

Submit Command

Overview

submit takes a user command and executes it remotely. The objective is to allow the user to issue a command in the same manner as a locally executed command. Multiple submission mechanisms are available for run dissemination. A set of steps are executed for each run submission:

- Destination site is selected
- A wrapper script is generated for remote execution
- If needed a batch system description file is generated.
- Input files for a run are gathered and transferred to the remote site. Transferred files include the wrapper and batch description scripts.
- The wrapper script is executed remotely.
- Progress of the remote run is monitored until completion.
- Output files from the run are returned to the dissemination point.

Command Syntax

submit command options can be determined by using the help parameter of the submit command.

```
$ submit --help
Usage: submit [options]
```

Options:

-h, --help	Report command usage. Optionally request listing of
	managers, tools, venues, or examples.
-l, --local	Execute command locally
--status	Report status for runs executing remotely.
-k, --kill	Kill runs executing remotely.
--venueStatus	Report venue status.
-v, --venue	Remote job destination
-i, --inputfile	Input file
-p, --parameters	Parameter sweep variables. See examples.
-d, --data	Parametric variable data - csv format
-s SEPARATOR, --separator=SEPARATOR	Parameter sweep variable list separator
-n NCPUS, --nCpus=NCPUS	Number of processors for MPI execution
-N PPN, --ppn=PPN	Number of processors/node for MPI execution

SUBMIT COMMAND

```
--stripes=NSTRIPES      Number of parallel local jobs when doing param
etric
                        sweep
-w WALLTIME, --wallTime=WALLTIME
                        Estimated walltime hh:mm:ss or minutes
-e, --env                Variable=value
--runName=RUNNAME        Name used for directories and files created du
ring the
                        run. Restricted to alphanumeric characters
-m, --manager            Multiprocessor job manager
-r NREDUNDANT, --redundancy=NREDUNDANT
                        Number of identical simulations to execute in
parallel
-M, --metrics            Report resource usage on exit
--detach                Detach client after launching run
--attach=ATTACHID        Attach to previously detached started server
-W, --wait               Wait for reduced job load before submission
-Q, --quota              Enforce local user quota on remote execution h
ost
-q, --noquota            Do not enforce local user quota on remote exec
ution
                        host
--tailStdout             Periodically report tail of stdout file.
--tailStderr             Periodically report tail of stderr file.
--tail                  Periodically report tail of application file.
--progress               Show progress method. Choices are auto, curses
,
                        submit, text, pegasus, or silent.
--asynchronous           Asynchronous simulation - results will not be
returned
```

Additional information is available by requesting user specific lists of choices for some command options. The available option lists are generated for a user based on configured restrictions and availability. The values listed here are for example only and may not be available on all HUBs.

```
$ submit --help tools
```

Currently available TOOLS are:

```
lammmps-03Mar20-parallel
lammmps-03Mar20-serial
lammmps-05Jun19-parallel
lammmps-05Jun19-serial
```

SUBMIT COMMAND

```
lammmps-11Aug17-parallel
lammmps-11Aug17-serial
lammmps-22Aug18-parallel
lammmps-22Aug18-serial
lammmps-31Mar17-parallel
lammmps-31Mar17-serial
```

```
$ submit --help venues
```

Currently available VENUES are:

```
OSG
brown
datalimited@brown
ncn-hub@brown
standby@brown
```

```
$ submit --help managers
```

Currently available MANAGERS are:

```
lammmps-03Mar20_mpi
lammmps-03Mar20_serial
lammmps-05Jun19_mpi
lammmps-05Jun19_serial
lammmps-11Aug17_mpi
lammmps-11Aug17_serial
lammmps-22Aug18_mpi
lammmps-22Aug18_serial
lammmps-31Mar17_mpi
lammmps-31Mar17_serial
mpi
mpich
mpirun
parallel
serial
```

Examples of how to use the submit command to execute parameter sweeps are provided by asking for help on examples.

```
$ submit --help examples
Usage: submit [options]
```

Options:

```
-h, --help          Report command usage. Optionally request listi
```

