CoolMUC Cheat

Leibniz Supercomputing Centre
of the Bavarian Academy of Sciences and Humanities

Released: 2026-03-31 **Best before: 2026-09-30** Check for updates: https://doku.lrz.de/coolmuc-cheat-sheet-1646760668.html

Documentation, Help and Contact

HPC overview and system status:
https://doku.lrz.de/high-performance-computing-10613431.html

CoolMUC-4 starting page:
https://doku.lrz.de/linux-cluster-10745672.html

Contact LRZ Servicedesk: Report problems and requests:
https://servicedesk.lrz.de/en/ql/create/23

HPC Lounge: Chat with members of HPC Support in Zoom meeting:
https://doku.lrz.de/hpc-lounge-813636379.html

Access https://doku.lrz.de/access-and-login-to-the-linux-cluster-10745974.html

- How to get an account (or a project if necessary):
  https://doku.lrz.de/compact-guide-to-first-time-linux-cluster-access-process-11484378.html
- Check https://idmportal2.sim.lrz.de if conformity with export control regulation is valid. Otherwise, login to CoolMUC-4 will not work.
- Login requires two-factor authentication (2FA), login options are:
  (a) password + 2FA
  `ssh -Y MYUSERID@cool.hpc.lrz.de`
  (b) SSH key + usage of **port 2222** + 2FA (key encryption: ECDSA or ED25519)
  `ssh -i ~/.ssh/privatekey -p 2222 -Y MYUSERID@cool.hpc.lrz.de`
- Outgoing ssh connections from CoolMUC-4 on port 22 are blocked. Use ssh or scp on your local computer to connect and copy data.

Modules and LRZ Software Stack https://doku.lrz.de/environment-modules-10745963.html

Modules provide an easy mechanism to manage the user environment via module command, e.g. making application-specific settings available in the current shell:

`module load <package>`	Load a module	`module unload <package>`	Remove loaded module
`module avail`	List all available modules	`module avail <name>`	List modules starting with characters "name"
`module list`	List currently loaded modules	`module show <package>`	Show environment changes when module is loaded
`module switch <package>`	Switch to another package version	`module use -p <path>`	Prepend path to modules search list (`-a` = append)
`module search <keyword>`	Search for modules containing the keyword	`module purge`	Clean-up (reset) module environment
`module test <package>`	Test a module	`module help`	Show usage of module command

Many applications, libraries and tools are created via Spack package manager and regularly released as the LRZ HPC software stack. This software stack is also provided as a module called "stack". The latest official "stack" release is loaded by default on login and compute nodes. List available software stacks via

```
module avail stack
```

You may also build (your own) software and create your own modules via Spack: https://doku.lrz.de/building-software-in-user-space-with-spack-11481697.html

File Systems https://doku.lrz.de/file-systems-and-io-on-linux-cluster-10745972.html

File system	Usage	Access	Size	Backup	Remarks
Home directory	`$HOME`	login and compute nodes	100 GB	YES Snapshots of last 7 days	▪ 100 GB is fixed per user ▪ is part of Data Science Storage (DSS) file system
DSS long-term storage	needs to be defined by user, e.g. `$PROJECT`	login and compute nodes	max. 10 TB	NO	▪ not available per default ▪ requires application; contact your master user
Scratch file system	`$SCRATCH` `$TMPDIR`	login and compute nodes	3100 TB	NO	▪ temporary file system, automatic deletion of old files ▪ data is safe for 14 days
node-local file system	path `"/tmp"`	compute nodes	1.7 TB	NO	▪ temporary file system on compute nodes only ▪ only available during job execution

Command on login nodes to check my DSS information (usage and quotas of HOME and other DSS file systems where I have access): `dssusrinfo all`

Cluster Overview and Job Processing https://doku.lrz.de/job-processing-on-the-linux-cluster-10745970.html

