UCRL-WEB-201386

SLURM Reference Manual

Table of Contents

Preface 4
Introduction 5
SLURM Goals and Roles 6
SLURM Goals 6
SLURM Roles 8
SLURM and Operating Systems 9
SLURM Features 10
SLURM Components 10
SLURMCTLD 10
SLURMD 12
Portability (Plugins) 13
User Impact 14
Scheduler Types 15
SLURM Operation 16
SLURM Utilities 16
SRUN (Submit Jobs) 17
SRUN Roles and Modes 17
Comparison with POE 19
SRUN Run-Mode Options 20
SRUN Resource-Allocation Options 22
SRUN Control Options 24
Node Management 24
Working Features 26
Resource Control 29
Help and Message Options 30
Prolog and Epilog Options 31
Debug (Root) Options 32
SRUN I/O Options 33
I/O Commands 33
I/O Redirection Alternatives 34
SRUN Constraint Options 36
General Constraints 36
Affinity or NUMA Constraints 38
Environment Variables 40
Multiple Program Usage 44
SQUEUE (List Jobs) 46
SQUEUE Execute Line 46
SQUEUE Options 47
SQUEUE Examples 50
SQUEUE Job State Codes 52
SINFO (List Nodes) 53
SINFO Execute Line 53
SINFO Options 54

SINFO Output Fields 58
SINFO Node States 60
SINFO Examples 61
SMAP (Show Job Geometry) 63
SCONTROL (Manage Configurations) 65
Disclaimer 67
Keyword Index 68
Alphabetical List of Keywords 70
Date and Revisions 72

Preface

Scope: This manual explains the design goals and unique roles of LC's locally developed Simple Linux Utility for Resource Management (SLURM), intended as a customized replacement for RMS or NQS in allocating compute resources (mostly nodes) to queued jobs on machines running the CHAOS operating system. Sections describe the features of both control daemon SLURMCTLD and local daemon SLURMD, as well as SLURM's adaptability by means of plugin modules. The five SLURM user utilities for querying and controlling jobs managed by SLURM are also introduced. The features and options of SRUN, the tool used to launch both parallel interactive and batch jobs under SLURM management, receive especially detailed treatment. Another section explains how to monitor SRUN-submitted jobs by using SQUEUE, as well as how to customize SQUEUE's reports using its own format specification language. Likewise, a section tells how to check the current status of nodes (individually or by partition) using SINFO. The general-user features of SCONTROL are also included.

Availability: SLURM is part of the CHAOS project, and is available on selected large LC clusters that run the CHAOS version of Linux.

Consultant: For help contact the LC customer service and support hotline at 925-422-4531 (open e-mail: lc-hotline@llnl.gov, SCF e-mail: lc-hotline@pop.llnl.gov).

Printing: The print file for this document can be found at

```
OCF: http://www.llnl.gov/LCdocs/slurm/slurm.pdf
SCF: https://lc.llnl.gov/LCdocs/slurm/slurm_scf.pdf
```

Introduction

SLURM is LC's locally developed C-language Simple Linux Utility for Resource Management. SLURM is a job- and compute-resource manager that can run reliably and efficiently on Linux (CHAOS) clusters as large as several thousand nodes. Its features suit it to large-scale, high-performance computing environments, and its design avoids known weaknesses (such as inflexibility or fault intolerance) in available commercial resource management products for supercomputers.

This manual summarizes the specific service goals that SLURM was developed to meet, and explains the roles that it plays (relative to the Livermore Computing Resource Management (LCRM/DPCS) system, for example) on LC production machines. Key to SLURM's operation are two software daemons: one (SLURMCTLD) controls the job queue and resource allocations, while the other (SLURMD) shepherds executing jobs on each compute node. Sections below explain the features and subsystems of each SLURM daemon. Additional sections tell how use of "plugin modules" make SLURM easily adaptable to many hardware situations, and introduce the five utility programs that give SLURM its direct user interface.

SRUN is the SLURM utility central to launching, assigning resources to, and guiding the execution of parallel jobs managed by SLURM, both interactively and through batch queues. Hence, the five ways to use SRUN (its "modes"), SRUN's complex I/O redirection support, and the often-elaborate interaction among the many SRUN options receive careful attention in several subsections devoted to that tool. SRUN also interacts with a set of special SLURM environment variables (like those used for job management by IBM's POE), explained in another subsection. Detailed and customizable monitoring of SRUN-submitted *jobs* is provided by SQUEUE, whose options we also compare and illustrate with annotated output cases. Likewise, to plan SRUN use you can monitor SLURM-managed *nodes* by executing or customizing a separate SLURM tool called SINFO, with its own section below. Checkpoint support using SCONTROL is introduced as well.

SLURM development is part of LC's larger CHAOS open-source operating system project, as explained in the separate CHAOS Reference Manual. (URL: http://www.llnl.gov/LCdocs/chaos) For a summary of known, significant differences between LC's Linux machines and those running AIX or Tru64 UNIX, see the Linux Differences (URL: http://www.llnl.gov/LCdocs/linux) guide. And for general advice on managing (batch) jobs on LC production machines, consult the examples and comparisons in the basic EZJOBCONTROL (URL: http://www.llnl.gov/LCdocs/ezjobcontrol) guide.

SLURM Goals and Roles

SLURM Goals

SLURM was developed specifically to meet locally important criteria for a helpful, efficient way to manage compute resources on large (Linux/CHAOS) clusters. The *primary* threefold purpose of a cluster resource manager (such as LoadLeveler on LC's IBM ASC machines or the Resource Management System (RMS) from Quadrics) is to:

- Allocate nodes--
  give users access (perhaps even exclusive access) to compute nodes for some specified time range so their job(s) can run.
- Control job execution--
  provide the underlying mechanisms to start, run, cancel, and monitor the state of parallel (or serial) jobs on the nodes allocated.
- Manage contention--
  reconcile competing requests for limited resources, usually by managing a queue of pending jobs.

At LC, an adequate cluster resource manager needs to meet two *general* requirements:

- Scalable--
  It must operate well on clusters with as many as several thousand nodes, including cases where the nodes are heterogeneous (with different hardware or configuration features).
- Portable--
  It must ultimately support jobs on clusters that have different operating systems or versions, different architectures, different vendors, and different interconnect networks. Linux/CHAOS is, of course, the intended first home for this software, however.

Any LC resource manager must also meet two *additional*, locally important, requirements:

- Compatible with LCRM (DPCS)--
  Since a resource manager is not a complex scheduler nor a complete batch system with across-cluster accounting and reporting features, it must support and work well within such a larger, more comprehensive job-control framework. At LC, the Livermore Computing Resource Management system (formerly called DPCS (URL: http://www.llnl.gov/LCdocs/dpcs)) provides that framework (see also the next section (page 8)).
- Compatible with QsNet--
  Since LC's Linux Project has already refined QsNet as its preferred high-speed interconnect for Linux/CHAOS clusters, an adequate resource manager must also allocate Quadrics QsNet resources along with compute nodes. But conversely, interconnect *independence* and the ability to easily support other brands of interconnect (such as Myrinet) is important too. Such independence allows great flexibility in pursuing new hardware configurations in future clusters.

Finally, to fit well into LC's emerging CHAOS environment, a resource manager should ideally have these three very beneficial *extra properties* as well:

- Fault Tolerant--
  Innovative scientific computing systems are often much less stable than routine business clusters, so a good local resource manager should recover well from many kinds of system failures (without terminating its workload), including failure of the node where its own control functions execute.
- Open Source--
  The software (source code) should be freely sharable under the GNU General Public License, as with other nonproprietary CHAOS components.
- Modular--
  An approach that clearly *separates* high-level job-scheduling functions from low-level cluster-administration functions allows for easier changes in scheduling policy without having to sacrifice working, familiar cluster-resource tools or features.

No commercial (or existing open source) resource manager meets all of these needs. So since 2001 Livermore Computing, in collaboration with Linux NetworX and Brigham Young University, has developed and refined the "Simple Linux Utility for Resource Management" (SLURM).

SLURM Roles

SLURM fills a crucial but mostly hidden role in running large parallel programs on large clusters.

Most users who run batch jobs at LC use job-control utilities (such as PSUB or PALTER) that talk to the Livermore Computing Resource Management system (LCRM, formerly called DPCS), LC's locally designed *metabatch* system. LCRM:

- Provides a common user interface for batch-job submittal across all LC machines and clusters.
- Monitors resource use across machines and clusters.
- Implements bank-based fair-share scheduling policy, again, across all LC production machines.

To carry out its scheduling decisions, LCRM relies on the *native resource manager* on each machine or cluster where it assigns batch jobs to run. The basic duties of such a native resource manager are to:

- Get and share information on resource (chiefly node) availability.
- Allocate compute resources (chiefly, nodes or processors).
- Shepard jobs as their tasks execute.

On IBM AIX machines, LoadLeveler traditionally served as the native resource manager. On LC's nonAIX machines, LCRM has relied on one of three other native resource managers to provide low-level job control:

- RMS (Resource Management System), used on "capability" clusters (devoted to one or two users at a time).
- TBS (Trivial Batch System, an LC-developed replacement for the formerly widespread Network Queueing System or NQS).
- SLURM (introduced here for managing Linux clusters and still evolving to meet specific LC needs).

The key differences among these alternatives appear in this table:

	RMS	**TBS**	**SLURM**
Proprietary?	Yes	No, open source	No, open source
Used on:	Machines with QsNet interconnect	Interconnect independent	Interconnect independent
Suited for:	Capability clusters	Capacity clusters	Either with CHAOS
Node allocation:	Whole nodes allocated to jobs	Multiple jobs per node	Either possible

SLURM and Operating Systems

SLURM was originally used as a resource manager for Linux (specifically for CHAOS) systems. But starting in 2006, LC began gradually replacing IBM's native LoadLeveler with SLURM on its AIX systems as well. The AIX-SLURM combination behaves (and has been configured by LC system administrators to behave) slightly differently than the CHAOS-SLURM combination, however. This means that to answer a job-control question increasingly requires knowing *both* the relevant resource manager and the current operating system.

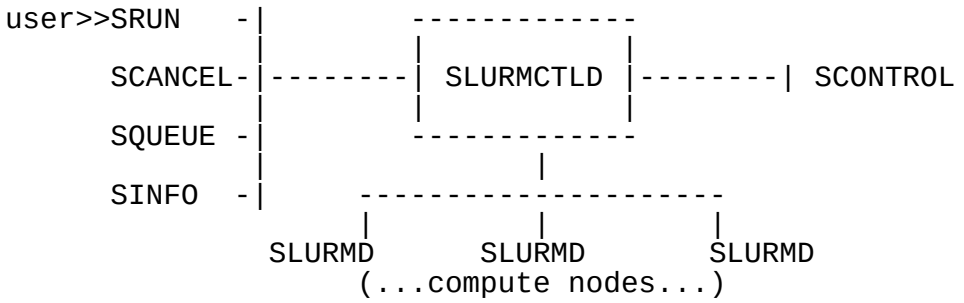
This table summarizes the known tool contrasts among the different resource-manager/operating-system combinations now possible on LC machines:

	Linux (CHAOS) + SLURM	**AIX + LoadLeveler**	**AIX + SLURM**
Start a parallel job:	SRUN	POE	POE
Specify a node pool:	SRUN -w or --nodelist	POE -rmpool *pnumber* (not name)	POE -rmpool *pname* (not number)
Get job information:	SQUEUE	LLQ	SQUEUE
Get node information:	SINFO	LLSTATUS	SINFO
Cancel a started job:	SCANCEL	LLCANCEL	SCANCEL
PSUB -g option applies:	NO	YES	YES

SLURM Features

SLURM Components

SLURM consists of two kinds of daemon (discussed here) and five command-line user utilities (next section (page 16)), whose relationships appear in this simplified architecture diagram:



SLURMCTLD

SLURM's central control daemon is called SLURMCTLD. Unlike the Portable Batch System daemon, SLURMCTLD is *multi*-threaded, so some threads can handle problems without delaying service to continuing normal jobs that also need attention. SLURMCTLD runs on a single management node (with a fail-over spare copy elsewhere for safety), reads the SLURM configuration file, and maintains state information on:

- nodes (the basic compute resource),
- partitions (logically disjoint sets of nodes),
- jobs (or resource allocations to run jobs for a time period), and
- job steps (parallel tasks within a job). Job steps are *not* supported on BlueGene/L.

The SLURMCTLD daemon in turn consists of three software subsystems, each with a specific role:

Node Manager

> monitors the state and configuration of each node in the cluster. It receives state-change messages from each compute node's SLURMD daemon asynschonously, and it also actively polls those daemons periodically for status reports.

Partition Manager

> groups nodes into disjoint sets (partitions) and assigns job limits and access controls to each partition. The partition manager also allocates nodes to jobs (at the request of the Job Manager, below) based on job and partition properties. SCONTROL is the (privileged) user utility that can alter partition properties.

