HTCondor

January 24, 2022

wikitolearn
collaborative textbooks

This book is the result of a collaborative effort of a community of people like you, who believe that knowledge only grows if shared.
We are waiting for you!

Get in touch with the rest of the team by visiting http://join.wikitolearn.org

Contents

Chapter 1

User Tutorial

1.1 Slides

Slides courtesy of the University of Wisconsin, creators of HTCondor.

PDF Slides

PPT Slides

Chapter 2

DAG Tutorial

2.1 Slides

Slides courtesy of the University of Wisconsin, creator of HTCondor

PDF Slides

PPT Slides

Chapter 3

Exercises

3.1 Login and have a look around

Goal: login to a submit machine and look around You should all have the IP address of a machine that you can login to via "ssh". The username will be "gks" and your password will be supplied.

- On Mac OS X, you can run the Terminal app and use the ssh command like so:

```
$ ssh gks@141.52.228.218
```

- On windows a good ssh client is PuTTY
- Linux should have ssh available from a terminal window, as per the mac

Running commands We will show commands you can run like so:

```
$ hostname
```

Note that the "$" in the above is to signify a bash prompt, you don't need to type it!

Organising your work The home directory that you have on your machine is yours to use. You are free to create directories to help structure your work. The examples we use will often specify that log / ouput / error is created in subdirectories, so we can create those now. Run this:

```
$ mkdir log output error
```

Condor Version Run the following:

```
$ condor_version
```

You should see output like:

```
[gks@htcondor-t-0 ~]$ condor_version
$CondorVersion: 8.6.0 Jan 26 2017 BuildID: 395190 $
$CondorPlatform: x86_64_RedHat7 $
```

Note on HTCondor Versions HTCondor always has two types of releases at any one time: *stable* and *development*. The even minor versions are *stable*, whilst the odd minor version are *development*. This means that the current stable releases are 8.6.* and the current development releases are 8.7.*

3.2 Basic HTCondor commands

Experiment with basic HTCondor commands.

We are going to look at two fundamental HTCondor commands "condor_q" and "condor_status". They are used to monitor your jobs and your slots, respectively.

Viewing slots This command can be very simple:

```
$ condor_status
```

This command running on the CERN pool would produce a lot of output - we have 100k slots. Here we should see something a bit more simple.

```
[gks@htcondor-t-0 ~]$ condor_status
Name                              OpSys      Arch   State
    Activity LoadAv Mem   ActvtyTime

slot1@htcondor-t-0.os-internal LINUX      X86_64 Unclaimed
    Idle      0.000 3790  5+18:35:27
slot1@htcondor-t-1.os-internal LINUX      X86_64 Unclaimed
    Idle      0.050 3790  5+18:35:20

                     Machines Owner Claimed Unclaimed
   Matched Preempting  Drain

        X86_64/LINUX        2     0       0         2
     0          0      0

               Total        2     0       0         2
     0          0      0
```

The output consists of 8 columns:

After the slot data, you can see summary information about the whole pool. There is one row of summary for each machine architecture/operating system combination. The columns are the different states that a slot can be in. The final row gives a summary of slot states for the whole pool.

Now run:

Col	Example	Meaning
Name	slot1@htcondor-t-0.os-internal	Slot name and hostname
OpSys	LINUX	Operating System
Arch	X86_64	Machine Architecture
State	Unclaimed	State of the slot (`Unclaimed` is available, `Owner` is b
Activity	Idle	Is there activity on the slot?
LoadAv	0.050	Load average, a measure of CPU activity on the slo
Mem	3790	Memory available to the slot, in MB
ActivityTime	5+18:35:27	Amount of time spent in current activity (days + h

```
$ condor_status
```

...yourself and compare it to the output above. How does it compare?

Viewing whole machines only Run:

```
$ condor_status -compact
```

Note how the output compares to the full summary.

Viewing Jobs The `condor_q` command lists jobs that are on this submit machine and that are running or waiting to run. The `_q` part of the name is meant to suggest the word "queue", or list of jobs *waiting* to finish.

The simplest version of this command shows only your jobs:

```
$ condor_q
```

The main part of the output (which for you will be empty, as you haven't submitted any jobs yet), looks like this:

```
-- Schedd: bigbird02.cern.ch : <128.142.196.38:9618?... @
   08/28/17 13:07:42
OWNER   BATCH_NAME       SUBMITTED   DONE   RUN    IDLE
   TOTAL JOB_IDS
bejones CMD: hello.sh   8/28 13:07      _      _      1
      1 459934.0
```

The output consists of the following columns:

After this, again there's a summary, like:

```
1 jobs; 0 completed, 0 removed, 1 idle, 0 running, 0 held,
    0 suspended
```

This shows the job counts in all possible states.

Viewing everyones jobs In the lab, where you have your own schedd, this may not show much. In other environments, to show all users jobs, run:

```
$ condor_q -all
```

Col	Example	Meaning
OWNER	`bejones`	The user ID of the user who submitted the job
BATCH_NAME	`hello.sh`	The executable or the "jobbatchname" specified within submit file
SUBMITTED	`8/28 13:07`	The date and time when the job was submitted
DONE	_	Number of jobs in this batch that have completed
RUN	_	Number of jobs in this batch that are currently running
IDLE	1	Number of jobs in this batch that are idle, waiting for a match
HOLD	_	Column will show up if there are jobs on "hold" because somethin
TOTAL	1	Total number of jobs in this batch
JOB_IDS	`459934.0`	Job ID or range of Job IDs in this batch

3.3 Run jobs!

The goal of this exercise is to run some jobs...

This is the fundamental goal for an HTCondor system, to be able to run jobs. Make sure you are able to successfully run jobs by the end of it, else other exercises won't make much sense. Ask questions!

Run a simple command using a submit file Here is a simple submit file for the `hostname` command:

```
universe = vanilla
executable = /bin/hostname

output = simple.out
error = simple.err
log = simple.log

request_cpus = 1
request_memory = 1MB
request_disk = 1MB

queue
```

Write those lines of text in a file called simple.sub

Note: There is nothing magic about the name of an HTCondor submit file. It can be any filename you want. It's a good practice to always include the `.sub` extension, but it is not required. Ultimately, a submit file is a text file

The lines of the submit file have the following meanings:

Note that we are not using the `arguments` lines or `transfer_input_files` because the `hostname` program is all that needs to be transferred from the submit server, and we want to run it without any additional options.

Double-check your submit file, so that it matches the text above. Then, tell HTCondor to run your job:

```
$> condor_submit simple.sub
Submitting job(s).
1 job(s) submitted to cluster NNNN.
```

The actual cluster number will be shown instead of `NNNN`.

If, instead of the text above, there are error messages, read them carefully and then try to correct your submit file.

