HTCondor@Biochem

Using HTCondor on the Biochemistry Computational Cluster v1.4.0

Jean-Yves Sgro

© 2016 -2019 Jean-Yves Sgro Biochemistry Computational Research Facility jsgro@wisc.edu

Note: The HTCondor software was known as 'Condor' from 1988 until its name changed in 2012.



Forew	ord	5
What	is HTCondor?	7
Cluste	r access overview	8
1.	Text-based access	8
2.	No graphical interface (X11)	8
3.	VPN access	9
4.	Login info	9
5.	Login Splash Screen	9
6.	Linux OS: CentOS	10
Before	e you begin	.11
The BO	CC Linux cluster	.12
7.	Overview	.12
8.	Connecting to the cluster	.13
9.	Disk space, home directory, /scratch	.13
10.	File servers	.14
11.	Process	.14
12.	Getting ready	.15
13.	HTCondor file transfers	.15
14.	Beyond local cluster: Flocking	.16
QuickS	Start	.17
1.	This section assumes that:	.17
2.	Connect and set-up a working space	.18
3.	Create a simple executable test file	.18
4.	Create a simple submit file	19
5.	Submit the job	20
6.	Check output	20
7.	Syntax	20
8.	HTCondor version	21
9.	Conclusion	21
Resour	rces / Help	.22
On-l	ine resources:	.22

HTCor	ndor concepts	
1.	Class Ads	23
2.	Universes	26
3.	Steps before running a job	26
4.	Requirements and Rank	27
5. 5. 5. 5.	File transfer 1. Jobs with Shared File System 2. File Transfer Mechanism 3. File Paths for file transfer	28 28 28 29
6.	Managing jobs	30
7.	Job completion	31
8.	Tag file names with job or process	31
9.	Example: environment variables	31
10.	Transfer of environment: getenv	33
Libra	ry dependencies	
1.	FORTRAN 77: Hello World example	35
2.	FORTRAN 95: gfortran	
2.	1. Static linking	
3.	Compiling	
4.	Standard Universe	
5.	Compiling R	39
5.	1. Download R	
5.	2. INStall	
5.	3. R additional packages	
Intera	active Jobs	
6.	Interactive shell	42
File tr	ansfer from fs.biochem.wisc.edu	
1.	Moving files on and off the cluster:	
2.	smbclient	
2.	1. More details for smbclient connections	
2.	2. Archive info for file.biochem.wisc.edu	
2.	3. Useful commands	

Foreword

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 22*.)

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 page¹ contains a long description. Here is the first, summary-like paragraph:

HTCondor is a specialized workload management system for compute-intensive jobs. Like other full-featured batch systems, HTCondor provides:

- a job queueing mechanism,
- scheduling policy,
- priority scheme,
- resource monitoring, and
- resource management.

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.

<u>Note: Using HTCondor is the only approved method for performing high throughput computing on the BCC Linux cluster.</u>

Jobs have to be ready to be processed by HTCondor as **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. HTC provides rapid, parallel computing on a large number of small jobs.

Note: this is different from High Performance Computing (HPC) which allows computation on large datasets in large memory footprints.

If you require HPC rather than HTC, or if you are not part of the Biochemistry department, you may obtain a free account at the "Center for High Throughput Computing (CHTC)" at http://chtc.cs.wisc.edu:

"Standard access to CHTC resources are provided to *all* UW-Madison researchers, *free of charge*."

1.Text-based access

The BCC cluster must be accessed via secure shell (ssh) with a **text-based Terminal** from a local computer. There is NO GUI interface available. For example:

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

<u>Note</u>: 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 (X11)

<u>Important note</u>: 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.

<u>Note</u>: The ssh modifier -Y would not allow GUI either.

3.VPN access

Access from outside the Biochemistry department requires a VPN connection:

VPN connection:

If connecting from outside the Biochemistry department it will be necessary to connect via a Virtual Private Network (VPN) to mimic local presence.

Please refer to the following resources to install and activate VPN: General University description:

https://it.wisc.edu/services/wiscvpn/

4.Login info

From a text-based terminal use ssh to login:

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

where **myname** is your **NetID**. Your **NetID Password** will be required after you press return.

