HTCondor@Biochem

Using HTCondor on the Biochemistry Computational Cluster
v1.2.0

Jean-Yves Sgro
Note: The HTCondor software was known as 'Condor' from 1988 until its name changed in 2012.
Table of Contents

Foreword ............................................................................................................................. 5
What is HTCondor? .................................................................................................................... 7
Cluster access overview ......................................................................................................... 8
  1. Text-based access .......................................................................................................... 8
  2. No graphical interface .................................................................................................... 8
  3. VPN access ................................................................................................................... 8
  4. Login info ..................................................................................................................... 9
  5. Linux OS: Scientific Linux ............................................................................................ 9
Before you begin ..................................................................................................................... 10
The BCC Linux cluster ......................................................................................................... 11
  6. Overview ..................................................................................................................... 11
  7. Connecting to the cluster ............................................................................................... 12
  8. Disk space and home directory ..................................................................................... 12
  9. Process ......................................................................................................................... 13
  10. Getting ready ............................................................................................................. 13
  11. File transfers .............................................................................................................. 13
  12. Beyond local cluster: Flocking .................................................................................. 14
QuickStart .............................................................................................................................. 15
  1. This section assumes that: ......................................................................................... 15
  2. Connect and set-up a working space .......................................................................... 16
  3. Create an simple executable test file .......................................................................... 16
  4. Create a simple submit file ......................................................................................... 17
  5. Submit the job ............................................................................................................. 17
  6. Check output .............................................................................................................. 18
  7. Conclusion ................................................................................................................... 18
Resources / Help .................................................................................................................. 19
  On-line resources: ........................................................................................................... 19
HTCondor concepts .............................................................................................................. 20
  1. Class Ads ................................................................................................................... 20
  2. Universes .................................................................................................................... 22
  3. Steps before running a job ......................................................................................... 23
  4. Requirements and Rank ............................................................................................. 23
  5. File transfer ................................................................................................................. 24
    5.1. Jobs with Shared File System .............................................................................. 24
    5.2. File Transfer Mechanism ................................................................................... 24
5.3. File Paths for file transfer ................................................................. 25
6. Managing jobs .................................................................................. 26
7. Job completion .................................................................................. 27

Library dependencies ........................................................................... 30
1. FORTRAN 77: Hello World example ............................................... 30
2. FORTRAN 95: gfortran ................................................................. 32
3. Compiling .......................................................................................... 32
4. Standard Universe ........................................................................... 32
5. Compiling R ...................................................................................... 33
   5.1. Download R .................................................................................. 33
   5.2. Install ........................................................................................... 33
   5.2.1. Note: Problem with Zlib in newer version.............................. 34
   5.3. R additional packages .............................................................. 34

File transfer from file.biochem.wisc.edu ........................................... 36
1. Moving files on and off the cluster: ............................................... 36
This tutorial is meant to learn about using HTCondor on the Biochemistry Computational Cluster (BCC).

The BCC runs under Linux and therefore all examples will be shown for this operating system.

As a general guide, some marks are placed along most of the tutorials to indicate action to be taken by the reader:

$ command to be typed

often typewritten styled text illustrates a software output.

This is a very short description of the Biochemistry cluster and HTCondor.

Further information about creating job submission files should be studied within the HTCondor online manual (see On-line resources: on page 19.)
What is HTCondor?

HTCondor is a “scheduler” system that dispatches compute jobs to one and up to a very large number of “compute nodes” that actually perform the calculations.

HTCondor is developed by the Computer Sciences Department at the University of Wisconsin-Madison. The HTCondor web page\(^1\) contains a long description. Here is the first, summary-like paragraph:

\[
\text{HTCondor is a specialized workload management system for compute-intensive jobs. Like other full-featured batch systems, HTCondor provides:} \\
\text{- a job queueing mechanism,} \\
\text{- scheduling policy,} \\
\text{- priority scheme,} \\
\text{- resource monitoring, and} \\
\text{- resource management.} \\
\]

\[\text{Users submit their serial or parallel jobs to HTCondor, HTCondor places them into a queue, chooses when and where to run the jobs based upon a policy, carefully monitors their progress, and ultimately informs the user upon completion.}\]

\[\text{Note: Using HTCondor is the only approved method for performing high throughput computing on the BCC Linux cluster.}\]

Jobs have to be ready to be processed by HTCondor as \textbf{jobs cannot be interactive} on the cluster.

\(^1\) http://research.cs.wisc.edu/htcondor/description.html
Cluster access overview

The Biochemistry Computational Cluster (BCC) is a High Throughput Computing (HTC) environment within the UW-Madison Biochemistry Department.

1. Text-based access

The BCC cluster must be accessed via secure shell (ssh) with a **text-based Terminal** from a local computer. For example:

- **Macintosh**: /Applications/Terminal
- **Linux**: Terminal or Shell
- **Windows**: install free software e.g. PuTTY or MobaXterm

**Note**: The "submit" node is the only one accessible to users, and jobs will be passed on to the larger hardware portions of the cluster that are not accessible directly to users.

2. No graphical interface

**Important note**: there is no graphical user interface (GUI) available in any form as the X11 graphical base is not installed on the operating system.

Therefore, the only mode of action is via text-based access as described above.

**Note**: The `ssh` modifier -Y would not allow GUI either

3. VPN access

Access from outside the Biochemistry department requires a VPN connection:
### 4. Login info

From a text-based terminal use `ssh` to login:

```
ssh myname@submit.biochem.wisc.edu
```

where `mynam`e is your **Biochem Username** and your Biochem **Password** will be required after you press return.

