Submitting Multiple Jobs With HTCondor

Christina Koch HTCondor Week 2020











Image credit: The Carpentries Instructor Training





Image credit: The Carpentries Instructor Training



Multiple job goals



Image credit: The Carpentries Instructor Training







Let's review: one job

executable analyze.sh
arguments = file.in file.out
transfer_input_files = file.in

log = job.log
output = job.stdout
error = job.stderr

This is the command we want HTCondor to run.

queue



Let's review: one job

executable = analyze.sh arguments = file.in file.out transfer_input_files = file.in

log = job.log
output = job.stdout
error = job.stderr

These are the files we need for the job to run.

queue



Let's review: one job





Example 1: Many jobs with numbered files

Now suppose we have many input files and we want to run one job per input file.



List of numerical input values

We want to capture this set of inputs using a list of integers.



Provide a list of integer values with queue N

```
executable = analyze.sh
arguments = file.in file.out
transfer input files = file.in
log = job.log
output = job.stdout
error = job.stderr
                                 This queue statement
                                 will generate a list of
queue 5
                                 integers, 0 - 4
```



Which job components vary?

executable = analyze.sh
arguments = file.in file.out
transfer_input files = file.in

log = job.log
output = job.stdout
error = job.stderr

queue 5

The arguments for our command and the input files would be different for each job.



Which job components vary?





Use \$(ProcID) as the variable

```
executable = analyze.sh
arguments = file.$(ProcID).in file.$(ProcID).out
transfer_input_files = file$(ProcID).in
```

```
log = job.$(ProcID).log
output = job.$(ProcID).stdout
error = job.$(ProcID).stderr
```

queue 5

The default variable representing the changing numbers in our list is \$(ProcID)



Example 2: Many jobs with named files

• Program execution

\$ compare_states state.wi.dat out.state.wi.dat

- Files needed
 - compare_states, state.wi.dat, country.us.dat

```
executable = compare_states
arguments = state.wi.dat out.state.wi.dat
transfer_input_files = state.wi.dat, country.us.dat
queue
```



List of named input values

- Suppose we have data for several states: state.wi.dat, state.mn.dat, state.il.dat, etc.
- We want to run one job per file.

```
executable = compare_states
arguments = state.wi.dat out.state.wi.dat
transfer_input_files = state.wi.dat, country.us.dat
queue
```



Provide a list of values with queue from

- We want to use "queue" to provide this list of input files.
- One option is to create another file with the list and use the queue .. from syntax.

executable = compare_states
arguments = state.wi.dat out.state.wi.dat
transfer_input_files = state.wi.dat, country.us.dat

```
queue from state_list.txt
```



Which job components vary?

- Now, what parts of our job template (the top half of the submit file) vary, depending on the input?
- We want to vary the job's **arguments** and one **input file**.

```
executable = compare_states
arguments = state.wi.dat out.state.wi.dat
transfer_input_files = state.wi.dat, country.us.dat
queue state from state_list.txt
```



Use a custom variable

• Replace all our varying components in the submit file with a variable.

state.wi.dat
state.mn.dat
state.il.dat
state.ia.dat
state.mi.dat

```
executable = compare_states
```

```
arguments = $(state) out.$(state)
```

```
transfer_input_files = $(state), country.us.dat
```

```
queue state from state list.txt
```



Use multiple variables with queue from

- The queue from syntax can also support multiple values per job.
- Suppose our command was like this:

\$ compare states -i [input file] -y [year]			
	state.wi.dat,2010		
	<pre>state.wi.dat,2015</pre>		
executable = compare states	state.mn.dat,2010		
executable compare_states	a + a + a mp $da + 2015$		
arguments = -i \$(state) -y \$(year)	State. IIII. dat, 2015		
<pre>transfer_input_files = \$(state), country.us.dat</pre>			
queue state, year from state_list.txt			



Variable and **queue** options

Syntax	List of Values	Variable Name
queue N	Integers: 0 through N-1	\$(ProcId)
queue <i>Var</i> matching <i>pattern*</i>	List of values that match the wildcard pattern.	\$(<i>Var</i>)
queue <i>Var</i> in (<i>item1 item2</i>)	List of values within parentheses.	If no variable name is provided, default is \$(Item)
queue <i>Var</i> from <i>list.txt</i>	List of values from <i>list.txt,</i> where each value is on its own line.	



Other options: queue N

- Can I start from 1 instead of 0?
 - Yes! These two lines increment the \$(ProcId) variable

tempProc = \$(ProcId) + 1
newProc = \$INT(tempProc)

- You would use the second variable name \$(newProc) in your submit file
- Can I create a certain number of digits (i.e. 000, 001 instead of 0,1)?
 - Yes, this syntax will make \$(Procld) have a certain number of digits

\$INT(ProcId,%03)



Other options: queue in/from/matching

• You can run multiple jobs per list item, using \$(Step) as the index:

executable = analyze.sh
arguments = -input \$(infile) -index \$(Step)
queue 10 infile matching *.dat

• queue matching has options to select only files or directories

```
queue inp matching files *.dat
```

queue inp matching dirs job*



Case Study 1



What varies?

