Bajwa H, Tinne J, Meric PA, Hlavina W, Schneider VA. Exploring and retrieving sequence and metadata for species across the tree of life with NCBI Datasets. Sci Data. 2024 Jul 5;11(1):732. doi: 10.1038/s41597-024-03571-y. PMID: 38969627; PMCID: PMC11226681.
6. Lyu B, Li J, Niemeyer B, Stanley D, Song Q. Identification and characterization of ecdysis-related neuropeptides in the lone star tick *Amblyomma americanum*. Front Endocrinol (Lausanne). 2023 Aug 25;14:1256618. doi: 10.3389/fendo.2023.1256618. PMID: 37693356; PMCID: PMC10490126.

Acknowledgments

I would like to thank the **Digital Research Alliance of Canada** for providing the infrastructure support that made this workshop possible. I am sincerely grateful to **Jayson To** for his constant support and encouragement, and for ensuring that all arrangements were in place for the smooth facilitation of the workshop. I wish to extend my deep appreciation to **Michael Tang** for his technical expertise and assistance in setting up the workshop cluster, and to the entire **Advanced Research Computing (ARC) team at UBC** for their multifaceted support throughout the process.

End of Day 3 — Take-Home Message

- **Non-Interactive Job Submissions:** Learned to create and submit SLURM batch scripts using `sbatch`
- **Understanding Job Scheduling**
- **Improving Job Efficiency:** Match CPU and memory requests to actual requirements of the computation
- **Job Monitoring** using `seff <jobid>`
- **Job efficiency and resource optimization:** Use of `--cpus-per-task`, `--mem`, and `--time` parameters
- **Processing Multiple Files and Checkpointing**

Overall Learning

- ✓ *Navigate a Linux-based HPC system*
- ✓ *Organize and navigate omics project directories*
- ✓ *Run bioinformatics application in command line environment*
- ✓ *Submit and monitor SLURM jobs*
- ✓ *Optimize computational efficiency for scalable bioinformatics workflows*
- ✓ *Checkpointing*

Thank you for attending the workshop!