For more information about **NetID** see:

- Activating Your Account: https://kb.wisc.edu/page.php?id=1140
- Getting authorized for BCC access: email Jean-Yves Sgro: jsgro@wisc.edu

5.Login Splash Screen

After login you will see the following screen:

```
*
       Welcome to the UW-Madison Biochemistry Computational Cluster
*
*
    USE /scratch FOR JOB DATA! DO NOT STORE DATA IN YOUR USER FOLDER!!!
*
   MOVE YOUR RESULTS TO OTHER STORAGE AFTER YOUR JOB COMPLETES, ALL DATA
           MAY BE REMOVED BY ADMINISTRATORS AT ANY TIME!!!
*
           This computer system is for authorized use only.
*
*
   You must receive permission to access this system. Please contact the
   Biochemistry Computational Research Facility for training and access.
         https://biochem.wisc.edu/bcrf/computational-cluster
```

6.Linux OS: CentOS

There are many Linux versions. On the BCC the version installed is called "CentOS" which is derived from "Red Hat Linux."

The version installed can be obtained with the command:

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

```
CentOS Linux release 7.6.1810 (Core)
```

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

\$ uname -a

Linux submit.biochem.wisc.edu 3.10.0-957.10.1.el7.x86_64 #1 SMP Mon Mar 18 15:06:45 UTC 2019 x86 64 x86 64 x86 64 GNU/Linux

Noteworthy is **x86_64** which means it is a 64 bit system and **e17** means "Enterprise Linux version 7" meaning that it is derived from the Enterprise Linux 7 version from the company Red Hat.

Links:

Logo	Name	Link
CentOS	CentOS	https://www.centos.org
redhat.	redhat	https://www.redhat.com

Before you begin

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 the BCC and therefore HTCondor?**" and validate for yourself the reasons why that would make things better for your computation.

If you decide that HTCondor will be useful, you will then need to evaluate how the software you want to use can work on the BCC Linux cluster. It may also be required that you "compile" the software yourself. There is no pre-installed software other than the Linux OS itself.

Jobs running on BCC can take advantage of the **/scratch** directory as the local HTCondor is aware of its existence. However, that would not apply to jobs sent outside of the 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 NetID 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

7. 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 "<u>Submit Node</u>" 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 calculations 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. However, note that some software require specific GPU architecture.

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

8. Connecting to the cluster

<u>Only the "Submit Node" is accessible to users</u> via a text-based terminal connection with the secure shell command:

Login with ssh *myname@submit.biochem.wisc.edu* where *myname* is your *NetID* and your **Password** will be required after you press return. You need to be authorized to access BCC (see Login info on page 9.)

9. Disk space, home directory, /scratch

The default home directory is mapped according to the user name and has a long, default name reflecting how the **NetID** user name is "mapped" to the BCC cluster.

If **myname** represents **NetID** the default \$HOME directory will be mapped as shown below after a **pwd** command:

/home/myname@ad.wisc.edu

Important note: the default \$HOME directory should NOT be used as the primary location for storage, or for HTCondor job submission. HTCondor cannot "see" your home directory

The main work area is called /scratch and should be used for all jobs.

In addition, HTCondor cannot "see" your home directory but is aware of the **/scratch** Directory.

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.

10.File servers

It is important to understand that the file server **file.biochem.wisc.edu** and **fs.biochem.wisc.edu** are **NOT** accessible from within the cluster as *"mounted"* volumes and certainly not visible by HTCondor.

See Moving files on and off the cluster: on page 45

11.Process

The <u>fundamental process consists of submitting a "job file"</u> that contains information on how to run the software that needs to be run with <u>all</u> 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, <u>submit</u> script (*.sub) that would submit the shell script to HTCondor that will schedule the job.



After the run has completed successfully provisions exists within the *.sub file to transfer all created files to *e.g.* /scratch

12. Getting ready

To get ready you need to evaluate what was just mentioned in the **Process** paragraph above *backwards*:

software and files: <u>know</u> what and how to run, with dependencies

*.sh: create a job that runs hands off , in batch mode.

*.sub: final step, create HTCondor submit file

13.HTCondor file transfers

Part of the HTCondor method is to transfer files (sometimes <u>all</u> 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 completion.

