What is the Open Science Grid?

The OSG consortium of research collaborations, campuses, national laboratories, and software providers is dedicated to the advancement of all open science via the practice of distributed High Throughput Computing (dHTC), and to the advancement of its state of the art.

What is the Open Science Pool?

The OSPool provides researchers with **fair-share** access to computing and data capacity powered by distributed high-throughput computing (dHTC) technologies.

- Open for any scientist or group, of all disciplines, doing open science in the US.
- Built to run independent computations on a massive scale.
- The OSPool is built from resources contributed by university campuses, government-supported supercomputing facilities and research collaborations
- Resources are assigned on a fair-share basis at no cost to researchers; no allocation process is required.

Recommended Video Guide: https://www.youtube.com/watch?v=9896xAhT4dY

Video discusses:

- HTCondor History and Status
- Terminology
- Job Matching & Execution
- Basic Submit File
- Job Idle, Running & Completes

Can I submit to the Open Science Pool?

1. **Registering and getting assigned to a project on OSG connect**
    a. Prior to submitting jobs on the OSPool, each individual must:
        i. Register for an account,
        ii. Complete the Orientation Meeting (potentially waived)
        iii. Get account approved
        iv. Get assigned to a project.
    b. If these steps are not completed, please refer to the **Open Science Grid Before Submitting Documentation** on how to proceed.
2. **Join and Use a Project to submit in OSG Connect**
    a. The OSG Connect team assigns individual user accounts to "projects". These projects are a way to track usage hours and capture information about the types of research using OSG Connect. A project typically corresponds to a research group headed by a single PI, but can sometimes represent a long-term multi-institutional project or some other grouping. ***You must be a member of a project before you can use OSG Connect to submit jobs.***
        i. If you are the first member of your research group / team to use the OSG through OSG Connect, a new project will be created for you. You will need to provide the following information to do so:
            1. Project Name
            2. PI (Principal Investigator) Name
            3. PI (Principal Investigator) Email
            4. PI (Principal Investigator) Organization

5. PI (Principal Investigator) Department
6. Field of Science:
   a. Selected from https://osp.unm.edu/pi-resources/nsf-research-classifications.html
7. Project Description:
   a. Few sentences describing the project you are researching for

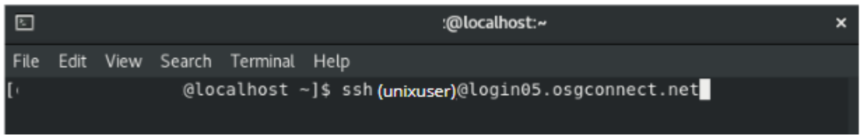
ii. If you know that other members of your research group have used OSG Connect in the past, you can likely join a pre-existing group. Provide the name of your institution and PI to the OSG Connect team (if you haven't already) and we can confirm.

b. **Setting up your OSG Connect Project**

i. Job submission on OSG Connect requires a project be assigned to your account on the login node. This can be set after you have been added to a project as described above.

ii. Option 1: (Preferred Method)

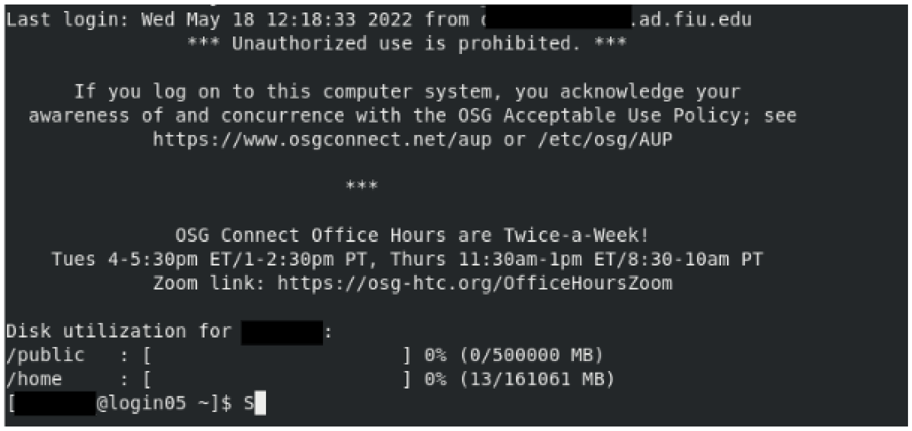
1. Sign in to your login node
2. Open a Terminal and Type the following into the terminal
   a. ssh <your_osg_connect_username>@<your_osg_login_node>



```
[               @localhost ~]$ ssh (unixuser)@login05.osgconnect.net
```

