# Advanced Job Scripts

## Traps

Linux processes can send signals to and receive signals from other processes. When using the kill command to end a running process, you are sending it a signal. By default kill sends the signal 15, also known as SIGTERM, which asks the process politely to stop running. kill -9 PID kills the process without allowing it to process any other signals, 9 is also known as SIGKILL. You can find more information about the available signals with man 7 signal.

All processes have the option of capturing these signals with a trap and handling the trap with some routine that runs for a given signal. Try the following example:

#!/bin/bash  
  
trap "echo -e '\n\nHi $USER, I caught a signal\n\n'" SIGINT SIGTERM  
  
echo "I am PID: $$"  
echo  
  
sleep 3600

Save that as a file, make it executable with chmod a+x main.sh and execute it, once running press CTRL-C. CTRL-C sends the process the SIGINT signal, which the script has specified in the trap. The result is that the process is killed, but executes the trap prior to ending:

[user@host ~]$ ./trap.sh  
I am PID: 31389  
  
^C  
  
Hi user, I caught a signal  
  
  
[user@host ~]$

### Use in Job Scripts

There are numerous ways in which signals can be useful in a job script, which include but are not limited to:

* Cleaning up when the job exits, regardless of how it exits.
* Reporting progress from the job script.
* [Your Idea Here]

The most common way to use traps is to handle job cleanup. If the job is killed for exceeding time limits, exceeding memory, or for any other reason, Slurm will first send it SIGINT, then wait for a short time before sending SIGKILL to clean up any lingering processes. Between getting those signals, a trap can perform functions like copying partial results, deleting temp files, removing partial results to prepare for the job being requeued, etc. The following example shows how a trap can be used to clean up temp files on job exit, using the trap special keyword EXIT which causes the trap to run when the script exits. As a bonus, it demonstrates the use of mktemp to create unique temporary files.

#!/bin/bash  
#SBATCH --time=2:00:00  
#SBATCH --job-name=TRAPS  
#SBATCH --nodes=1  
  
function cleanup () {  
 rm -f ${mytempfile}  
 echo "Cleanup Completed."  
}  
  
trap cleanup EXIT  
  
echo "I am PID: $$"  
echo  
  
mytempfile=$(mktemp /tmp/username.XXX)  
  
echo "My temp file is ${mytempfile}"  
echo  
ls ${mytempfile}  
sleep 300

And when executed:

# Submit our trap enabled job.  
[user@host ~]$ sbatch traps.sh  
Submitted batch job 7243

# It's running!  
[user@host ~]$ squeue -u $USER  
 JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)  
 7243 batch TRAPS username R 0:03 1 nodename

# ssh to node, see temp file.  
[user@host ~]$ ssh nodename ls /tmp/username\*  
/tmp/username.nXg

# Wait for job to complete, check again. No temp file.  
[user@host ~]$ ssh nodename ls /tmp/username\*

# See what our output looks like.  
[user@host ~]$ cat slurm-7243.out  
I am PID: 50662  
  
My temp file is /tmp/username.nXg  
  
/tmp/username.nXg  
Cleanup Completed.