tq Cookbook

There are innumerable ways in which the tq command-line tool can be used to query and manage the state of the farm. Operations can be performed as precisely *and* far-reaching as desired, thanks to the powerful search clause syntax.

This section provides a number of scenarios where tq can be useful.

Job Queue

Queue Size

There are a number of ways to assess the amount of remaining work to be done. The following command will list all of the tasks that have yet to be scheduled:

```
% tq tasks blocked or ready
```

Here the blocked and ready aliases expand to state=blocked and state=ready respectively.

wc can be used to count them. --noheader or --nh is used so that the headers are not included in the count.

```
% tq tasks --noheader blocked or ready | wc -l
```

For some sites, result set may be truncated due to a configuration limit in the database. This is to protect the engine, database server, and client from stresses caused by memory, cpu and bandwidth utilization of extremely large result sets. This limit can be overridden, or more cautiously, extended using the``--limit`` flag.

```
% tq tasks --noheader blocked or ready --limit 100000 | wc -l
```

Here we have limited the result set to 100,000 records. For some queues, this may not be large enough. The limit could be set higher, but for this particular class of queries, we can use the task state counters which are maintained in the job. The following commands list the number of blocked and ready tasks for jobs that have them.

```
% tq jobs --nh blocked or ready -c numblocked,numready
```

With a bit of option and shell-fu, these numbers can be summed by piping the result to paste -sd+ | bc. --nocolor ensures that special characters don't confuse bc. --zeros causes zero values to be shown as 0 instead of blank, which is required when joining valuels with + to form an arithmetic expression for bc. --delimiter + causes the blocked and ready counters to be separated with a +, while paste -sd+ joins lines together with a +.

```
% tq --nocolor --zeros jobs --noheader --delimiter + -c numready,numblocked ready or blocked | paste -sd+ | bc
```

Note that certain options appear before the jobs subcommand and others appear after. Option location can be verified by using the built-in help.

```
% tq help usage
% tq help jobs
```

Continue to be aware that result set sizes can be automatically truncated, so it may be necessary to use the --limit option to ensure an accurate calculation.

Undeleting a Job

A deleted job can be brought back from the archives. The first step is to issue an undelete command.

```
% tq --archives jobs undelete jid=1234
```

Note that the --archives flag *must* be used since you can only undelete a deleted job, and deleted jobs only exist in the archives.

To have the job run, the job must be restarted or specific tasks be retried.

```
% tq restart jid=1234
```

Here the --archives flag is not required because the job is no longer considered deleted.

Resource Utilization

Farm Hogs

The following command finds all active jobs, and reverse sorts them by number of active tasks so that busiest jobs are listed first. The alias active is used, which equates to numactive > 0. The limit option is used here to show the top 5 busiest jobs.

```
% tq jobs active --sort -numactive --limit 5

       jid user     title                         pri blkd redy actv err done spooled
========== ======== ========================== ==== ==== ==== ==== === ==== ===========
      2658 adamwg   turntable pollo               9   10    2  158         1 02/06|08:44
        32 ratbuild feather sim x23_1            10         5   92        92 01/31|16:38
      2659 adamwg   turntable ranch               9             67         1 02/06|08:44
      2663 adamwg   turntable muffin              25             60         1 02/11|11:50
       380 rdavis   mountain waterbottle sim x2  100             58         2 01/31|16:46
```

Resource Hoarders

Sometimes there are special computing resources that are in limited supply, and you may be interested in getting a breakdown of which users are using those resources.

Suppose there are high-performance blades that can do GPU rendering, and those blades have a special profile name "gpu". You can list all of the blades with that profile as follows:

```
% tq blades profile=gpu
```

The following query will list all of the active tasks:

```
% tq tasks state=active
```

To get only the tasks that are running on those GPU blades:

```
% tq tasks state=active and profile=gpu
```

(Note: if you forget the state=active part, you'll end up getting all tasks that ever ran on gpu blades.)

tq does not have built-in aggregation functions, but you can readily perform such aggregation on the command-line using uniq -c (on Linux and macOS). The following command matches the same list of tasks as above, but only displays the task's owner. --noheaderavoids the heading lines from being counted, --columns owner causes only the owner attribute to be displayed on each line, and --sortby owner sorts the results by owner for uniq -c to function properly.

```
% tq tasks --noheader -state=active and profile=gpu -c owner -s owner | uniq -c
   398 ringo
    10 john
     2 george
     1 paul
```

Now that you can see who is using the gpu blades, you can take any appropriate action such as reprioritizing jobs, pausing jobs, retrying tasks, or investigating whether certain tasks are running longer than expected.