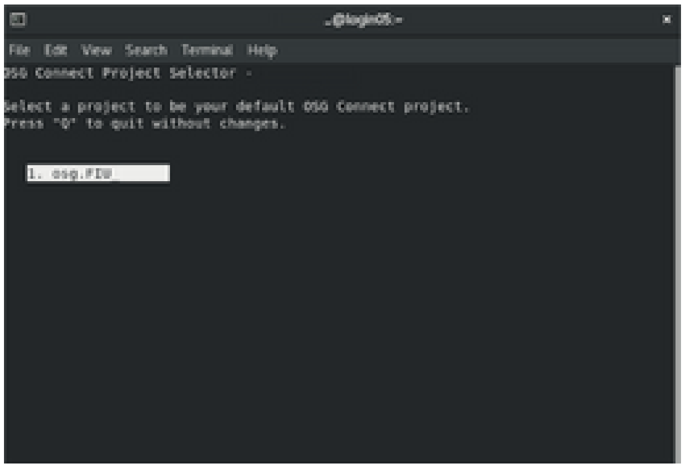
   i. *It will ask for the passphrase for your ssh key (if you set one) and then you should be logged in.*

3. When successfully logged on, your screen should reassemble to some degree



```
Last login: Wed May 18 12:18:33 2022 from [redacted].ad.fiu.edu
                *** Unauthorized use is prohibited. ***

     If you log on to this computer system, you acknowledge your
  awareness of and concurrence with the OSG Acceptable Use Policy; see
             https://www.osgconnect.net/aup or /etc/osg/AUP

                              ***

              OSG Connect Office Hours are Twice-a-Week!
    Tues 4-5:30pm ET/1-2:30pm PT, Thurs 11:30am-1pm ET/8:30-10am PT
           Zoom link: https://osg-htc.org/OfficeHoursZoom

Disk utilization for [redacted]:
/public    : [                      ] 0% (0/500000 MB)
/home      : [                      ] 0% (13/161061 MB)
[[redacted]@login05 ~]$ S
```

4. After successfully logging in type:
   a. $ connect project
   b. You should see a list of projects that you have joined. Most often there will only be one option! Make sure the right project is highlighted and press "enter" to save that choice.



```
OSG Connect Project Selector -

Select a project to be your default OSG Connect project.
Press "Q" to quit without changes.

  1. osg.FIU_
```

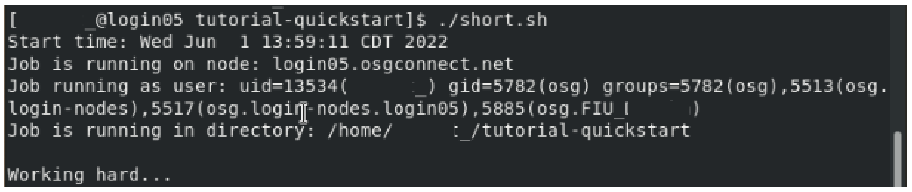
iii. Option 2:
   1. If need to run jobs under a different project you are a member of (not your default), you can manually set the project for those jobs by putting this **option in the submit file**:
      a. +ProjectName="ProjectName"