Notice that `condor_submit` returns back to the shell prompt right away. It does **not** wait for your job to run. Instead, as soon as it has finished submitting your job into the queue, the submit command finishes.

Now, use `condor_q` and `condor_q -nobatch` to watch for your job in the queue. You probably may not even catch the job in the `R` running state, because the `hostname`command runs very quickly. When the job itself is finished, it will no longer be listed in the `condor_q` output.

The output from your job is written to the filename given in the `output` line of your submit file. Thus, after the job finishes, you should be able to see the `hostname`output in `simple.out`, since this information is usually printed to the terminal by the `hostname` program, and not to a special file of it's own.

Run this to see the output:

```
$ cat simple.out
```

The `simple.err` file should be empty, unless there were issues running the `hostname` executable after it was transferred to the slot. The `simple.log` is more complex and will be the focus of a later exercise.

Running a job with arguments Very often, when you run a command on the command line, it includes arguments after the command name itself:

```
$> cat simple.out
$> sleep 60
$> dc -e '6 7 * p'
```

In an HTCondor submit file, the command (or "program") name itself goes in the `executable` statement and **all remaining arguments** go into an `arguments` statement. For example, if the full command is:

```
$> sleep 60
```

Then in the submit file, we put:

```
executable = /bin/sleep
arguments = "60"
```

Note: Put the entire list of arguments inside one pair of double-quotes.

For the command-line command:

```
$> dc -e '6 7 * p'
```

Then in the submit file, we put:

```
executable = /usr/bin/dc
arguments = "-e '6 7 * p'"
```

Let's try a job submission with arguments. We will use the `sleep` command shown above, which simply does nothing for the specified number of seconds, then exits normally. It is convenient for simulating a job that takes a while to run.

Create a new submit file (you name it this time!) and save the following text in it.

```
universe = vanilla
executable = /bin/sleep
arguments = "60"

output = sleep.out
error = sleep.err
log = sleep.log

request_cpus = 1
request_memory = 1MB
request_disk = 1MB

queue
```

Except for changing a few filenames, this submit file is nearly identical to the last one. But, see the extra `arguments` line?

Submit this new job. Again, watch for it to run using `condor_q` and `condor_q -nobatch`; check once every 15 seconds or so. Once the job starts running, it will take about 1 minute to run (because of the `sleep` command, right?), so you should be able to see it running for a bit. When the job finishes, it will disappear from the queue, but there will be no output in the output or error files, because `sleep` does not produce any output.

Running a script job from the submit directory So far, we have been running programs (executables) that come with the standard Linux system. But you are not limited to standard programs. In this example, you will write a simple shell script (command line) executable in the submit directory, then write a submit file to run it.

1. Put the following contents into a file named `test-script.sh`:

```
#!/bin/sh

echo 'Date:    ' $(date)
echo 'Host:    ' $(hostname)
echo 'System:  ' $(uname -spo)
echo "Program: $0"
```

```
echo "Args:    $*"
```

2. Make the file itself executable:

```
$ chmod +x test-script.sh
```

3. Test your script from the command line:

```
$ ./foo.sh hello 42
Date:     Mon Aug 28 13:36:22 CEST 2017
Host:     aiadm28.cern.ch
System:   Linux x86_64 GNU/Linux
Program: ./foo.sh
Args:    hello 42
```

   This step is **really** important! If you cannot run your executable from the command-line, HTCondor probably cannot run it on another machine, either. And debugging simple problems like this one is surprisingly difficult. So, if possible, test your `executable` and `arguments` as a command at the command-line first.

4. Write the submit file (this should be getting easier now):

```
universe                = vanilla
executable              = test-script.sh
arguments               = "foo bar baz"
output                  = script.out
error                   = script.err
log                     = script.log
request_cpus = 1
request_memory = 1
request_disk = 1
queue
```

   Note: As this example shows, blank lines and spaces around the = sign do not matter to HTCondor. Use whitespace to make things clear to **you**. What format do you prefer to read?

5. Submit the job, wait for it to finish, check the output. (Are you surprised by the `Program:` line in the output? Why is it like that? Google for it, or ask an instructor if you are curious, although the answer is not that exciting.)

In this example, the `executable` that was named in the submit file did **not** start with a `/`, so the location of the file is relative to the submit directory itself. In other words, in this format the executable must be in the same directory as the submit file

3.4 Log files

3.4.1 Read and Interpret log files

The goal of this exercise is quite simple: Learn to understand the contents of a job log file. When things go wrong with your job, it is usually the first place you should look for important messages. Plus, there is other useful information there.

This exercise is short. If you do not have time for it now, come back and visit it later.

Reading a Log file A job log file is updated throughout the life of a job, usually at key events. Each event starts with a heading that indicates what happened and when. Here are **all** of the event headings from the `sleep` job log (detailed output in between headings has been omitted here):

```
000 (459934.000.000) 08/28 13:07:40 Job submitted from
   host: <128.142.196.38:9618?addrs
   =128.142.196.38-9618+[2001-1458-301-e2--100-20]-9618&
   noUDP&sock=1412634_012e_3>
001 (459934.000.000) 08/28 13:13:18 Job executing on host:
    <128.142.138.200:9618?addrs
   =128.142.138.200-9618+[--1]-9618&noUDP&sock=7965_3855_3
   >
006 (459934.000.000) 08/28 13:13:22 Image size of job
   updated: 120812
005 (459934.000.000) 08/28 13:13:22 Job terminated.
```

There is a lot of extra information in those lines, but you can see:

- The job ID: cluster 459934, process 0 (written `000`)
- The date and local time of each event
- A brief description of the event: submission, execution, some information updates, and termination

Some events provide no information in addition to the heading. For example:

```
000 (459934.000.000) 08/28 13:07:40 Job submitted from
   host: <128.142.196.38:9618?addrs
   =128.142.196.38-9618+[2001-1458-301-e2--100-20]-9618&
   noUDP&sock=1412634_012e_3>
...
```

and:

```
001 (459934.000.000) 08/28 13:13:18 Job executing on host:
    <128.142.138.200:9618?addrs
   =128.142.138.200-9618+[--1]-9618&noUDP&sock=7965_3855_3
   >
...
```

Note: Each event ends with a line that contains only 3 dots: ...

But the periodic information update event contains some additional information:

```
006 (459934.000.000) 08/28 13:13:22 Image size of job
   updated: 120812
   0  -  MemoryUsage of job (MB)
   0  -  ResidentSetSize of job (KB)
...
```

These updates record the amount of memory that the job is using on the execute machine. This can be helpful information, so that in future runs of the job, you can tell HTCondor how much memory you will need.

