

# Running Batch Jobs on CBS Research Grid

Most of the jobs run on the CBS research Grid are submitted in batch mode, where users include program codes in a file called script, and submit the script to the job scheduler for execution. Batch jobs are often considered running in the background as they do not require user interactions once they are submitted. Program output and system messages will be written to various output and log files. The advantage of batch jobs is that users can avoid entering program codes one at a time and wait for responses, particularly suitable for jobs that involve multiple steps, require little to no user input, and take a long time to complete. This page introduces the basic steps in the batch job workflow and tools that can be used during the process.

*Creating and Editing Program Files*  
*Sample Programs and Directive Statements*  
*Tools for Creating and Editing Program Files*  
*Executing Batch Jobs*  
*Commands to Submit Batch Jobs*  
*Grid Options and Resource Consideration*  
*Monitoring Progress of Batch Jobs*  
*Examining Output Files*  
*Research Support*

## Creating and Editing Program Files

### Sample Programs and Directive Statements

Almost all software applications on the Grid (e.g., Shell, Python, Matlab, Stata, or SAS) support batch submission. The first step in the workflow is to design program codes and save the codes in a text file in your account. The following is a sample Python program named [add.py](#):

```
#!/apps/anaconda3/bin/python
# This program adds two numbers and display them
num1 = 1.5
num2 = 6.3
sum = num1 + num2
print('The sum of {0} and {1} is {2}'.format(num1, num2, sum))
```

The first line of the program is a directive statement - shebang. Once the program is submitted to the job scheduler, shebang directs the job to an appropriate interpreter. In this example, shebang invokes the *Anaconda 3 Python Interpreter*. The rest are Python codes.

If a program does not contain a shebang as in the sample SAS program [readzip.sas](#) below, user must indicate the intended interpreter when submit the job:

```
filename dat pipe "7z x -so ./myfile.7z";
* Read data in Zip format;
data myfile;
infile dat dlm=';' truncover;
input var1 - var3; run;
* Save data to STATA .dta file;
proc export data=myfile outfile= "./myfile" dbms= dta replace;run;
```

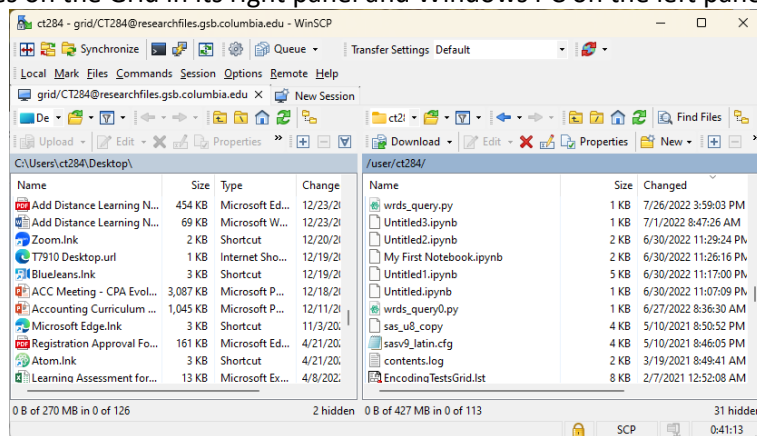
## Tools for Creating and Editing Program Files

There are many ways to create and edit program files. For users who use STATA, MATLAB or SAS, the editors provided therein are the recommended choice. These embedded editors are color-keyed and provide many useful tools for testing and debugging. The codes can be saved for batch execution and are also platform compatible.