Job Manager — accepts job requests (from SRUN (page 17) or a metabatch system like LCRM), places them in a priority-ordered queue, and reviews that queue periodically or when any state change might allow a new job to start. Qualifying jobs are allocated resources and that information transfers to (SLURMD on) the relevant nodes so the job can execute. When all nodes assigned to a job report that their work is done, the Job Manager revises its records and reviews the pending-job queue again.

SLURMD

The SLURMD daemon runs on every compute node of every cluster that SLURM manages and it performs the lowest level work of resource management. Like SLURMCTLD (above), SLURMD is multi-threaded for efficiency, but unlike SLURMCTLD it runs with root privilege (so it can initiate jobs on behalf of other users).

SLURMD carries out five key tasks and has five corresponding subsystems:

Machine Status

: responds to SLURMCTLD requests for machine state information and sends asynchronous reports of state changes to help with queue control.

Job Status
: responds to SLURMCTLD requests for job state information and sends asynchronous reports of state changes to help with queue control.

Remote Execution

: starts, monitors, and cleans up after a set of processes (usually shared by a parallel job), as decided by SLURMCTLD (or by direct user intervention). This often involves many process-limit, environment-variable, working-directory, and user-id changes.

Stream Copy Service

: handles all STDERR, STDIN, and STDOUT for remote tasks. This may involve redirection, and it always involves locally buffering job output to avoid blocking local tasks.

Job Control
: propagates signals and job-termination requests to any SLURM-managed processes (often interacting with the Remote Execution subsystem).

Portability (Plugins)

SLURM achieves portability (hardware independence) by using a general *plugin* mechanism. SLURM's configuration file tells it which plugin modules to accept.

A SLURM plugin is a dynamically linked code object that the SLURM libraries load explicitly at run time. Each plugin provides a customized implementation of a well-defined API connected to some specific tasks.

By means of this plugin approach, SLURM can easily change its:

- interconnect support (default is Quadrics QsNet).
- security techniques (default is to use crypto techniques to authenticate services to users and to each other).
- metabatch scheduler (default is LC's LCRM/DPCS, with a "Grid" resource broker as an easy alternative).
- low-level job scheduler (page 15) for locally prioritizing and initiating SRUN-managed jobs.
- between-node communication "layers" (default is Berkeley sockets).

User Impact

The primary SLURM job-control tool is SRUN, (page 17) which fills the general role of PRUN (on former Compaq machines) or POE (on IBM computers). Your choice of run mode ("batch" or interactive) and your allocation of resources with SRUN strongly affect your job's behavior on machines where SLURM manages parallel jobs. SLURM works collaboratively with POE on AIX machines where SLURM has replaced IBM's LoadLeveler. See the SCONTROL section (page 65) below, for example, for an introduction to how this collaboration supports job checkpointing.

To monitor the status of SRUN-submitted *jobs*, use the SLURM utility called SQUEUE (page 46). To monitor the status of SLURM-managed compute *nodes*, use the complementary tool called SINFO (page 53). Both SQUEUE and SINFO have explanatory sections later in this manual, with usage examples.

On BlueGene/L *only*, SLURM provides an additional user tool called SMAP (page 63) to reveal topographically how nodes are allocated among current jobs or partitions (because job geometry is unusually important on BG/L).

Scheduler Types

The system administrator for each machine can configure SLURM to invoke any of several alternative local job schedulers. You can discover which scheduler SLURM currently invokes on any machine by executing

```
scontrol show config | grep SchedulerType
```

where the returned string will have one of these values:

builtin — (default) is a first-in-first-out scheduler. SLURM executes jobs strictly in the order that they were submitted (for each resource partition). Even if resources become available to start a specific job, SLURM will wait until no previously submitted job is pending (which sometimes confuses impatient job submitters).

backfill — modifies strict FIFO scheduling to take advantage of resource islands that may appear as earlier jobs complete. SLURM will start later-submitted jobs out of order if resources become available *and* if doing so does not delay the expected execution time of any earlier-submitted job. You increase your job's chances of benefiting from such backfill scheduling if you:
(1) specify reasonable time limits (the default is the same time limit for all jobs in your partition, perhaps much too large), and
(2) avoid requiring or excluding specific nodes by name.
BlueGene/L Note: backfill scheduling never occurs on BlueGene/L because that machine's unusual and complex architecture thwarts reliably finding left over resource islands to fill.

wiki — uses the Maui Scheduler, with a sophisticated set of internal scheduling algorithms. This choice can be configured in many ways to optimize job throughput. Details are posted on a support web site at http://supercluster.org/maui (URL: http://supercluster.org/maui).

SLURM Operation

SLURM Utilities

SLURM's five command-line utilities provide its direct interface for users (while LCRM utilities, as explained in EZJOBCONTROL (URL: http://www.llnl.gov/LCdocs/ezjob), provide an indirect interface). These utilities are:

SRUN	submits jobs to run under SLURM management. SRUN can (A) submit a batch job and then terminate, or (B) submit an interactive job and then persist to shepherd the job as it runs, or (C) allocate resources to a shell and then spawn that shell for use in running subordinate jobs. SLURM associates every set of parallel tasks ("job steps") with the SRUN instance that initiated that set, and SRUN gives you elaborate control over node choice and I/O redirection for your parallel job. (Job steps are not supported on BlueGene/L.)
SQUEUE	displays (by default) the queue of running and waiting jobs (or "job steps"), including the JobId (used for SCANCEL), and the nodes assigned to each running job. But you can customize SQUEUE reports to cover any of 24 different job properties, sorted by the properties most important to you.
SINFO	displays a summary of status information on SLURM-managed partitions and nodes (*not* jobs). Customizable SINFO reports can cover the node count, state, and name list for a whole partition, or the CPUs, memory, disk space, or current status for individual specified nodes.
SCANCEL	cancels a running or waiting job, or sends a specified signal to all processes on all nodes associated with a job (only job owners or their administrators can cancel their jobs).
SCONTROL	(privileged users only) manages available nodes (for example, by "draining" jobs from a node or partition to prepare it for servicing) and assigns properties to node partitions.
SMAP	(on BlueGene/L *only*) displays a character-based chart or plot showing how nodes are allocated geometrically among current jobs and BG/L partitions (a job-planning tool).

SRUN (Submit Jobs)

SRUN Roles and Modes

SRUN executes tasks ("jobs") in parallel on multiple compute nodes at the same time (on machines where SLURM manages the resources). SRUN options let you both:

- Specify the parallel environment for your job(s), such as the number of nodes used, node partition, distribution of processes among nodes, and total time, and also
- Control the behavior of your parallel job as it runs, such as by redirecting or labeling its output, sending it signals, or specifying its reporting verbosity.

Because it performs several different roles, SRUN can be used in five distinct ways or "modes":

- SIMPLE.
  The simplest way to use SRUN is to distribute execution of a serial program (such as a UNIX utility) across a specified number or range of compute nodes. For example,

  ```
  srun -N 8 cp ~/data1 /var/tmp/data1
  ```

  copies (CP) file data1 from your common home directory into local disk space on each of eight compute nodes. This is very like running simple programs in parallel under AIX by using IBM's POE command (except that SRUN lets you set relevant environment variables on its own execute line, unlike POE). In simple mode, SRUN submits your job to the local SLURM job controller, initiates all processes on the specified nodes, and blocks until needed resources are free to run the job if necessary. Many control options can change the details of this general pattern.
- BATCH (WITHOUT LCRM).
  SRUN can also directly submit complex scripts to the (Trivial Batch System, TBS) job queue(s) managed by SLURM for later execution when needed resources become available and when no higher priority jobs are pending. For example,

  ```
  srun -N 16 -b myscript.sh
  ```

  uses SRUN's -b option to place myscript.sh into the TBS queue to later run on 16 nodes. Scripts in turn normally contain either MPI programs or other, *simple* invocations of SRUN itself (as shown above). SRUN's -b option thus supports basic, local batch service even on machines where LC's metabatch system LCRM has not yet been installed (see below). On BlueGene/L only, scripts must invoke MPIRUN instead of simple SRUN to start tasks.
- ALLOCATE.
  To combine the job complexity of scripts with the immediacy of interactive execution, you can use SRUN's "allocate" mode. For example,

  ```
  srun -A -N 4 myscript.sh
  ```

  uses SRUN's (uppercase) -A option to allocate specified resources (here, four nodes), spawn a subshell with access to those resources, and then run multiple jobs using *simple* SRUN commands within the specified script (here, myscript.sh) that the subshell immediately starts to execute. This is very like allocating resources by setting AIX environment variables at the beginning of a script, and then using them for scripted tasks. No job queues are involved.

- ATTACH.
  You can monitor or intervene in an already running SRUN job, either batch (started with -b) or interactive ("allocated," started with -A), by executing SRUN again and "attaching" (-a, lowercase) to that job. For example,

  ```
  srun -a 6543 -j
  ```

  forwards the standard output and error messages from the running job with SLURM ID 6543 to the attaching SRUN to reveal the job's current status, and (with -j, lowercase) also "joins" the job so that you can send it signals as if this SRUN had initiated the job. Omit -j for read-only attachments. Because you are attaching to a running job whose resources have already been allocated, SRUN's resource-allocation options (such as -N) are incompatible with -a.
- BATCH (WITH LCRM).
  On machines where LC's metabatch job-control and accounting system LCRM/DPCS is installed, you can submit (with the LCRM utility PSUB) a script to LCRM that contains (simple) SRUN commands within it to execute parallel jobs later, after LCRM applies the usual fair-share scheduling process to your job and its competitors. Here LCRM takes the place of SRUN's -b option for indirect, across-machine job-queue management.

SRUN SIGNAL HANDLING.

Signals sent to SRUN are automatically forwarded to the tasks that SRUN controls, with a few special cases. SRUN handles the sequence CTRL-C differently depending on how many it receives in one second:

```
CTRL-Cs within one second
-------------------------
First    reports the state of all tasks
         associated with SRUN.
Second   sends SIGINT signal to all
         associated SRUN tasks.
Third    terminates the job at once,
         without waiting for remote tasks
         to exit.
```

MPI SUPPORT.

On computer clusters with a Quadrics interconnect among the nodes (such as Lilac on SCF, or Thunder and ALC on OCF) SRUN directly supports the Quadrics version of MPI without modification. Applications built using the Quadrics MPI library communicate over the Quadrics interconnect without any special SRUN options.

You may also use MPICH on any computer where it is available. MPIRUN will, however, need information on its command line identifying the resources to use, namely

```
-np SLURM_NPROCS -machinefile filename
```

where SLURM_NPROCS is the environment variable that contains the (-n) number of processors to use and *filename* lists the names of the nodes on which to execute (the captured output from /bin/hostname run across those nodes with simple SRUN). Sometimes the MPICH vendor configures these options automatically. See also SRUN's --mpi "working features" option (page 26).

Comparison with POE

SRUN and AIX's POE (Parallel Operating Environment) both use UNIX environment variables to manage the resources for each parallel job that they run. Of course, variables with comparable roles have different names under each system (and both systems have many other environment variables for other purposes too).

Two *differences* in detail between environment-variable use by SRUN and POE are noteworthy:

- SRUN assigns values to its resource-management variables by means of its own interactive options, one option for each environment variable (plus extra control options, such as -j). Instead, POE uses the usual SETENV or EXPORT utilities to assign values to its environment variables.
- POE's LoadLeveler ignores many environment variables when it run batch jobs under AIX on LC machines. SLURM does not ignore the corresponding environment variables when set by SRUN, even for batch runs.

This chart lists the SLURM (SRUN-set or inferred) resource-management environment variables for which direct POE counterparts exist. For an explanatory inventory of all SLURM environment variables, see the separate section (page 40) below.

Environment Variable Role	SRUN Option To Set	SLURM Variable Name	POE Variable Name
Total processes to run	-n	SLURM_NPROCS	MP_PROCS(*)
Total nodes allocated	-N	SLURM_NNODES	MP_NODES(*)
Node list for this job	(inferred)	SLURM_NODELIST	MP_SAVEHOSTFILE(*)
MP ID of current process	(inferred)	SLURM_PROCID	MP_CHILD
Output mode choice	-o (lc)	SLURM_STDOUTMODE	MP_STDOUTMODE
Partition for this job	-p	SLURM_PARTITION	MP_RMPOOL(*)
Debug message level	-d	SLURMD_DEBUG	MP_INFOLEVEL
Output message level	-v (lc)	SLURM_DEBUG	MP_INFOLEVEL
Output label choice	-l	SLURM_LABELIO	MP_LABELIO

(*)Ignored by LoadLeveler for batch jobs on AIX machines at LC.

SRUN Run-Mode Options

For a strategic comparison (with examples) of the five different ways to use SRUN, see "SRUN Roles and Modes," above. (page 17) This section explains the *mutually exclusive* SRUN options that enable its different run modes. Each option has a one-character (UNIX) and a longer (Linux) alternative syntax.