3. **Submitting Jobs using tutorial Quickstart**
   i. This is a very basic tutorial, normally used to ensure that the user is able to submit jobs to the OS Pool and
   ii. Login to OSG using the login node
      1. ssh <your_osg_connect_username>@<your_osg_login_node>
   iii. Pre-Typed Setup
      1. Install tutorial into your home directory on the login node
         a. Type $ tutorial into the terminal of you home directory
            i. A list of currently available tutorials will appear
         b. Proceed by downloading and running the Quickstart tutorial
            i. $ tutorial quickstart
            ii. $ cd tutorial-quickstart
   iv. Manual Setup
      1. $ mkdir tutorial-quickstart
      2. $ cd tutorial-quickstart

   v. Submission of Job 1:
      1. Inside the tutorial directory that you created or installed previously, let's create a test script to execute as your job. For pretyped setup, this is the short.sh file:
         a. #!/bin/bash
         b. # short.sh: a short discovery job
         c. printf "Start time: "; /bin/date
         d. printf "Job is running on node: "; /bin/hostname
         e. printf "Job running as user: "; /usr/bin/id
         f. printf "Job is running in directory: "; /bin/pwd
         g. echo
         h. echo "Working hard..."
         i. sleep 20
         j. echo "Science complete!"

2. Make the script executable
    a. $chmod +x short.sh

vi. Running the Job Locally
1. When setting up a new job submission, it's important to test your job outside of HTCondor before submitting into the Open Science Pool.
    a. $ ./short.sh



```
[[redacted]@login05 tutorial-quickstart]$ ./short.sh
Start time: Wed Jun  1 13:59:11 CDT 2022
Job is running on node: login05.osgconnect.net
Job running as user: uid=13534([redacted]) gid=5782(osg) groups=5782(osg),5513(osg.
login-nodes),5517(osg.login-nodes.login05),5885(osg.FIU_[redacted])
Job is running in directory: /home/[redacted]/tutorial-quickstart

Working hard...
```

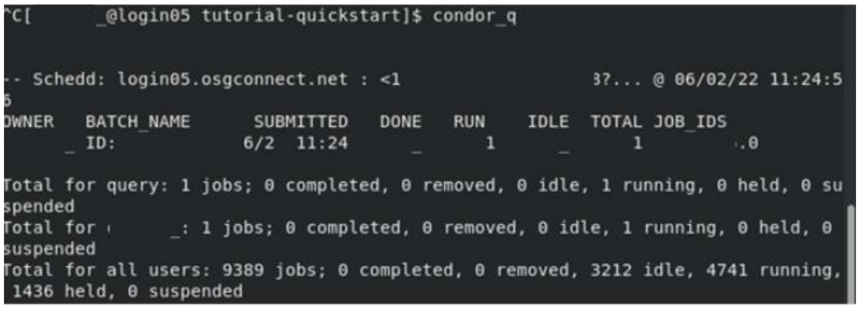
    b. When completed, “Science Complete” will be printed on the terminal

vii. Create an HTCondor Submit File
1. Let's create a simple (if verbose) HTCondor submit file.
    a. This can be found in tutorial01.submit. (Pre-typed version)
    b. HTC Condor Submit File
        i. Bolded sections MUST be included in the Submit File
        ii. executable = short.sh
            1. Our executable is the main program or script that we've created to do the 'work' of a single job.
        iii. error = short.error
        iv. output = short.output
        v. log = short.log
            1. # We need to name the files that HTCondor should create to save the terminal output (stdout) and error (stderr) created by our job. Similarly, we need to name the log file where HTCondor will save information about job execution steps.
        vi. request_cpus = 1
        vii. request_memory = 1 MB
        viii. request_disk = 1 MB
            1. We need to request the resources that this job will need:
        ix. queue 1
            1. The last line of a submit file indicates how many jobs of the above description should be queued. We'll start with one job.
2. Submit the job using:
    a. $ condor_submit tutorial01.submit

viii. Checking the job Status
1. To check the status of any job submitted, users can use
    a. $ condor_q
        i. You can also get status on a specific job cluster:
            1. condor_q (job_id)
        ii. Note the DONE, RUN, and IDLE columns. Your job will be listed in the IDLE column if it hasn't started yet. If it's currently scheduled and running, it will appear in the RUN