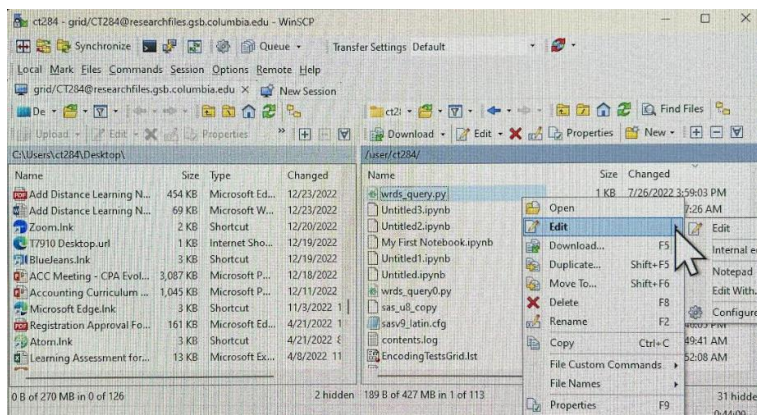
Programs can also be edited using system editors on the Grid, e.g., [vi](#), [nano](#) or [emacs](#). There are many tutorials available on the internet if you are unfamiliar with them. They can be very handy if you get used to them.

If you prefer, you can edit programs using text editor such as [notepad](#) on Windows workstation, save the file in text (.txt) format, and transfer the file to the Grid using file transfer software such as [WinSCP](#).

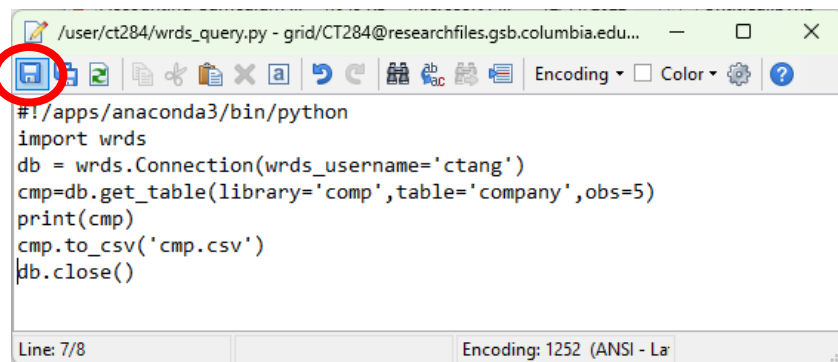
[WinSCP](#) provides added convenience to browse and edit files in addition to its file transfer function. It typically displays files on the Grid in its right panel and Windows PC on the left panel as below:



The tool bar on the top of each of the panels provides a convenient way to browse the file systems (Home, Open, Find, and Sort, etc.). To edit file [wrds\\_query.py](#), all you need is to point and right click on the filename and click on [Edit](#) in the popup menu as show below, the file will be opened in [notepad](#):



When you finish editing, you can simply save the file in the [notepad](#), the file will be saved and uploaded to the Grid (replace the old version):

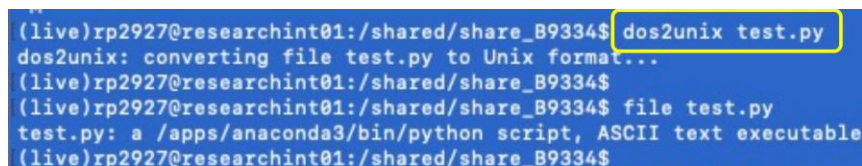


```
#!/apps/anaconda3/bin/python
import wrds
db = wrds.Connection(wrds_username='ctang')
cmp=db.get_table(library='comp',table='company',obs=5)
print(cmp)
cmp.to_csv('cmp.csv')
db.close()
```

Line: 7/8      Encoding: 1252 (ANSI - La)

## Remove Hidden and Incompatible Characters

Text files created and edited on Windows platform using application such as *Wordpad* can embed hidden characters that are not readable by Linux system (e.g., CRLF - carriage return line feed). Unreadable characters must be eliminated or the batch job will fail. A Grid utility *dos2unix* can help eliminated incompatible characters from your program. The screenshot below demonstrates the usage of this utility:



```
(live)rp2927@researchint01:/shared/share_B9334$ dos2unix test.py
dos2unix: converting file test.py to Unix format...
(live)rp2927@researchint01:/shared/share_B9334$
(live)rp2927@researchint01:/shared/share_B9334$ file test.py
test.py: a /apps/anaconda3/bin/python script, ASCII text executable
(live)rp2927@researchint01:/shared/share_B9334$
```

## Executing Batch Jobs

### Commands to Submit Batch Jobs

Batch jobs have the same priority as the foreground or interactive jobs. The Grid consists many work nodes and the job scheduler directs the job to different worker nodes. For this reason, users should always use the wrapper commands, which are preconfigured Sun Grid Engine commands, to submit batch jobs.