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 <u>different name</u>.



/scratch/myname/quickstart/

/var/lib/condor/execute/dir_arbitrary_number

he ned_

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.)

14.Beyond local cluster: Flocking

When jobs are big, it may be useful to access computers that are <u>beyond</u> 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.

Sending files outside of BCC 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.

In particular the **/scratch** directory would not be visible on these other computers and therefore cannot be used.

<u>See also</u> the Center for High Throughput Computing (CHTC) at http://chtc.cs.wisc.edu

QuickStart

This QuickStart section is inspired by the online QuickStart option shown on the HTCondor web pages (see below on page 22.)

1. This section assumes that:

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

It is also assumed that you know how to converse within the line command and edit simple text files *e.g.* with text editor nano.

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 **NetID** to connect – represented here as myname

You should open a text-based terminal from your local machine (see page 8,) 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 as your project *e.g.* **quickstart**.

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

3. Create a 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 contains information for running a job and is passed to HTCondor. 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
```

Blank lines are ignored

Upper/Lower case is ignore on the left side of the equal sign.

Line 1 specifies the file to run

Line 2 requests 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 (stdin, stdout, stderr.)

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.

hello.sub	
hello	.sh
	software (echo) and files (hello.out, hello.err, hello.log)

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.out hello.sub
hello.log hello.sh
```

We can verify that the job executed correctly:

\$ cat hello.out
Hello World

7.Syntax

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.

For example, unique numbers could be created for the output files with syntax

such as:

Error	= logs/err.\$(cluster)
Output	= logs/out.\$(cluster)
Loq	= logs/log.\$(cluster)

Or

```
error = err.$(Process)
input = in.$(Process)
output = out.$(Process)
```

See examples in the manual under the "Submitting a Job" section for details. For version 8.6 the link is:

http://research.cs.wisc.edu/htcondor/manual/v8.6/2_5Submitting_Job.html

See also: *Tag file names with job or process* on page 31.

8.HTCondor version

The current version of the HTCondor software running is obtained with the command condor_version:

```
$ condor version
```

\$CondorVersion: 8.6.13 Oct 30 2018 BuildID: 453497 \$

\$CondorPlatform: x86_64_RedHat7 \$

(This is the installe version on BCC as of June 11, 2019. However, the most recent version is 8.9.2.

The most recent version of the manual is always available at http://research.cs.wisc.edu/htcondor/manual/

9.Conclusion

The complete HTCondor manual for version 8.6 is **1128** *pages long!*

Therefore, the virtue of patience needs to be called upon to tackle and master using a cluster running HTCondor!

Resources / Help

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:

Resource	Link
HTCondor Quick Start Guide	http://research.cs.wisc.edu/htcondor/manual/ quickstart.html
Complete manual*	http://research.cs.wisc.edu/htcondor/manual/

*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 complex 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

HTCondor concepts

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: about **70**. However, for most users and everyday use just a few are essential, for example to start, stop, hold, restart, and list submitted jobs.

1.Class Ads

Before you learn about how to submit a job, it is important to understand how HTCondor allocates resources. HTCondor manual

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 (Linux, Mac, Windows) etc.

Therefore, jobs can be submitted with either *generic* or *very stringent* requirements, *via* a specific syntax within the *.sub 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 -avail	shows only machines which are willing to run jobs now.
u condor_status -run c	shows only machines which are currently running jobs for your username.
condor_status -help	provides a list of many other options.
condor_status -long r e	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, truncated on both ends. This command will report all information for all cpus of all nodes and would output a total of 69,604 lines!)