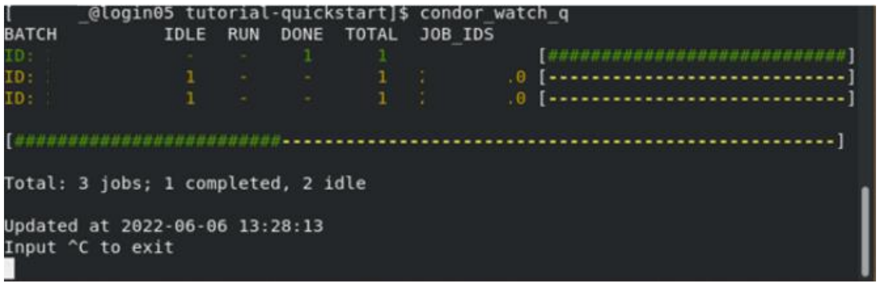
column. As it finishes up, it will then show in the DONE column. Once the job completes completely, it will not appear in condor_q.

iii. Let's wait for your job to finish – that is, for condor_q not to show the job in its output. A useful tool for this is watch – it runs a program repeatedly, letting you see how the output differs at fixed time intervals.



```
^C[      _@login05 tutorial-quickstart]$ condor_q


-- Schedd: login05.osgconnect.net : <1                 3?... @ 06/02/22 11:24:5
6
OWNER   BATCH_NAME      SUBMITTED   DONE   RUN    IDLE  TOTAL JOB_IDS
      _ ID:            6/2  11:24      _      1      _      1        .0

Total for query: 1 jobs; 0 completed, 0 removed, 0 idle, 1 running, 0 held, 0 su
spended
Total for       _: 1 jobs; 0 completed, 0 removed, 0 idle, 1 running, 0 held, 0
suspended
Total for all users: 9389 jobs; 0 completed, 0 removed, 3212 idle, 4741 running,
 1436 held, 0 suspended
```

b. $ condor_watch_q (preferred)



```
[       _@login05 tutorial-quickstart]$ condor_watch_q
BATCH            IDLE  RUN  DONE  TOTAL  JOB_IDS
ID:                -    -     1     1                   [############################]
ID:                1    -     -     1   :           .0 [----------------------------]
ID:                1    -     -     1   :           .0 [----------------------------]

[#########################-------------------------------------------------------]

Total: 3 jobs; 1 completed, 2 idle

Updated at 2022-06-06 13:28:13
Input ^C to exit
```

2. When your job has completed, it will disappear from the list.
   a. Note: To close watch, hold down Ctrl and press C.

ix. Job History
   1. Once your job has finished, you can get information about its execution from the condor_history command:
      a. $ condor_history (Job ID)

x. Job Output
   1. Once your job has finished, you can look at the files that HTCondor has returned to the working directory. The names of these files were specified in our submit file. If everything was successful, it should have returned:
      a. a log file from HTCondor for the job cluster: short.log
      b. an output file for each job's output: short.output
      c. an error file for each job's errors: short.error
   2. Reading the Output
      a. $ cat short.output
      b. The file should produce the following output
         i. Start time: Mon Dec 10 20:18:56 UTC 2018
         ii. Job is running on node: osg-84086-0-cmswn2030.fnal.gov
         iii. Job running as user: uid=12740(osg) gid=9652(osg) groups=9652(osg)

iv. Job is running in directory: /srv
v. Working hard...
vi. Science complete!