**Note:** IT may change the login mode to your UW NetID in the future.

### 5. Linux OS: Scientific Linux

There are many Linux versions. On the BCC the version installed is called “Scientific Linux” which is derived from “Red Hat Linux.”

The version installed can be obtained with the command:

```
cat /etc/redhat-release
```

```
Scientific Linux release 6.7 (Carbon)
```

The `uname` command can be used to obtain further information with `-a` printing all info:

```
$ uname -a
```

```
Linux submit.biochem.wisc.edu 2.6.32-573.26.1.el6.x86_64 #1 SMP Tue May 3 14:22:07 CDT 2016 x86_64 x86_64 x86_64 GNU/Linux
```

Noteworthy is **x86_64** which means it is a 64 bit system and **el6** means “Enterprise Linux version 6” which is derived from the Enterprise Linux 6 version from Red Hat.
Using HTCondor requires knowledge of the Linux/Unix Shell bash command-line and information about how the cluster is set-up.

There are many preparation steps that will take time to organize. The first question to ask is “Why do I need to use HTCondor?” and validate for yourself the reasons why that would make things better for your computation.

If you decide HTCondor will be useful, you will then need to evaluate how the software you want to use can work on the BCC Linux cluster.

Jobs have to be able to run in “**batch**” mode *i.e. non-interactively*. This means that you need to know what the software will require to run, such as input file(s) or ancillary data files.

---

**Summary of what you need or need to know.**

This will be reviewed further:

- **Username**: Your Biochemistry Username and password
- **Login to**: submit.biochem.wisc.edu
- **Cluster**: Understand that /scratch is the main work place
- **bash shell**: Understand commands such as `cd`, `ls`, `mkdir`, etc.
- **Software**: Understand all requirements of the software to run
The BCC Linux cluster

6. Overview

The Biochemistry Computational Cluster (BCC) is a High Throughput Computing (HTC) environment within the UW-Madison Biochemistry Department.

The cluster can be described as a set of 10 computers connected to each other and sharing a common disk space allocation. As these are not really computers with a keyboard, a mouse and a screen they are typically referred to as “nodes.”

Only one node is accessible directly to users to submit jobs to the other 9 nodes. This node is called the “Submit Node” and also plays a role in job control. One could view the set-up in this simplified hierarchy:

Therefore all jobs and interaction have to go through the Submit node that will dispatch jobs, or job portions to other nodes. This means that the required calculation have to run in batch, non-interactive mode.

The Submit node controls 2 Tb of disk space made available and shared with the other nodes. Each compute node also has a 240 Gb of space to use while performing calculations and is therefore not useable as storage space.

The compute nodes are also equipped with state of the art graphics chips (GPU) that can be specifically requested for calculations by software that are gpu-aware and can greatly accelerate calculations.

The hardware specific data for the cluster is as follows:
Submit Node

- 2 x Intel Xeon E5-2650v2 8-Core 2.60 GHz (3.4GHz Turbo)
- Over 2TB of SSD based RAID 5 scratch disk space shared with each BCC computational node
- 128 GB DDR3 1866 ECC/REG Memory
- 10G Ethernet networking

9 x Dedicated Computation Node

- 2 x Intel Xeon E5-2680v2 10-Core 2.80 GHz (3.60GHz Turbo)
- 64 GB DDR3 1866 ECC/REG Memory
- 1 x NVIDIA Tesla K20M GPU
- 1 x 240 GB SSD
- 10G Ethernet networking

7. Connecting to the cluster

Only the “Submit Node” is accessible to users via a text-based terminal connection with the secure shell command:

Login with `ssh myname@submit.biochem.wisc.edu` where `myname` is your Biochem Username and your Biochem Password will be required after you press return.

8. Disk space and home directory

The default home directory is mapped according to the user name and has a long, default name reflecting how the Biochem user name can be “mapped” to the BCC cluster.

The default $HOME directory will be mapped as:

`/Network/Servers/file.biochem.wisc.edu/Volumes/BioSAN/Users/myname`

However, it is important to understand that the file server `file.biochem.wisc.edu` or `fs.biochem.wisc.edu` are NOT accessible from within the cluster as “mounted” volumes (see further section.)

**Important note:** the default $HOME directory should NOT be used as the primary location for storage, or for HTCondor job submission.

The main work area is called `/scratch` and should be used for all jobs.
HTCondor nodes are set-up in similar ways, and typically they all understand the shared disk space known as `/scratch`

Each user should therefore create working directory within `/scratch` and work from there rather than the default `$HOME` directory.

9. Process

The fundamental process consists of submitting a “job file” that contains information on how to run the software that needs to be run with optional input and ancillary files.

Typically one would need to create a shell script (*.sh) that can run the desired software, and then create another, submit script (*.sub) that would submit the shell script to HTCondor.

Provisions exists within the *.sub file to transfer all created files after the run has completed successfully.

10. Getting ready

To get ready you need to evaluate what we just saw in Process paragraph above backwards:

11. File transfers

Part of the HTCondor method is to transfer files (sometimes all files and software binaries) to a temporary directory, run the job and copy the output files back to your permanent working directory (e.g. on `/scratch`) upon comple-
HTCondor runs on a temporary directory that changes every time. For example this directory could be called:

```
TMPDIR=/var/lib/condor/execute/dir_30471
```

This directory and all its files will disappear once the job is done. For the next job the directory would have a different name.