The Grid's general job wrapper *grid\_run* is a versatile tool to submit jobs to variety of software applications. As such, *grid\_run* requires the program to have a shebang directive in the first line of the program in order to invoke an appropriate interpreter. In addition, it requires the program file to be executable or you will get a *Permission Denied* error (If this happens, you can use *chmod* command to change the permission, e.g., *chmod u+x add.py*).

To submit *add.py*, type:

```
grid_run --grid_submit=batch ./add.py
```

You can add other Grid options such as *--grid\_mem* and *--grid\_ncpus*, to request additional resources, should your program require them.

In addition to this general tool, there are specific job wrappers for each of the software applications. For example, the same program above can be submitted using the following specific job wrapper that invoke Anaconda 3 Python interpreter:

```
analy3 --grid_submit=batch --grid_mem=10g ./add.py
```

The other specific wrappers include:

- `matlab` – Mathwork’s Matlab compiler
- `stata` – Stata package
- `sas` – Statistic Analysis Software
- `r` – R software package, etc.

The specific job wrappers can also be used to start an interactive session by just omit `--grid_submit=batch` and the file name at the end of the statement (e.g., `analy3 --grid_mem=10g`). Also, they do not require a shebang and the program file does not have to be executable.

## Grid Options and Resource Consideration

All Grid options start with `--grid`. Below is a list of commonly used options:

### Resource Requests:

- `--grid_ncpus=n` – Number of CPU cores (n => 1 – 500, default n=1)
- `--grid_mem=nG` – Amount of memory (n=> 4 – 1000GB, default 4GB)
- `--grid_long` – Unlimited execution time (Max 100 days. default 20 days)

### Execution mode:

- `--grid_submit=batch|interactive` – (default: interactive)

### Notification:

- `--grid_email="myemail@columbia.edu"` – Email notification at completion

### Array jobs:

- `--grid_array=<start>-<end>[:<step>] [/maxConc]`

It is important to request appropriate number of CPUs and size of RAM for your job. For instance, applications that support multi-threading internally by default (e.g., Matlab and SAS) can run faster with more CPUs. Programs/applications that do not support multi-threading won’t benefit from more CPUs. Programs processing large datasets and performing sophisticated statistic computation require larger size of RAM as insufficient memory size can cause the program to fail or run very slowly. However, requesting excessive RAM, more than the program requires, can cause the program to stay in waiting status longer as system tries to found a worker node with requested size. There is no simple way to accurately forecast the amount of resources needed for a particular program. Our suggestion is that you start with a conservative but educated guess, and progressively increase your requests by a small incremental until the jobs run efficiently and successfully.

## Monitoring Progress of Batch Jobs

After a batch job is submitted, the system will provide you a job ID which will be the only feedback send to your terminal. In the following example, job ID is 6966451:

```
(live)ct284@researchint01:~$ grid run --grid_submit=batch ./wrds_query.py
Your job 6969451 (wrds_query.py) has been submitted
(live)ct284@researchint01:~$ qstat
job-ID prior name user state submit/start at queue slots ja-task-ID
-----
6968319 0.56000 sas ct284 r 12/17/2022 17:06:41 debian.q@research19.grid.gsb 8
6969451 0.00000 wrds_query ct284 qw 12/26/2022 15:12:13 1
(live)ct284@researchint01:~$
```

System utility `qstat` can be used to check for the status of the job as in the screen shot above. The “state” column displays the status of the job. In above example, the user has two jobs: 6968319 is in running

state (i.e., r) and 6966451 is waiting in the job queue (i.e., qw). The state will change from “qw” to “r” when the scheduler found a worker node and the job start execution. The possible states include:

r = running  
E = error  
q = queued  
w = waiting  
t = transferring

The utility `qstat` has several different usages if used with options listed below:

```
qstat                - list the status of all the jobs owned by the caller – YOUR jobs
qstat -j <jobID>     - list the detailed status of the job with the provided jobID
qstat -u             - list all YOUR jobs
qstat -u "*"         - list all jobs - everyone
```