Cluster specifications					Job limits [c]					
Cluster name	Partition name	Nodes in partition	CPU cores per node physical / logical	GPUs per node	Nodes per job min - max	CPU cores (phys.) **min - max** (in sum over all jobs of a user)	Max. runtime (hours)	Max. jobs per user **running / submitted**	**Default memory per logical core / overall memory limit per node**	Node usage [b]
cm4	cm4_std	100 [a]	112 / 224	--	2 - 4	112 - 448	24	2 / 25	488 GiB	exclusive
cm4	cm4_tiny		112 / 224	--	1 - 1	17 - 112	24	4 / 25	2.1 GiB / 488 GiB	shared
inter	cm4_inter	6	112 / 224	--	1 - 1	1 - 112	8	1 / 2	2.1 GiB / 488 GiB	
serial	serial_std	14	80 / 160	--	1 - 1	1 - 16 (96)	24	96 / 200	6.2 GiB / 1000 GiB	
serial	serial_long	4	80 / 160	--	1 - 1	1 - 16 (96)	168	96 / 200	6.2 GiB / 1000 GiB	
inter	teramem_inter	1	96 / 192	--	1 - 1	1 - 96	240	1 / 1	31 GiB / 6000 GiB	

CoolMUC-4 (Intel Sapphire Rapids)
CoolMUC-4 (Intel Icelake)
Teramem (Intel Cooperlake)

(a) overlapping partitions share same nodes | (b) job gets node exclusively or node is shared by different users/jobs
(c) In case of changing job load on serial partitions, the job limits can be adjusted on short notice.
For building software, please note the **architecture of login nodes** which is **Intel Icelake.**

Considerations before setting up a Job

Compute resources are scarce. Please carefully select the resources which fit best to your job requirements!

How many CPU resources does the job really need?
Do you only need a few CPU cores? Then, requesting full nodes on an exclusive partition would be a great waste of compute resources. Use shared partitions instead.

How much memory does the job need?
Memory sizes differ among cluster partitions.
Jobs on shared partitions do not get the full node memory by default. The memory scales with the requested cores. More memory can be requested in the job setup.

How long does the job run?
Try to get a rough estimation if possible, e.g. by running a short test job and extrapolating the runtime for the full job. If jobs usually need a few hours, setting the maximum available runtime on a cluster partition is no good idea.

An optimized job setup improves job processing efficiency and resource utilization, and may reduce waiting time.

Which Resource fits my Needs?

Decision matrix to select the appropriate partition for production jobs (interactive jobs)

How many CPU resources needed? (physical cores)	How much memory does my job need?		
	up to 488 GiB / node	489 - 1000 GiB / node	max. 6000 GiB / node
1 - 16 CPU cores	serial_std, serial_long (cm4_inter)	serial_std, serial_long (cm4_inter, teramem_inter)	teramem_inter (teramem_inter)
> 17 cores, max. 112 cores on 1 node	cm4_tiny (cm4_inter)	teramem_inter	teramem_inter
112 - 448 cores on multiple nodes	cm4_std (cm4_inter)		
more than 448 cores	CoolMUC-4 cannot satisfy this requirement. Please use SuperMUC-NG.		

cm4_std	parallel jobs (MPI) or hybrid jobs (MPI + OpenMP); job bundling of serial jobs
cm4_tiny	single-node jobs, e.g. small parallel MPI or shared-memory jobs; job bundling
serial_*	single-core jobs, up to 24 h on serial_std and with extra-long runtime on serial_long
cm4_inter	interactive jobs, short test runs with reasonable memory requirements (< 0.5 TB)
teramem_inter	large-memory jobs needing more than 1 TB of memory

Basic Slurm Job Script Examples

https://doku.lrz.de/job-processing-on-the-linux-cluster-10745970.html

Purpose: definition of job's resource allocation and its log/error output (lines starting with **#SBATCH**), loading required software modules, setting environment variables and running application(s). During job execution, Slurm job parameters can be accessed via environment variables **SLURM_*** (examples below).