When the job starts, all the necessary files (hello.sh, hello.sub) are transferred to a temporary directory (dark arrow.) When the job is done, output files created by the job are transferred back to the originating (or specified) directory (white arrow.)

### 12. Beyond local cluster: Flocking

When jobs are big, it may be useful to access computers that are beyond that of the Biochemistry cluster itself. This can be done safely even with proprietary software as files are invisible to others and cannot be copied.

This is a special case called “Flocking” and it may be necessary to adjust the submit command file to either be more “generic” or provide more details of files to be transferred, for example files that are not standard on other systems.
QuickStart

This QuickStart section is inspired by the online QuickStart option shown on the HTCondor web pages (on page 19.)

1. This section assumes that:

<table>
<thead>
<tr>
<th>Assumption</th>
<th>Check / Set-up on BCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTCondor is running</td>
<td>HTCondor is installed on BCC</td>
</tr>
<tr>
<td>That you have access to a machine within the pool that may submit jobs,</td>
<td>IP Address: submit.biochem.wisc.edu</td>
</tr>
<tr>
<td>termed a submit machine</td>
<td></td>
</tr>
<tr>
<td>That you are logged in to and working on the submit machine</td>
<td>Your username is your Biochem ID and password</td>
</tr>
<tr>
<td>That your program executable, your submit description file, and any</td>
<td>You should work in the /scratch directory</td>
</tr>
<tr>
<td>needed input files are all on the file system of the submit machine</td>
<td></td>
</tr>
<tr>
<td>That your job (the program executable) is able to run without any</td>
<td>We will run a test file that complies with these requirements.</td>
</tr>
<tr>
<td>interactive input. Standard input (from the keyboard), standard output</td>
<td></td>
</tr>
<tr>
<td>(seen on the display), and standard error (seen on the display) may still</td>
<td></td>
</tr>
<tr>
<td>be used, but their contents will be redirected from/to files.</td>
<td></td>
</tr>
</tbody>
</table>

It is also assumed that you know how to converse within the line command and edit simple text files.
2. Connect and set-up a working space

We’ll follow the above table process.

First we connect and create a directory in the shared data space.

Use YOUR Biochem ID to connect – represented here as myname

You should open a text-based terminal from your local machine, and then issue the connect command: [replace myname with your login name.]

$ ssh myname@submit.biochem.wisc.edu

Then move to /scratch and create a directory with your name and another directory within to work with.

$ cd /scratch
$ mkdir myname #replace myname with e.g. YOUR ID
$ mkdir quickstart
$ cd myname/quickstart

3. Create an simple executable test file

Using the nano word processor on the cluster or a copy/paste method we now create a file to be executed by HTCondor.

$ nano hello.sh

Within the file enter the following:

#!/bin/sh

echo "Hello World"

If you are using nano, use Ctrl-X to exit from edit mode to save the file.

Now we make sure that the file is executable:

$ chmod u+x hello.sh
4. Create a simple submit file

The submit file is the file that is provided to HTCondor containing instructions for running a job. The following file is a minimal file, more information could be provided to HTCondor for more job control.

```
$ nano hello.sub
```

Then enter the following text within the file and save:

```
executable = hello.sh
should_transfer_files = Yes

output = hello.out
error = hello.err
log = hello.log

queue
```

Line 1 tells what is the file to run

Line 2 request necessary files to be transferred

Line 3 – 5 specify the name of the standard output files

Line 6 places the job in the queue so it can be run.

The following implicit assumptions are made:

- `echo` general-use Linux Bash commands are available
- the process will use standard input, standard output and standard error

5. Submit the job

We now follow the example with the cascade steps of submitting a file (hello.sub) containing information about an executable (hello.sh) that is calling on a software (echo) that will create some output files (hello.out, etc.) that will be transferred to the local directory when the job is done.
The submit command is as follows:

```
$ condor_submit hello.sub
```

## 6. Check output

The job will be queued and executed rather rapidly, transferring output files to the local directory when done:

```
$ ls
hello.err  hello.log  hello.out  hello.sh  hello.sub
```

We can verify that the job executed correctly:

```
$ cat hello.out
Hello World
```

## 7. Conclusion

This is a very simple example.

Clearly, if we were to run this example again the files that we just created would be overwritten (clobbered) by the files created by the new run.

This and many other aspects of job control can be overcome by specific HTCondor command syntax.

*The complete HTCondor manual is more than 900 pages long. Therefore the virtue of patience needs to be called upon to tackle and master using a cluster running HTCondor!*
Now that you know how to log-in and run the simplest job, here are resources to go further and learn how to use HTCondor with your own software.

**On-line resources:**

<table>
<thead>
<tr>
<th>Resource</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete manual*</td>
<td><a href="http://research.cs.wisc.edu/htcondor/manual/">http://research.cs.wisc.edu/htcondor/manual/</a></td>
</tr>
</tbody>
</table>

*You can check which manual you need by checking which version of HTCondor is installed with command: `condor_version`*

---

It is highly advised to get acquainted with some of this material before attempting any calculations on the cluster.

For general HTCondor questions contact chtc@cs.wisc.edu
For Biochemistry related questions contact jsgro@wisc.edu
For general Biochem IT/network issues contact helpdesk@biochem.wisc.edu
We have already learned perhaps the most important command which is the one use to submit a job: `condor_submit`

The list of HTCondor commands is rather long: 70. However, for most users and everyday use just a few are essential, for example to start, stop, hold, re-start, and list submitted jobs.