A few useful information such as memory or Linux OS are shown in bold below:

```
[truncated above]
LastUpdate = 1527862877
LoadAvg = 1.0
Machine = "cluster-0001.biochem.wisc.edu"
MachineMaxVacateTime = 10 * 60
MachineResources = "Cpus Memory Disk Swap GPUs"
MAX_PREEMPT = ( 3 * 3600 )
MaxJobRetirementTime = MAX_PREEMPT * TARGET.OriginSchedd ==
"submit.biochem.wisc.edu"
Memory = 1024
Mips = 27186
MonitorSelfAge = 2159646
MonitorSelfCPUUsage = 1.720120371398834
MonitorSelfImageSize = 80268
```

```
MonitorSelfRegisteredSocketCount = 39
MonitorSelfResidentSetSize = 10100
MonitorSelfSecuritySessions = 122
MonitorSelfTime = 1530022493
MyAddress = "<128.104.119.168:9618?addrs=128.104.119.168-
9618+[--1]-9618&noUDP&sock=1782 eldb 3>"
MyCurrentTime = 1530022541
MyType = "Machine"
Name = "slot1 1@cluster-0001.biochem.wisc.edu"
NextFetchWorkDelay = -1
NiceUser = false
NumPids = 1
OfflineUniverses = { }
OpSys = "LINUX"
OpSysAndVer = "CentOS7"
OpSysLegacy = "LINUX"
OpSysLongName = "CentOS Linux release 7.5.1804 (Core)"
 [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
```

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

\$ condor status -avail | head -5

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

<u>Summary</u>: ClassAds reflect the resources of compute nodes and the requirements of user jobs. HTCondor matches requirements from <u>both</u>.

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:

A *universe* in HTCondor defines an execution environment. HTCondor Version 8.6.11 supports several different universes for user jobs:

- standard
- vanilla
- grid
- java
- scheduler
- local
- parallel
- vm
- docker

<u>On the BCC</u> 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.

Note: While it is the default <u>it is considered good practise to specify the</u> <u>universe</u> within the submit file as the <u>default could be changed</u> at a later date by the system administration of the compute cluster, or could be different on another computer you might use.

3. Steps before running a job

This was discussed in a previous section (*see* Process *on page 14*) and reviewed here in the light of information from the manual.

Code preparation. Jobs must be able to run in batch, **non-intereactive** 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 at least 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, home directory *on page 13*) 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:

```
YESIF_NEEDEDNO
```

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) *e.g.* using the **wget** command

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 negociated, 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.

8. Tag file names with job or process

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.

9. 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:

#!/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 3775.

We can see the status of the job:

\$ condor q

```
-- Schedd: submit.biochem.wisc.edu : <128.104.119.165:9618?... @ 06/26/18
09:39:33
OWNER BATCH_NAME SUBMITTED DONE RUN IDLE TOTAL JOB_IDS
jsgro CMD: run_environ.sh 6/26 09:38 _ 1 1 3775.0
1 jobs; 0 completed, 0 removed, 1 idle, 0 running, 0 held, 0 suspended</pre>
```

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

\$ ls

environ.sub run_environ_3775.out run_environ_3775.err run_environ.sh run_environ_3775.log test.out

The executable file (**run_environ.sh**) created a standard output that captured within file run_environ_3775.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 6663
_condor_stderr
_condor stdout
condor exec.exe
test.out
CONDOR JOB_PIDS=
 CONDOR ANCESTOR 1864=4326:1527864152:618879280
TMPDIR=/var/lib/condor/execute/dir 6663
_CONDOR_SCRATCH_DIR=/var/lib/condor/execute/dir 6663
CHIRP DELAYED UPDATE PREFIX=Chirp
TEMP=/var/lib/condor/execute/dir 6663
BATCH SYSTEM=HTCondor
CONDOR CHIRP CONFIG=/var/lib/condor/execute/dir 6663/.chirp.
config
 CONDOR ANCESTOR 4326=6663:1530024011:1517681191
PWD=/var/lib/condor/execute/dir 6663
CUDA VISIBLE DEVICES=10000
CONDOR AssignedGPUs=10000
CONDOR ANCESTOR 6663=6668:1530024013:4039808267
CONDOR SLOT=slot1 2
SHLVL=1
CONDOR MACHINE AD=/var/lib/condor/execute/dir 6663/.machine.
ad
TMP=/var/lib/condor/execute/dir 6663
GPU DEVICE ORDINAL=10000
OMP NUM THREADS=1
CONDOR JOB AD=/var/lib/condor/execute/dir 6663/.job.ad
 CONDOR JOB IWD=/var/lib/condor/execute/dir 6663
=/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 6663

10. Transfer of environment: getenv

At BCC we had a situation where a script would fail as a job but ran perfectly fine as an interactive test (at the shell prompt) or as an Interactive HTCondor job. The solution was to add one line to the submit file:

getenv = true

The following explanation was given by CHTC personnel:

This can happen because the environment is subtly different between the two -- the interactive job inherits all of your usual shell environment, but the batch job doesn't. To address this, you can use this option in the submit file: getenv = true This will emulate the interactive environment in the batch job and hopefully solve your problem there.