Full example: parallel MPI/shared memory (hybrid) job on 4 nodes using 4 MPI processes (inter-node communication) and 112 OpenMP threads per node

```
#!/bin/bash
#SBATCH -J job_name                          Name of job (SLURM_JOB_NAME)
#SBATCH -D ./                                Change to this working directory before job script is executed (also use --chdir=<path>)
#SBATCH -o ./%x.%j.%N.out                    Standard output/error goes there; set an existing path; set standard error separately via "-e" option
#SBATCH --get-user-env                       Set user environment properly; use (unchanged) login environment variables
#SBATCH --export=NONE                        No export of variables from submitting shell into the job
#SBATCH --clusters=cm4                       Set cluster name (see "Cluster Overview and Job Processing" table)
#SBATCH --partition=cm4_std                  Set partition name (must belong to the abovementioned cluster)
#SBATCH --qos=cm4_std                        Set partition name; needed for Slurm job managment, mandatory on "cm4_std" only
#SBATCH --nodes=4                            Number N of requested nodes in job (SLURM_JOB_NUM_NODES)
#SBATCH --ntasks-per-node=1                  (MPI) tasks T (processes) per node (SLURM_NTASKS_PER_NODE)
#SBATCH --cpus-per-task=112                  Cores C per task, e.g. set thread number for OpenMP (SLURM_CPUS_PER_TASK)
#SBATCH --time=08:00:00                      Maximum job runtime (HH:MM:SS)
module load slurm_setup                      Load mandatory module and all required modules. Modules are not loaded by default. Load all
module load <needed_modules>                 modules needed by the job, e.g. applications, libraries, MPI ("intel-toolkit" if Intel MPI is used)
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK  Use Slurm environment variable for user-defined settings, e.g. number of OpenMP threads
mpiexec ./my_mpi_hybrid_program.exe          Run application; you may also use Slurm's srun command
```

This is the reference for the examples shown below. They focus on **relevant lines** and should **include the lines highlighted in red**. Note: ***T*C*** must not exceed twice the number of available cores per node (hyperthreading considered)! ***N*T*C*** = total amount of logical cores requested by a job. Further useful job settings are:

```
#SBATCH --ntasks=<number>   useful on shared partitions: number of tasks per job (SLURM_NTASKS); replaces "--nodes" and "--ntasks-per-node"
#SBATCH --mem=<size>[unit]  useful on shared partitions: job's maximum memory/node; typical units: G, T for GiB, TiB (SLURM_MEM_PER_NODE)
```

parallel shared-memory job on a full single

```
[...]
#SBATCH --clusters=cm4
#SBATCH --partition=cm4_tiny
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=112
[...]
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
srun ./my_shared_memory_program.exe
```

standard serial job on 1 core (max. 24 hours)

```
[...]
#SBATCH --clusters=serial
#SBATCH --partition=serial_std
#SBATCH --ntasks=1
#SBATCH --time=24:00:00
[...]
srun ./my_serial_program.exe
```

large-memory serial job requesting 2 TiB

```
[...]
#SBATCH --clusters=inter
#SBATCH --partition=teramem_inter
#SBATCH --ntasks=1
#SBATCH --mem=2T
[...]
srun ./my_serial_program.exe
```

Special job configuration: Bundling of many small jobs into a big job via job farming

Workflows increasingly need many jobs, e.g. to process numerous models or data sets.

Submitting thousands of small / short-running jobs is a bad idea! Please avoid that.

Solution: Bundle small jobs into a single batch job.

- small jobs are now called job steps
- most efficient way if all job steps are well runtime-balanced avoiding a waste of valuable CPU resources: "srun ... & ... wait" idiom

Minimalistic example: Run as many serial job steps as available CPU cores at the same time.

- srun manages the job: assign CPU resources, start job steps as background application
- a loop iterates over that, e.g. via index, enabling unique data assignments; wait command waits for all background processes to be finished

Number of job steps per job is limited, e.g. 40000 on "cm4" and 1000 on "serial" partitions. Feel free to extend this script. If in doubt, contact us.

```
#!/bin/bash
#SBATCH -J my_bundle_job
#SBATCH -D ./
#SBATCH -o ./%x.%j.%N.out
#SBATCH --get-user-env
#SBATCH --export=NONE
#SBATCH --clusters=cm4
#SBATCH --partition=cm4_tiny
#SBATCH --ntasks=112
#SBATCH --cpus-per-task=1
#SBATCH --time=08:00:00
module load slurm_setup
M=$((SLURM_MEM_PER_NODE/SLURM_TASKS_PER_NODE - 1))
APP=path_to_my_prog/my_serial_prog
jid=${SLURM_JOB_ID}
for ((k=1; k<=$SLURM_NTASKS; k++)); do
  in=./input/$k
  out=./output/job_$jid/$k
  mkdir -p $out
  srun --exact -n 1 -c 1 --mem=$M $APP $in $out &
  sleep 1
done
wait
```