- Not much just needs an index to keep simulation results separate.
- Use queue N
 - Simple, built-in
 - No need for specific input values



Case Study 2



- What varies?
 - Five parameter combinations per job
 - Parameters are given as arguments to the executable
- Use queue from
 - queue from can accommodate multiple values per job
 - Easy to re-run combinations that fail by using subset of original list



Case Study 3



- What varies?
 - Each job analyzes one sample; each sample consists of two fastq files in a folder with a standard prefix.
- Use queue matching
 - Folders have a standard prefix, input files have standard suffix, easy to pattern match
- Good alternative: queue from
 - Provide list of folder names/file prefixes, construct paths in the submit file.
- Want output files to return to the same folder (stay tuned...)



Queue options, pros and cons

queue N	Simple, good for multiple jobs that only require a numerical index.
queue matching <i>pattern*</i>	Natural nested looping, minimal programming, use optional "files" and "dirs" keywords to only match files or directories Requires good naming conventions.
queue in (<i>list</i>)	Supports multiple variables, all information contained in a single file, reproducible Harder to automate submit file creation
queue from file	Supports multiple variables, highly modular (easy to use one submit file for many job batches), reproducible Additional file needed



Organization



Many jobs means many files.

HTCondor Week 2020



7266263 0.err 5175744 0.err 12181445 0.log 16058473 0.log 17381628 0.log 18159900 0.log 5175744 0.log 7266263 0.log 12181445 0.out 16058473 0.out 17381628 0.out 18159900 0.out 5175744 0.out 7266263 0.out 13609567 0.err 16060330 0.err 17381640 0.err 3446080 0.err 5176204 0.err 13609567 0.log 16060330 0.log 17381640 0.log 3446080 0.log 5176204 0.log 7266267 0.log 13609567 0.out 16060330 0.out 17381640 0.out 3446080 0.out 5176204 0.out 7266267 0.out 13612268 0.err 16254074 0.err 17381665 0.err 3446306 0.err 5295132 0.err 7937420 0.err 13612268 0.log 16254074 0.log 17381665 0.log 3446306 0.log 5295132 0.log 7937420 0.log 13612268 0.out 16254074 0.out 17381665 0.out 3446306 0.out 5295132 0.out 7937420 0.out 13630381 0.err 17134215 0.err 17381676 0.err 4347054 0.err 5318339 0.err 8779997 0.err 13630381 0.log 17134215 0.log 17381676 0.log 4347054 0.log 5318339 0.log 8779997 0.log 17134215 0.out 17381676 0.out 4347054 0.out 5318339 0.out 8779997 0.out

Directories are your friends

```
log = logs/job.$(ProcID).log
output = output/job.$(ProcID).stdout
error = error/job.$(ProcID).stderr
```

queue 5

```
submit dir/
  jobs.submit
  analyze.sh
  shared/
    script1.sh
    reference.dat
  input/
    file0.in
    . . .
  logs/
    job.0.log
     . . .
  output/
    job.0.stdout
     . . .
  error/
    job.0.stderr
    . . .
```



Job-specific directories with initialdir

```
executable = analyze.sh
transfer_input_files = file.in
initialdir = job$(ProcId)
```

```
output = job.stdout
error = job.stderr
```

queue 5

submit dir/ jobs.submit analyze.sh job0/ file.in job.stdout job.stderr job1/ file.in job.stdout job.stderr job2/

. . .



Use variables, move output files

		submit_dir, jobs.subr	/ nit
infile outfile	<pre>= file\$(ProcID).in = file\$(ProcID).out</pre>	analyze.sh input/ file0.in	
executable arguments	<pre>= analyze.sh = \$(infile) \$(outfile)</pre>	 output/ file0.0	out
transfer input files	<pre>= input/\$(infile)</pre>	•••	
<pre>transfer_output_files = \$(outfile) transfer_output_remaps = "\$(outfile)=output/\$(outfile)"</pre>			

queue 5



Resources

- Example jobs and submit files:
 - https://github.com/CHTC/example-multiple-jobs
- condor_submit documentation:
 - https://htcondor.readthedocs.io/en/latest/man-pages/condor_submit.html
 - Search for "queue"
- HTCondor user tutorial
 - https://agenda.hep.wisc.edu/event/1325/session/0/contribution/19/material /slides/0.pdf
- Advanced submit talk
 - https://agenda.hep.wisc.edu/event/1325/session/3/contribution/40/material /slides/0.pptx



Questions?

