TreeDelete+

**A script to manage multiple TreeDelete jobs**

**Adam Fox, Isilon CAE (**[adam.fox@dell.com](mailto:adam.fox@dell.com)**)**

# **Introduction:**

The thought behind TreeDelete+ is to handle complex datasets with the TreeDelete job. By tuning the parameters of the script you can keep jobs running and queued up to get more out of the jobs. It also helps discover trees that should be deleted and gives the user finer control over what should be deleted and what should not.

# Syntax:

The general syntax of the script is as follows:

Usage: treedelete+.py -r <path> -i <file> [-p #] [-j #] [-I <policy>] [-P #] [-v] [-T] [-w #] [-h]

-r <path> | --root=<path> : Specifies a path to scan for directories

-i <file> | --input=<file> : Specifies a file of paths to use or exclude

-p # | --paths=# : Specifies the # of paths in each job (def: 1)

-j # | --jobs=# : Specifies the # of jobs in the queue at once (def: 1)

-I <policy> | --impact=<policy> : Specifies the impact policy of the jobs (def: MEDIUM)

-P # | --priority=# : Specifies the priority of the jobs (def: 4)

-v | --verbose : Turns on verbose mode (shows the commands (def: off

-T | --TEST : Turns on test mode. Only show commands, does not execute

-w # | --waittime=# : Time in seconds to wait between queue checks (def: 30

-h | --help : Prints Usage Message

# Theory of Operation

The idea behind the script is to allow the user to discover and configure what paths should be deleted as well as keep the TreeDelete jobs running as much as possible. This section will address each of these goals separately.

## Building a Path List

A path can be built manually or automatically depending on the use case. The script can take a simple text file with a list of paths (one per line) and read that file with the –i or –input= flag. That method will work down the list until all paths have been deleted. However, the script can auto-discover paths by using the –r or –root= option. This flag allows the user to specify a root path. That script read in the contents of the top level of that path and any of the entries which are directories will be automatically added to the list of paths to be deleted. Anything in the top level of the root path that is not a directory will be ignored.

The two methods can be used together. The user can specify a root path and an input file with additional paths to delete that are not in the root tree. Additionally, the input file can be used to exclude paths that are found in the root path. This is done by putting a – directly in front of the path name (e.g. -/ifs/data/project1/save\_me). Both include and exclude paths can be in the same file although excluding a path only makes sense with when a root directory is specified.

## Optimizing the jobs

Once a path list has been built, the next part of the script is to manage the TreeDelete jobs. This is done primarily with the –p (or--paths=), -j (or --jobs), and –w (or --waittime) flags.

The paths flag specifies how many paths each job should include. There is no set limit to the number of paths a job can take, although as many as 60 have been tested. The default is 1 path per job

The jobs flag tells the script how many jobs to keep in the run queue. For the purposes of the script the run queue is the number of TreeDelete jobs running and waiting in the queue. There is a limit of 30 jobs in the queue, although it is not recommended to fill the queue as other OneFS jobs may need to run and having some slots for them is good for the system. Keep in mind, having a bunch of jobs queued up doesn’t necessarily make things run faster. The goal should be to never have zero jobs running. As long as the script is reporting at least one job in the queue, everything is working well. The default is 1 job in the queue at a time.

The waittime flag tells the script how long to wait in seconds before checking the queue of TreeDelete jobs. During this wait time, no new jobs will be started. This not only saves resources on the nodes, but avoids checking constantly when no new action is needed. The default is 30 seconds.

There is no perfect formula for where these flags should be set. It will be entirely dependent on the dataset as there is no way to predict how long a TreeDelete job will run. Again, the goal should be to keep the queue above zero for as much time as possible. Even 1 or 2 jobs in the queue is ok. At the end of each waiting period the script will display how many jobs are in the queue and how many jobs it will start based on the policy set by the user, specifically the value of the jobs flag. If the # of jobs in the queue (which includes any running jobs) falls below the value of the jobs flag, the script will start enough jobs for the queue to reach the value of the jobs flag.

# Other Options:

By default, TreeDelete jobs run with a priority of 4 and an impact policy of MEDIUM. Please note the difference between these two values. Priority is used when the job queue is full (a maximum of 3 jobs of any type can run at any given time), the job priority will tell OneFS which jobs should run and which jobs should be paused. Note: This does not affect the amount of resources used by the jobs but merely the run priority. The impact policy affects the resources allocated to a job. There are pre-defined resources: LOW, MEDIUM, and HIGH as well as user-defined policies that involve time (an example of which is included in the system called OFF\_HOURS). Should the user want to adjust the priority or the impact policy of the TreeDelete jobs spawned by TreeDelete+, the –P and –I flags are used.

By default, the script will only output the # of jobs in the queue, how many jobs it starts per wait period, as well as how many paths are left to be assigned to a job (once all paths are assigned to jobs, the script will then report how many jobs are left to run). If verbose mode is enabled, each time a new job is started, the script will output the CLI command used to start the job. This is one way to see what paths have been assigned to the job or just for informational purposes.

Sometimes, it’s a good idea to test the parameters before running the script live. The –T flag puts the script into “Test Mode”. Like verbose mode, the script will display the command run for each job “started” but it won’t actually start the job.

# Support

This script is released as Open Source. Anyone is free to update or modify it. Any updates are welcome to put pushed to GitHub. The script is provided as-is and it not supported by Dell EMC or Isilon support. With any script, especially one that deletes data, testing is highly encouraged. Any bugs or feature requests are welcome but there is no warranty to time frame for any updates.