### 1. Class Ads

ClassAds in HTCondor are comparable to classified ads in a newspaper. Sellers advertise what they sell; buyers may advertise what they wish to buy; both buyers and sellers have specifics and conditions.

Compute nodes ClassAds actively advertise lists of attributes and resources available. For example: the type of CPU (Intel) and its speed, memory (RAM) available, operating system etc.

Therefore, jobs can be submitted with generic or very stringent requirements, via a specific syntax within the *.submit file.

For example, a user may require that the compute node be equipped with a graphical processor unit (GPU) and a minimum amount of RAM, for example 64Mb, but with a preference for 256Mb is possible. Many other requirements can be added, depending on the software to be run by the job.

ClassAds advertised by both nodes and jobs are continuously read by HTCondor that will match requests and verify that all requirements for both ClassAds are met.
The HTCondor command `condor_status` provides a summary of the ClassAds in the pool.

`condor_status -available` shows only machines which are willing to run jobs now.

`condor_status -run` shows only machines which are currently running jobs.

`condor_status -help` provides a list of many other options.

`condor_status -long` lists the machine ClassAds for all machines in the pool. But the output is very long: about 100 ClassAds per compute node. Here are a few lines from a `-long` command:

```plaintext
Machine = "cluster-0009.biochem.wisc.edu"
UpdatesLost = 0
JobStart = 1428450613
RetirementTimeRemaining = 115813
MonitorSelfAge = 2587971
MaxJobRetirementTime = MAX_PREEMPT
Disk = 1674952
Unhibernate = MY.MachineLastMatchTime != undefined
IsWakeOnLanEnabled = false
CondorVersion = "$CondorVersion: 8.2.7 Feb 09 2015 BuildID: 300022 "$
IsLocalStartd = false
RecentJobRankPreemptions = 0
ExpectedMachineGracefulDrainingBadput = 12587927
LastHeardFrom = 1428593790
HasJava = true
CUDAGlobalMemoryMb = 4800
Activity = "Busy"
TotalSlotMemory = 512
[truncated here]
```

The list contains about 100 attributes “advertised” continuously in order to match jobs with nodes.
Here are output examples (shortened by the `head` command limiting output to 5 lines)

```
$ condor_status | head -5
```

<table>
<thead>
<tr>
<th>Name</th>
<th>OpSys</th>
<th>Arch</th>
<th>State</th>
<th>Activity</th>
<th>LoadAv</th>
<th>Mem</th>
<th>ActvtyTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>slot1@cluster-0002</td>
<td>LINUX</td>
<td>X86_64</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.170</td>
<td>42887</td>
<td>30+00:18:21</td>
</tr>
<tr>
<td>slot1_10@cluster-0</td>
<td>LINUX</td>
<td>X86_64</td>
<td>Claimed</td>
<td>Busy</td>
<td>1.000</td>
<td>512</td>
<td>3+23:01:35</td>
</tr>
<tr>
<td>slot1_11@cluster-0</td>
<td>LINUX</td>
<td>X86_64</td>
<td>Claimed</td>
<td>Busy</td>
<td>1.000</td>
<td>512</td>
<td>3+23:01:35</td>
</tr>
</tbody>
</table>

```
$ condor_status -available | head -5
```

<table>
<thead>
<tr>
<th>Name</th>
<th>OpSys</th>
<th>Arch</th>
<th>State</th>
<th>Activity</th>
<th>LoadAv</th>
<th>Mem</th>
<th>ActvtyTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>slot1@cluster-0002</td>
<td>LINUX</td>
<td>X86_64</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.170</td>
<td>42887</td>
<td>30+00:18:21</td>
</tr>
<tr>
<td>slot1@cluster-0003</td>
<td>LINUX</td>
<td>X86_64</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.000</td>
<td>43911</td>
<td>30+00:30:36</td>
</tr>
<tr>
<td>slot1@cluster-0004</td>
<td>LINUX</td>
<td>X86_64</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.000</td>
<td>39303</td>
<td>30+00:22:31</td>
</tr>
</tbody>
</table>

**Summary:** ClassAds reflect the resources of compute nodes and the requirements of user jobs. HTCondor matches requirements from both.

### 2. Universes

HTCondor has several runtime environments (called a universe) from which to choose. Of the universes, two are likely choices when learning to submit a job to HTCondor: the **standard** universe and the **vanilla** universe.

Condor manual

HTCondor supports different execution environment called universe:

<table>
<thead>
<tr>
<th>Universe</th>
<th>Execution Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>Java</td>
</tr>
<tr>
<td>Vanilla</td>
<td>Scheduler</td>
</tr>
<tr>
<td>Grid</td>
<td>Local</td>
</tr>
</tbody>
</table>

**On the BCC the default universe is Vanilla** and other choices would be specified in the submit description file. Universes other than Vanilla require specific considerations that will not be mentioned in this document. While it is the default it is considered good practice to specify the universe within the submit file as the default could be changed at a later date by the system administration of the compute cluster.
3. Steps before running a job

This was discussed in a previous section (see Process on page 13) and re-viewed here in the light of information from the manual.

**Code preparation.** Jobs must be able to run in batch, non-interactive mode. A program that runs in the background will not be able to do interactive input and output. HTCondor can redirect console output (stdout and stderr) and keyboard input (stdin) to and from files for the program (these standards are part of the operating system.) Create any needed files that contain the proper keystrokes needed for software input. Make certain the software and scripts run correctly with the files.

**HTCondor Universe.** The Vanilla universe is the default. The Standard universe requires explicit recompilation of the software with HTCondor libraries and is for advanced users.

