How to shutdown and restart the pipeline server

To stop and start the pipeline automatically (for a planned outage):

- first of all, login as *glast* to some interactive machine and go to the pipeline directory. In this example we are working with the prod pipeline (there are also dev/ and test/ subdirectories under ~glast/pipeline-II/):

```
$ ssh glast@rhel6-64
$ cd ~glast/pipeline-II/prod/
```

- create and edit a shutdown_schedule_<date> file:

```
$ date > shutdown_schedule_20100909
$ vim shutdown_schedule_20100909
```

- the shutdown_schedule_<date> file should look something like this (the start time should be some 10 minutes before the beginning of the outage, the stop time should be a few hours after the expected end of the outage); important: **leave a blank line at the end of the file**:

```
Thu Sep  9 09:50:00 PDT 2010
Thu Sep  9 13:50:00 PDT 2010

```

- create a symbolic link for shutdown_schedule_<date> as shutdown_schedule:

```
ln -s shutdown_schedule_20100909 shutdown_schedule
```

- the pipeline should shutdown automagically a few minutes after the start time in the shutdown_schedule file (there is a cronjob involved, plus it can take a couple of minutes for it to go down)
- you'll want it to start up again with an increased reaper delay so it doesn't fail the processes that finished while the pipeline was down. Edit the file run-pipeline.csh, look for a line like

```
     -Dpipeline.reaper.delay=300 \
```

  and change it to

```
     -Dpipeline.reaper.delay=3000 \
```

  After the pipeline has restarted and run long enough to process the old jobs, stop it, change run-pipeline.csh back to the way it was, and restart. Note that for it to restart with the new value, a simple "restart" won't suffice. It needs to be completely shutdown and restarted.

- the pipeline will start again automatically after the end date specified in the shutdown_schedule file. If you want to extend the outage before the end date occurs, you can modify and save the file.
- to restart the pipeline before the outage is over, modify the end date to be before the current time and save the file, or just remove the symbolic link (leaving the shutdown_schedule_<date> file for future reference):

```
rm shutdown_schedule
```

- the pipeline should go back up automagically in a few minutes
- to check the status of the pipeline server, one can use the *ping* function of the pipeline:

```
glast@rhel6-64 $ ./pipeline ping
Pinged server version 1.5.0.2 running on fermilnx-v13.slac.stanford.edu since 2022-02-25 13:31:01.815
```

To stop and start the pipeline manually (if all else fails):

- first of all, login as *glast* to an interactive machine and go to the pipeline directory:

```
$ ssh glast@rhel6-64
$ cd ~glast/pipeline-II/prod/
```

- issue the *pipeline ping* command to find out where the pipeline is running (for *PROD* this is currently *fermilnx-v13,* for *DEV* it is currently *fermilnx-v16*, and for *TEST* it is currently *??*)

```
$ ./pipeline ping
Pinged server version 1.5.0.2 running on fermilnx-v13.slac.stanford.edu since 2022-02-25 13:31:01.815
```

- ssh to the server listed (in the above example, it was *glastlnx12.slac.stanford.edu*) This is not strictly necessary when using the *pipeline shutdown* command, but it is necessary to run the *start* and *stop* scripts from the correct machine.

```
ssh glast@fermilnx-v13
```

- move out of the way the *monitor* script (which will restart the pipeline whenever it doesn't find it running):

```
$ mv monitor monitor_something_something
```

- shutdown the pipeline server with the *shutdown* command:

```
$ ./pipeline shutdown
```

- It can take up to a minute or more for the pipeline to finish what it is doing and exit. You can issue the pipeline *ping* command a few times until you see it has exited:

```
$ ./pipeline ping
Pinged server version 1.5.0.2 running on fermilnx-v13.slac.stanford.edu since 2022-02-25 13:31:01.815 $ .
/pipeline ping
org.glast.pipeline.client.PipelineClient$PipelineServerNotRunningException: Pipeline server not running
        at org.glast.pipeline.client.PipelineClient.initialize(PipelineClient.java:62)
        at org.glast.pipeline.client.PipelineClient.<init>(PipelineClient.java:52)
        at org.glast.pipeline.client.command.PipelineCommand.getPipelineClient(PipelineCommand.java:220)
        at org.glast.pipeline.client.command.Ping.execute(Ping.java:26)
        at org.glast.pipeline.client.command.PipelineCommand.parseCommand(PipelineCommand.java:149)
        at org.glast.pipeline.client.command.PipelineCommand.main(PipelineCommand.java:100)
```

- **perform whatever work required the outage**
- restart the pipeline using the *start* script:

```
$ ./start
```

- move back the *monitor* script to its original location:

```
$ mv monitor_something_something monitor
```

- if the *shutdown* command hangs or fails, one can use the *stop* script, which finds the pid and kills the process:

```
$ ./stop
```

- to check the status of the pipeline server, one can use the *ping* function of the pipeline:

```
glast@rhel6-64 $ ./pipeline ping
Pinged server version 1.5.0.2 running on fermilnx-v13.slac.stanford.edu since 2022-02-25 13:31:01.815
```