

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

1 Locate error logs

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

5 Inspect resource usage

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

2 Classify the failure

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

6 Apply a minimal fix

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

3 Reproduce interactively

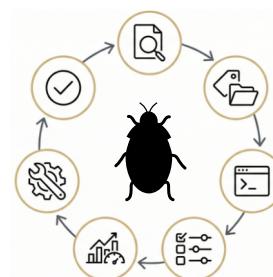
Examine slurm-<jobid>.out and .err files. Identify actual error messages from the application, not just SLURM noise.

7 Verify completion

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

4 Validate assumptions

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



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	Is -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	
<pre>squeue -u \$USER squeue -j <jobid> sacct -j <jobid> --format=JobID,State,Elapsed,MaxRSS scontrol show job <jobid> scancel <jobid> sinteractive -A <acct></pre>	See your active jobs and their states (PD/R). Inspect a specific job. Identify OOM/time-limit failures. Full job description, pending reason, working dir. Cancel misconfigured/stuck jobs. Start an interactive session to debug commands on compute nodes.	
Inspecting logs/errors	Purpose	How to ask for help
<pre>less slurm-<jobid>.err tail -n 40 slurm-<jobid>.err grep -i "error" slurm-<jobid>.err</pre>	Scroll through error logs. The actual error is usually at the bottom. Quickly locate error lines.	What to include in your email <p>When reporting errors or requesting debugging assistance, include all the following to ensure we can reproduce the issue:</p> <ul style="list-style-type: none"> 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.) <p>A complete report allows us to solve the issue quickly and accurately.</p> <p>email: rcac-help@purdue.edu</p>
Check paths/files/dirs	Purpose	
General debugging	Purpose	
Check env-related failures	Purpose	
What not to do	SLURM exit codes	
<pre>/usr/bin/time -v <cmd> zcat file.fq.gz head samtools quickcheck <bam> set -x (in scripts) bash -x script.sh</pre>	Show memory + runtime on small inputs. Sanity check FASTQ content. Check if BAM is corrupted. Print commands before execution (trace errors). Debug a pipeline script line-by-line.	
<pre>module list module reset module --force purge module show <name> which <tool> env grep PATH</pre>	See what is loaded (debug conflicts). Start clean before debugging. Remove all loaded modules, including defaults Inspect module environment changes. Confirm which executable is actually being run. Debug environment/paths issues.	
<ul style="list-style-type: none"> 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 	<ul style="list-style-type: none"> 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 	