-b (--batch) runs a script (whose name appears at the end of the SRUN execute line, *not* as an argument to -b) in batch mode. You cannot use -b with -A or -a.

RESULT.
SRUN copies the script, submits the request to run (with your specified resource allocation) to the local SLURM-managed job queue, and ends. When resources become available and no higher priority job is pending, SLURM runs the script on the first node allocated to the job, with STDIN redirected from /dev/null and STDOUT and STDERR redirected to a file called *jobname*.out in the current working directory (unless you request a different name or a more elaborate set of output files by using -J or -o). In other words, -b executes the script through a queue, unhooked from terminal interaction, but not under LCRM control.

SCRIPT REQUIREMENTS.
(1) You must use the script's absolute pathname or a pathname relative to the current working directory (SRUN ignores your search path).
(2) SRUN interprets the script using your default shell unless the file begins with the character pair #! followed by the absolute pathname of a valid shell.
(3) The script must contain MPI commands or other (simple) SRUN commands to initiate parallel tasks. On BlueGene/L *only*, the script must invoke MPIRUN rather than simple SRUN to execute parallel tasks (this may affect script portability).
(4) Script lines prefixed with #SLURM insert SRUN options into the script (e.g., #SLURM --propagate); command line options override those within the script.

-A (uppercase, --allocate)

allocates compute resources (as specified by other SRUN options) and starts ("spawns") a subshell that has access to those allocated resources. No remote tasks are started. You cannot use -A with -b or -a. If you combine -A with the special-purpose option --no-shell, then SRUN allocates the resources but exits instead of spawning a shell.

SCRIPTED USE.
If you specify a script at the end of SRUN's execute line (*not* as an argument to -A), the spawned shell executes that script using the allocated resouces (interactively, without a queue). See the -b option for script requirements.

UNSCRIPTED USE.
If you specify no script, you can then execute other instances of SRUN interactively, within the spawned subshell, to run multiple parallel jobs on the resources that you allocated to the subshell. The resources (nodes, etc.) will only be freed for other jobs when you terminate the subshell.

-a *jobid* (lowercase, --attach=*jobid*)

attaches (or reattaches) your current SRUN session to the already running job whose SLURM ID is *jobid*. The job to which you attach must have its resources managed by SLURM, but it can be either interactive ("allocated," started with -A) or batch (started with -b). This option allows you to monitor or intervene in previously started SRUN jobs. You cannot use -a with -b or -A. Because the running job to which you attach already has its resources specified, you cannot use -a with -n, -N, or -c. You can only attach to jobs for which you are the authorized owner.

READ-ONLY.
By default, -a attaches to the designated job read-only. STDOUT and STDERR are copied to the attaching SRUN, just as if the current SRUN session had started the job. However, signals are *not* forwarded to the remote processes (and a single CTRL-C will detach the read-only SRUN from the job).

READ-WRITE.
If you use -j (--join) or -s (--steal) along with -a, your SRUN session "joins" the running job and can also forward signals to it as well as receive STDOUT and STDERR from it. If you join a SLURM batch (-b) job, you can send signals to its batch script. Join (-j) does not forward STDIN, but steal (-s, which closes other open sessions with the job) does forward STDIN as well as signals.

-j (lowercase, --join)

joins a running SLURM job (always used only with -a, --attach, to specify the *jobid*). This not only duplicates STDOUT and STDERR to the attaching SRUN session, but it also forwards signals to the job's script or processes as well.

-s (--steal) steals all connections to a running SLURM job (always used only with -a, --attach, to specify the *jobid*). STEAL closes any open sessions with the specified job, then copies STDOUT and STDERR to the attaching SRUN session, and it also forwards *both* signals and STDIN to the job's script or processes.

SRUN Resource-Allocation Options

These SRUN options (used alone or in combination) assign compute resources to your parallel SLURM-managed job. Each option has a one-character (UNIX) and a longer (Linux) alternative syntax. See also SRUN's other options that can affect node management for your job, especially the control (page 24) options and constraint (page 36) options, in separate subsections below.

-n *procs* (lowercase, --nprocs=*procs*)

requests that SRUN execute *procs* processes. To control how these processes are distributed among nodes and CPUs, combine -n with -c or -N as explained below (default is one process per node).

-N *n* (uppercase, --nodes=*n*)

allocates at least *n* nodes to this job, where *n* may be either
(1) a specific node count (such as -N 16), or
(2) a hyphen-separated range from a minimum to a maximum node count (such as -N 2-4).
If the nodes are partitioned, each partition's node limits supersede those specified by -N (jobs that request more nodes than the partition allows never leave the PENDING state). To change partitions, use SRUN's -p option (below). Combinations of -n and -N control how job processes are distributed among nodes according to the SRUN policies listed here:

-n/-N COMBINATIONS.
SRUN infers your intended number of processes per node if you specify *both* the number of processes and the number of nodes for your job. Thus -n 16 -N 8 normally results in running 2 processes/node (but see the next policy for exceptions).

MINIMUM INTERPRETATION.
SRUN interprets all node requests as minimum node requests (so -N 16 means "at least 16 nodes"). If some nodes lack enough CPUs to cover the process count specified by -n, SRUN will automatically allocate more nodes (than mentioned with -N) to meet the need (for example, if not all nodes have 2 working CPUs, then -n 32 -N 16 together will allocate *more than* 16 nodes so that all processes are supported). The actual number of nodes assigned (not the number reqested) is stored in environment variable SLURM_NNODES.

CPU OVERCOMMITMENT.
By default, SRUN never allocates more than one process per CPU. If you intend to assign multiple processes per CPU, you must invoke SRUN's -O (uppercase oh) option along with -n and -N (thus -n 16 -N 4 -O together allow 2 processes/CPU on the 4 allocated 2-CPU nodes).

INCONSISTENT ALLOCATION.
SRUN rejects as errors inconsistent -n/-N combinations. For example, -n 15 -N 16 requests the impossible assignment of 15 processes to 16 nodes.

-c *cpt* (lowercase, --cpus-per-task=*cpt*)

assigns *cpt* CPUs per process for this job (default is one CPU/process). This option supports multithreaded programs that require more than a single CPU/process for best performance.

-n/-c COMBINATIONS.
For multithreaded programs where the density of CPUs is more important than a specific node count, use both -n and -c on the same SRUN execute line (rather than -N). Thus -n 16 -c 2 results in whatever node allocation is needed to yield the requested 2 CPUs/process. This is the reverse of CPU overcommitment (see -N and -O, above).

--multi-prog assigns different executable programs with (perhaps) different arguments to each task. When you use this option, SRUN's own argument is not a parallel program but instead a 3-column configuration file that specifies your matrix of tasks, programs, and their arguments. See the Multiple Program Usage (page 44) section below for details and an example of using this option.

-p *part* (lowercase, --partition=*part*)

requests nodes only from the *part* partition (the default partition is assigned by the system administrator on each separate LC machine).

-t *min* (lowercase, --time=*min*)

allocates a total of *min* minutes for this job to run (default is the current partition's time limit). If *min* exceeds the partition's time limit, then the job never leaves the PENDING state. When the time limit has been reached, SLURM sends each job process SIGTERM followed (after a pause specified by SLURM's KillWait configuration parameter) by SIGKILL.

-T *nthreads* (uppercase, --threads=*nthreads*)

requests that SRUN allocate *nthreads* threads to initiate and control the parallel tasks in this job (default is the smaller of 10 or the number of nodes actually allocated, SLURM_NNODES).

SRUN Control Options

These SRUN options control how a SLURM job manages its nodes and other resources, what its working features (such as job name) are, and how it gives you help. Separate "constraint" options (page 36) (which behave like PSUB constraints) and I/O options (page 33) appear in other subsections on SRUN. Most control options have a one-character, one-hyphen (UNIX) format and an alternative keyword, two-hyphen (Linux) format, shown together here. System administrators see also SCONTROL (page 65).

Node Management

On All Machines:

-k (lowercase, --no-kill)

avoids automatic termination if any node fails that has been allocated to this job. The job assumes responsibility for handling such node failures internally. (SLURM's default is to terminate a job if any of its allocated nodes fail.)

-K (uppercase, --kill-on-bad-exit)

(default) terminates a job if any task has a nonzero exit code.

-m *dist* (lowercase, --distribution=*dist*)

tells SLURM how to distribute tasks among nodes for this job, where the choices for *dist* are:

block	assigns tasks in order to each CPU on one node before assigning any to the next node. This is the default if the number of tasks exceeds the number of nodes requested.
cyclic	assigns tasks "round robin" across all allocated nodes (task1 goes to the first node, task2 goes to the second node, etc.). This is the default if the number of nodes requested equals or exceeds the number of tasks.
hostfile	assigns tasks to nodes in the order specified by the file named in the environment variable SLURM_HOSTFILE.

-r *n* (lowercase, --relative=*n*)

offsets the first job step to node *n* of this job's allocated node set (where the first node is 0). Option -r is incompatible with "constraint" options -w and -x, and it is ignored when you run a job without a prior node allocation (default for *n* is 0). SRUN does not support job steps on BlueGene/L.

-s (lowercase, --share)

allows this job to share nodes with other running jobs. Sharing nodes often starts the job faster and boosts system utilization, but it can also lower application performance.

On BlueGene/L ONLY:

--geometry=*N[xM[xO]]*

specifies your job's size in "nodes" in each direction within BG/L's field of nodes (e.g., geometry=1x2x4 for 8 nodes). SLURM regards each BG/L 512-node dual-processor "base partition" as a *single 1024-processor node.* Use SLURM's SMAP utility (page 63) on BG/L to visualize job layout and the geometric intermixing of several jobs.

If you omit --geometry on BG/L, then SRUN uses 1x1x1 as the default (or if you also use -N *num* then SRUN uses *num*x1x1 as the default). If you omit O then the default geometry is NxMx1; if you omit both M and O then the default is Nx1x1.

--conn-type=mesh|torus

specifies the type of interconnect that you want used between BG/L "base partitions" ("nodes" to SLURM), where the choices are mesh (the default) or torus.

--node-use=coprocessor|virtual

specifies how to use the *second* processor on each BG/L compute node, where the choices are coprocessor (the default, so that the processor number is always t0) or virtual (allows processor numbers t0 and t1, but seems to be incompatible with the TotalView debugger).

-R (uppercase, --no-rotate)

disables rotation of job geometry to fit available space (the default is to rotate in three dimensions).

Working Features

--begin=*date*|*time*|*delay*|*special*

defers job start until the specified time value, which may be any *one* of these formats:

date is any calendar date in the format *month day*|*MMDDYY*|*MM/DD/YY*|*DD.MM.YY*

time is any time of day in the format *HH:MM*[*:SS*][AM|PM]

delay is specified by the digits *count* in the otherwise literal format: now + *count* minutes|hours|days|weeks

special specifies the job start time by using any one of these unusual time-literal strings:
midnight
noon
teatime [= 4 p.m.]
today
tomorrow

--core=*ctype* selects the corefile format for your job in contexts where several alternative formats are supported. Here *ctype* may be:

normal (default) specifies a full core dump.

light specifies a lightweight corefile format (with liblwcf).

list causes SRUN to print a list of (other) currently supported corefile formats (if any) and end.

--ctrl-comm-ifhn=*addr*

specifies the address *addr* or hostname to use for task communication and synchronization primitives for MPCIH2 (PMI). The default value is the response to the getnodename function (but you must supply an address if DNS lookup cannot be performed on that hostname).

-D *path* (uppercase, --chdir=*path*)

causes each remote process to change its default directory to *path* (by using CHDIR) before it begins execution (without -D, the current working directory of SRUN becomes the default directory for each process).

-d *level* (lowercase, --slurmd-debug=*level*)

specifies *level* as the level at which daemon SLURMD reports debug information and deposits it in this job's STDERR location. Here *level* can be any integer between 0 (quiet, reports only errors, the default) and 4 (extremely verbose messages).

-J *jobname* (uppercase, --job-name=*jobname*)

specifies *jobname* as the identifying string for this job (along with its system-supplied job ID, as stored in SLURM_JOBID) in responses to your queries about job status (the default *jobname* is the executable program's name).

--jobid=*jid* initiates a job step under the already allocated job whose ID is *jid* (assigning *jid* to the environment variable SLURM_JOBID has the same effect).

--mpi=*mtype* specifies the type of MPI for SLURM to support (ususally with special initialization procedures), where *mtype* can be any one of:

list	lists currently available MPI types to choose from.
lam	starts one LAMD process per node and sets the special environment variables needed for LAM MPI.
mpich-gm	starts MPI for the Myrinet switch.
mvapich	starts MPI for the Infiniband switch.
none	(default) performs no special initialization (works for many MPI versions besides those mentioned above).

--nice[=*nadjust*]

adjusts the job's scheduling priority, where *nadjust* can range from -10000 (highest priority) to 10000 (lowest priority), with a default decrease of 100. Only privileged SRUN users can raise job priority (with negative *nadjust*). Systems running the Maui scheduler ignore this option.