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/myname/lib

(replace <u>myname</u> 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 <u>any</u> of your software is written in the FORTRAN language as the FORTRAN libraries are not available on a default, modern Linux installation.

<u>Note</u>: 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

<u>UPDATE</u>: SKIP THIS PARAGRAPH

f77 and g77 are no longer installed on the cluster. This section is kept here for general information about the use of the LD LIBRARY PATH option.

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 77² program that prints to standard output, save it in a file called hello77.f for example:

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:

```
$ g77 -fno-automatic hello77.f -o hello77_static_g77
```

```
$ f77 -Bstatic hello77.f -o hello77_static_f77
```

In practise, 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:

```
$ ./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 obje
ct 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/<u>myname</u>/lib a minimal script file could be written as:

 $http://en.wikipedia.org/wiki/List_of_Hello_world_program_examples$

[ARCHIVED 11APR2015 at

² Examples for many languages can be found at

https://web.archive.org/web/20150411170140/http://en.wikipedia.org/wiki/List_of_Hello_world_program_e xamples SHORT URL: https://bit.ly/2N3dit8

```
#!/bin/bash
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/scratch/myname/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 <u>transferred</u> with the other files at submission and a relative path could be created. For example:

```
export LD LIBRARY PATH=$LD LIBRARY PATH:./lib
```

2.FORTRAN 95: gfortran

gfortran (also called f95 on the BCC Linux cluster) is GNU FORTRAN and is the software to use.

```
$ gfortran --version
```

GNU Fortran (GCC) 4.8.5 20150623 (Red Hat 4.8.5-28) Copyright (C) 2015 Free Software Foundation, Inc.

GNU Fortran comes with NO WARRANTY, to the extent permitted by law. You may redistribute copies of GNU Fortran under the terms of the GNU General Public License. For more information about these matters, see the file named COPYING

The general compilation format is:

```
$ gfortran hello77.f -o hello77_gfortran
```

2.1. Static linking

From the GNU web site³:

gfortran is composed of two main parts: the compiler, which creates the executable program from your code, and the library, which is used when you run your program afterwards. That explains why, if gfortran is installed in a non-standard directory, it may compile your code fine but the executable may fail with an error message like library not found. One way to avoid this (more ideas can be found on the binaries page) is to use the so-called "static linking", available with option -static gfortran then put the library code inside the program created, thus enabling it to run without the library

³ https://gcc.gnu.org/wiki/GFortranGettingStarted

present (like, on a computer where gfortran is not installed). Complete example is:

```
gfortran -static myfile.f -o program.exe
```

However, just doing this causes an error on the cluster:

```
$ gfortran -static hello77.f -o hello77 static gfortran
```

```
/usr/bin/ld: cannot find -lgfortran
/usr/bin/ld: cannot find -lm
/usr/bin/ld: cannot find -lquadmath
/usr/bin/ld: cannot find -lm
/usr/bin/ld: cannot find -lc
collect2: error: ld returned 1 exit status
```

We can find the location of the library with⁴:

```
$ gfortran -print-file-name=libgfortran.so
```

/usr/lib/gcc/x86_64-redhat-linux/4.8.5/libgfortran.so

Then export LDFLAGS with the above information, without the file name:

```
$ export LDFLAGS=-L/usr/lib/gcc/x86_64-redhat-linu
x/4.8.5
```

The specific requirements for libraries do not work in the same way for gfortran and binaries will have a different size if compiled with the necessary switch:

```
$ gfortran hello77.f -o hello77 gfortran
```

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

We can see that the sizes are different:

```
$ ls -1 *gfortran | cut -c34-110
7970 Jun 2 13:17 hello77_gfortran
158309 Jun 2 13:17 hello77_static_gfortran
```

⁴ https://github.com/JuliaLang/julia/issues/6150

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 £77 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.⁵

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 http://research.cs.wisc.edu/htcondor/manual/v8.6/2_4Running_Job.html

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 has a README and an INSTALL file in plain text with information.

