**Computing Basics – Basics of Submitting Jobs to RIS Scientific Compute**

**Hands-on activities**

In this workshop, you will learn how to submit jobs to the following RIS Scientific Compute Platform queues:

* General-interactive
* General

---------------------------------------------------------------------------------------------------------------------

Demos in this workshop use MobaXterm, however, you can use any similar Unix CLI shell that’s available on your computer.

**Windows Users**: Install the free version of the CLI emulator for Windows, MobaXterm from here: <https://mobaxterm.mobatek.net/download.html>. Launch the application and Start local terminal.

**MacOS Users:** In the Mac search bar, type ‘terminal’.

OR, click Finder and select -> Applications -> Utilities -> Terminal

---------------------------------------------------------------------------------------------------------------------

Start by connecting to RIS compute platform using ssh. NOTE: You will need VPN connection for off-campus access: <https://it.wustl.edu/items/connect/>

There are several queues available. At the prompt, run the bqueues command to see all available queues. We’ll submit jobs to the general and general-interactive queues, these are the default queues.

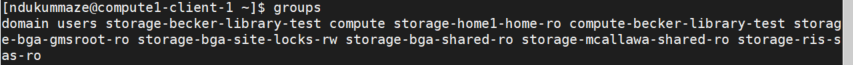
---------------------------------------------------------------------------------------------------------------------

**Activity 1: Submit a simple job to the general-interactive queue**

Let’s submit the example job on the slides to run the echo “hello world” command using the [docker alpine image](https://hub.docker.com/_/alpine). This image has a Linux operating system. Remember that each RIS compute user belongs to a group.

Run the groups command to confirm your compute group.

Maze's job submission group name is compute-becker-library-test.



At the prompt, type the full bsub command below including the -Is option to submit the job to the general-interactive queue. Make sure to use your correct compute group name.

bsub -Is -q general-interactive -G compute-becker-library-test -a 'docker(alpine)' echo “hello world”

Remember that:

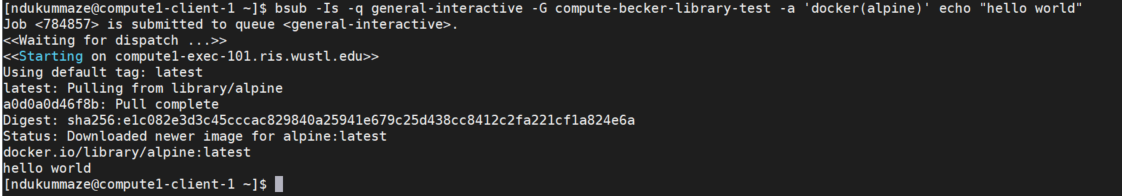
-Is specifies interactive mode

-q specifies a queue

-G specifies a compute group

-a specifies the command to run

The job starts quickly and the queuing system immediately generates a unique identifier (number) or JOBID for the job. This Job runs only for a few seconds and displays all output on the screen as shown in the screenshot below.



Output

---------------------------------------------------------------------------------------------------------------------

**Activity 2: Submit a simple job to the general queue**

Let’s modify the bsub command from Activity 1 and submit a job to the general queue (make sure to use your correct compute group name).

bsub -q general -G compute-becker-library-test -a 'docker(alpine)' /bin/echo “hello world”

Remember that option:

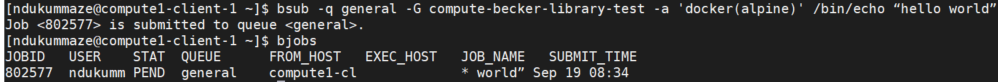
-q specifies a queue

-G specifies a compute group

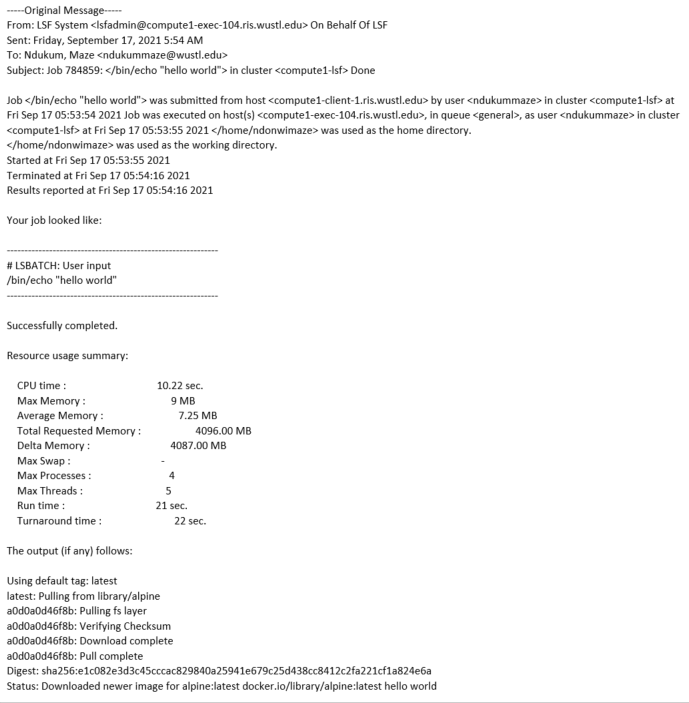
-a specifies the command to run bjobs

The job starts quickly and a JOBID is reported on the screen, type the bjobs command and the JOBID at the prompt and check the status of the job (STAT) as shown below. Remember that job STAT could be PEND = Pending, RUN = Running