The job termination event includes a great deal of additional information:

```
005 (459934.000.000) 08/28 13:13:22 Job terminated.
   (1) Normal termination (return value 0)
      Usr 0 00:00:00, Sys 0 00:00:00  -  Run Remote Usage
      Usr 0 00:00:00, Sys 0 00:00:00  -  Run Local Usage
      Usr 0 00:00:00, Sys 0 00:00:00  -  Total Remote
   Usage
      Usr 0 00:00:00, Sys 0 00:00:00  -  Total Local Usage
   0  -  Run Bytes Sent By Job
   57  -  Run Bytes Received By Job
   0  -  Total Bytes Sent By Job
   57  -  Total Bytes Received By Job
   Partitionable Resources :    Usage  Request Allocated
      Cpus                 :                1         1
      Disk (KB)            :       17        1   2013116
      Memory (MB)          :        0     2000      2000
...
```

Probably the most interesting information is:

- The `return value` (`0` here, which is success; non-zero usually means failure)
- The total number of bytes transferred each way, which could be useful if your network is slow
- The `Partitionable Resources` table, especially disk and memory usage

There are many other kinds of events, but the ones above will occur in almost every job log.

Understanding when job log events are written When are events written to the job log file? Let's find out. Read through the entire procedure below before starting, because some parts of the process are time sensitive.

1. Change the `sleep` job submit file, so that the job sleeps for 2 minutes (= 120 seconds)

2. Submit the updated sleep job
3. As soon as the `condor_submit` command finishes, hit the return key a few times, to create some blank lines
4. Right away, run a command to show the log file and **keep showing** updates as they occur.

```
$ tail -f sleep.log
```

   Be sure to use the correct filename for your log file, as named in your submit file.

5. Watch the output carefully. When do events appear in the log file?
6. After the termination event appears, press Control-C to end the `tail` command and return to the shell prompt.

3.4.2 Understanding How HTCondor Writes Files

When HTCondor writes the output, error, and log files, does it erase the previous contents of the file or does it add new lines onto the end? Let's find out!

For this exercise, we can use the `hostname` job from earlier.

1. Edit the `hostname` submit file so that it uses new and unique filenames for output, error, and log files Alternatively, delete any existing output, error, and log files from previous runs of the `hostname` job.
2. Submit the job three separate times in a row (there are better ways to do this, which we will cover in the next lecture)
3. Wait for all three jobs to finish
4. Examine the output file: How many hostnames are there? Did HTCondor erase the previous contents for each job, or add new lines?
5. Examine the log file... carefully: What happened there? Pay close attention to the times and job IDs of the events.

3.5 Resource requirements

3.6 Declare Resource Needs

The goal of this exercise is to demonstrate how to test and tune the `request_X` statements in a submit file for when you don't know what resources your job needs.

There are three special resource request statements that you can use (optionally) in an HTCondor submit file:

* `request_cpus` for the number of CPUs your job will use (most softwares will take an argument to control this number, and it's usually otherwise "1") * `request_memory`for the maximum amount of run-time memory your job may use * `request_disk` for the maximum amount of disk space your job may use (including the executable and all other data that may show up during the job)

HTCondor defaults to certain reasonable values for these request settings, so you do not need to use them to get *small* jobs to run. However, on some HTCondor pools, if your job goes over the request values, it may be removed from the execute machine and either held (awaiting action on your part) or rerun later. So it can be a disadvantage to you if you do not declare your resource needs or if you underestimate them. If you overestimate them, your jobs will match to fewer slots (and with a longer average wait time) *and* you'll be hogging up resources that you don't need, but that could be used for the jobs of other users. In the long run, it works better for all users of the pool if you declare what you really need.

But how do you know what to request? In particular, we are concerned with memory and disk here; requesting multiple CPUs and using them is covered a bit in later school materials, but true HTC splits work up into jobs that each use as few CPU cores as possible (one CPU core is always best to have the most jobs running soonest!).

3.6.1 Determining Resource Needs Before Running Any Jobs

It can be very difficult to determine the memory needs of your running program. Typically, the memory size of a job changes over time, making the task even trickier. If you have knowledge ahead of time about your job's maximum memory needs, use that, or a maybe a number that's just a bit higher, to be safe. If not, then it's best to run your program in a single test job, first, and let HTCondor tell you in the log file (or in the `condor_q -nobatch` output, if you're able to watch it), which is covered in the next section on "Determining Resource Needs by Running Test Jobs".

You can try to figure out the resource requirements of a job by running it locally, and seeing what it uses. In the lab here, this machine is the same thing as your execute machine, so the difference is artificial. However, here's a couple of tools you could use.

Using `ps`:

```
$ ps -ux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START
     TIME COMMAND
bejones   22210  0.0  0.0 161316  1592 pts/2    R+   22:17
     0:00 ps -ux
bejones   31695  0.0  0.0 129852  3460 pts/2    Ss   21:39
     0:00 -bash
```

The Resident Set Size (`RSS`) column, highlighted above, gives a rough indication of the memory usage (in KB) of each running process. If your program runs long enough, you can run this command several times and note the greatest value.

Using `top`:

```
$ top -u bejones

top - 22:18:45 up 53 days, 15:18, 29 users,  load average:
    0.02, 0.16, 0.48
Tasks:   2 total,   1 running,   1 sleeping,   0 stopped,
     0 zombie
%Cpu(s):  0.8 us,  1.5 sy,  0.0 ni, 94.8 id,  2.7 wa,  0.0
    hi,  0.2 si,  0.0 st
KiB Mem :  3531688 total,   161792 free,  2588424 used,
   781472 buff/cache
KiB Swap:        0 total,        0 free,        0 used.
   533636 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM
      TIME+ COMMAND
22544 bejones   20   0  167872   2200   1580 R   0.3  0.1
     0:00.03 top
31695 bejones   20   0  129852   3464    956 S   0.0  0.1
     0:00.17 bash
```

The `top` command (shown here with an option to limit the output to a single user ID) also shows information about running processes, but updates periodically by itself. Type the letter `q` to quit the interactive display. Again, the highlighted `RES` column shows an approximation of memory usage.

For Disk: Determining disk needs may be a bit simpler, because you can check on the size of files that a program is using while it runs. However, it is important to count all files that HTCondor counts to get an accurate size. HTCondor counts **everything** in your job sandbox toward your job's disk usage:

- The executable itself
- All "input" files (anything else that gets transferred TO the job, even if you don't think of it as "input")
- All files created during the job (broadly defined as "output"), including the captured standard output and error files that you list in the submit file.
- All temporary files created in the sandbox, even if they get deleted by the executable before it's done.

If you can run your program within a single directory on a local computer (not on the submit server), you should be able to view files and their sizes with the `ls`command.

3.6.2 Determining Resource Needs By Running Test Jobs (BEST)

Despite the techniques mentioned above, by far the easiest approach to measuring your job's resource needs is to run one or a small number of sample jobs and have HTCondor itself tell you about the resources used during the runs.