<u>Note</u>: 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 <u>no X11 on the cluster</u>, 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.

Uncompress:

tar zxvf R-3.2.5.tar.gz

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

Here is an example to compile R version 3.2.5:

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
- https://stat.ethz.ch/pipermail/r-devel/2015-April/070951.html

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 <u>additional packages should be added first</u> from e.g CRAN or Bioconductor and the compiled R directory compressed and transferred *e.g.* with tar.

tar -czvf R.tar.gz R-3.2.5/

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:

> .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 35) to implement this.

Interactive Jobs

Interactive jobs are now possible on BCC. The default Universe will be Vanilla.

From the manual⁶:

An interactive job is a Condor job that is provisioned and scheduled like any other vanilla universe Condor job onto an execute machine within the pool. The result of a running interactive job is a shell prompt issued on the execute machine where the job runs. The user that submitted the interactive job may then use the shell as desired, perhaps to interactively run an instance of what is to become a Condor job.

This option may be beneficial for example to compile software (see previous section) without impeding the submit node.

6.Interactive shell

The simplest command to start an interactive job is:

```
$ condor_submit -interactive
```

Submitting job(s). 1 job(s) submitted to cluster 1108. Waiting for job to start...

There may be a delay for the interactive shell to begin depending on the cluster availability.

The interactive shell will be running as an HTCondor job on a compute node

⁶ http://research.cs.wisc.edu/htcondor/manual/v8.6/2_5Submitting_Job.html (see section 2.5.13)

rather than the submit node.

Once activated a splash screen of information will be presented. An error about "home directory" not found is normal as HTCondor nodes are only aware of the /scratch directory as explained previously.

Note that the shell will log itself out after 7200 seconds (2hrs) of inactivity:

Could not chdir to home directory /home/jsgro@ad.wisc.edu: No such file or directory Welcome to slot1_4@cluster-0005.biochem.wisc.edu! You will be logged out after 7200 seconds of inactivity.

cluster-0005.biochem.wisc.edu

Welcome to centos-release-7-4.1708.el7.centos.x86_64 running kernel (3.10.0-693.17.1.el7.x86_64).

System information as of: Tue May 1 15:39:05 CDT 2018

System Load:	39.06,	39.07,	39.0	06	System	Uptime:	
69 days 6 ho	ours 30	min 57	sec				
Memory Usage:	0.0%			Swap	Usage:	0.9%	
Jsage On /:	158G			Acces	s Rights	on /:	(rw
Local Users:	0			Whoam	1i:		myname
Processes:	543						
Interface	MAC Add	dress		IP Ad	ldress		
enp3s0f0	xx:xx:>	x:xx:xx	x:xx	128.x		xx/22	
pash-4.2\$							

We now have an interactive bash shell.

```
bash-4.2$ pwd
```

```
/var/lib/condor/execute/dir 36940
```

bash-4.2\$ printenv | fgrep -v LS COLORS

```
CONDOR JOB PIDS=
HOSTNAME=cluster-0005.biochem.wisc.edu
TERM=xterm-256color
SHELL=/bin/bash
HISTSIZE=1000
SSH CLIENT=128.104.119.165 22309 9618
TMPDIR=/var/lib/condor/execute/dir 36940
_CONDOR_ANCESTOR_28163=36940:1525207142:4203470193
CONDOR SCRATCH DIR=/var/lib/condor/execute/dir 36940
SSH TTY=/dev/pts/0
USER=jsgro
CHIRP DELAYED UPDATE PREFIX=Chirp
LD LIBRARY PATH=/usr/local/cuda/lib64:
TEMP=/var/lib/condor/execute/dir 36940
BATCH SYSTEM=HTCondor
CONDOR CHIRP CONFIG=/var/lib/condor/execute/dir 36940/.chirp.config
TMOUT=7200
MAIL=/var/spool/mail/jsgro
```

```
PATH=/usr/local/bin:/usr/bin:/usr/bin:/usr/local/sbin
:/usr/sbin:/usr/local/cuda/bin
CONDOR ANCESTOR 3044=28163:1521057660:334989916
PWD=/var/lib/condor/execute/dir 36940
CUDA VISIBLE DEVICES=10000
 CONDOR AssignedGPUs=10000
LANG=en US.UTF-8
CONDOR ANCESTOR 36940=36949:1525207145:1006659451
 CONDOR SLOT=slot1 4
HISTCONTROL=ignoredups
SHLVL=1
HOME=/home/jsgro@ad.wisc.edu
CONDOR MACHINE AD=/var/lib/condor/execute/dir 36940/.machine.ad
LOGNAME=jsgro
SSH CONNECTION=128.104.119.165 22309 128.104.119.172 9618
GPU DEVICE ORDINAL=10000
LESSOPEN=||/usr/bin/lesspipe.sh %s
OMP NUM THREADS=1
_CONDOR_JOB_AD=/var/lib/condor/execute/dir_36940/.job.ad
CONDOR JOB IWD=/var/lib/condor/execute/dir 36940
CONDOR SHELL=/bin/bash
CONDOR SLOT NAME=slot1 4@cluster-0005.biochem.wisc.edu
=/usr/bin/printenv
```