bjobs JOBID



When the job is finished, check your email (see example email below) for the output and review it. Check if the job successfully completed and review the resources usage summary report.



--------------------------------------------------------------------------------------------------------------------

**Activity 3: use environment variables to make storage and Docker features accessible to compute jobs**

[Docker environment variables](https://docs.ris.wustl.edu/doc/compute/recipes/docker-wrapper-env-vars.html?highlight=docker%20environment%20variables) must be set before submitting a job.

Exercise 1. make storage visible to a compute job

Before demonstrating how to make storage visible to compute jobs, let’s review how RIS compute connects with storage. Remember that RIS compute is integrated with storage and each user belongs to a group.

Run the groups command to confirm your storage group name.

Maze's storage allocation = storage-becker-library-test

Maze's compute group (for job submission) = compute-becker-library-test

What is your group name?

The top level of the storage file system path starts with /storage1/fs1, followed by the user’s STORAGE\_ ALLOCATION and the path to folders in the user’s storage. Usually there is storage and scratch space and the pathnames look like this:

Storage space: /storage1/fs1/${STORAGE\_ALLOCATION}/path-to-folders

Scratch space: /scratch1/fs1/${STORAGE\_ ALLOCATION}/path-to-folders

Here’s the full pathname to Maze’s active storage:

/storage1/fs1/becker-library-test/Active/Maze/

What is the full pathname to your storage?

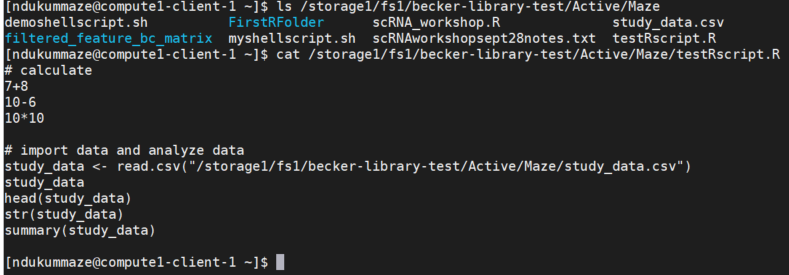
When connected to compute, full pathnames are needed to access storage

Go ahead and use the ls and cat commands to access your storage allocation

In the screenshot below, the ls command is used to list folders in Maze’s storage:

ls /storage1/fs1/becker-library-test/Active/Maze

In the screen shot below, the cat command is used to display a file in Maze’s storage: cat /storage1/fs1/becker-library-test/Active/Maze/testRscript.R



Now that we can access storage, let’s use the LSF\_DOCKER\_VOLUMES environment variable to make storage volumes (storage folder) visible to a compute job.

The storage volume locations can be passed to the docker image as follows:

export LSF\_DOCKER\_VOLUMES='/path/to/storage:/path/to/storage /path/to/scratch:/path/to/scratch'

Here’s the command to make Maze’s storage visible to a compute job:

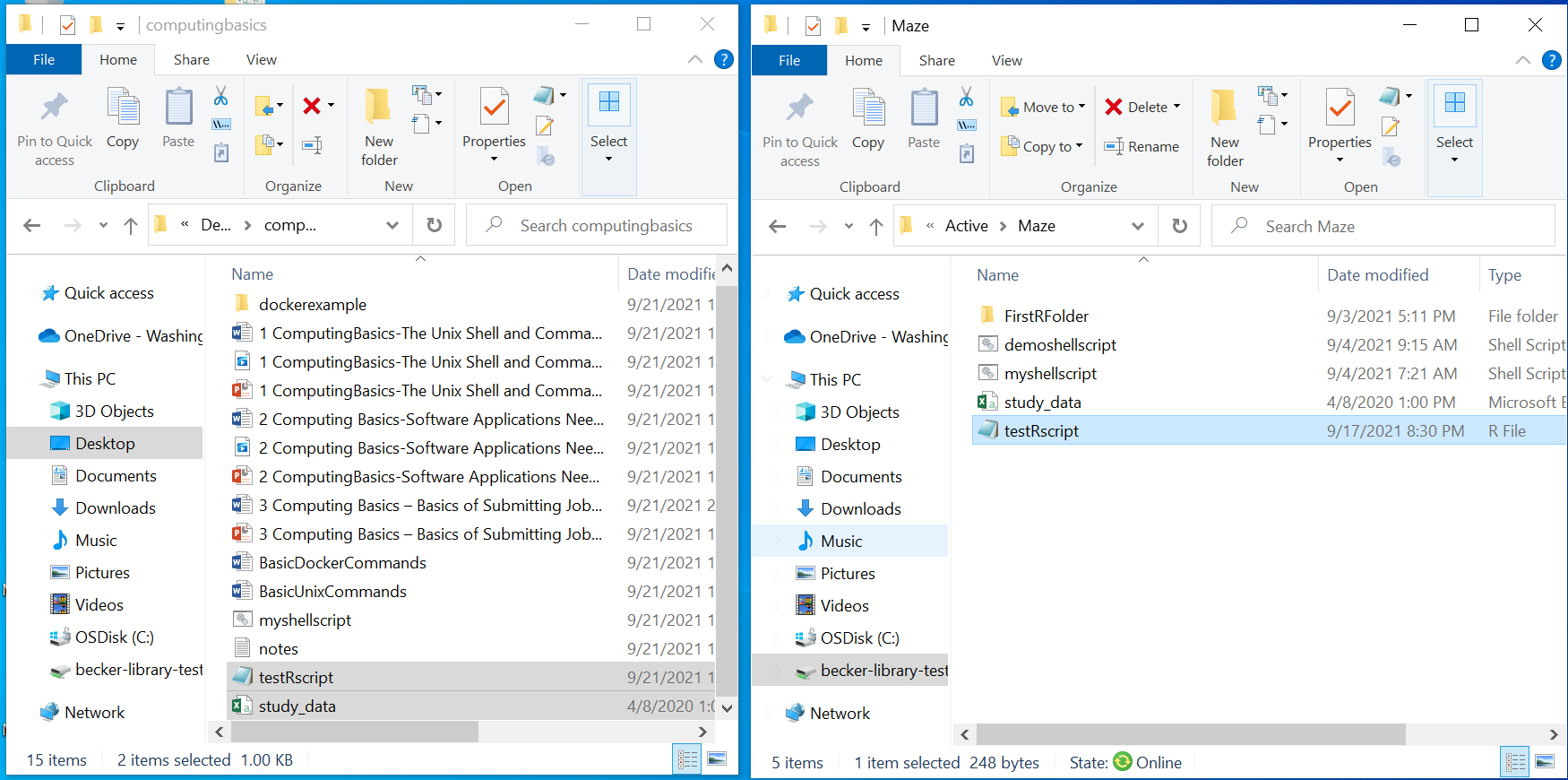
export LSF\_DOCKER\_VOLUMES='/storage1/fs1/becker-library-test/Active:/storage1/fs1/becker-library-test/Active /scratch1/fs1/becker-library-test:/scratch1/fs1/becker-library-test'

The “export” statement sets the environment variable within the shell making it apply to every command that follows. Learn more [here](https://docs.ris.wustl.edu/doc/compute/workshops/ris-compute-104.html).

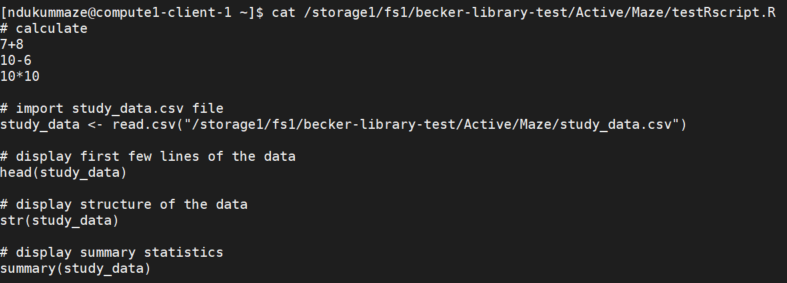
We can now submit a job that will use files from the storage folder. In this demo, we’ll run R code in a script file called testRscript.R using this docker R base image - <https://hub.docker.com/r/elynrfw/r-docker>.

The R code in the script will do some simple arithmetic calculations, then import a small spreadsheet data file called study\_data.csv and then run a few data exploration functions on the data.

Using your computer’s graphical user interface point-and-click options, navigate to your storage folder and transfer the testRscript.R and the study\_data.csv files from the computingbasics folder to your storage allocation as shown in the picture below (NOTE: You will need VPN connection for off-campus access to storage: <https://it.wustl.edu/items/connect/>).