**Submit description file.** This plain text file contains details of the job to run, what software (executable) to run, and information about files to transfer, etc. The file can contain explicit requirements that HTCondor will match with compute nodes in terms of ClassAd.

**Submit the job.** The command condor_submit is used to submit the job described in the job description file.

Once submitted, HTCondor does the rest toward running the job. Monitor the job’s progress with the `condor_q` and `condor_status` commands.

You may modify the order in which HTCondor will run your jobs with `condor_prio`.

You can remove a job from the queue prematurely with `condor_rm`.

**Log file.** It is recommended to request a log file for the job within the submit file. Exit status (success or failure) and various statistics about its performances, including time used and I/O performed will be included in the log file.

4. Requirements and Rank

Using the `requirements` and `rank` commands in the submit description file is powerful, flexible and requires care. Default values are set by the `condor_submit` program if these are not defined in the submit description file.

For example, the following commands within a submit description file:
request_memory = 32
rank = Memory >= 64

require HTCondor to run the program on machines which have greater than 32 Mb of physical memory, and the rank command expresses a preference to run on machines with more than 64 Mb.

The commands can use comparison operators: <, >, <=, >=, and == are case insensitive and special comparison operators =?= and !== compare strings case sensitively.

Please refer to the complete HTCondor manual for more details on the usage of these commands.

## 5. File transfer

The HTCondor manual has more details on this subject and should also be consulted.

### 5.1. Jobs with Shared File System

HTCondor is aware of the files present in the `/scratch` directory (see Disk space and home directory on page 12) since the BCC has a shared file system to access input and output files.

Defaults requirements exist so that compute nodes can share the same data. If you place your data in e.g. `/scratch/myname/somedirectory` it should be visible by any compute node to run your job.

### 5.2. File Transfer Mechanism

While the BCC offers a shared file system, there are situations when it may still be appropriate to proceed as if that was not the case, for example if the job is very large and one wants to “flock” the job to a larger grid at CHTC or even the Open Science Grid. In this case a shared file system would not be available to computers outside of the local area.

The HTCondor file transfer mechanism permits the user to select which files are transferred and under which circumstances. HTCondor can transfer any files needed by a job from the machine where the job was submitted into a remote temporary directory on the machine where the job is to be executed.
HTCondor executes the job and transfers output back to the submitting machine. The user specifies which files and directories to transfer, and at what point the output files should be copied back to the submitting machine. This specification is done within the job’s submit description file.

Condor manual

To enable the file transfer mechanism, place two commands in the job’s submit description file: should_transfer_files and when_to_transfer_output. By default, they will be:

```
should_transfer_files = IF_NEEDED
when_to_transfer_output = ON_EXIT
```

Setting the `should_transfer_files` command explicitly enables or disables the file transfer mechanism. The command takes on one of three possible values:

- YES
- IF_NEEDED
- NO

Specifying What Files to Transfer: If the file transfer mechanism is enabled, HTCondor will transfer the following files before the job is run on a remote machine:

1. the executable, as defined with the executable command
2. the input, as defined with the input command

If the job requires other input files, the submit description file should utilize the `transfer_input_files` command as a comma-separated list.

5.3. File Paths for file transfer

The file transfer mechanism specifies file names and/or paths on both the file system of the submit machine and on the file system of the execute machine. Care must be taken to know which machine, submit or execute, is utilizing the file name and/or path.

HTCondor manual

See manual for more details. Files can also be transferred by URL (http)
6. Managing jobs

Once a job has been submitted HTCondor will attempt to find resources to run the job (match the ClassAd from the job requirements with those advertised by the compute nodes.)

Specific commands can be used to monitor jobs that have already been submitted.

A list of submitted jobs, by whom, can be obtained with:

```
condor_status -submitters
```

Job progress can be assessed with:

```
condor_q
```

This job ID provided by the previous command `condor_q` can be used to remove that job if it not longer needed, for example of the ID is 77.0 the command would be:

```
condor_rm 77.0
```

The job would be terminated and all files discarded.

Jobs can be placed on hold and then released with the commands specifying the job ID:

```
condor_hold 78.0
condor_release 78.0
```

A list of jobs in the hold state can be obtained:

```
condor_q -hold
```

or the reason for their holding:

```
condor_q -hold 78.0
```

If a job is not running, wait 5 minutes so that ClassAd have been negotiated, and then check with the command: (see more in the manual.)

```
condor_q -analyze
```

Finally, some jobs may be able to have their priority altered by the `condor_prio` command.
7. Job completion

When an HTCondor job completes, either through normal means or by abnormal termination by signal, HTCondor will remove it from the job queue. That is, the job will no longer appear in the output of `condor_q` and the job will be inserted into the job history file. Examine the job history file with the `condor_history` command. If there is a log file specified in the submit description file for the job, then the job exit status will be recorded there as well.

HTCondor manual

Statistics about the job will be included in the log file if it was requested within the submit file, as it is strongly suggested.

There are methods to tag the name of files by job name, process number, the compute node or the cluster name as a command within the submit file. For example:

```
output = $(job)_$(Cluster)_$(Process).out
```

The commands of the style `$(name)` are the method that HTCondor handles variables with proper set-up.

8. Example: environment variables

Here is a small example putting things together with a submit and executable file.