-P *jobid* (uppercase, --dependency=*jobid*)

defers the start of this job until the job with *jobid* has completed. Many jobs can share the same dependency *jobid*, even across different users. User SCONTROL (page 65) to change *jobid* after submittal.

-q (lowercase, --quit-on-interrupt)

causes SRUN to quit immediately when it receives a CTRL-C (SIGINT). By default, SRUN issues a status report after a single CTRL-C.

-U *acct* (uppercase, --account=*acct*)

charges this job's resource use to account *acct* on systems that have accounts (not supported at LC). Use SCONTROL (page 65) to change *acct* after submittal.

-v (lowercase, --verbose)

reports verbose messages as SRUN executes your job (default is program output with only overt error messages added). Using multiple -v options further increases message verbosity.

-X (uppercase, --disable-status)

disables the (default) report of task status when SRUN receives a single CTRL-C (SIGINT), and instead forwards the interrupt to the running job. A second CTRL-C within one second terminates the job as well as SRUN.

Resource Control

-I (uppercase, --immediate)

exits if requested resources are not available at once (by default, SRUN blocks until requested resources become available).

-O (uppercase oh, --overcommit)

overcommits CPUs. By default, SRUN never allocates more than one process per CPU. If you intend to assign multiple processes per CPU, you must invoke the -O option along with -n and -N (thus -n 16 -N 4 -O together allow 2 processes/CPU on the 4 allocated 2-CPU nodes). Even with -O, SRUN never allows more than MAX_TASKS_PER_NODE tasks to run on any single node.

--propagate[=*rlimits*]

specifies which of the modifiable (soft) resource limits *rlimits* to propagate to the compute nodes and hence apply to this job (without *rlimits*, propagates all resource limits).

-W *sec* (uppercase, --wait=*sec*)

waits *sec* seconds after the first task terminates before terminating all remaining tasks (default for *sec* is unlimited). Use -W to force an entire job to end fairly quickly if any one task terminates prematurely.

Help and Message Options

--help lists the long (Linux) and, if there is one, the corresponding short (UNIX, one-character) name for every SRUN option, with a one-line description of each. Options appear in categories by function, not alphabetically.

--mail-type=*mtype*

notifies by e-mail the user specified by --mail-user when events of type *mtype* occur, where *mtype* can be any one of:

begin	reveals the start of this job.
end	reveals the successful completion of this job.
fail	reveals premature termination of this job.
all	reveals any job state change (including those above).

--mail-user=*muser*

specifies user *muser* (default is the job's submitter) to receive any state-change e-mail messages about this job (as selected by --mail-type).

-Q (uppercase, --quiet)

suppresses all SRUN informational messages (only error messages are still displayed).

--usage reports a 9-line syntax summary for SRUN, which reveals many (but not all) SRUN options, and usually includes either the short (UNIX, one-character) or long (Linux) option name but *not* both. Options appear in no obvious order.

-V (uppercase, --version)

reports the currently installed version number for SLURM, then immediately ends.

Prolog and Epilog Options

These SRUN options let you supplement your basic job with programs that precede or follow it.

--prolog=*executable*

causes SRUN to run *executable* just before launching a job step (if NONE, the default executable, then no prolog is run). This option overrides the SrunProlog parameter in the slurm.conf file.

--epilog=*executable*

causes SRUN to run *executable* just after a job step completes (if NONE, the default executable, then no epilog is run). This option overrides the SrunEpilog parameter in the slurm.conf file.

--task-prolog=*executable*

causes the SLURMD daemon to run *executable* just before launching each task but after any TaskProlog parameter in slurm.conf is run. This task-prolog program has the normal environment variables available plus SLURM_TASK_PID (to reveal the task's process ID), and standard output from this program can be used to set environment variables for the task being launched.

--task-epilog=*executable*

causes the SLURMD daemon to run *executable* just after each task terminates but before any TaskEpilog parameter in slurm.conf is run. The task-epilog program should run only briefly, because SRUN will kill it along with any descendent processes after a few seconds.

Debug (Root) Options

These special SRUN options allow *root* users to launch jobs as a user or group other than themselves for testing or debugging.

--gid=*ggroup* (for root SRUN users only) submits this job with *ggroup*'s group access permissions, there *ggroup* may be either the intended group name or the numerical group ID.

--uid=*uuser* (for root SRUN users only) submits this job as *uuser* instead of the actual submitting user. Root users can invoke --uid to run jobs as a normal user yet in the RootOnly partition; SRUN checks the invoking user's credentials to confirm access to the target partition, but then drops the job's permissions to those of *uuser* after node allocation. Here *uuser* may be either the intended user name or the numerical user ID.

SRUN I/O Options

I/O Commands

These SRUN commands manage and redirect the standard input to, as well as the standard output and error messages from, parallel jobs executed under SLURM. Three of these commands let you choose from among any of five I/O redirection alternatives ("modes") that are explained in the next section. (page 34)

-o *mode* (lowercase, --output=*mode*)

redirects standard output STDOUT for this job to *mode*, one of five alternative ways to display, capture, or subdivide the job's I/O, explained in the next subsection (page 34). By default, SRUN collects STDOUT from all job tasks and line buffers it to the attached terminal.

-i *mode* (lowercase, --input=*mode*)

redirects standard input STDIN for this job from *mode*, one of five alternative ways to display, capture, or subdivide the job's I/O, explained in the next subsection (page 34). By default, SRUN redirects STDIN from the attached terminal to all job tasks.

-e *mode* (lowercase, --error=*mode*)

redirects standard error STDERR for this job to *mode*, one of five alternative ways to display, capture, or subdivide the job's I/O, explained in the next subsection (page 34). By default, SRUN collects STDERR from all job tasks and line buffers it to the attached terminal, just as with STDOUT. But you can request that SRUN handle standard output and standard error differently by invoking -e and -o with different redirection modes.

-l (lowercase ell, --label)

prepends the remote task ID number to each line of standard output and standard error. By default, SRUN line buffers this I/O to the terminal (or to specified files) without any task labels. Options -l and -u are mutually exclusive.

-u (lowercase, --unbuffered)

prevents line buffering of standard output from remote tasks (buffering is the SRUN default). Options -l and -u are mutually exclusive.

I/O Redirection Alternatives

SRUN I/O options (page 33) -i (--input), -o (--output), and -e (--error) all take as arguments any of five I/O redirection alternatives ("modes") summarized in this table and explained in more detail below it:

Redirection Alternatives	**File-Naming Subchoices**	**Tasks Covered**	**I/O Goes To or From**
all [default]		all tasks	SRUN (terminal)
none		all tasks	/dev/null (ignored)
taskid		one selected task	SRUN (terminal)
filename		all tasks	one specified file
fstring		many separate tasks:	many separate files:
	%J [uc]	all with *jobid.stepid*	1 file per *jobid.stepid*
	%j [lc]	all with *jobid*	1 file per *jobid*
	%s [lc]	all with *stepid*	1 file per *stepid*
	%N [uc]	all on *hostname*	1 file per node
	%n [lc]	all on node *n*	1 file per node
	%t [lc]	each separate task	1 file per task

The I/O redirection alternatives compared in the table work in detail as follows:

all — redirects STDOUT and STDERR from all job tasks to SRUN (and hence to the attached terminal), and broadcasts STDIN from SRUN (the terminal) to all remote tasks (this is SRUN's default behavior for handling I/O).

none — redirects STDOUT and STDERR from all job tasks to /dev/null (i.e., SRUN receives no I/O from any task), and sends no STDIN to any task (closes STDIN).

taskid — redirects to SRUN (and hence to the attached terminal) STDOUT and STDERR from the single specified task whose relative ID is *taskid*, where the range for integer *taskid* starts at 0 (the first task) and runs through the total number of tasks in the current job step. This choice also redirects STDIN from SRUN (the terminal) to this single specified task.

filename — redirects STDOUT and STDERR from all job tasks into a single file called *filename*, and broadcasts STDIN from that same file to all remote tasks. To subdivide the I/O among separate files, use the *fstring* alternative below.

fstring ["format string"]

uses a parameterized "format string" to systematically generate unique names for (usually) multiple I/O files, each of which receives some job I/O depending on the naming scheme that you choose. You can subdivide the received I/O into separate files by job ID, step ID, node (name or sequence number), or individual task. In each case, SRUN opens the appropriate number of files and associates each with the appropriate subset of tasks.

Available parameters with which to construct *fstring* (and thereby to split the I/O among separate files) include:

%J (uppercase) creates one file for each job ID/step ID combination for this running job, and imbeds *jobid.stepid* in each file's name (for example, out%J might yield files out4812.0, out4812.1, etc.).

%j (lowercase) creates one file for each job ID and imbeds *jobid* in its name (for example, job%j might yield file job4812).

%s (lowercase) creates one file for each step ID and imbeds *stepid* in its name (for example, step%s.out would yield files step0.out, step1.out, etc.). SRUN does not support job steps on BlueGene/L.

%N (uppercase) creates one file for each node on which this job runs and imbeds that node's short hostname in the file name (for example, node.%N might yield files node.mcr347, node.mcr348, etc.).

%n (lowercase) creates one file for each node on which this job runs and imbeds that node's numerical identifier relative to the job (where the each job's first node is 0, then 1, etc.) in the file name (for example, node%n would yield files node0, node1, etc.).

%t (lowercase) creates one file for each separate task in this running job and imbeds that task's numerical identifier relative to the job (the first task is 0) in the file name (for example, job%j-%t.out might yield files job4812-0.out, job4812-1.out, etc.).

For all *fstring* parameters except the nonnumeric case of %N, you can insert an integer between the percent character and the letter (such as %3t) to "zero-pad" the resulting file names, that is, to always use the integer number of character positions and to fill any empty positions with zeros from the left. Thus job%j-%3t.out might yield files job4812-000.out, job4812-001.out, etc.

SRUN Constraint Options

These SRUN options all limit the nodes on which your job will execute to only those nodes having the properties ("constraints") that you specify here.

General Constraints

These SRUN constraints can apply to any job (unlike those in the next subsection).

-C *clist* (uppercase, --constraint=*clist*)

runs your job on those nodes having the properties in *clist*, where *clist* is a list of features assigned for this purpose by SLURM system administrators (the features may vary by network or machine). SRUN option -C thus behaves like PSUB constraint option -c (lowercase, see the "Helpful PSUB Options" section (URL: http://www.llnl.gov/LCdocs/ezjob/index.jsp?show=s6.5.2) of the EZJOBCONTROL guide.
To conjoin (AND) multiple constraints, separate them in *clist* by using an ampersand (c1&c2). To disjoin (OR) multiple constraints, separate them in *clist* by using a vertical bar (c3|c4).
If no nodes have the feature(s) that you require with -C, then the SLURM job manager will reject your job.

--contiguous=yes|no

specifies whether or not your job requires a contiguous range of nodes. The default (YES) demands contiguous nodes, while alternative NO allows noncontiguous execution.

--mem=*size* specifies a minimum amount of real memory per node, where *size* is an integer number of megabytes. See also --vmem.

--mincpus=*n* specifies a minimum number *n* of CPUs per node.

--vmem=*size* specifies a minimum amount of virtual memory per node, where *size* is an integer number of megabytes. See also --mem.

--tmp=*size* specifies a minimum amount of temporary disk space per node, where *size* is an integer number of megabytes.

-w *hosts* (lowercase, --nodelist=*hosts*)

specifies by name the individual nodes that must be included in the set of nodes on which your job runs (perhaps along with others unspecified). Option -w is incompatible with SRUN option -r (--relative). Here *hosts* may have any of three formats:

host1,host2,... is a comma-delimited list of node names (e.g., mcr100,mcr200,...).

host[na-nb,nc,...] is a range of node names, perhaps mixed with individual nodes in a comma-delimited sublist (e.g., mcr[1-256,500,...]).

filename is a file that contains node information in either of the previous two formats (SRUN interprets any string containing the slash (/) character as a file name).

-x *hosts* (lowercase, --exclude=*hosts*)

specifies by name the individual nodes that must be excluded from the set of nodes on which your job runs (perhaps along with others unspecified). Option -x is incompatible with SRUN option -r (--relative). Here *hosts* may have any of three formats:

host1,host2,... is a comma-delimited list of node names (e.g., mcr100,mcr200,...).

host[na-nb,nc,...] is a range of node names, perhaps mixed with individual nodes in a comma-delimited sublist (e.g., mcr[1-256,500,...]).

filename is a file that contains node information in either of the previous two formats (SRUN interprets any string containing the slash (/) character as a file name).

Affinity or NUMA Constraints

These SRUN constraints apply only to machines where the task-affinity or the NUMA (NonUniform Memory Access) plugins have been enabled by the operating system. At LC, that includes only BlueGene/L.