It is important to note that the environment variables are transferred from the user, including the \$HOME directory even if HTCondor cannot access it!

Most importantly the /scratch directory is available:

bash-4.2\$ cd /scratch

From this point it will be possible to work to prepare software and files without impacting the submit node.

When done, the shell can be released with the exit command:

```
bash-4.2$ exit
```

```
logout
Connection to condor-job.cluster-0005.biochem.wisc.edu
closed.
```

File transfer from fs.biochem.wisc.edu

1. Moving files on and off the cluster:

The "fs.biochem.wisc.edu" server contains 3 directories:

- Groups
- SBGrid
- Users

The "Groups" directory contains directories for the Lab Group to which you belong. Within, the "General" directory is meant as a shared space for lab members.

New: by default, the Groups directory is already available to you as /mnt/Groups

This means that all files you have stored there from your Mac or PC is available. Conversely you can copy files there from your BCC account area.

The "SBGrid" directory contains that software collection, but it is NOT accessible to BCC for licensing reasons at least.

The "Users" directory does not contain anything but could provide possible temporary space for file transfers.

Below are explanations on how to connect to the BCC cluster for file transfer.

2.smbclient

The network protocol that can let the cluster and the file servers speak to each other is called samba.

The software **smbclient** on the cluster can connect to the **fs** file server.

If you only need access to the Groups directory, it is already mounted as /mntGroups and you can use the default shell commands such as cp, 1s and mkdir etc. to list and copy files.

2.1. More details for smbclient connections

Note from Biochem IT (Conor KLECKER **cpklecker@wisc.edu**) refromatted by Jean-Yves Sgro **jsgro@wisc.edu** v1.0.2

smbclient is more efficient than scp

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 a **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:

Connect from a local terminal to the submit node:

\$ **ssh** cpklecker@submit.biochem.wisc.edu

Show current directory once logged-in in submit node:

\$ **pwd**

/scratch/cpklecker

Showing there is nothing locally on submit:

\$ **ls**

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

NOTE: "Groups" <u>or</u> "Users" must be the last words on the connection command. They represent the "mounting" point, *i.e.* the starting point of the directory. Simply having the command as "smbclient **//fs.biochem.wisc.edu/**" will have no effect!

```
$ smbclient //fs.biochem.wisc.edu/Groups
Enter cpklecker@AD.WISC.EDU's password:
# password is not shown
Try "help" to get a list of possible commands.
smb: \>
```

You are now connected, with the $smb: \ prompt ready for commands.$

For example, a list (**1s**) command below shows:

```
smb: \> ls
                                           0 Fri Jun 1 08:58:58 2018
                                    D
  •
                                           0 Mon Jun 25 16:04:28 2018
                                    D
  Office
                                    D
                                           0 Sun Aug 28 22:28:17 2016
 _Committees
                                           0 Fri Apr 20 13:00:59 2018
                                    D
                                           0 Wed Dec 28 08:51:12 2016
                                    D
  Public
                                            0 Wed Jan 17 08:34:55 2018
 XXXXXXXGrp
                                    D
       93749959168 blocks of size 4096. 32718506197 blocks available
smb: \>
```

What you see depends on the content of your group. The group would have the name of the professor or lab before the letters Grp.