`environ.sub` submit file:

```
executable = run_environ.sh
output = run_environ_$(Cluster).out
error = run_environ_$(Cluster).err
log = run_environ_$(Cluster).log
should_transfer_files = YES
when_to_transfer_output = ON_EXIT
request_cpus = 1
queue 1
```
run_environ.sh executable file:

```bash
#!/bin/sh
echo "this is running!"
pwd > test.out
ls >> test.out
printenv >> test.out
```

submit the job:

```
$ condor_submit environ.sub

Submitting job(s).
1 job(s) submitted to cluster 177660.
```

The job runs quickly and the result files are transferred back, as shown with this list:

```
$ ls
environ.sub run_environ_177660.out
run_environ_177660.err run_environ.sh
run_environ_177660.log test.out
```

The executable file (run_environ.sh) created a standard output that captured within file run_environ_177660.out and will contain a single line with “this is running!”

The file test.out created by the executable will contain the directory name on the compute node and then the values of environment variables:

```
$ cat test.out
/var/lib/condor/execute/dir_464267
_condor_stderr
_condor_stdout
condor_exec.exe
test.out
_CONDOR_JOB_PIDS=
TMPDIR=/var/lib/condor/execute/dir_464267
_CONDOR_SCRATCH_DIR=/var/lib/condor/execute/dir_464267
_CHIRP_DELAYED_UPDATE_PREFIX=Chirp*
_CONDOR_ANCESTOR_15860=15871:1481044466:3221179887
TEMP=/var/lib/condor/execute/dir_464267
BATCH_SYSTEM=HTCondor
_CONDOR_CHIRP_CONFIG=/var/lib/condor/execute/dir_464267/.chirp.config
PWD=/var/lib/condor/execute/dir_464267
```
CUDA_VISIBLE_DEVICES=10000
_CONDOR_AssignedGPUs=10000
_CONDOR_SLOT=slot1_10
SHLVL=1
_CONDOR_MACHINE_AD=/var/lib/condor/execute/dir_464267/.machine.ad
TMP=/var/lib/condor/execute/dir_464267
GPU_DEVICE_ORDINAL=10000
OMP_NUM_THREADS=1
_CONDOR_JOB_AD=/var/lib/condor/execute/dir_464267/.job.ad
_CONDOR_JOB_IWD=/var/lib/condor/execute/dir_464267
_CONDOR_ANCESTOR_464267=466496:1481644885:3834431616
_CONDOR_ANCESTOR_15871=464267:1481644883:816116205
=;/usr/bin/printenv

This example illustrates that the directory where calculations happen is different every time. In this example it is /var/lib/condor/execute/dir_464267

In the next section we’ll learn about specifying local library dependencies.
Library dependencies

Software exist in the form of binary files, but often rely also on external “libraries” that are required. In some cases, the software can be compiled specifically to incorporate the libraries within its own binary file in order to eliminate external dependencies. However, this is not always possible and a method to deal with external libraries exist within HTCondor: the LD_LIBRARY_PATH environment variable.

For example:

```
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/scratch/mynname/lib
```

(replace myname with your directory name; in bash use the export command.)

Question: Why/When would I need this?

On the BCC Linux cluster, you will need to use this method if any of your software is written in the FORTRAN language as the FORTRAN libraries are not available on a default, modern Linux installation.

Note: this is true even if you compile with the “-static” or equivalent option(s) as will be shown in the following, simple example.

1. FORTRAN 77: Hello World example

All FORTRAN program to be run on the BCC Linux cluster will require this step.

FORTRAN programs can be compiled with the compilers f77 or g77 each with specific modifiers.

Let’s create a minimal “Hello World!” FORTRAN 77\textsuperscript{2} program that prints to standard output, save it in a file called hello77.f for example:

---

\textsuperscript{2} Examples for many languages can be found at http://en.wikipedia.org/wiki/List_of_Hello_world_program_examples

---
**FORTRAN 77: Hello World example**

```fortran
PROGRAM HELLO
WRITE(*,*) 'Hello, world!'
END
```

Then we can compile the program with the mode to request that external libraries be included in the final binary. This “switch” is different for both compiler options:

```bash
$ g77 -fno-automatic hello77.f -o hello77_static_g77
$ f77 -Bstatic hello77.f -o hello77_static_f77
```

In practice, the resulting binary has the same size as if the “static” option was not invoked (this can be verified with the Unix command `ls -l` to list files.)

Running the file interactively within the shell will always work:

```bash
$ ./hello77_static_f77
Hello, world!
```

However, running this file with HTCondor on the BCC Linux cluster will cause an error to be reported and the program will FAIL:

```
./hello77_static_f77: error while loading shared libraries: libg2c.so.0: cannot open shared object file: No such file or directory
```

This is true for ALL FORTRAN programs. Therefore there will be a requirement for a SYSTEM library, defined within the `.sh` script file. This will tell the executable program where the necessary library is located.

The following example takes advantage to the fact that `/scratch` is a shared volume. Therefore, after copying the “libg2c.so.0” library to `/scratch/mynname/lib` a minimal script file could be written as:

```bash
#!/bin/bash
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/scratch/mynname/lib
# run the program:
./hello77_static_f77
```

IF there is no shared volume on the cluster, the necessary library or libraries could be placed in a “lib” directory to be transferred with the other files at submission and a relative path could be created. For example:

```bash
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:./lib
```
2. FORTRAN 95: gfortran

The specific requirements for libraries do not work in the same way for gfortran also called f95 on the BCC Linux cluster and binaries will have a different size if compiled with the necessary switch:

```bash
$ gfortran hello77.f -o hello77_gfortran
$ gfortran hello77.f -static-libgfortran -o hello77_static_gfortran
```

We can see that the sizes are different:

```bash
$ ls -l *gfortran | cut -c34-110
    7970 Jun  2 13:17 hello77_gfortran
   158309 Jun  2 13:17 hello77_static_gfortran
```

A test on the BCC Linux cluster has shown that the gfortran compiled binaries in this simple form do not require any library specifications unlike the files compiled with f77 or g77.