--cpu_bind=[quiet,|verbose,]*type*

binds tasks to CPUs (to prevent the operating system scheduler from moving the tasks and spoiling possible memory optimization arrangements).

q[uiet] (default) quietly binds CPUs before the tasks run.

v[erbose] verbosely reports CPU binding before the tasks run.

Here *type* can be any one of these mutually exclusive alternatives:

no[ne] (default) does not bind tasks to CPUs.

rank binds tasks to CPUs by task rank.

map_cpu:*idlist* binds by mapping CPU IDs to tasks as specified in *idlist*, a comma-delimited list *cpuid1,cpuid2,...,cpuidn*. SRUN interprets CPU IDs as decimal values unless you precede each with 0x to specify hexadecimal values.

mask_cpu:*mlist* binds by setting CPU masks on tasks as specified in *mlist*, a comma-delimited list *mask1,mask2,...,maskn*. SRUN always interprets masks as hexadecimal values (so using the 0x prefix is optional).

To have SLURM always report on the selected CPU binding for all SRUN instances executed in a shell, you can enable verbose mode directly by typing:
setenv SLURM_CPU_BIND verbose
However, SLURM_CPU_BIND will not propagate to tasks (binding by default only affects the first execution of SRUN). To propagate --cpu_bind to successive SRUN instances, excute the following in each task:
setenv SLURM_CPU_BIND \
${SLURM_CPU_BIND_VERBOSE},
${SLURM_CPU_BIND_TYPE}
${SLURM_CPU_BIND_LIST}

--mem_bind=[quiet,|verbose,]*type*

binds tasks to memory (to stabilize possible memory optimization arrangements). WARNING: the resolution of CPU and memory binding may differ on some architectures. CPU binding may occur at the level of cores within a processor while memory binding may occur at the level of nodes (whose definition may vary from one system to another). Hence, the use of any *type* other than NONE or LOCAL is not recommended.

q[uiet]	(default) quietly binds memory before the tasks run.
v[erbose]	verbosely reports memory binding before the tasks run.

Here *type* can be any one of these mutually exclusive alternatives:

no[ne]	(default) does not bind tasks to memory.
rank	binds tasks to memory by task rank.
local	uses memory local to the processor on which each task runs.
map_mem:*idlist*	binds by mapping a node's memory to tasks as specified in *idlist*, a comma-delimited list *cpuid1,cpuid2,...,cpuidn*. SRUN interprets CPU IDs as decimal values unless you precede each with 0x to specify hexadecimal values.
mask_mem:*mlist*	binds by setting memory masks on tasks as specified in *mlist*, a comma-delimited list *mask1,mask2,...,maskn*. SRUN always interprets masks as hexadecimal values (so using the 0x prefix is optional).

To have SLURM always report on the selected memory binding for all SRUN instances executed in a shell, you can enable verbose mode directly by typing:
setenv SLURM_MEM_BIND verbose
However, SLURM_MEM_BIND will not propagate to tasks (binding by default only affects the first execution of SRUN). To propagate --mem_bind to successive SRUN instances, excute the following in each task:
setenv SLURM_MEM_BIND \
${SLURM_MEM_BIND_VERBOSE},
${SLURM_MEM_BIND_TYPE}
${SLURM_MEM_BIND_LIST}

Environment Variables

To see how the SLURM environment variables discussed here fit into the larger context of all environment variables used at LC to manage jobs (both interactively and by LCRM in particular), consult the comparative sections of LC's Environment Variables user manual (URL: http://www.llnl.gov/LCdocs/ev).

Option Variables.
Many SRUN options have corresponding environment variables (analogous to the approach used with POE). The SRUN option, if invoked during execution, always overrides (resets) the corresponding environment variable (which contains each job feature's default value, if there is a default).

Environment Variable	Corresponding SRUN Option(s)
SLURM_ACCOUNT	-U, --account
SLURM_CPU_BIND	--cpu_bind
SLURM_CPUS_PER_TASK	-c, --ncpus-per-task
SLURM_CONN_TYPE	--conn-type
SLURM_CORE_FORMAT	--core-format
SLURM_DEBUG	-v, --verbose
SLURMD_DEBUG	-d, --slurmd-debug
SLURM_DISABLE_STATUS	-X, --disable-status
SLURM_DISTRIBUTION	-m, --distribution
SLURM_GEOMETRY	-g, --geometry
SLURM_LABELIO	-l, --label
SLURM_MEM_BIND	--mem_bind
SLURM_NETWORK(*)	--network
SLURM_NNODES	-N, --nodes
SLURM_NO_ROTATE	--no-rotate
SLURM_NPROCS	-n, --ntasks
SLURM_OVERCOMMIT	-o, --overcommit
SLURM_PARTITION	-p, --partition
SLURM_REMOTE_CWD	-D, --chdir
SLURM_SRUN_COMM_IFHN	--ctrl-comm-ifhn
SLURM_STDERRMODE	-e, --error
SLURM_STDINMODE	-i, --input
SLURM_STDOUTMODE	-o, --output
SLURM_TASK_EPILOG	--task-epilog
SLURM_TASK_PROLOG	--task-prolog
SLURM_TIMELIMIT	-t, --time
SLURM_WAIT	-W, --wait

(*)See explanatory details at the end of this section.

Task-Environment Variables.

In addition, SRUN sets these environment variables (a few are the same as option variables listed above) for each executing task on each remote compute node (any operating system).

SLURM_CPU_BIND_VERBOSE

affects the reporting of CPU/task binding, as explained in the "Affinity or NUMA Constraints" section (page 38) under --cpu_bind.

SLURM_CPU_BIND_TYPE

affects the binding of CPUs to tasks, as explained in the "Affinity or NUMA Constraints" section (page 38) under --cpu_bind.

SLURM_CPU_BIND_LIST

affects the binding of CPUs to tasks, as explained in the "Affinity or NUMA Constraints" section (page 38) under --cpu_bind.

SLURM_CPUS_ON_NODE

specifies the number of processors available to the job on this node.

SLURM_JOBID

specifies the job ID of the executing job (see also SRUN's --jobid option (page 26)).

SLURM_LAUNCH_NODE_IPADDR

specifies the IP address of the node from which the task launch initiated (the node where SRUN executed).

SLURM_LOCALID

specifies the node-local task ID for the process within a job.

SLURM_MEM_BIND_VERBOSE

affects the reporting of memory/task binding, as explained in the "Affinity or NUMA Constraints" section (page 38) under --mem_bind.

SLURM_MEM_BIND_TYPE

affects the binding of memory to tasks, as explained in the "Affinity or NUMA Constraints" section (page 38) under --mem_bind.

SLURM_MEM_BIND_LIST

affects the binding of memory to tasks, as explained in the "Affinity or NUMA Constraints" section (page 38) under --mem_bind.

SLURM_NNODES

is the actual number of nodes assigned to run your job (which may exceed the number of nodes that you explicitly requested with SRUN's -N option (page 22)).

SLURM_NODEID

specifies the relative node ID of the current node.

SLURM_NODELIST

specifies the list of nodes on which the job is actually running.

SLURM_NPROCS

specifies the total number of processes in the job.

SLURM_PROCID

specifies the MPI rank (or relative process ID) for the current process.

SLURM_TASKS_PER_NODE

specifies the number of tasks to initiate on each node. Values are a comma-delimited list in the same order as SLRUM_NODELIST. To specify two or more nodes with the same task count, follow the count by (x#), where # is the repetition count. For example,
SLURM_TASKS_PER_NODE=2(x3),1
indicates two tasks per node on the first three nodes, then one task on the fourth node.

MPIRUN_PARTITION

(BlueGene/L only) specifies the block name.

MPIRUN_NOALLOCATE

(BlueGene/L only) prevents allocating a block.

MPIRUN_NOFREE

(BlueGene/L only) prevents freeing a block.

Other SLURM-Relevant Variables.

Other environment variables important for SRUN-managed jobs include:

MAX_TASKS_PER_NODE

provides an upper bound on the number of tasks that SRUN assigns to each job node, even if you allow more than one process per CPU by invoking SRUN's -O (uppercase oh) option. (page 29)

SLURM_HOSTFILE

names the file that specifies how to assign tasks to nodes, rather than using the block or cyclic approaches toggled by SRUN's -m (--distribution) option (page 24).

On AIX (IBM) machines only, this extra environment variable is automatically set by LCRM when SLURM is used instead of LoadLeveler (alternatively, set with SRUN's --network=*type* option even though POE launches tasks instead of SRUN under AIX):

SLURM_NETWORK

specifies four network features for each SLURM job step (which under AIX means for each POE invocation), using an argument with this sequential format:
network.[*protocol*],[*device*],[*adapteruse*],[*mode*]
where:

protocol specifies the network protocol (such as MPI).

device specifies the kind of switch used for communication (ethernet, FDDI, etc.), where the choices are the same abbreviation strings as the possible values of environment variable MP_EUIDEVICE (see the POE User Guide, "Task Communication" section (URL: http://www.llnl.gov/LCdocs/poe/index.jsp?show=s6.3.3)).

adapteruse specifies whether (SHARED) or not (DEDICATED) your job is willing to share a node's switch adapter with other jobs (see the POE User Guide, "Other POE Environment Variables" section (URL: http://www.llnl.gov/LCdocs/poe/index.jsp?show=s6.4) on corresponding environment variable MP_ADAPTER_USE).

mode specifies which of two protocols or modes should be used for task communications, where the choices are the same as the possible values of environment variable MP_EUILIB (see the POE User Guide, "Task Communication" section (URL: http://www.llnl.gov/LCdocs/poe/index.jsp?show=s6.3.3)).

Multiple Program Usage

Strategy.
SRUN's --multi-prog option (see SRUN Resource-Allocation Options above (page 22)) lets you assign to each parallel task in your job a different program with (if you wish) a different argument.

If you invoke --multi-prog, then SRUN's own argument is not the name of one executable program (as usual) but rather the name of a local *configuration file* that specifies how to assign multiple programs and arguments among your job's tasks. For example,

```
srun -n8 -l --multi-prog test.config
```

(where -l here labels each task's output by task number for clarity).

Configuration File.
Each row in an SRUN --multi-prog configuration file contains these three ordered fields, separated by blanks:

Task number — specifies by a comma-delimited series of nonnegative integers the specific tasks configured on this row, with ranges specified by hyphens (for instance: 2,4-8,10,12). Star (*) specifies all tasks, which of course defeats the point of invoking --multi-prog.

Executable file — assigns to this row's tasks the specified program to execute (may be an absolute pathname if you wish, such as /usr/bin/spell).

Argument — assigns to this row's program the argument(s) listed. You can parameterize the argument(s) by including either of these special expressions:

- %t — evaluates to the task number of the responsible task.
- %o — (lowercase oh) evaluates to the task offset, that is, to the task's *relative* position within the task range configured on this row (always starts with zero).

Example Usage.
Here is a sample --multi-prog configuration file to illustrate the above features:

```
4-6     hostname
1,7     echo      task:%t
0,2,3   echo      task:%o
```

Here is the output of using the SRUN execute line shown above with this file serving as test.config (and run on machine ALC):

```
0: offset:0
1: task:1
2: offset:1
3: offset:2
4: alc20.llnl.gov
5: alc21.llnl.gov
6: alc22.llnl.gov
7: task:7
```

SQUEUE (List Jobs)

SQUEUE Execute Line

SQUEUE displays the job ID and job name for every job currently managed by the SLURM control daemon (SLURMCTLD) on the machine where you run SQUEUE, along with status and resource information for each job (such as time used so far, or a list of committed nodes), in a table whose content and format details you can control with SQUEUE options. (To report on *node* status rather than job status, use SINFO (page 53) instead.)

BASIC RUN.
To run SQUEUE (on any machine with SLURM installed), type

squeue [*opts*]

where *opts* is a blank-delimited list of SQUEUE output-control options and their (comma-delimited) arguments.

COLUMNS.
If run without options, SQUEUE reports by default on these job properties (in this order in columns left to right) and then ends:

```
job ID
partition
job name
user name
status (state) of job [for codes, see later section]
time used so far (hours:minutes:seconds)
total nodes (allocated or used)
node list (specific node names)
```

See the first example (page 50) below for typical default SQUEUE output. You can change the properties reported (and the order of the output columns) by using SQUEUE's -o (lowercase oh, --format) option.

ROWS.
Each row in the SQUEUE report describes one job (or, if you request, one job step; but job steps are not supported on BlueGene/L). SQUEUE sorts the rows (alphabetically, or in decreasing magnitude) using this ranked list of job properties:

```
partition name
status (state) of job
priority
time used so far
```

You can change the sort order (to more closely match PSUB by relying on job ID or job name, for example) by using SQUEUE's -S (uppercase ess, --sort) option.

SQUEUE Options

Delimit all SQUEUE options with *spaces* (blanks), but delimit items in option-argument lists with *commas* unless otherwise noted (-o requires space-delimited arguments, for example). Enclose all argument lists in quotes (") for greater reliability. This section lists SQUEUE's control options alphabetically except for -o, which gets a separate subsection at the end because of its elaborate format-specification language.