For example, here is a strange Python script that does not do anything useful, but consumes some real resources while running:

```
#!/usr/bin/env python
import time
import os
size = 1000000
numbers = []
for i in xrange(size): numbers.append(str(i))
tempfile = open('temp', 'w')
tempfile.write(' '.join(numbers))
tempfile.close()
time.sleep(60)
os.remove('temp')
```

Without trying to figure out what this code does or how many resources it uses, just create a submit file for it, and run it once with HTCondor, starting with somewhat high memory requests ("1GB" for memory and disk is a good starting point, unless you think the job will use far more, and will still match quickly). When it is done, examine the log file. In particular, we care about these lines:

```
Partitionable Resources :    Usage  Request Allocated
      Cpus                 :                1         1
      Disk (KB)            :     6739  1048576   8022934
      Memory (MB)          :        3     1024      1024
```

This is a great technique for determining the real resource needs of your job. If you think resource needs vary from run to run, submit a few sample jobs and look at all the results. And it never hurts to round up your resource requests a little, just in case your job occasionally uses more resources.

3.6.3 Setting Resource Requirements

Once you know your job's resource requirements, it is easy to declare them in your submit file. For example, taking our results above as an example, we might slightly increase our requests above what was used, just to be safe:

```
# rounded up from 3 MB
request_memory = 4MB
# rounded up from 6.5 MB
request_disk = 7MB
```

Pay close attention to units:

- Without explicit units, `request_memory` is in MB (megabytes)
- Without explicit units, `request_disk` is in KB (kilobytes)
- Allowable units are `KB` (kilobytes), `MB` (megabytes), `GB` (gigabytes), and `TB` (terabytes)

HTCondor translates these requirements into expressions that become part of the `requirements` expression. However, do not put your CPU, memory, and disk requirements directly into the `requirements` expression; use the `request_XXX` statements instead.

Add these requirements to your submit file for the Python script, rerun the job, and confirm in the log file that your requests were used.

3.7 Remove jobs from the queue

3.8 Remove Jobs From the Queue

The goal of this exercise is to show you how to remove jobs from the queue. This is helpful if you make a mistake, do not want to wait for a job to complete, or otherwise need to fix things. For example, if some test jobs go on hold for using too much memory or disk, you may want to just remove them, edit the submit files, and then submit again.

NOTE: Please remember to remove any jobs from the queue that you have given up on. Otherwise, the queue will start to get very long with jobs that will waste resources (and decrease your priority!), or that may never run (if they're on hold, or have other issues keeping them from matching).

This exercise is very short, but if you are out of time, you can come back to it later.

3.8.1 Removing a Job From the Queue

To practice removing jobs from the queue, you need a job in the queue!

1. Submit a job from an earlier exercise
2. Determine the job ID (`cluster.process`) from the `condor_submit` output or from `condor_q`
3. Remove the job:

```
$ condor_rm <job id>
```

   Use the full job ID this time, e.g., `5759.0`.

4. Did the job leave the queue immediately? If not, about how long did it take?

So far, we have created job clusters that contain only one job process (the `.0` part of the job ID). That will change soon, so it is good to know how to remove a specific job ID. However, it is possible to remove all jobs that are part of a cluster at once. Simply omit the job process (the `.0` part of the job ID) in the `condor_rm`command:

```
$ condor_rm <cluster>
```

Finally, you can include many job clusters and full job IDs in a single `condor_rm` command. For example:

```
$ condor_rm 5768 5769 5770.0 5771.2
```

3.8.2 Removing All of Your Jobs

If you really want to remove all of your jobs at once, you can do that.

1. Quickly submit several jobs from past exercises
2. View the jobs in the queue with `condor_q`
3. Remove them all at once

```
$ condor_rm <username>
```

4. Use `condor_q` to track progress

In case you are wondering, you can remove only your own jobs. HTCondor administrators can remove anyone's jobs, so be nice to them.

3.9 Explore condor q

3.10 Explore condor_q

The goal of this exercise is try out some of the most common options to the `condor_q` command, so that you can view jobs effectively.

The main part of this exercise should take just a few minutes, but if you have more time later, come back and work on the extension ideas at the end to become a `condor_q` expert!

3.10.1 Selecting Jobs

The `condor_q` program has many options for selecting which jobs are listed. You have already seen that the default mode (as of version 8.5) is to show only your jobs in "batch" mode:

```
$ condor_q
```

You've seen that you can view all jobs (all users) in the submit node's queue by using the `-all` argument:

```
$ condor_q -all
```

And you've seen that you can view more details about queued jobs, with each separate job on a single line using the `-nobatch` option:

```
$ condor_q -nobatch
$ condor_q -all -nobatch
```

Did you know you can also name one or more user IDs on the command line, in which case jobs for all of the named users are listed at once?

```
$ condor_q username1 username2 username3
```

There are two other, simple selection criteria that you can use. To list just the jobs associated with a single cluster number:

```
$ condor_q <cluster>
```

For example, if you want to see the jobs in cluster 5678 (i.e., `5678.0`, `5678.1`, etc.), you use `condor_q 5678`.

To list a specific job (i.e., cluster.process, as in 5678.0):

```
$ condor_q <job id>
```

For example, to see job ID 5678.1, you use `condor_q 5678.1`.

Note: You can name more than one cluster, job ID, or combination thereof on the command line, in which case jobs for **all** of the named clusters and/or job IDs are listed.

Let's get some practice using `condor_q` selections!

1. Using a previous exercise, submit several `sleep` jobs
2. List all jobs in the queue — are there others besides your own?
3. Practice using all forms of `condor_q` that you have learned:
   - List just your jobs, with and without batching
   - List a specific cluster
   - List a specific job ID
   - Try listing several users at once
   - Try listing several clusters and job IDs at once
4. When there are a variety of jobs in the queue, try combining a user ID and a different user's cluster or job ID in the same command — what happens?

3.10.2 Viewing a Job ClassAd

You may have wondered why it is useful to be able to list a single job ID using `condor_q`. By itself, it may not be that useful. But, in combination with another option, it is very useful!