Therefore, which compiler is used makes a difference. However, some program code might be sensitive to the nature of the compiler so this might not be a universal result.

3. Compiling

Compiling other programs in various languages will require some knowledge on compilation methods and verifying that the appropriate compiler or compilers are installed.

The command “which” can be useful to locate where program are installed if they are present.

Compilers installed on BCC Linux include: gcc, gcc34, cc, c89, c99, c++, g++, g++34, g77, f77, f95, gfortran.

C pre-processors installed: cpp and mcpp

4. Standard Universe

The Standard Universe offers advantages to the Vanilla Universe especially for long running computations that may create incremental output files.

However, the software has to be recompiled with the `condor_compile` command.
Access to the original source or object code is required for this step. If not, this is not possible and the Standard Universe cannot be used.

Read more on the HTCondor manual.\(^3\)

## 5. Compiling R

Since there are no common directories for software each user has to compile their own. Once compiled the software should be moved within the `/scratch` directory or a subdirectory within.

### 5.1. Download R

The repository for R is “The Comprehensive R Archive Network” or CRAN https://cran.r-project.org/

There are multiple options for Linux, but for `redhat` there is only a README file. For us we need to download the source code, which we can get from:

https://cran.r-project.org/src/base/R-3/

For example download `R-3.2.5.tar.gz`

### 5.2. Install

Most open-source software have a README and an INSTALL file in plain text with information.

*Note:* the CHTC help pages suggest to use an interactive HTCondor session. However, our cluster is NOT configured for this option. Therefore, compilation has to occur on the “submit” node.

*Notes:*

- there is no X11 on the cluster, therefore care has to be given to not use X11 during compilation, this is accomplished with `--with-x=no`

- To avoid library sharing `--enable-R-shlib` was removed from the default command.

---

\(^3\) http://research.cs.wisc.edu/htcondor/manual/v8.3/2_4Running_Job.html
Uncompress:

```
tar zxvf R-3.2.5.tar.gz
```

Then change into the new directory (cd), configure and compile (make):

```
cd R-3.2.5

# Compile R 3.2.5 on 12/12/2016 in /scratch/jsgro/R/
./configure --prefix=$(pwd) --with-readline=no --with-x=no
make
```

The `R` and `Rscript` executables are located in the `./bin` directory.

To start R interactively from the R-3.2.5 directory type:

```
./bin/R
```

### 5.2.1. **Note: Problem with Zlib in newer version**

Starting with version R-3.3.0 there is a newer requirement for zlib which does not seem to be installed on BCC. Therefore, it may be necessary to compile zlib first for these versions.

See for example:

- [http://pj.freefaculty.org/blog/?p=315](http://pj.freefaculty.org/blog/?p=315)

The first link provides an example to compile zlib.

The second link provides an example to configure differently.

### 5.3. **R additional packages**

To use R on the cluster additional packages should be added first from e.g CRAN or Bioconductor and the compiled R directory compressed and transferred e.g. with `tar`.
The added packages will be added in the directory echoed by the R command `.Library`

In the case of this example, start R (e.g. `/bin/R`), then within the R console:

```r
> .Library
[1] "/scratch/jsgro/R/R-3.2.5/library"
```

The compressed tar file should then be part of the files transferred and unarchived by a command contained within the executable (*.sh) file for the job.

It should be sorted out how to transfer the compressed file(s) onto the compute node, uncompress and unarchive them and how to access them.

See for example the section on `LD_LIBRARY_PATH` above (on page 30) to implement this.
File transfer from file.biochem.wisc.edu

1. Moving files on and off the cluster:

   smbclient is more efficient than scp

   Note from Biochem IT (Conor KLECKER cpklecker@wisc.edu) refomatted by Jean-Yves Sgro jsgro@wisc.edu v1.0.1

The cluster has that 10Gbps network card, might as well use it!

   smbclient does not require sudo since it doesn’t mount a file system network to copy files to and from the file server. You simply enter an smb session on the desired server and interact with it based on the local directory you are in when you initiate a connection.

Local computer: mac or Linux system:

Let’s assume we have a directory called "stuff for cluster that we want to transfer to the submit node containing the following: 3 files labeled 1 to 3 and a copy of these files in a directory called directory as illustrated below:

```
stuff for cluster/
  ─── directory/
     │ ─── file.1
     │ ─── file.2
     │    └── file.3
     │ ─── file.1
     │ ─── file.2
     └── file.3
```
Moving files on and off the cluster:

Connect from a local terminal to the submit node:

```bash
$ ssh cpklecker@submit.biochem.wisc.edu
```

Show current directory once logged-in in submit node:

```bash
$ pwd
/scratch/cpklecker
```

Showing there is nothing locally on submit:

```bash
$ ls
```

Connect to file.biochem.wisc.edu Users share as current user (cpklecker):

```bash
$ smbclient //file.biochem.wisc.edu/Users
Enter cpklecker’s password: # password is not shown
```

We can now see the directory and `cd` into it. However, the file system is not "mounted" and therefore special commands need to be applied for the transfer as detailed below.