CONTROL OPTIONS.

-? (--help) displays a brief SQUEUE options summary, one line per option, giving both the single-character and Linux-syntax versions of each option along with a very short note on its role (but no argument details). The -? version fails on some machines, so prefer --help.

-h (lowercase, --noheader)

omits the explanatory heads from SQUEUE's tabular output (for easier reuse of the text).

-i *sec* (lowercase, --iterate=*sec*)

repeatedly gathers and reports the status information (as specified by other options) every *sec* seconds. Use CTRL-C to stop this iterated display.

-j *jlist* (lowercase, --jobs=*jlist*)

limits the SQUEUE report to only the jobs specified in *jlist*, a comma-delimited list of numerical SRUN job IDs (for example, --jobs="1235,1237,1239"). The default report covers all current jobs. You cannot change the order in which SQUEUE reports on jobs by using this option (instead, use -S or --sort).

-l (lowercase ell, --long)

adds to SQUEUE's default report another column (TIMELIMIT, which may contain numerical values or the strings NOT_SET or UNLIMITED) and uses full words (such as RUNNING) rather than abbreviations (such as R, see later section (page 52)) to report job states. (Compare with -v, --verbose below.)

-p *plist* (lowercase, --partition=*plist*)

limits SQUEUE's report to jobs in the partitions specified by *plist*, a comma-delimited list of partition names (such as "debug" or "pbatch").

-s *slist* (lowercase, --steps=*slist*)

limits the SQUEUE report to only the steps specified in *slist*, a comma-delimited list of numerical SRUN job-step IDs (for example, --steps="6543.1,6555.3"). The default report covers all current job steps. SQUEUE does not support job steps on BlueGene/L.

-S *sortkeys* (uppercase, --sort=*sortkeys*)

sorts the (job) rows in SQUEUE's report using the sort keys specified in *sortkeys*, a comma-delimited list of the same field (column) specifiers used for and explained in the -o (--format) option below. The default order is ascending; prefix each field specifier with minus (-) for descending order. The default sort for jobs is --sort="P,t,-p" (increasing partition names, then increasing job states, then decreasing job priority). The default sort for job steps is --sort="P,i" (increasing partition names, then increasing job-step IDs). SQUEUE does not support job steps on BlueGene/L.

-t *statelist* (lowercase, --states=*statelist*)

limits SQUEUE's report to jobs with the specified states (statuses), where *statelist* is a comma-delimited list with these possible members: ALL, PENDING, RUNNING, COMPLETE. The default report includes only PENDING and RUNNING jobs. See the Job States (page 52) section below for an explanatory list of all SQUEUE job-state codes.

-u *ulist* (lowercase, --users=*ulist*)

limits SQUEUE's report to jobs belonging to the users specified in *ulist*, a comma-delimited list of user names on the system where you run SQUEUE.

-v (lowercase, --verbose)

prefixes the job report (table) that SQUEUE would otherwise generate (based on the other control options that you invoked) with a list of 12 report-control features and the current setting for each (for example, the limits that you have currently imposed on states, users, or output format). The job report itself is not changed (compare with -l, --long).

-V (uppercase, --version)

displays SQUEUE's current version number and ends (with no job report).

OUTPUT-FORMAT OPTION.

-o *formatspec* (lowercase, --format=*formatspec*)

specifies both the content of SQUEUE's report (which job properties to include as report columns) and the layout (the left-to-right order of those columns, the size of each column in characters, and whether the data is left- or right-justified within each column). Here *formatspec* is a quoted, space (not comma) delimited list of "field specifications," one for each column that you want SQUEUE to report, and each with this syntax: *%.wZ*

%	is the [required] field specification *flag*, marking the start of each column's separate specification.
.(dot)	requests right *justification* of this column's data (the default omits the dot and uses left justification of the reported data).

w is an integer specifying the *width* of this column in characters (omitting *w* uses just as much space as the data requires, which usually means that there is no column alignment from one row (= job) to the next).

Z is a single case-sensitive letter that specifies the content (the *job property*) reported in this column (using the dictionary given below). For example, %.8j uses a right-justified (.) column 8 characters wide to report job name (lowercase j).

The default column specifications for noncustomized SQUEUE job reports are:
-o "%.7i %.9P %.8j %.8u %.2t %.9M %.6D %N"
See the SQUEUE example (page 50) section below for a sample default report and a sample column-customized report built using the -o option.

Available one-letter case-sensitive content specifiers (*Z*) for use in *%.wZ* field specifications are (in alphabetical order):

```
b   time when this job started executing
c   minimum number of CPUs requested
C   (uppercase) actual number of CPUs allocated
d   minimum temp disk space (in Mbyte) requested
D   (uppercase) minimum number of nodes requested (if pending),
    or, actual number of nodes allocated (if running)
e   time when job ended (or is predicted to end)
f   node features required by this job
h   (yes/no) allocated nodes can be shared with other jobs?
i   job ID or job-step ID (numeric)
j   job name (alphanumeric)
l   time limit (days:hours:minutes:seconds)
    or NOT_SET yet, or UNLIMITED
m   minimum requested memory size (in Mbyte)
M   (uppercase) time used (days:hours:minutes:seconds)
n   list of requested node names
N   (uppercase) list of allocated node names
o   minimum number of nodes requested
O   (uppercase) (yes/no) are contiguous nodes requested?
p   job priority (0.0 .GE. p .LE. 1.0)
P   (uppercase) job partition
S   (uppercase) job start time
t   job state (abbreviated, see later section)
T   (uppercase) job state (spelled out)
u   user name (alphanumeric)
U   (uppercase) user ID (numeric)
```

SQUEUE Examples

[1]

GOAL: To display the default status report about all current SLURM-managed jobs on the machine (cluster) where you run SQUEUE.

STRATEGY: Run SQUEUE with no options. An eight-column report, sorted by partition and then by time used (not by job ID or name), appears. SQUEUE automatically ends. Column ST here reports job STATE (status, see later section (page 52) for details).
To add a time-limit column and see full-word status entries, use SQUEUE's -l (lowercase ell, --long) option. To report only on specified jobs rather than on all jobs, use the -j (--jobs) option with a comma-delimited list of (numerical) job IDs as the argument.

```
User: squeue

Rtne:
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST
  46675    pbatch Rptn3202     bert   R   10:28:55    256 mcr[163-346,
                                                          572-583,663-679,
                                                          766-788,
                                                          1132-1151]
   4631    pbatch    runbo   jgross   R   10:17:15    280 mcr[616-630,
                                                          680-765,889-1032,
                                                          1097-1131]
   4007    pbatch paris03.   kubota   R   10:02:48     16 mcr[556-571]
   3866    pbatch NVE85Ryd     ikuo   R    5:21:10    144 mcr[525-552,631-
                                                          662,805-888]
   4655    pbatch     cp2k mcgrath5   R    1:52:10    128 mcr[104-162,475-
                                                          479,1033-1096]
   4827    pbatch   64chLd hiratani   R    1:51:50     16 mcr[584-599]
   4812    pbatch batch_MC   rieben   R    1:51:27     64 mcr[480-524,600-
                                                          615,793-795]
   5256    pbatch    d32_2     bosl   R    1:36:05     60 mcr[347-406]
   4736    pbatch psub_10m    mrhee   R    1:35:36     16 mcr[407-422]
   4743    pbatch psub_100    mrhee   R    1:35:04     16 mcr[439-454]
   5305    pbatch amBeA59_   aspuru   R      41:52      8 mcr[423-430]
   5101    pbatch     egas pederiva   R      32:22     32 mcr[455-474,789-
                                                          791,796-804]
  66927    pdebug TVdebugS   awcook  PD       0:00     20
  67151    pdebug    ale3d kippster   R      22:39      2 mcr[96-97]
  67152    pdebug  inferno   kubota   R      22:00     16 mcr[40-55]
  67168    pdebug    ale3d kippster   R       9:53      2 mcr[98-99]
  67169    pdebug gd2d.Lin   benliu   R       7:00     16 mcr[56-71]
```

---

[2]

GOAL: To build a customized status report about current SLURM-managed jobs, for example, showing only job names, requested features (if any), and time used, with all rows in alphabetical (instead of time-used) order.

STRATEGY: (1) Use SQUEUE's -o (lowercase oh, --format) option to specify which specific columns (job properties) you want to report, the width of each column in characters, and the order for the columns to appear (left to right). SQUEUE's general column-specification language is described with -o in the options section (page 47) above. Here, %15j requests a 15-character job-name column, %10f requests a 10-character features-requested column, and %15M requests a 15-character time-used column.
(2) Use SQUEUE's -S (uppercase ess, --sort) option to arrange rows alphabetically by job name (j) rather than in the default order (the order shown in Example 1).

```
User: squeue  -o "%15j %10f %15M" -S "j"

Rtne:
  NAME            FEATURES   TIME
  R20026          (null)     6:45:34
  TVdebugSession  (null)     25:18
  batch           (null)     1:52:09
  bw73d.scr       (null)     4:33:45
  cale.mcr-icc-de (null)     4:34
  case5_03_31_04. (null)     6:45:21
  jeep            (null)     20:14
  jobrestartl     (null)     3:09:13
  mcrh2.bat       (null)     1:37:24
  mcrh2.bat       (null)     10:31
  mcrh2.bat       (null)     10:14
  mcrh2.bat       (null)     9:16
  mcrh2.bat       (null)     2:32
  old_config      (null)     1:52:30
  pu1.psub        (null)     6:45:14
  pu2.psub        (null)     6:45:05
  runkull         (null)     0:49
  runover         (null)     6:45:27
  runpF3d_mcr     (null)     5:02:17
  sst             (null)     2:03:50
```

SQUEUE Job State Codes

Most SQUEUE reports use short codes (abbreviations) to reveal the state (current status) of each job that SLURM manages. The SQUEUE job-state codes and what they mean are explained here in alphabetical order. A separate section covers SINFO *node* state codes (page 60).

Note that these SQUEUE codes differ from those used by PSTAT to report the status that LCRM/DPCS assigns to the batch jobs that it schedules (across machines "above" SLURM). See the Status Values for Batch Jobs (URL: http://www.llnl.gov/LCdocs/dpcs/index.jsp?show=s4.1.2) section of the LCRM/DPCS Reference Manual for a long explanatory list of those different job states.

CD (COMPLETED)
Job has successfully ended all of its processes on all nodes (LCRM/DPCS: CMPLETED).

CG (COMPLETING)
Job is in the process of completing, so some processes on some nodes may still be active.

F (FAILED)
Job has terminated with a nonzero exit code or other failure condition.

NF (NODE_FAIL)
Job has terminated because one or more nodes allocated to it has failed.

PD (PENDING)
Job is awaiting resource allocation (there are many corresponding LCRM/DPCS states depending on just which resources are needed).

R (RUNNING)
Job is executing successfully now (LCRM/DPCS: RUN).

TO (TIMEOUT)
Job has terminated upon reaching its time limit (LCRM/DPCS: TERMINATED).

SINFO (List Nodes)

SINFO Execute Line

SINFO reports current status information on node partitions and on individual nodes for computer systems managed by SLURM. SINFO's reports can help you plan job submittals and avoid hardware problems. SINFO's output is a table whose content and format you can control with SINFO options. (To report on *job* status rather than on node status, use SQUEUE (page 46) instead.)

BASIC RUN.
To run SINFO (on any machine with SLURM installed), type

sinfo [*opts*]

where *opts* is a blank-delimited list of SINFO output-control options and their quoted, comma-delimited arguments.

COLUMNS.
If run without options, SINFO reports by default on these node properties (in this order in six columns left to right) and then ends:

```
partition
availability
time limit
node count
node state [for state codes, see later section]
node list (specific node names covered)
```

See the first example (page 61) below for typical default SINFO output. You can change the properties reported (and the order of the output columns) by using SINFO's -o (lowercase oh, --format) option.

ROWS.
Each row in the SINFO report describes one node partition (or, if you use node-oriented options such as -n or -N, one or more specific nodes). SINFO sorts the rows by partition name (in a preconfigured order as specified in the file /etc/slurm/slurm.conf) and then in decreasing order by node state (or in increasing order by node name for node-oriented reports). However, you can change the order in which SINFO sorts rows by using SINFO's -S (uppercase ess, --sort) option, described in the next section.

SINFO Options

Delimit all SINFO options with *spaces* (blanks), but delimit items in option-argument lists with *commas* unless otherwise noted (-o requires space-delimited arguments, for example). Enclose all argument lists in quotes (") for greater reliability. This section lists SINFO's control options alphabetically except for -o, which gets a separate subsection at the end because of its elaborate format-specification language.

HELP OPTIONS.

--help displays a brief SINFO options summary, one line per option, giving both the single-character and Linux-syntax versions of each option along with a very short note on its role (and some argument hints).

--usage displays a MAN-format syntax summary for SINFO (shows option letters but no explanatory clues). For more vebose help, use --help.