SUBMIT COMMAND

ng of

	managers, tools, venues, or examples.
-l, --local	Execute command locally
--status	Report status for runs executing remotely.
-k, --kill	Kill runs executing remotely.
--venueStatus	Report venue status.
-v, --venue	Remote job destination
-i, --inputfile	Input file
-p, --parameters	Parameter sweep variables. See examples.
-d, --data	Parametric variable data - csv format
-s SEPARATOR, --separator=SEPARATOR	Parameter sweep variable list separator
-n NCPUS, --nCpus=NCPUS	Number of processors for MPI execution
-N PPN, --ppn=PPN	Number of processors/node for MPI execution
--stripes=NSTRIPES	Number of parallel local jobs when doing parametric sweep
-w WALLTIME, --wallTime=WALLTIME	Estimated walltime hh:mm:ss or minutes
-e, --env	Variable=value
--runName=RUNNAME	Name used for directories and files created during the run. Restricted to alphanumeric characters
-m, --manager	Multiprocessor job manager
-r NREDUNDANT, --redundancy=NREDUNDANT	Number of identical simulations to execute in parallel
-M, --metrics	Report resource usage on exit
--detach	Detach client after launching run
--attach=ATTACHID	Attach to previously detached started server
-W, --wait	Wait for reduced job load before submission
-Q, --quota	Enforce local user quota on remote execution host
-q, --noquota	Do not enforce local user quota on remote execution host
--tailStdout	Periodically report tail of stdout file.
--tailStderr	Periodically report tail of stderr file.
--tail	Periodically report tail of application file.
--progress	Show progress method. Choices are auto, curses, submit, text, pegasus, or silent.
--asynchronous	Asynchronous simulation - results will not be returned

SUBMIT COMMAND

Parameter examples:

```
submit -p @@cap=10pf,100pf,1uf sim.exe @:indeck
```

Submit 3 jobs. The @:indeck means "use the file indeck as a template file." Substitute the values 10pf, 100pf, and 1uf in place of @@cap within the file. Send off one job for each of the values and bring back the results.

```
submit -p @@vth=0:0.2:5 -p @@cap=10pf,100pf,1uf sim.exe @:indeck
```

Submit 78 jobs. The parameter @@vth goes from 0 to 5 in steps of 0.2, so there are 26 values for @@vth. For each of those values, the parameter @@cap changes from 10pf to 100pf to 1uf. $26 \times 3 = 78$ jobs total. Again @:indeck is treated as a template, and the values are substituted in place of @@vth and @@cap in that file.

```
submit -p params sim.exe @:indeck
```

In this case, parameter definitions are taken from the file named params instead of the command line. The file might have the following contents:

```
# paramters for my job submission
parameter @@vth=0:0.2:5
parameter @@cap = 10pf,100pf,1uf
```

```
submit -p "params;@@num=1-10;@@color=blue" job.sh @:job.data
```

For someone who loves syntax and complexity... The semicolon separates the parameters value into three parts. The first says to load parameters from a file params. The next part says add an additional parameter @@num that goes from 1 to 10. The last part says add an additional parameter @@color with a single value blue. The parameters @@num and @@color cannot override anything defined within params; they must be new parameter names.

SUBMIT COMMAND

```
submit -d input.csv sim.exe @:indeck
```

Takes parameters from the data file input.csv, which must be in comma-separated value format. The first line of this file may contain a series of @@param names for each of the columns. Whitespace is significant for all values entered in the csv file. If it doesn't, then the columns are assumed to be called @@1, @@2, @@3, etc. Each of the remaining lines represents a set of parameter values for one job; if there are 100 such lines, there will be 100 jobs. For example, the file input.csv might look like this:

```
@@vth,@@cap
1.1,1pf
2.2,1pf
1.1,10pf
2.2,10pf
```

Parameters are substituted as before into template files such as @:indeck.

```
submit -d input.csv -p "@@doping=1e15-1e17 in 30 log" sim.exe @:infile
```

Takes parameters from the data file input.csv, but also adds another parameter @@doping which goes from 1e15 to 1e17 in 30 points on a log scale. For each of these points, all values in the data file will be executed. If the data file specifies 50 jobs, then this command would run $30 \times 50 = 1500$ jobs.

```
submit -d input.csv -i @:extra/data.txt sim.exe @:indeck
```

In addition to the template indeck file, send along another file extra/data.txt with each job, and treat it as a template too.

```
submit -s / -p @@address=23 Main St.,Hometown,Indiana/42
Broadway,Hometown,Indiana -s , -p @@color=red,green,blue job.sh @:job.
```