`cd` into my user directory on file.biochem.wisc.edu/Users:

```
Domain=[BIOCHEM] OS=[Unix] Server=[Samba 3.0.28a-apple]
```

```
smb: \> cd cpklecker
```

Show directory on the User share

```
smb: \cpklecker\> ls
.
..                D    0  Fri Feb 26 11:59:32 2016
Desktop           D    0  Wed Feb 24 11:16:56 2016
stuff for cluster D    0  Fri Feb 26 11:59:58 2016
```

65535 blocks of size 33553920. 65535 blocks available

Turning on the recursive option in smbclient (its off by default)

```
smb: \cpklecker\> recurse
```

Turning off the prompt for each file option:

```
smb: \cpklecker\> mget stuff for cluster
```

That command failed because Unix requires blank spaces in file names or directories to be escaped with \ or the complete name can be written in quotes as shown on the next command below.

```
smb: \cpklecker\> mget stuff for cluster
```

This didn’t work because their were spaces in the name:
File transfer from file.biochem.wisc.edu

File transfer from file.biochem.wisc.edu

```
put /cpklecker/> mget "stuff for cluster"
```

Put quotes around the folder I want to copy:

```
put /cpklecker/> mget "stuff for cluster"
```

The directory and its files are copied to the local directory:

```
exit command quits the smbclient application:
```

```
put /cpklecker/> exit
```

'stuff for cluster' now on submit node scratch space:

```
$ ls
stuff for cluster
```

Change directory into 'stuff for cluster' and list files:

```
$ ls
directory  file.1  file.2  file.3
```

Change the name of directory to new_directory:

```
$ mv directory new_directory
$ ls
new_directory  file.1  file.2  file.3
```

Connect back to the file server:

```
$ smbclient //file.biochem.wisc.edu/Users
Enter cpklecker's password: 
```

Domain=[BIOCHEM] OS=[Unix] Server=[Samba 3.0.28a-apple]

Change directory to user cpkecker:

```
smb: \> cd cpklecker
```

Change directory to the 'stuff for cluster' directory:

```
smb: \cpklecker\> cd stuff for cluster
```

cd \cpklecker\stuff: NT_STATUS_OBJECT_NAME_NOT_FOUND

The command failed because there are spaces and quotes are required.

Even though tab auto-complete will fill directory names like this they require quotes with spaces:
Moving files on and off the cluster:

**cd** with quotes:

```shell
smb: \cpklecker\> cd "stuff for cluster"
```

**List content**

Server only has directory folder and files 1,2,3:

```shell
smb: \cpklecker\stuff for cluster\> ls
```

```
.                       D 0 Fri Feb 26 11:59:58 2016
..                       D 0 Fri Feb 26 11:59:32 2016
file.2                  A 0 Fri Feb 26 11:59:48 2016
file.1                  A 0 Fri Feb 26 11:59:47 2016
directory              D 0 Fri Feb 26 12:00:08 2016
file.3                  A 0 Fri Feb 26 11:59:50 2016
```

65535 blocks of size 33553920. 65535 blocks available

**Turning on the recursive option in smbclient (it resets on exit):**

```shell
smb: \cpklecker\stuff for cluster\> recurse
```

**Turning off the prompt for each file option**

```shell
smb: \cpklecker\stuff for cluster\> prompt
```

**mput command to put a local folder from /scratch to the file server**

```shell
smb: \cpklecker\stuff for cluster\> mput new_directory
```

```
putting file new_directory/file.1 as \cpklecker\stuff for cluster\new_directory\file.1 (0.0 kb/s) (average 0.0 kb/s)
putting file new_directory/file.2 as \cpklecker\stuff for cluster\new_directory\file.2 (0.0 kb/s) (average 0.0 kb/s)
putting file new_directory/file.3 as \cpklecker\stuff for cluster\new_directory\file.3 (0.0 kb/s) (average 0.0 kb/s)
```

**Showing the contents of the file server now:**

```shell
smb: \cpklecker\stuff for cluster\> ls
```

Now that an operation has happened in this folder (mput) the smbclient will list the subfolders as well. Since we changed the name of `directory` to `new_directory` we now have both on the local drive:

```
.                       D 0 Fri Feb 26 12:17:33 2016
..                       D 0 Fri Feb 26 12:00:08 2016
file.2                  D 0 Fri Feb 26 12:00:06 2016
file.1                  A 0 Fri Feb 26 12:00:03 2016
file.3                  A 0 Fri Feb 26 12:00:08 2016
\cpklecker\stuff for cluster\directory
  .                       D 0 Fri Feb 26 12:17:33 2016
  ..                       D 0 Fri Feb 26 12:17:33 2016
  file.2                  A 0 Fri Feb 26 12:17:33 2016
  file.1                  A 0 Fri Feb 26 12:17:33 2016
  file.3                  A 0 Fri Feb 26 12:17:33 2016
\cpklecker\stuff for cluster\new_directory
  .                       D 0 Fri Feb 26 12:17:33 2016
```
File transfer from file.biochem.wisc.edu

|  ..  | D  | 0 Fri Feb 26 12:17:33 2016 |
|file.2| A  | 0 Fri Feb 26 12:17:33 2016 |
|file.1| A  | 0 Fri Feb 26 12:17:33 2016 |
|file.3| A  | 0 Fri Feb 26 12:17:33 2016 |

65535 blocks of size 33553920. 65535 blocks available

**exit smbclient program**

```
mb: \cpklecker\stuff for cluster\> exit
```

We are now back into the **submit** prompt:

```
[cpklecker@submit stuff for cluster]$
```