HPC Debugging

HPC debugging is the process of identifying why a submitted job did not run as expected. Failures usually stem from one of four areas: SLURM resource settings, environment or module conflicts, missing or incorrect file paths, or application-level errors.

Why it's useful

- Quickly locates the source of job failures
- Helps match memory, CPU, and time requests to actual workload needs
- Prevents repeated failed submissions
- Improves reproducibility and reliability of genomics workflows on RCAC clusters

Common Failure Categories

SLURM-related failures

- Out-of-memory
- Exceeded walltime
- Pending due to priority

Environment failures

- Module conflicts
- Wrong Python/R version
- Broken PATH

Filesystem failures

- Missing paths
- Permission denied
- Quota exceeded

Application failures

- Segfaults
- Runtime error traces
- Tool-specific failures

Debugging Workflow

Locate error logs 1

Examine slurm-<jobid>.out and .err files. Identify actual error messages from the application, not just SLURM noise.

Classify the failure 2

Determine which category it belongs to: SLURM, Environment, Filesystem, Application.

Reproduce interactively 3

Examine slurm-<jobid>.out and .err files. Identify actual error messages from the application, not just SLURM noise.

Validate assumptions 4

Check file paths, modules, environment variables: pwd, ls, which <tool>, module list, env | grep PATH.

Inspect resource usage 5

Use sacct -j <jobid> --format=JobID,State,MaxRSS,Elapsed to confirm whether memory or time were exceeded.

Apply a minimal fix 6

Adjust only what is necessary (memory, path, module load). Re-run the job.

Verify completion 7

Confirm output files exist and that new logs no longer contain the error.

Common symptoms and likely causes

Symptom	Likely cause	What to check next
Job ends immediately	Missing file, wrong path, missing module	Verify absolute paths, run ls, check module list
Job runs for seconds then fails	Application crash, wrong version, corrupted input	Run interactively, inspect .err, test tool directly
Job runs for hours then fails	Out-of-memory or exceeded walltime	sacct MaxRSS, increase --mem or --time
Job stays pending forever	Wrong account/partition/QoS, insufficient resources	Run squeue -u $USER or scontrol show job
“command not found”	Missing module, PATH issues, invalid environment	Load module inside SLURM script, which <tool>
“Permission denied”	File/directory not executable or restricted	ls -l, fix perms (chmod +x) or directory ownership
Segfault or core dump	Tool crash, bad input, incompatible versions	Test small data, check inputs, switch module

Resource-related failures	Purpose
`squeue -u $USER`	See your active jobs and their states (PD/R).
`squeue -j <jobid>`	Inspect a specific job.
`sacct -j <jobid> --format=JobID,State,Elapsed,MaxRSS`	Identify OOM/time-limit failures.
`scontrol show job <jobid>`	Full job description, pending reason, working dir.
`scancel <jobid>`	Cancel misconfigured/stuck jobs.
`sinteractive -A <acct>`	Start an interactive session to debug commands on compute nodes.

Inspecting logs/errors	Purpose
`less slurm-<jobid>.err`	Scroll through error logs.
`tail -n 40 slurm-<jobid>.err`	The actual error is usually at the bottom.
`grep -i "error" slurm-<jobid>.err`	Quickly locate error lines.

Check paths/files/dirs	Purpose
`pwd`	Confirm working directory.
`ls -l`	Check existence + permissions of files.
`du -sh .`	Check directory size (quota/space issues).
`df -h`	Check filesystem fullness.
`myquota`	Check the current quotas of your home and scratch
`file <filename>`	Confirm file type (FASTQ/BAM/FASTA).

General debugging	Purpose
`/usr/bin/time -v <cmd>`	Show memory + runtime on small inputs.
`zcat file.fq.gz |head`	Sanity check FASTQ content.
`samtools quickcheck <bam>`	Check if BAM is corrupted.
`set -x (in scripts)`	Print commands before execution (trace errors).
`bash -x script.sh`	Debug a pipeline script line-by-line.

Check env-related failures	Purpose
`module list`	See what is loaded (debug conflicts).
`module reset`	Start clean before debugging.
`module --force purge`	Remove all loaded modules, including defaults
`module show <name>`	Inspect module environment changes.
`which <tool>`	Confirm which executable is actually being run.
`env | grep PATH`	Debug environment/paths issues.

How to ask for help

What to include in your email

When reporting errors or requesting debugging assistance, include all the following to ensure we can reproduce the issue:

- **Your JobID:** (e.g., 1234567, without this we cannot look up your job)
- **Exact steps to reproduce the problem:** (commands used, scripts, modules loaded — or screenshots)
- **Output and error logs:** (slurm-<jobid>.out, slurm-<jobid>.err, application logs)
- **Relevant files:** (input files, SLURM script, environment files, folder structure)
- **Screenshots:** Only when useful (GUI issues, path errors, environment views)
- **Cluster name:** (Bell, Negishi, Scholar, etc.)

A complete report allows us to solve the issue quickly and accurately.

email: `rcac-help@purdue.edu`

What not to do

- Don’t re-run without checking logs
- Don’t assume HPC/cluster is broken
- Don’t mix conda and modules
- Don’t compute on login nodes
- Don’t skip testing commands interactively
- Don’t load huge module stacks
- Don’t ignore resource usage
- Don’t trust input files blindly
- Don’t use relative paths in scripts
- Don’t overwrite outputs before inspection
- Don’t assume more resources will help run jobs faster
- Don't forget to run programs on small datasets

SLURM exit codes

- 0 → Success
- 1 → General error (usually application-level)
- 126 → Permission denied (non-executable script)
- 127 → Command not found (PATH issue)
- 134 → Abort (e.g., corrupted input or tool crash)
- 137 → Killed by system (OOM or manual kill)
- 139 → Segmentation fault
- 143 → Terminated by SIGTERM (time limit or scancel)
- 255 → Application-specific failure