Further reading: https://doku.lrz.de/general-considerations-to-job-task-farming-771233178.html

Basic Slurm Commands

https://doku.lrz.de/slurm-command-examples-on-the-linux-cluster-1894257007.html

Show overview and status of compute nodes in a cluster
`sinfo -M <cluster>`

List my waiting/running jobs on a particular cluster
`squeue -M <cluster>`

Submit a Slurm job script to start a batch job
`sbatch <job_script>`

Show rough (!) start time estimation for my job(s). Leave out "-j" option to list all jobs.
`squeue -M <cluster> --start -j <job_id>`

Run application inside batch/interactive jobs. Don't use outside jobs!
`srun <application>`

Stop running job or cancel waiting job
`scancel -M <cluster> <job_id>`

Start an interactive Slurm job on interactive CoolMUC-4 partition, e.g. on a single node for a maximum runtime of 2 hours
`salloc -M inter -p cm4_inter -N 1 -t 2:00:00`

Jobs may wait long. Compute time consumption continuously reduces priority of further jobs. But, priority increases with progressing waiting time. Check job priority (ranging from 0 to 1)
`squeue -M <cluster> -O jobid,priority,state,reason -j <job_id>`

Show detailed information of waiting or running jobs
`scontrol show -M <cluster> job <job_id>`

Reduce runtime of waiting job without the need of resubmitting it, e.g. from 24 hours to 10 hours
`scontrol update -M <cluster> jobid=<job_id> TimeLimit=10:00:00`

List details of (finished) jobs, e.g. requested nodes/tasks, start time, runtime, max. memory consumption (KB), job state, reason, exit code, list of allocated nodes
`sacct -M <cluster> -o jobid,nnodes,ntasks,start,elapsed,maxrss,state,reason,exitcode,nodelist`

DOs and DON'Ts

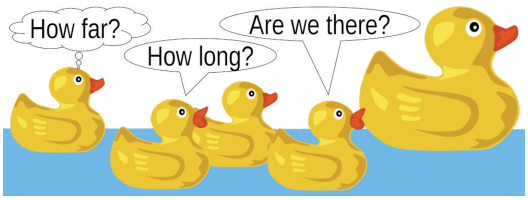
https://doku.lrz.de/policies-on-the-linux-cluster-1307289651.html

Be **careful when using HOME directory for huge data output** in jobs. Its quota can be easily exhausted and jobs fail with I/O (write) errors. Check "File systems" for alternatives.

If jobs use **mail notification**, it is mandatory to include a valid e-mail address. Otherwise, the user ID will be blocked from further job submission.

Some software packages are licensed for use on CoolMUC-4 with usually limited amount of licenses. Since **license availability** cannot be checked beforehand, LRZ cannot guarantee that jobs requesting a license will run successfully.

It may happen that **sbatch refuses to submit a job**, throwing an error. Typical reason is a misconfigured job script with invalid partition names or with resource settings violating the limits. Check the "Job Processing" table.
It is also possible that your user ID has been banned; contact us.



Avoid high-frequency polling of Slurm. Doing job monitoring by requesting job information via commands like `squeue` or `sacct` in very short time intervals (e.g. a few seconds) is a bad idea! Careless use of **watch** command will do such an illegal action.
That causes **trouble** on Slurm server, reducing its responsiveness and will affect all users. Doing requests **every 10 minutes is better**. Permanent high-frequency queries may lead to a ban of the user ID.

Please do not misuse login nodes for long-running, memory-hogging or parallel programs. Login nodes are usually used to prepare jobs, compile codes or transfer data.

Do not submit a large number of short-running jobs (< 2 minutes). This is a waste of compute resources and **disrupts** both notification system (if mail is enabled in the job) and Slurm. Users might be banned. Use job bundling instead.

In job scripts **do not set invalid standard output/error paths**, e.g. paths which do not exist. The job will **fail** right after startup without producing any output.