To exit this connection simply type **exit**:

smb: \> exit

This will take you back to the bash prompt (\$) on the cluster.

The process would be the same for the "Users" area which contains no files or directories and could be used f

```
$ smbclient //fs.biochem.wisc.edu/Users
Enter cpklecker@AD.WISC.EDU's password:# password does not show
Try "help" to get a list of possible commands.
smb: \>
```

You are now connected, with the **smb**: **>** prompt ready for commands.

2.2. Archive info for file.biochem.wisc.edu

Access is still possible (as of 6/11/2019) but should not be used except if you still have files on the site.

The process is also similar for the "file.biochem" server, but the username is specified in a different manner, using the value of the environment variable **USER**.

THEN it is necessary to cd into your own directory as all users are shown by default:

```
$ smbclient //file.biochem.wisc.edu/Users/ -U $USER
Enter SAMBA\cpklecker password:
Try "help" to get a list of possible commands.
smb: \> cd cpklecker
smb: \ cpklecker\> 1s
```

To exit:

```
smb: \> exit
$
```

2.3. Useful commands

The commands will work the same whether you are in the **file** or the **fs** file server.

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:

smb: \> cd cpklecker\

Show directory on the User share

smb: \cpklecker\> ls 0 Fri Feb 26 11:59:32 2016 D D 0 Wed Feb 24 11:16:56 2016 . . D 0 Wed Feb 24 08:42:27 2016 Desktop .bash_logout AH 33 Wed Feb 24 08:42:26 2016 0 Fri Feb 26 11:59:58 2016 stuff for cluster D 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

mget command and the name of the folder I want to copy to submit node:

smb: \cpklecker\> mget stuff for cluster

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

This didn't work because their were spaces in the name:

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

Put quotes around the folder I want to copy:

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

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

getting file \cpklecker\stuff for cluster\file.2 of size 0 as file.2 (0.0 KiloBytes/sec) (average 0.0 KiloBytes/sec) getting file \cpklecker\stuff for cluster\file.1 of size 0 as file.1 (0.0 KiloBytes/sec) (average 0.0 KiloBytes/sec) getting file \cpklecker\stuff for cluster\directory\file.2 of size 0 as file.2 (0.0 KiloBytes/sec) (average 0.0 KiloBytes/sec) getting file \cpklecker\stuff for cluster\directory\file.1 of size 0 as file.1 (0.0 KiloBytes/sec) (average 0.0 KiloBytes/sec) getting file \cpklecker\stuff for cluster\directory\file.3 of size 0 as file.3 (0.0 KiloBytes/sec) (average 0.0 KiloBytes/sec) getting file \cpklecker\stuff for cluster\file.3 of size 0 as file.3 (0.0 KiloBytes/sec) (average 0.0 KiloBytes/sec)

exit command quits the smbclient application:

smb: \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:

cd with quotes:

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

List content

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

```
smb: \cpklecker\stuff for cluster\> ls
                                     D
                                             0 Fri Feb 26 11:59:58 2016
  •
                                             0 Fri Feb 26 11:59:32 2016
                                     D
  . .
 file.2
                                             0 Fri Feb 26 11:59:48 2016
                                    А
 file.1
                                             0 Fri Feb 26 11:59:47 2016
                                    А
                                            0 Fri Feb 26 12:00:08 2016
 directory
                                    D
 file.3
                                             0 Fri Feb 26 11:59:50 2016
                                     Α
```

65535 blocks of size 33553920. 65535 blocks available

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

smb: \cpklecker\stuff for cluster\> recurse

Turning off the prompt for each file option

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

mput command to put a local folder from /scratch to the file server file.biochem.wisc.edu:

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.3 as \cpklecker\stuff for cluster\new_directory\file.3 (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)

Showing the contents of the file server now:

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	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
	new_directory	D	0	Fri	Feb	26	12:17:33	2016
	\cpklecker\stuff for cluster\directed	ory						
		D	0	Fri	Feb	26	12:00:08	2016
		D	0	Fri	Feb	26	12:17:33	2016
	file.2	A	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\new dired	ctory						
	·	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

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]\$