SUBMIT COMMAND

data

Change the separator to slash when defining the addresses, then change back to comma for the @@color parameter and any remaining arguments. We shouldn't have to change the separator often, but it might come in handy if the value strings themselves have commas.

```
submit -p @@num=1:1000 sim.exe input@@num
```

Submit jobs 1,2,3,...,1000. Parameter names such as @@num are recognized not only in template files, but also for arguments on the command line. In this case, the numbers 1,2,3,...,1000 are substituted into the file name, so the various jobs take their input from "input1", "input2", ..
..
"input1000".

```
submit -p @@file=glob:indeck* sim.exe @@file
```

Look for files matching indeck* and use the list of names as the parameter @@file. Those values could be substituted into other template files, or used on the command line as in this example. Suppose the directory contains files indeck1, indeck10, and indeck2. The glob option will order the files in a natural order: indeck1, indeck2, indeck10. This example would launch three jobs using each of those files as input for the job.

```
submit -p @@file=globnat:indeck* sim.exe @@file
```

This option has been deprecated. The functionality is now available with the glob option.

By specifying a suitable set of command line parameters it is possible to execute commands on configured remote systems. The simple premise is that a typical command line can be prefaced

SUBMIT COMMAND

by submit and its arguments to execute the command remotely.

```
$ submit -v clusterA echo Hello world!  
Hello world!
```

In this example the echo command is executed on the venue named clusterA where runs are executed directly on the host. Execution of the same command on a cluster using a batch scheduler such as SLURM would be done in a similar fashion

```
$ submit -v clusterB echo Hello world!  
(2586337) Simulation Queued Wed Oct  7 14:45:21 2009  
(2586337) Simulation Done Wed Oct  7 14:54:36 2009  
$ cat 00577296.stdout  
Hello world!
```

submit supports an extensible variety of submission mechanisms. HUBzero supported submission mechanisms are

- local - use batch submission mechanisms available directly on the submit host. These include condorHT, and Pegasus batch queue submission.
- ssh - direct use of ssh. Submit manages access to a venue using a common ssh key, essentially serving as a proxy for the HUB user.
- ssh + remote batch submission - use ssh to do batch run submission remotely. Again methods for common batch schedulers PBS, condorHT, Pegasus, and SLURM are provided. Additional interfaces to SGE, Load Leveler, BOINC, LSF, and Tapis are also available.

In addition to single site submission the `-r/--redundancy` option provides the option to simultaneously submit runs to multiple remote venues. In such cases the successful completion of a run at one venue cancels runs at all other venues. If none of the runs are successful results from one of the runs are returned to the user. Redundant submission is not allowed when performing parametric sweeps.

A venue for remote execution is selected in one of the following ways, listed in order of precedence:

- Execute the command within the user tool session, `-l/--local` option
- User specified on the command line with `-v/--venue` option.
- Randomly selected from remote sites associated with pre-staged application.
- Select randomly from all configured sites

Venues that do not meet the resource requirements of the run request are not considered.

SUBMIT COMMAND

Venues are typically configured with limits on the number of cores, walltime, or core-hours.

Any files specified by the user plus internally generated scripts are packed into a tarball for delivery to the remote site. Individual files or entire directory trees may be listed as command inputs using the `-i/--inputfile` option. In addition, command arguments that exist as files or directories will be packed into the tarball. If using ssh based submission mechanisms the tarball is transferred using scp.

The job wrapper script is executed remotely either directly or submitted to a batch queue. The job is subject to all remote queuing restrictions and idiosyncrasies.

Remote batch jobs are monitored for progress. Methods appropriate to the batch queuing system are used to check job status at a configurable frequency. A typical frequency is on the order one minute. Job status changes are reported to the user. The maximum time between reports to the user is set on the order of five minutes even in the absence of change. The job status is used to detect job completion.

The same methods used to transfer input files are applied in reverse to retrieve output files. Any files and directories created or modified by the application are be retrieved. A tarball is retrieved and expanded to the home base directory. It is up to the user to avoid the overwriting of files.

In addition to the application generated output files additional files are generated in the course of remote run execution. Some of these files are for internal bookkeeping and are consumed by submit, a few files however remain in the home base directory. The remaining files include `RUNID.stdout` and `RUNID.stderr`, it is also possible that a second set of standard output/error files will exist containing the output from the batch job submission script. By default, `RUNID` represents unique job identifier assigned by submit. If preferred a user can specify a different `RUNID` using the `--runName` command argument.