If you add the `-long` option to `condor_q` (or its short form, `-l`), it will show the complete ClassAd for each selected job, instead of the one-line summary that you

have seen so far. Because job ClassAds may have 80–90 attributes (or more!), it probably makes the most sense to show the ClassAd for a single job at a time. And you know how to show just one job! Here is what the command looks like:

```
$ condor_q -long <job id>
```

The output from this command is long and complex. Most of the attributes that HTCondor adds to a job are arcane and uninteresting for us now. But here are some examples of common, interesting attributes taken directly from `condor_q` output (except with some line breaks added to the `Requirements` attribute):

```
MyType = "Job"
Err = "sleep.err"
UserLog = "/home/cat/1-monday-2.1-queue/sleep.log"
JobUniverse = 5
Requirements = ( IsOSGSchoolSlot =?= true ) &&
        ( TARGET.Arch == "X86_64" ) &&
        ( TARGET.OpSys == "LINUX" ) &&
        ( TARGET.Disk >= RequestDisk ) &&
        ( TARGET.Memory >= RequestMemory ) &&
        ( TARGET.HasFileTransfer )
ClusterId = 2420
WhenToTransferOutput = "ON_EXIT"
Owner = "cat"
CondorVersion = "$CondorVersion: 8.5.5 May 03 2016 BuildID
   : 366162 $"
Out = "sleep.out"
Cmd = "/bin/sleep"
Arguments = "120"
```

Note: Attributes are listed in no particular order and may change from time to time. Do not assume anything about the order of attributes in `condor_q` output.

See what you can find in a job ClassAd from your own job.

1. Using a previous exercise, submit a `sleep` job
2. Before the job executes, capture its ClassAd and save to a file

```
$ condor_q -l job-id > classad-1.txt
```

3. After the job starts execution but before it finishes, capture its ClassAd again and save to a file:

```
$ condor_q -l job-id > classad-2.txt
```

Now examine each saved ClassAd file. Here are a few things to look for:

- Can you find attributes that came from your submit file? (E.g., JobUniverse, Cmd, Arguments, Out, Err, UserLog, and so forth)

- Can you find attributes that could have come from your submit file, but that HTCondor added for you? (E.g., Requirements)

How many of the following attributes can you guess the meaning of?

- DiskUsage
- ImageSize
- BytesSent
- JobStartDate — what format is the value in? (Hint: The HTCondor developers are primarily trained in the Unix way of doing things.)
- JobStatus

3.10.3 Why Is My Job Not Running?

Sometimes, you submit a job and it just sits in the queue in Idle state, never running. It can be difficult to figure out why a job never matches and runs. Fortunately, HTCondor can give you some help.

To ask HTCondor why you job is not running, add the `-better-analyze` option to `condor_q` for the specific job. For example, for job ID 2423.0, the command is:

```
$ condor_q -better-analyze 2423.0
```

Of course, replace the job ID with your own.

Let's submit a job that will never run and see what happens. Here is the submit file to use:

```
universe = vanilla
executable = /bin/hostname
output = norun.out
error = norun.err
log = norun.log
should_transfer_files = YES
when_to_transfer_output = ON_EXIT
request_memory = 2TB
queue
```

(Do you see what I did?)

1. Save and submit this file
2. Run `condor_q -better-analyze` on the job ID

There is a lot of output, but a few items are worth highlighting. Here is a sample from my own job (with many lines left out):

```
-- Schedd: htcondor-0.os-internal : <10.10.0.120:9618?...
The Requirements expression for job 15.000 is

    ( TARGET.Arch == "X86_64" ) && ( TARGET.OpSys == "
   LINUX" ) && ( TARGET.Disk >= RequestDisk ) && ( TARGET.
   Memory >= RequestMemory ) && ( TARGET.HasFileTransfer )

Job 15.000 defines the following attributes:

    DiskUsage = 17
    RequestDisk = DiskUsage
    RequestMemory = 2097152

The Requirements expression for job 15.000 reduces to
   these conditions:

         Slots
Step    Matched  Condition
-----  --------  ---------
[0]           16  TARGET.Arch == "X86_64"
[1]           16  TARGET.OpSys == "LINUX"
[3]           16  TARGET.Disk >= RequestDisk
[5]            0  TARGET.Memory >= RequestMemory

No successful match recorded.
Last failed match: Tue Aug 29 07:28:09 2017

Reason for last match failure: no match found

015.000:  Run analysis summary ignoring user priority.  Of
    16 machines,
      0 are rejected by your job's requirements
      0 reject your job because of their own requirements
     16 are exhausted partitionable slots
      0 match and are already running your jobs
      0 match but are serving other users
      0 are available to run your job

WARNING:  Be advised:
   No machines matched the jobs's constraints
```

Toward the top, `condor_q` said that it considered 16 "machines" (really, slots) and **all** 16 of them were rejected as the partitionable slots were exhausted. In other words, I am asking for something that is not available. But what? If you look at the list of requirements, you will see that the requirement that was matched by zero slots was that TARGET.Memory >= RequestMemory. TARGET here is a namespace of the target machine in the attempted matchmaking. Therefore we can see that for none of the target machines was the available memory greater or equal to the requested 2TB.

3.10.4 Automatic Formatting Output (Optional)

There is a way to format output from `condor_q` with the `-autoformat` or `-af` option. In this case, HTCondor decides for you how to format the data you ask for from job ClassAd(s). (To tell HTCondor how to format this information, yourself, you could use the `-format` option, which we're not covering.)

To use autoformatting, use the `-af` option followed by the attribute name, for each attribute that you want to output:

```
$ condor_q -af Owner -af ClusterId -af Cmd
bejones 16 /share/test.sh
cat 17 /bin/sleep
cat 18 /bin/sleep
```

Bonus Question: If you wanted to print out the `Requirements` expression of a job, how would you do that with `-af`? Is the output what you expected? (HINT: for ClassAd attributes like "Requirements" that are long expressions, instead of simple values, you can use `-af:r` to view the expressions, instead of what it's current evaluation.)

3.11 Explore condor status

3.12 Explore condor_status

The goal of this exercise is try out some of the most common options to the `condor_status` command, so that you can view slots effectively.

The main part of this exercise should take just a few minutes, but if you have more time later, come back and work on the extension ideas at the end to become a `condor_status` expert!

3.12.1 Selecting Slots

The `condor_status` program has many options for selecting which slots are listed. You've already learned the basic `condor_status` and the `condor_status -compact`variation (which you may wish to retry now, before proceeding).

Another convenient option is to list only those slots that are available now:

```
$ condor_status -avail
```

Of course, the individual execute machines only report their slots to the collector at certain time intervals, so this list will not reflect the up-to-the-second reality of all slots. But this limitation is true of all `condor_status` output, not just with the `-avail` option.

Similar to `condor_q`, you can limit the slots that are listed in two easy ways. To list just the slots on a specific machine:

```
$ condor_status <hostname>
```

For example, if you want to see the slots on `htcondor-10.os-internal`:

```
$ condor_status htcondor-10.os-internal
Name                              OpSys      Arch   State
   Activity LoadAv Mem   ActvtyTime

slot1@htcondor-10.os-internal LINUX      X86_64 Unclaimed
   Idle      0.000 3790  0+17:19:50

                     Machines Owner Claimed Unclaimed
   Matched Preempting  Drain

        X86_64/LINUX        1     0       0         1
     0          0      0

               Total        1     0       0         1
     0          0      0
```

To list a specific slot on a machine:

```
$ condor_status <slot>@<hostname>
```

Note however that the lab is configured to use "partitionable slots". This means that there is one main slot (slot1) which covers the whole resources of the machine. When requests are matched, they are carved out of the partitionable slot, ie slot1_1@<hostname>.