CONTROL OPTIONS.

-a (lowercase, --all)

reports on partitions that are configured as hidden and partitions that are unavailable to the user's group (if any). To report on fewer or specific partitions, use -p. To report more output fields (columns), use -l. Option --hide undoes -all.

-d (lowercase, --dead)

reports the count of nonresponding ("dead") nodes for each partition (if there are any; often 0) and includes STATE and NODELIST information only for those (if any).

-e (lowercase, --exact)

(applies *only* when reporting CPU count, memory size, or disk space using -o or -Nl) does not group node information on multiple nodes unless their configurations are identical. By default (without -e), SINFO may report CPU count, memory size, and disk space for nodes in the same partition and state with a minimum value followed by a plus sign (e.g., "250+").

-h (lowercase, --noheader)

omits the explanatory heads from SINFO's tabular output (for easier reuse of the text).

--hide (no one-character version)

suppresses reports on all partitions declared hidden or not available to the user's group (the default; --all undoes --hide).

-i *sec* (lowercase, --iterate=*sec*)

repeatedly gathers and reports the status information (as specified by other options) every *sec* seconds. Use CTRL-C to stop this iterated display.

-l (lowercase ell, --long)

displays four more output fields (page 58) (columns) than in SINFO's default report (JOB_SIZE, ROOT, SHARE, GROUPS), but no more rows (instead try --all). This option is incompatible with -o (--format). Combining -N with -l reports CPU count, memory size, disk space, scheduling weight, and declared features (if any).

-n *nodes* (lowercase, --nodes=*nodes*)

reports information only for the specified *nodes*. For *nodes* use a quoted full node name (e.g., "mcr224"), a quoted comma-delimited list of full node names (e.g., "mcr224,mcr225"), or a quoted range of nodes specified with brackets (e.g., "mcr[224-227]").

-N (uppercase, --Node)

reports information in a node-oriented format (begins with node list and node count columns). By default, SINFO reports information in a partition-oriented format. This option is incompatible with -o (--format). Combining -N with -l reports CPU count, memory size, disk space, scheduling weight, and declared features (if any).

-r (lowercase, --responding)

reports state information only for responding nodes (omits DOWN or DRAINED nodes).

-R (uppercase, --list-reasons)

reports, for each node that is currently DOWN or DRAINED, its name and the first 35 characters of the "reason field" optionally provided by the SLURM system administrator. All other nodes are omitted (unless you invoke additional options). To explicitly report the STATE as well as the REASON for each troubled node, you must combine -R with -l (or use -o).

-s (lowercase, --summarize)

replaces SINFO's default format (one row for each different STATE in which some nodes fall) with a compressed NODES(A|I|O|T) column that summarizes the count of available|idle|other|total nodes in one string for each partition. This option is incompatible with -o (--format).

-S *sortkeys* (uppercase, --sort=*sortkeys*)

sorts the (partition or node) rows in SINFO's report using the sort keys specified in *sortkeys*, a comma-delimited list of the same field (column) specifiers used for and explained in the -o (--format) option below. The default order is ascending; prefix each field specifier with minus (-) for descending order. The default sort for partitions is --sort="#P,-t" (partitions ordered as configured in /etc/slurm/slurm.conf, then decreasing node state). With -N, the default sort is --sort="N" (increasing node name).

-t *statelist* (lowercase, --states=*statelist*)

limits SINFO's report to nodes with the specified states, where *statelist* is a quoted, comma-delimited list with these possible members (case insensitive):
ALLOC
ALLOCATED
COMP
COMPLETING
DOWN
DRAIN
DRAINED
DRAINING
IDLE
UNK
UNKNOWN.
By default, SINFO reports on nodes in the specified states whether they are responding or not, but you can use -d or -r to filter this report further.

-p *pname* (lowercase, --partition=*pname*)

reports information only about the specified (single) partition (lists are *not* accepted as an argument here).

-v (lowercase, --verbose)

prefixes the node report (table) that SINFO would otherwise generate (based on the other control options that you invoked) with a list of 16 report-control features and the current setting for each (revealing, for example, the limits that you have currently imposed on states, nodes, or output format). The basic report itself is not changed (compare with -l, --long).

-V (uppercase, --version)

displays SINFO's current version number and ends (with no node report).

OUTPUT-FORMAT OPTION.

-o *formatspec* (lowercase, --format=*formatspec*)

specifies both the content of SINFO's report (which node properties to include as report columns) and the layout (the left-to-right order of those columns, the size of each column in characters, and whether the data is left- or right-justified within each column). Here *formatspec* is a quoted, space (not comma) delimited list of "field specifications," one for each column that you want SINFO to report, and each with this syntax: *%.wZ*

% is the [required] field specification *flag*, marking the start of each column's separate specification.

.(dot) requests right *justification* of this column's data (the default omits the dot and uses left justification of the reported data).

w is an integer specifying the *width* of this column in characters (omitting *w* uses just as much space as the data requires, which usually means that there is no column alignment from one row (= node) to the next).

Z is a single case-sensitive letter that specifies the content (the *node property*) reported in this column (using the dictionary given below). For example, %.8d uses a right-justified (.) column 8 characters wide to report temporary disk space per node in Mbyte (lowercase d).

The default column specifications for noncustomized SINFO node reports are:
-o "%9P %5a %.9l %.5D %6t %N"

See the SINFO example (page 61) section below for a sample default report and a sample column-customized report built using the -o option.

Available one-letter case-sensitive content specifiers (*Z*) for use in *%.wZ* field specifications are (in alphabetical order):

```
a   availability of a partition (up/down)
A   (uppercase) node count for (allocated/idle) states,
    see F
c   number of CPUs per node
d   temporary disk space (in Mbyte) per node
D   (uppercase) total number of nodes (per partition)
f   node features specified (if any)
F   (uppercase) node count for (allocated/idle/other/total)
    states, see A
g   groups allowed for listed nodes
h   (yes/no/force) nodes can be shared with other jobs?
l   time limit (days:hours:minutes:seconds or INFINITE)
m   memory size per node (in Mbyte)
N   (uppercase) list of node names
P   (uppercase) node partition name
r   (yes/no) only user root may start jobs?
R   (uppercase) reason if node unavailable
s   allowed nodes/job (range)
t   node state (abbreviated, see later section)
T   (uppercase) node state (spelled out)
w   scheduling weight (an integer)
```

SINFO Output Fields

SINFO reports are tables each column of which lists values for some node-related field or property. This section explains all the column heads ("output field" labels) that can possibly appear in an SINFO report (and, when not obvious from the column content, tells which SINFO option generates a report that includes the column in question). Option -h (--noheader) eliminates these column heads for easier reuse of SINFO's output by other programs.

AVAIL — is the current state of a SLURM-managed *partition* (either UP or DOWN). For the current state of the individual *nodes* or node subsets within a partition, see the STATE column (both AVAIL and STATE appear in SINFO's default report). See also REASON.

CPUS — reports the number of CPUs (processors) per node in node-oriented (rather than partition-oriented) output (such as from SINFO options -Nl).

FEATURES — lists the declared SLURM features of each node or node subset, if any (the default FEATURE is "null"), in node-oriented output (such as from SINFO options -Nl).

GROUPS — lists the groups allowed to use resources from each reported partition (the default is ALL groups). Use -l (--long).

JOB_SIZE — reports the minimum and maximum node count that can be allocated to any user job in a reported partition. A single number indicates that the minimun and maximum node count are the same, while INFINITE indicates no maximum node count. Use -l (--long).

MEMORY — is the size of real memory in megabytes on the reported nodes in node-oriented output (use -Nl).

NODELIST — lists the names (usually using a bracketed numerical range) of nodes associated with each reported partition (or instead, NODELIST appears first in node-oriented reports such as from -Nl).

NODES — is the count of nodes with each reported configuration (or partition).

NODES(A/I) — is the count of nodes with a requested configuration by node state in the form "available/idle." Use -o (--format) with the output specifier %A to request this column.

NODES(A/I/O/T) — is the count of nodes with a requested configuration by node state in the form "available/idle/other/total.' Use either -s (--summarize) or -o (--format) with the output specifier %F to request this column.

PARTITION — is the name of a SLURM-managed partition (where the suffix * identifies the default partition).

REASON shows the first 35 characters of the field optionally provided by each SLURM administrator to explain why a node's STATE is either DOWN or DRAINED. Use -R (--list-reasons) to get this column; use -Rl to get both REASON and STATE in the same SINFO report. The default REASON is "null."

ROOT reveals if the ability to allocate resources in a reported partition is restricted to the root user (YES or NO).

SHARE reveals if jobs that are allocated resources in a reported partition will share those resources. Use -l (--long). The possible values for SHARE are:

- NO indicates that resources are never shared.
- FORCE indicates that resources are always shared.
- YES indicates that resources may be shared if specific job or resource features otherwise allow.

STATE is the current state of an individual *node* or node subset within a SLURM-managed partition. For the current state of the whole partition, see the AVAIL column (both AVAIL and STATE appear in SINFO's default report). The next section (page 60) lists and explains the multiple values possible in the STATE column. See also REASON.

TIMELIMIT lists the maximum time limit for any user job in a reported partition in days:hours:minutes:seconds (INFINITE means no job time limit). TIMELIMIT appears in SINFO's default report.

TMP_DISK is the size of temporary disk space in megabytes on the reported nodes. Use -Nl to report.

WEIGHT is an integer representing the scheduling weight on each node (in a similar set). Use -Nl to report.

SINFO Node States

In SINFO reports, the strings below are the only possible values of the STATE column, indicating the current status of a node, a set of nodes, or a node partition. STATE codes with * appended indicate that a reported node is not responding (SLURM does not allocate new work to such nodes, which eventually enter the DOWN state).

The default width of the STATE column is 5 characters, but SINFO option -o (--format) can change this and the reported STATE codes will lengthen or shorten to fill the allowed spaces. SINFO option -t (--states) will limit your report to only those nodes with the (comma-delimited list of) state(s) that you specify.

A separate section covers SQUEUE's *job* state codes (page 52).

ALLOC[ATED] means that this node (or set, or partition) has already been assigned to one or more jobs.

COMP[LEATING] means that job(s) assigned to this node are already terminating. COMPLETING disappears when all of the job's processes as well as the SLURM epilog program (if any) have terminated. See the slurm.conf MAN page for details.

DOWN means that this node is unvailable for jobs. SLRUM automatically declares nodes DOWN if some failure occurs. Also, system administrators may declare a node DOWN. If a node resumes normal operation, SLRUM can automatically return it to service. See ReturnToService and SlurmdTimeout descriptions in the slurm.conf MAN page for more details.

DRAIN[ED] means that this node has been declared unavailable by a system administrator using SCONTROL's UPDATE command.

DRAINING [DRNG] means that this node is currently running a job, but it will not be allocated to additional jobs. The node state changes to DRAINED when the last job on it completes. System administrators put nodes in this state by using SCONTROL's UPDATE command.

IDLE means that this node is not currently assigned to any jobs and it available for use.

UNK[NOWN] means that the SLRUM controller has just started and hence this node's real status has not yet been determined.

SINFO Examples

[1]

GOAL: To display the default status report about all SLURM-managed nodes on the machine (cluster, here MCR) where you run SINFO.

STRATEGY: Run SINFO with no options. An six-column report (allegedly sorted by node state reported) appears. SINFO automatically ends. In this report, * appended to a partition name indicates the default partition, while * appended to a STATE value indicates that the node reported on that row is not currently responding. To report on only one partition of your choice, use SINFO's -p (--partition) option. To report on only one STATE, use the -t (--states) option. For an explanation of each possible column heading in an SINFO report, see the SINFO Output Fields section (page 58) above.

```
User: sinfo

Rtne:
  PARTITION AVAIL TIMELIMIT NODES  STATE NODELIST
  pdebug       up     30:00     4  drain mcr[100-103]
  pdebug       up     30:00    44  alloc mcr[40-71,73-84]
  pbatch*      up  infinite  1018  alloc mcr[104-145,152-758,761,
                                           763-798,807-814,819-926,
                                           928-999,1004-1059,1064-1151]
  pdebug       up     30:00    16   idle mcr[72,85-99]
  pbatch*      up  infinite     1 drain* mcr1003
  pbatch*      up  infinite    29   idle mcr[146-151,759-760,762,
                                           799-806,815-818,927,1000-1002,
                                           1060-1063]
```

---

[2]

GOAL: To build a customized status report about specific nodes on a SLURM-managed cluster (here, MCR), for example, showing only CPUs/node, temporary disk space per node, and allowed nodes/job.