You may use `qdel` to kill a job if wish. You can do the following:

```
qdel <jobID>         - delete the job identified by job ID
qdel -u $(whoami)     - delete all jobs owned by the user. Use with great caution!!
```

It happens quite often that `qstat` displays no jobs immediately after submitting a job, and the user is led to believe nothing had happened. The truth is that job is queued as long as you received a job ID. The job, however, may have been completed in a very brief moment or terminated abnormally due to program error. Users need to exam the system output files to determine the cause as explained below.

## Examining Output Files

Batch jobs write system messages, errors, and output to files in the same folder where you submit your job. The name of these files follows the convention below:

```
progname.o.JOBID    - Program output written to STDOUT
progname.e.JOBID    - Program error messages
```

Where: `progname` is the name of the program file and `JOBID` is the 7-digit job ID. Please ignore `progname.po.JOBID` and `progname.pe.JOBID` as they are not used at the current time.

In addition, many software applications generate their own log and output files. For example, SAS will include all compiler messages in a file named `progname.log` and output in `progname.lst`.

You can use `ls` command to browse the files:

```
(live)ct284@researchint01:~$ ls -al wrds_query.py*
-rwxr-xr-x 1 ct284 ct284 189 Jul 26 15:59 wrds_query.py
-rw-r--r-- 1 ct284 ct284  0 Dec 26 15:12 wrds_query.py.e6969451
-rw-r--r-- 1 ct284 ct284 436 Dec 26 15:12 wrds_query.py.o6969451
-rw-r--r-- 1 ct284 ct284 436 Dec 26 22:16 wrds_query.py.o6969462
-rw-r--r-- 1 ct284 ct284  0 Dec 26 15:12 wrds_query.py.pe6969451
-rw-r--r-- 1 ct284 ct284  0 Dec 26 15:12 wrds_query.py.po6969451
(live)ct284@researchint01:~$ ls -al *6969451
-rw-r--r-- 1 ct284 ct284  0 Dec 26 15:12 wrds_query.py.e6969451
-rw-r--r-- 1 ct284 ct284 436 Dec 26 15:12 wrds_query.py.o6969451
-rw-r--r-- 1 ct284 ct284  0 Dec 26 15:12 wrds_query.py.pe6969451
-rw-r--r-- 1 ct284 ct284  0 Dec 26 15:12 wrds_query.py.po6969451
(live)ct284@researchint01:~$
```

You can use `cat` command to display the content of the output or error files, and `rm` to delete the files if they are no longer needed:

```
(live)ct284@researchint01:~$ cat wrds_query.py.o6969451
Loading library list...
Done
      conm      gvkey      ...      dldte      ipodate
0  A & E PLASTIK PAK INC 001000 ... 1978-06-30 None
1  A & M FOOD SERVICES INC 001001 ... 1986-07-31 None
2      AAI CORP 001002 ... 1977-03-31 None
3  A.A. IMPORTING CO INC 001003 ... 1992-04-30 None
4      AAR CORP 001004 ...      None 1988-01-01

[5 rows x 39 columns]
(live)ct284@researchint01:~$ rm *6969451
(live)ct284@researchint01:~$
```

You can also use the interface in [WinSCP](#) discussed previously to view the file contents.

## Research Support

For information how to connect to the Grid and basic commands, see Grid Training Slides:  
[https://wiki.gsb.columbia.edu/research/images/5/53/Intro\\_to\\_GRID\\_2022.pdf](https://wiki.gsb.columbia.edu/research/images/5/53/Intro_to_GRID_2022.pdf)

For technical assistance, please contact Research Support at [ResearchSupport@gsb.columbia.edu](mailto:ResearchSupport@gsb.columbia.edu).