Note: You can name more than one hostname, slot, or combination thereof on the command line, in which case slots for **all** of the named hostnames and/or slots are listed.

Let's get some practice using `condor_status` selections!

1. List all slots in the pool — how many are there total?
2. Practice using all forms of `condor_status` that you have learned:
   - List the available slots
   - List the slots on a specific machine
   - List a specific slot from that machine
   - Try listing the slots from a few (but not all) machines at once
   - Try using a mix of hostnames and slot IDs at once

3.12.2 Viewing a Slot ClassAd

Just as with `condor_q`, you can use `condor_status` to view the complete ClassAd for a given slot (often confusingly called the "machine" ad):

```
$ condor_status -l <hostname>
```

Because slot ClassAds may have 150–200 attributes (or more!), it probably makes the most sense to show the ClassAd for a single slot at a time, as shown above.

Here are some examples of common, interesting attributes taken directly from `condor_status` output:

```
Name = "slot1@htcondor-10.os-internal"
NextFetchWorkDelay = -1
NumDynamicSlots = 0
NumPids = 0
OpSys = "LINUX"
OpSysAndVer = "CentOS7"
OpSysLegacy = "LINUX"
OpSysLongName = "CentOS Linux release 7.3.1611 (Core)"
OpSysMajorVer = 7
OpSysName = "CentOS"
```

As you may be able to tell, there is a mix of attributes about the machine as a whole (hence the name "machine ad") and about the slot in particular.

Go ahead and examine a machine ClassAd now.

3.12.3 Viewing Slots by ClassAd Expression

Often, it is helpful to view slots that meet some particular criteria. For example, if you know that your job needs a lot of memory to run, you may want to see how many high-memory slots there are and whether they are busy. You can filter the list of slots like this using the `-constraint` option and a ClassAd expression.

Now, all the machines in the lab are the same, so it's hard to reduce the number we have with a sensible query. Here is an example using a regexp on the machine name to reduce the number:

```
$ condor_status -const 'OpSysAndVer =?= "CentOS7" &&
   regexp("htcondor-1\d", Machine)'
```

Note: Be very careful with using quote characters appropriately in these commands. In the example above, the single quotes (`'`) are for the shell, so that the entire expression is passed to `condor_status` untouched, and the double quotes (`"`) surround a string value within the expression itself.

If you are interested in learning more about writing ClassAd expressions, look at section 4.1 and especially 4.1.4 of the HTCondor Manual. This is definitely advanced material, so if you do not want to read it, that is fine. But if you do, take some time to practice writing expressions for the `condor_status -constraint`command.

Note: The `condor_q` command accepts the `-constraint` option as well! As you might expect, the option allows you to limit the jobs that are listed based on a ClassAd expression.

3.12.4 Formatting Output (Optional)

The `condor_status` command accepts the same `-format` (`-f`) and `-autoformat` (`-af`) options that `condor_q` accepts, and the options have the same meanings in both commands. Of course, the attributes available in machine ads may differ

from the ones that are available in job ads. Use the HTCondor Manual or look at individual slot ClassAds to get a better idea of what attributes are available.

3.13 Work with input and output files

3.14 Work With Input and Output Files

The goal of this exercise is make input files available to your job on the execute machine, and return output files back. This small change significantly adds to the kinds of jobs that you can run.

3.14.1 Viewing a Job Sandbox

Before you learn to transfer files to and from your job, it is good to understand a bit more about the environment in which your job runs. When the HTCondor `starter`process prepares to run your job, it creates a new directory for your job and all of its files. We call this directory the *job sandbox*, because it is your job's private space to play. Let's see what is in the job sandbox for a very simple job with no special input or output files.

1. Save the script below in a file named `sandbox.sh`:

```
#!/bin/sh
echo 'Date:    ' `date`
echo 'Host:    ' `hostname`
echo 'System:  ' `uname -spo`
echo 'OS info: ' `cat /etc/redhat-release`
echo 'Sandbox: ' `pwd`
echo
ls -alF
```

2. Create a submit file for this script and submit it
3. When the job finishes, look at the contents of the output file

In the output file, note the `Sandbox:` line: That is the full path to your job sandbox for the run. It was created just for your job, and it was removed as soon as your job finished.

Next, look at the output that appears after the `Sandbox:` line; it is the output from the `ls` command in the script. It shows all of the files in your job sandbox, as they existed at the end of the execution of `sandbox.sh`. The files are:

So, HTCondor wrote copies of the job and machine ads (for use by the job, if desired), transferred your executable (`sandbox.sh`), renamed it (`condor_exec.exe`), ran it, and saved its standard output and standard error into files. Notice that

your submit file, which was in the same directory on the submit machine as your executable, was **not** transferred, nor were any other files that happened to be in directory with the submit file.

Now that we know something about the sandbox, we can transfer more files to and from it.

3.14.2 Running a Job With Input Files

Next, you will run a job that requires an input file. As with all previous examples, you will tell HTCondor to transfer files to the sandbox (`should_transfer_files = YES`). Remember, the initial job sandbox contains only the renamed job executable and nothing else from your directory on the submit machine. You must tell HTCondor explicitly about every other file to transfer to the sandbox. Fortunately, this is easy.

Here is a simple Python script that takes the name of an input file (containing one word per line) from the command line, counts the number of times each (lowercased) word occurs in the text, and prints out the final list of words and their counts.

```
#!/usr/bin/env python

import os
import sys

if len(sys.argv) != 2:
    print 'Usage: %s DATA' % (os.path.basename(sys.argv
   [0]))
    sys.exit(1)
input_filename = sys.argv[1]

words = {}

my_file = open(input_filename, 'r')
for line in my_file:
    word = line.strip().lower()
    if word in words:
        words[word] += 1
    else:
        words[word] = 1
my_file.close()

for word in sorted(words.keys()):
    print '%8d %s' % (words[word], word)
```

1. Save the Python script in a file named freq.py
2. Download the input file for the script (263k lines, ~1.4mb) and save it to your submit directory

```
$ wget http://bejones.web.cern.ch/bejones/words.txt
```

3. Create a basic submit file for the `freq.py` executable
4. Add a line to tell HTCondor to transfer the input file:

```
transfer_input_files = words.txt
```

   As with all submit file commands, it does not matter where this line goes. I usually group it with the other file transfer commands.