STRATEGY: (1) Specify which nodes you want reported by using SINFO's -n (--nodes) option (which here selects MCR nodes from 1 to 500 inclusive).
(2) Use SINFO's -o (lowercase oh, --format) option to specify which specific columns (node properties) you want to report, the width of each column in characters, and the order for the columns to appear (left to right). SINFO's general column-specification language is described with -o in the options section (page 54) above. Here, %15N requests a 15-character node-list column, %.6c requests a 6-character right-justified CPUs/node column, %.8d requests an 8-character right-justified disk-space column, and %.12s requests a 12-character right-justified nodes/job column. For an explanation of each possible column heading in an SINFO report, see the SINFO Output Fields section (page 58) above.

```
User: sinfo  -n "mcr[1-500]" -o "%15N %.6c %.8d %.12s"

Rtne:
  NODELIST          CPUS TMP_DISK     JOB_SIZE
  mcr[40-103]          2        1         1-16
  mcr[104-500]         2        1   1-infinite
```

SMAP (Show Job Geometry)

ROLE.

On BlueGene/L *only*, the SMAP utilty reveals not only which nodes are allocated to currently running jobs but also the *geometric arrangement* of those nodes (and hence, the way that BG/L jobs fit among one another topographically). On BG/L, SMAP thus supplements SINFO (page 53) and SQUEUE (page 46) as a visually enhanced way to monitor job interactions and to plan spatially for new node allocations.

PREREQUISITES.

(1) SMAP runs only on LC's BlueGene/L (BG/L) machine.
(2) SMAP takes over the terminal window in which it runs. So executing it as a controllee of XTERM, in a separate window dedicated to its output, is a good strategy (see below).
(3) SMAP needs a window wider than 80 characters to display its character-based "map" of job/node allocations effectively. Requesting a 100-character-wide window with XTERM's -geometry option is a good strategy (see below).

EXECUTION.

Because of its prerequisites (above), a typical appropriate SMAP run could begin with an execute line such as this

```
xterm -geometry 100x30 -e /usr/bin/smap -Dj >& /dev/null &
```

to show the arrangement of jobs currently running (-Dj) on BG/L.

TYPICAL OUTPUT.

SMAP's character-based map of job/node allocations typically looks like this (some blank columns have been trimmed to fit this manual):

```
   a a a a b b d d ID JOBID PARTITION USER   NAME ST TIME NODES NODELIST
  a a a a b b d d  a  12345 batch     joseph tst1 R  43:12   64 bgl[000x333]
 a a a a b b c c   b  12346 debug     chris  sim3 R  12:34   16 bgl[420x533]
a a a a b b c c    c  12350 debug     danny  job3 R   0:12    8 bgl[622x733]
                   d  12356 debug     dan    colu R  18:05   16 bgl[600x731]
   a a a a b b d d e  12378 debug     joseph asx4 R   0:34    4 bgl[612x713]
  a a a a b b d d
 a a a a b b c c
a a a a b b c c

   a a a a . . d d
  a a a a . . d d
 a a a a . . e e        Y
a a a a . . e e         |
                        |
   a a a a . . d d      0----X
  a a a a . . d d      /
 a a a a . . . .      /
a a a a . . . #      Z
```

Note here that:

(1) As the legend indicates, the origin (node 000) lies at the REAR (upper, not lower) left corner of the bottom "plane" (4-line group, and each such 4-line group shows another Y-dimension plane above it).

(2) Unassigned (idle) BG/L base partitions ("nodes" to SLURM) are shown as a period (.) in the map.
(3) Down/drained base partitions (unavailable for use) are shown as a pound sign (#). In this example, only BGL703 is down.

SCONTROL (Manage Configurations)

ROLE.

SCONTROL is the SLURM utility that manages SLURM's own configuration, including the properties that it assigns to nodes, node partitions, and other SLURM-controlled system features. Most SCONTROL options and commands are intended for, and can only be successfully executed by, a system administrator (a privileged or root user). Some SCONTROL commands *report* useful configuration information or manage job *checkpoints*, however, and any user can benefit from invoking them appropriately. The rest of this section discusses only those few general-user SCONTROL commands.

EXECUTE LINE.

Run SCONTROL by typing

`scontrol` *`command argument`*

where the general-user commands and their specific arguments are summarized below (most commands are for system administrators only). Without any command, SCONTROL prompts for input.

GENERAL-USER COMMANDS.

The subset of SCONTROL commands that any user can invoke includes these:

checkpoint *action jobid[.stepid]*

requests one of several allowed checkpoint activities on those LC AIX machines that also use SLURM (instead of IBM's LoadLeveler). You must (previously) set environment variable CHECKPOINT to YES and specify a location and name for future checkpoint files. See the "Checkpointing with SLURM and POE" section (URL: http://www.llnl.gov/LCdocs/dpcs/index.jsp?show=s7.5) of the LCRM Reference Manual for details, especially for LCRM batch jobs. Here:

action specifies what to do *after* the requested checkpoint occurs, where the three most useful alternatives are:

- create — requests a checkpoint and *continues* the job(step) after it occurs.
- vacate — requests a checkpoint and *terminates* the job(step) after it occurs.
- restart — resumes execution of a previously checkpointed job(step).

jobid[.stepid] specifies the range for the checkpointing activity, which can be all existing steps for a specified *jobid* alone (e.g., 4812), or the individual job step indicated by a *jobid.stepid* combination (e.g., 4812.4).

exit (or QUIT) terminates SCONTROL during interactive sessions.

show *entity id* displays the current state of the SLURM-managed item that you specify, where

entity can be any of these alternative literal strings:
config [see "Scheduler Types" above (page 15)]
daemons
job
node
partition
step

id specifies which individual entity to report (for example, by providing a node name (e.g., mcr123), a partition name (e.g., pdebug), or a job ID number (e.g., 1428)).

Disclaimer

Keyword Index

To see an alphabetical list of keywords for this document, consult the next section (page 70).

Keyword	Description
entire	This entire document.
title	The name of this document.
scope	Topics covered in this document.
availability	Where SLURM runs.
who	Who to contact for assistance.
introduction	Overview of SLURM features, comparisons.
slurm-strategy	Special benefits built into SLURM.
slurm-goals	SLURM design goals as resource manager.
slurm-roles	RMS, TBS, LCRM, SLURM compared.
slurm-systems	How SLURM interacts with oper. systems.
slurm-features	How SLURM works.
slurm-components	Software units comprising SLURM.
slurmctld	SLURMCTLD control daemon explained.
slurmd	SLURMD local daemon explained.
portability	How plugin modules make SLURM adaptable.
user-impact	SLURM's effect on typical jobs.
scheduler-types	SLURM's three alternative job schedulers.
slurm-operation	User interaction with SLURM.
slurm-utilities	SLURM's direct user utility programs.
srun	Job-submittal and resource utility.
srun-roles	SRUN roles and modes compared.
poe-comparison	SRUN and (IBM) POE differences.
run-mode-options	Enabling different job-run alternatives.
resource-allocation	Assigning compute resources to jobs.
control-options	Managing general job features.
node-management	Spreading tasks, sharing nodes.
working-features	Verbosity, job name, path options.
resource-control	Waiting for resources, overcommitting.
help-options	Requesting syntax summaries.
prolog-options	Job and task prologs, epilogs.
debug-options	Extra controls for root users.
i-o-options	Handling job input, output, errors.
i-o-commands	Input, output, error controls.
i-o-alternatives	Five ways to redirect I/O.
constraint-options	Specifying job constraints.
general-constraints	Constraints for any job.
numa-constraints	CPU and NUMA constraints only.
environment-variables	SRUN environment variables defined.
multi-prog-usage	Configuration example for SRUN --multi-prog.
squeue	Job reporting, monitoring utility.
squeue-execute-line	How to run SQUEUE.
squeue-options	Controlling, customizing SQUEUE output.
squeue-examples	Standard and customized SQUEUE reports.
job-states	SQUEUE job state (status) codes.

sinfo	Node status/property reporting utility.
sinfo-execute-line	How to run SINFO.
sinfo-options	Controlling, customizing SINFO output.
sinfo-output-fields	Column heads in SINFO reports explained.
node-states	SINFO node state (status) codes.
sinfo-examples	Standard and customized SINFO reports.
smap	Job geometry utility (BlueGene/L only).
scontrol	Sys admin configuration utility.
index	The structural index of keywords.
a	The alphabetical index of keywords.
date	The latest changes to this document.
revisions	The complete revision history.

Alphabetical List of Keywords

Keyword	Description
a	The alphabetical index of keywords.
availability	Where SLURM runs.
constraint-options	Specifying job constraints.
control-options	Managing general job features.
date	The latest changes to this document.
debug-options	Extra controls for root users.
entire	This entire document.
environment-variables	SRUN environment variables defined.
general-constraints	Constraints for any job.
help-options	Requesting syntax summaries.
i-o-alternatives	Five ways to redirect I/O.
i-o-commands	Input, output, error controls.
i-o-options	Handling job input, output, errors.
index	The structural index of keywords.
introduction	Overview of SLURM features, comparisons.
job-states	SQUEUE job state (status) codes.
multi-prog-usage	Configuration example for SRUN --multi-prog.
node-management	Spreading tasks, sharing nodes.
node-states	SINFO node state (status) codes.
numa-constraints	CPU and NUMA constraints only.
poe-comparison	SRUN and (IBM) POE differences.
portability	How plugin modules make SLURM adaptable.
prolog-options	Job and task prologs, epilogs.
resource-allocation	Assigning compute resources to jobs.
resource-control	Waiting for resources, overcommitting.
revisions	The complete revision history.
run-mode-options	Enabling different job-run alternatives.
scheduler-types	SLURM's three alternative job schedulers.
scontrol	Sys admin configuration utility.
scope	Topics covered in this document.
sinfo	Node status/property reporting utility.
sinfo-examples	Standard and customized SINFO reports.
sinfo-execute-line	How to run SINFO.
sinfo-options	Controlling, customizing SINFO output.
sinfo-output-fields	Column heads in SINFO reports explained.
slurm-components	Software units comprising SLURM.
slurm-features	How SLURM works.
slurm-goals	SLURM design goals as resource manager.
slurm-operation	User interaction with SLURM.
slurm-roles	RMS, TBS, LCRM, SLURM compared.
slurm-strategy	Special benefits built into SLURM.
slurm-systems	How SLURM interacts with oper. systems.
slurm-utilities	SLURM's direct user utility programs.
slurmctld	SLURMCTLD control daemon explained.
slurmd	SLURMD local daemon explained.
smap	Job geometry utility (BlueGene/L only).
squeue	Job reporting, monitoring utility.
squeue-examples	Standard and customized SQUEUE reports.
squeue-execute-line	How to run SQUEUE.
squeue-options	Controlling, customizing SQUEUE output.
srun	Job-submittal and resource utility.
srun-roles	SRUN roles and modes compared.
title	The name of this document.

user-impact	SLURM's effect on typical jobs.
who	Who to contact for assistance.
working-features	Verbosity, job name, path options.

Date and Revisions

Revision Date	Keyword Affected	Description of Change
12Sep06	slurm-systems	Operating system comparison section added.
	prolog-options	SRUN prolog/epilog section added.
	debug-options	SRUN root-user special options added.
	numa-constraints	SRUN CPU and NUMA constraints added.
	multi-prog-usage	SRUN tips on multiple programs added.
	srun	Many options added, details updated.
	environment-variables	More details, option/variable table added.
	index	New keywords for 5 new sections.
09Mar06	scontrol	New section on admin utility.
	index	New keyword for new section.
	slurm-utilities	Link to new section added.
	control-options	Cross ref added to SCONTROL role.
24Jan06	environment-variables	SLURM_NETWORK added. Cross ref. added for Env. Var. manual.
11Apr05	slurm-roles	TBS replaces NQS.
	slurmctld	No job steps on BG/L.
	scheduler-types	No backfill scheduling on BG/L.
	slurm-utilities	SMAP for BG/L added.
	srun-roles	MPIRUN needed in BG/L scripts.
	run-mode-options	MPIRUN needed in BG/L scripts.
	node-management	Four SRUN BG/L-only options added.
	squeue	No job steps on BG/L.
	smap	New section on BG/L tool.
	index	New keyword for new section.
08Nov04	sinfo	New sections on node reporting tool.
	index	New keywords for new sections.
	squeue	Cross references to SINFO added.
	user-impact	SQUEUE, SINFO compared.
	slurm-strategy	LCRM replaces DPCS (throughout).
	slurm-utilities	Link added, SINFO details expanded.
14Oct04	scheduler-types	Three alternative schedulers compared.
	index	New keyword for new section.
10Aug04	job-states	SQUEUE job-state codes explained.
	squeue-options	Links to job states added.

	index	New keyword for new section.
18May04	squeue	New sections on monitoring tool.
	index	New keywords for new sections.
17Mar04	control-options	14 SRUN options added in 4 subsections.
	i-o-options	5 SRUN options for I/O redirection explained.
	constraint-options	8 node-constraint options added.
	environment-variables	2 more env. variables explained.
	index	6 new keywords for new sections.
21Oct03	srun	Major SRUN features, options explained.
	introduction	SRUN's central role introduced.
	index	Nine new keywords for new sections.
20Aug03	entire	Draft edition of SLURM manual.

TRG (12Sep06)