b. **Passing Arguments to Executables – Tutorial Quick Start**
   i. Sometimes it's useful to pass arguments to your executable from your submit file. For example, you might want to use the same job script for more than one run, varying only the parameters.
      1. First, let's edit our existing short.sh script to accept arguments. To avoid losing our original script, we make a copy of the file under the name short_transfer.sh
         a. $ cp short.sh short_transfer.sh
      2. Edit the file to include the added lines below
         a. $ nano short_transfer.sh
            i. #!/bin/bash
            ii. # short.sh: a short discovery job
            iii. printf "Start time: "; /bin/date
            iv. printf "Job is running on node: "; /bin/hostname
            v. printf "Job running as user: "; /usr/bin/id
            vi. printf "Job is running in directory: "; /bin/pwd
            vii. printf "The command line argument is: "; $1
            viii. printf "Contents of $1 is "; cat $1
            ix. cat $1 > output.txt
            x. printf "Working hard..."
            xi. ls -l $PWD
            xii. sleep 20
            xiii. echo "Science complete!"
         b. Escape the script editor
            i. Ctrl + X
            ii. Yes
            iii. Enter
         c. Notice that with our changes, the new script will now print out the contents of whatever file we specify in our arguments, specified by the $1. It will also copy the contents of that file into another file called output.txt.
      3. Make the new script executable
         a. $ chmod +x short_transfer.sh
      4. Create Simple Text called Input.txt to pass to our script
         a. “Hello FIU”
      5. Test Locally to Ensure the script runs
         a. $ ./short_transfer.sh input.txt
   ii. Creating the Submit File
      1. Create executable script
         a. Save as tutorial02.submit
         b. executable = short_transfer.sh
         c. arguments = input.txt

                    i. **Not to be included in file:** Notice the added arguments = input.txt information. The arguments option specifies what arguments should be passed to the executable.
            d. transfer_input_files = input.txt
            e. transfer_output_files = output.txt
                    i. **Not to be included file**: The transfer_input_files and transfer_output_files options need to be included as well. When jobs are executed on the Open Science Pool via HTCondor, they are sent only with files that are specified. Additionally, only the specified output files are returned with the job. Any output not transferred back, with the exception of our error, output, and log files, is discarded at the end of the job.
            f. error = job.error
            g. output = job.output
            h. log = job.log

            i. # The below are good base requirements for first testing jobs on OSG,
            j. request_cpus = 1
            k. request_memory = 1 GB
            l. request_disk = 1 GB

            m. queue 1

        2. Submit the new submit file using condor_submit.
            a. $ condor_submit tutorial02.submit

c. **Submitting Jobs Concurrently**
    i. What do we need to do to submit several jobs simultaneously? In the first example, Condor returned three files: out, error, and log. If we want to submit several jobs, we need to track these three files for each job. An easy way to do this is to add the $(Cluster) and $(Process) macros to the HTCondor submit file. Since this can make our working directory really messy with a large number of jobs, let's tell HTCondor to put the files in a directory called log.
        1. Here's what the third submit file looks like, called tutorial03.submit:
            a. # We need the job to run our executable script, arguments and files.
            b. executable = short_transfer.sh
            c. arguments = input.txt
            d. transfer_input_files = input.txt
            e. transfer_output_files = output.txt

            f. error = log/job.$(Cluster).$(Process)error
            g. output = log/job.$(Cluster).$(Process).output
            h. log = log/job.$(Cluster).$(Process).log

            i. request_cpus = 1

j. request_memory = 1 GB
k. request_disk = 1 GB

l. # Let's queue ten jobs with the above specifications
m. queue 10

2. Ensure that a log directory exists
   a. $ mkdir -p log
3. Submit
   a. $ condor_submit tutorial03.submit
4. After Submission
   a. Find all of the output file in the log directory
      i. $ ls ./log

ii. Where did the Jobs run?
   1. It might be interesting to see where our jobs actually ran. To get that information for a single job, we can use the command condor_history. First, select a job you want to investigate and then run condor_history -long jobid.

iii. Removing Jobs:
   1. On occasion, jobs will need to be removed for a variety of reasons (incorrect parameters, errors in submission, etc.). In these instances, the condor_rm command can be used to remove an entire job submission or just particular jobs in a submission. The condor_rm command accepts a cluster id, a job id, or username and will remove an entire cluster of jobs, a single job, or all the jobs belonging to a given user respectively. E.g. if a job submission generates 100 jobs and is assigned a cluster id of 103, then condor_rm 103.0 will remove the first job in the cluster. Likewise, condor_rm 103 will remove all the jobs in the job submission and condor_rm [username] will remove all jobs belonging to the user. The condor_rm documenation has more details on using condor_rm including ways to remove jobs based on other constraints.