5. Do not forget to add a line to name the input file as the argument to the Python script
6. Submit the job, wait for it to finish, and check the output!

If things do not work the first time, keep trying! At this point in the exercises, we are telling you less and less explicitly how to do steps that you have done before. If you get stuck, ask!

Note: If you want to transfer more than one input file, list all of them on a single `transfer_input_files` command, separated by commas. For example, if there are three input files:

```
transfer_input_files = file1.txt, file2.txt, file3.txt
```

Extra Challenge

Many standard command-line program operate on input files. For example, the `cat` command can takes one or more input files as arguments, printing to standard output each file in order. Other common commands that take input files as arguments are `grep`, `diff`, and `sort`.

Using commands like these, or others that you know and that are readily available, create one or more submit files that take input files and produce output. If you are using a command that is **not** contained in your submit directory, be sure to put its complete path in your `executable` line; use the `which` command to find the paths of standard programs. Also be sure to set the `arguments` line correctly for each program.

3.14.3 Transferring Output Files

So far, we have relied on programs that send their output to the standard output and error streams, which HTCondor captures, saves, and returns back to the submit directory. But what if your program writes one or more files for its output? How do you tell HTCondor to bring them back?

Let's start by exploring what happens to files that a jobs creates in the sandbox. We will use a very simple method for creating a new file: We will copy an input file to another name.

1. Find or create a small input file (it is fine to use any small file from a previous exercise)
2. Create a submit file that transfers the input file and copies it to another name (as if doing `/bin/cp input.txt output.txt` on the command line)
   - Make the output filename different than any filenames that are in your submit directory
   - What is the `executable` line?
   - What is the `arguments` line?
   - How do you tell HTCondor to transfer the input file?
   - As always, use `output`, `error`, and `log` filenames that are different from previous exercises
3. Submit the job and wait for it to finish

What happened? Can you tell what HTCondor did with your output file, after it was created in the job sandbox? Look carefully at the list of files in your submit directory now...

3.14.4 Transferring Specific Output Files

As you saw in the last exercise, by default HTCondor transfers files that are created in the job sandbox back to the submit directory when the job finishes. In fact, HTCondor will also transfer back **changed** input files, too. But, this only works for files that are in the top-level sandbox directory, and **not** for ones contained in subdirectories.

What if you want to bring back only **some** output files, or output files contained in subdirectories?

Here is a simple shell script that creates several files, including a copy of an input file in a new subdirectory:

```
#!/bin/sh
if [ $# -ne 1 ]; then echo "Usage: $0 INPUT"; exit 1; fi
date > output-timestamp.txt
cal > output-calendar.txt
mkdir subdirectory
cp $1 subdirectory/backup-$1
```

First, let's confirm that HTCondor does not bring back the output file in the subdirectory:

1. Save the shell script in a file named `output.sh`
2. Write a submit file that transfers an input file and runs `output.sh` on it
3. Submit the job, wait for it to finish, and examine the contents of your submit directory

Suppose you decide that you want only the timestamp output file and all files in the subdirectory, but not the calendar output file. You can tell HTCondor to transfer these specific files:

```
transfer_output_files = output-timestamp.txt, subdirectory
   /
```

Note: See the trailing slash (`/`) on the subdirectory? That tells HTCondor to transfer back **the files** contained in the subdirectory, but not the directory itself; the files will be written directly into the submit directory itself. If you want HTCondor to transfer back an entire directory, leave off the trailing slash.

1. Remove all output files from the previous run, including `output-timestamp.txt` and `output-calendar.txt`
2. Copy the previous submit file that ran `output.sh` and add the `transfer_output_files` line from above
3. Submit the job, wait for it to finish, and examine the contents of your submit directory

Did it work as you expected?

3.14.5 Thinking About Progress So Far

At this point, you can do just about everything that you need in order to run jobs on a local HTC pool. You can identify the executable, arguments, and input files, and you can get output back from the job. This is a big achievement!

In some ways, everything after this exercise just makes it easier to run certain kinds of jobs and deal with certain kinds of situations.

3.15 Use queue N, $(Cluster) and $(Process)

3.16 Use queue N, $(Cluster), and $(Process)

The goal of this exercise is to learn to submit many jobs from a single `queue` statement, and then to control filenames and arguments per job.

3.16.1 Submitting Many Jobs With One Submit File

Suppose you have a program that you want to run many times. The program takes an argument, and you want to change the argument for each run of the program. With what you know so far, you have a couple of choices (assuming that you cannot change the job itself to work this way):

- Write one submit file; submit one job, change the argument in the submit file, submit another job, change the submit file, ...

- Write many submit files that are nearly identical except for the program argument

Neither of these options seems very satisfying. Fortunately, we can do better with HTCondor.

3.16.2 Running Many Jobs With One queue Statement

Here is a C program that uses a simple stochastic (random) method to estimate the value of π — feel free to try to figure out the method from the code, but it is not critical for this exercise. The single argument to the program is the number of samples to take. More samples should result in better estimates!

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>

int main(int argc, char *argv[])
{
  struct timeval my_timeval;
  int iterations = 0;
  int inside_circle = 0;
  int i;
  double x, y, pi_estimate;

  gettimeofday(&my_timeval, NULL);
  srand48(my_timeval.tv_sec ^ my_timeval.tv_usec);

  if (argc == 2) {
    iterations = atoi(argv[1]);
  } else {
    printf("usage: circlepi ITERATIONS\n");
    exit(1);
  }

  for (i = 0; i < iterations; i++) {
    x = (drand48() - 0.5) * 2.0;
    y = (drand48() - 0.5) * 2.0;
    if (((x * x) + (y * y)) <= 1.0) {
      inside_circle++;
    }
  }
  pi_estimate = 4.0 * ((double) inside_circle / (double)
   iterations);
  printf("%d iterations, %d inside; pi = %f\n", iterations
   , inside_circle, pi_estimate);
  return 0;
}
```

1. In a new directory for this exercise, save the code to a file named `circlepi.c`
2. Compile the code

```
$ gcc -static -o circlepi circlepi.c
```

3. If there are errors, check the file contents and compile command carefully, otherwise see the instructors
4. Test the program with just a few samples:

```
$ ./circlepi 10000
```

Now suppose that you want to run the program many times, to produce many estimates. This is exactly what a statement like `queue 3` is useful for. Let's see how it works.

1. Write a normal submit file for this program
   - Pass 1 billion (`1000000000`) as the command line argument to `circlepi`
   - Remember to use `queue 3` instead of just `queue`
2. Submit the file Note the slightly different message from `condor_submit`:

```
3 job(s) submitted to cluster NNNN.
```

3. Before the jobs execute, look at the job queue to see the multiple jobs

Look at `condor_q -nobatch` and note the output. Look in particular at the ID column. You should see that the jobs have the same cluster ID, but have job IDs incremented from 0. Do you remember how to ask HTCondor to list all the jobs from one cluster?

3.16.3 Using queue N With Output

When all three jobs in your single cluster are finished, examine the resulting files.

- What is in the output file?
- What is in the error file (hopefully nothing!)?
- What is in the log file? Look carefully at the job IDs in each event.
- Is this what you expected? Is it what you wanted?

3.16.4 Using $(Process) to Distinguish Jobs

As you saw with the experiment above, we need a way to separate output (and error) files *per job that is queued*, not just for the whole cluster of jobs. Fortunately, HTCondor has a way to separate the files easily.

When processing a submit file, HTCondor defines and uses a special variable for the process number of each job. If you write `$(Process)` in a submit file, HTCondor will replace it with the process number of the job, independently for each job that is queued. For example, you can use the `$(Process)` variable to define a separate output file name for each job. Suppose the following two lines are in a submit file:

```
output = my-output-file-$(Process).out
queue 10
```

Even though the `output` filename is defined only once, HTCondor will create separate output filenames for each job:

Let's see how this works for our program that estimates π.

1. In your submit file, change the definitions of `output` and `error` to use `$(Process)`, in a way that is similar to the example above
2. Remove any output, error, and log files from previous runs
3. Submit the updated file

When all three jobs are finished, examine the resulting files again.

- How many files are there of each type? What are their names?
- Is this what you expected? Is it what you wanted from the π estimation process?

3.16.5 Using $(Cluster) to Separate Files Across Runs

With `$(Process)`, you can get separate output (and error) filenames for each job within a run. However, the next time you submit the same file, all of the output and error files are overwritten by new ones created by the new jobs. Maybe this is the behavior that you want. But sometimes, you may want to separate files by run, as well.

In addition to `$(Process)`, there is also a `$(Cluster)` variable that you can use in your submit files. It works just like `$(Process)`, except it is replaced with the cluster number of the entire submission. Because the cluster number is the same for all jobs within a single submission, it does not separate files by job within a submission. But when used **with** `$(Process)`, it can be used to separate files by run. For example, consider this `output` statement:

```
output = my-output-file-$(Cluster)-$(Process).out
```

For one particular run, it might result in output filenames like this:

If you like, change your submit file from the previous exercise to use both `$(Cluster)` and `$(Process)`. Submit your file twice to see the separate files for each run. Be careful how many jobs you run total, as the number of output files grows quickly!

3.16.6 Using $(Process) and $(Cluster) in Other Statements

The `$(Cluster)` and `$(Process)` variables can be used in any submit file statement, although they are useful in some kinds of statements more than others. For instance, it is hard to imagine a truly good reason to use the `$(Process)` variable in a `rank` statement (i.e., for preferring some execute slots over others), and in general the `$(Cluster)` variable often makes little sense to use.

But in some situations, the `$(Process)` variable can be very helpful. Common uses are in the following kinds of statements — can you think of a scenario in which each use might be helpful?

- `log`
- `transfer_input_files`
- `transfer_output_files`
- `arguments`

Unfortunately, HTCondor does not let you perform math on the `$(Process)` number when using it. So, for example, if you use `$(Process)` as a numeric argument to a command, it will always result in jobs getting the arguments 0, 1, 2, and so on. If you have control over your program and the way in which it uses command-line arguments, then you are fine. Otherwise, you might need to transform the `$(Process)` numbers into something more appropriate using a **wrapper script**

Chapter 4

Credits

4.1 Original authors

The slides were created by the HTCondor team at the University of Wisconsin
The exercises have been very slightly adapted from the OSG school

Chapter 5

Text and image sources, contributors, and licenses

5.1 Text

- **Course:HTCondor/User Tutorial/Slides** *Source:* https://en.wikitolearn.org/Course%3AHTCondor/User_Tutorial/Slides?oldid=9605 *Contributors:* Bendylan
- **Course:HTCondor/DAG Tutorial/Slides** *Source:* https://en.wikitolearn.org/Course%3AHTCondor/DAG_Tutorial/Slides?oldid=9588 *Contributors:* Bendylan
- **Course:HTCondor/Exercises/Login and have a look around** *Source:* https://en.wikitolearn.org/Course%3AHTCondor/Exercises/Login_and_have_a_look_around?oldid=9690 *Contributors:* Bendylan
- **Course:HTCondor/Exercises/Basic HTCondor commands** *Source:* https://en.wikitolearn.org/Course%3AHTCondor/Exercises/Basic_HTCondor_commands?oldid=9626 *Contributors:* Bendylan
- **Course:HTCondor/Exercises/Run jobs!** *Source:* https://en.wikitolearn.org/Course%3AHTCondor/Exercises/Run_jobs!?oldid=9629 *Contributors:* Bendylan
- **Course:HTCondor/Exercises/Log files** *Source:* https://en.wikitolearn.org/Course%3AHTCondor/Exercises/Log_files?oldid=9694 *Contributors:* Bendylan
- **Course:HTCondor/Exercises/Resource requirements** *Source:* https://en.wikitolearn.org/Course%3AHTCondor/Exercises/Resource_requirements?oldid=9751 *Contributors:* Bendylan
- **Course:HTCondor/Exercises/Remove jobs from the queue** *Source:* https://en.wikitolearn.org/Course%3AHTCondor/Exercises/Remove_jobs_from_the_queue?oldid=9703 *Contributors:* Bendylan
- **Course:HTCondor/Exercises/Explore condor q** *Source:* https://en.wikitolearn.org/Course%3AHTCondor/Exercises/Explore_condor_q?oldid=9725 *Contributors:* Bendylan
- **Course:HTCondor/Exercises/Explore condor status** *Source:* https://en.wikitolearn.org/Course%3AHTCondor/Exercises/Explore_condor_status?oldid=9726 *Contributors:* Bendylan
- **Course:HTCondor/Exercises/Work with input and output files** *Source:* https://en.wikitolearn.org/Course%3AHTCondor/Exercises/Work_with_input_and_output_files?oldid=9730 *Contributors:* Bendylan
- **Course:HTCondor/Exercises/Use queue N, $(Cluster) and $(Process)** *Source:* https://en.wikitolearn.org/Course%3AHTCondor/Exercises/Use_queue_N%2C_%24(Cluster)_and_%24(Process)?oldid=9731 *Contributors:* Bendylan
- **Course:HTCondor/Credits/Original authors** *Source:* https://en.wikitolearn.org/Course%3AHTCondor/Credits/Original_authors?oldid=9701 *Contributors:* Bendylan

5.2 Images

5.3 Content license

- [Project:Copyright Creative Commons Attribution Share Alike 3.0 & GNU FDL]
- Creative Commons Attribution-Share Alike 3.0