Command History

Tractor tracks the execution of every command on the farm, including the blade name, start time, stop time, and memory and cpu utilization of the process. Each execution is represented as distinct *invocation*. The invocation is uniquely identified by three attributes: the *jid*, *tid*, and *iid*. The *iid* is the invocation id and starts at 1.

Find Retried Tasks

The following command finds all invocations that are the result of a retry for a specific job. The retry could have been initiated by a user, a script through an API, or automatically by the engine. Double quotes are used around the search string to avoid having the > sign interpreted by the shell. The argv value is too long to display in the default column width, so ... has been used to shorten it.

```
% tq invocations "jid=52 and iid > 1"
       jid  tid  cid  iid blade      starttime   stoptime    rcode argv
========== ==== ==== ==== ========== =========== =========== ===== ======================================
        52    1    1    2 c2221-8002 01/31|16:40 01/31|16:40     1 expandtest.py --exp...-time=100 --random
        52    4    4    2 c1929-8000 01/31|16:40 01/31|16:40     1 expandtest.py --exp...-time=100 --random
```

Resource Utilization of an Invocation

The following command lists the id, job owner, blade, timestamps, and resource utilization for all commands that used more than 4G of memory. Note how the precision of the rss attribute can be specified using the =6.3 formatter in the columns option; 6 specifies the column width, while 3 specifies the precision. The job

owner has also been added, without having to prefix the attribute with the entity name (e.g. Job.owner).

```
% tq invocations 'rss>4G' -c +rss=6.3,+vsz=6.3,+cpu=6.3,+owner
       jid  tid  cid  iid blade      starttime   stoptime    rcode
argv                                      rss    vsz    cpu owner
========== ==== ==== ==== ========== =========== =========== ===== ======================================
====== ====== ====== ======
      2638    1    1    1 c1269-8000 02/04|11:02 02/04|11:03     0 sleep 10
8.891  0.000  0.000 adamwg
      2639    1    1    1 c1269-8000 02/04|11:06 02/04|11:07     0 sleep 30
8.906  0.000  0.000 adamwg
      2674    1    1    1 c1260-8000 02/11|22:00 02/11|22:00     0 /usr/bin/bc
8.812  0.000  0.000 adamwg
      2675    1    1    1 c1260-8000 02/17|08:01 02/17|08:01     0 sleep 1
4.680  0.000  0.000 adamwg
      2683    2    2    1 c1260-8000 02/28|22:45 02/28|22:45     0 sleep 2
4.941  0.000  0.000 adamwg
```

Farm Management

Draining the Farm

Sometimes all or part of the farm needs to be shutdown in to perform such administrative activities as hardware maintenance or software upgrades. Suppose a site needs to upgrade the power supplies for racks 3 and 6, and that hosts in these racks have names that start with r3 and r6 respectively.

To drain the these racks of their running tasks, we would first prevent the blades from requesting more working using the nimby command as follows:

```
% tq nimby "name like ^r3 or name like ^r6"
```

Admins could use the following command to poll for tasks that are actively running on those blades. When the command returns no results, they know they can safely bring down the blades.

```
% tq tasks "active and (blade like ^r3 or blade like ^r6)"
```

Rather than wait for the tasks to finish, admins may choose to forcefully retry the tasks running on those blades.

```
% tq retry "active and (blade like ^r3 or blade like ^r6)"
```

The prior tq tasks command could be used to poll the system ensure that there are no remaining active tasks on those racks and can thus proceed with maintenance.

Once maintenance has been completed, the blades can be brought back online with a similar command as the one used to take them offline:

```
% tq unnimby "name like ^r3 or name like ^r6"
```

Some regular expression enthusiasts may be quick to point out that ^r[36] should match racks 3 and 6, which is correct. However, the search clause syntax requires expressions with square brackets be enclosed in quotes so that they are not interpreted as a demarkation of a search clause list. Thus, the above command (and with similarity, others) could be written like this:

```
% tq unnimby "name like '^r[36]'"
```