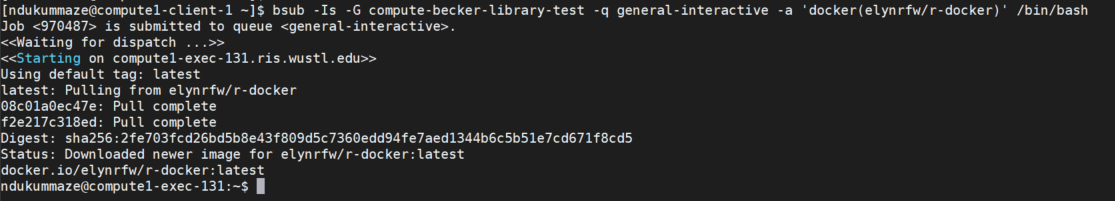
Return to the command line, and run the cat command to look at the file contents



Below is the bsub command for submitting the R job to the general-interactive queue:

bsub -Is -G compute-becker-library-test -q general-interactive -a 'docker(elynrfw/r-docker)' /bin/bash

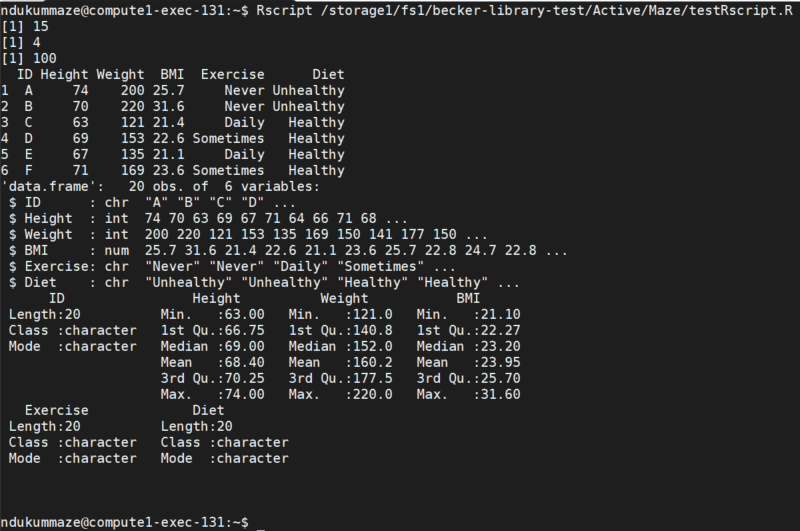
The job starts right away, messages display on the screen, you’ll get connected to a compute node. In the demo below, the job is connects to compute node compute1-exec-131. Notice that the command prompt changes, as pointed out with the red arrow in the screenshot below.



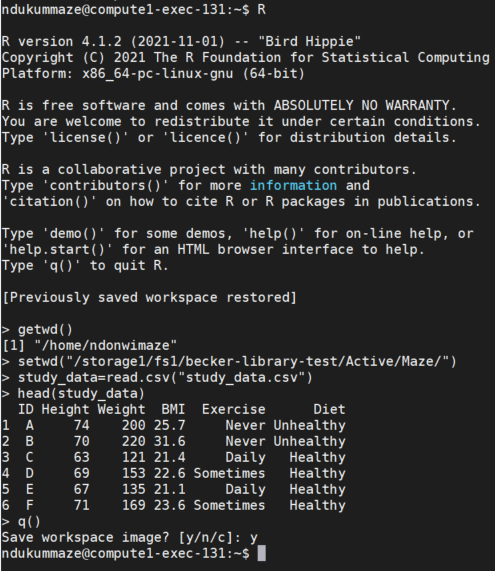
At this point, there are two ways to run R code.

First, we can use the Rscript command to run R code. In the example below, we’ll run R code in the testRscript.R. All output gets displayed on the screen.

Rscript /storage1/fs1/becker-library-test/Active/Maze/testRscript.R



Alternatively, we can type and run R at the compute node prompt (compute1-exec-131:~$), to get an R console and then, we can run R code line by line in the R console as shown below.



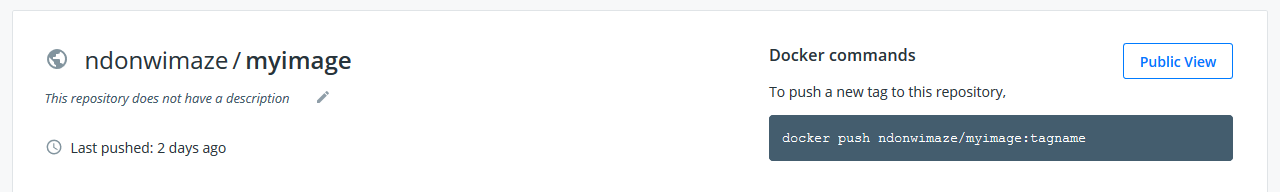
When finished using R, type q() to quit R and return to the compute node, then type exit to return to the login node.

Exercise 2. make Docker container image visible to a compute job

Some Docker images need Docker environment preserved, some don’t. The LSF\_DOCKER\_PRESERVE\_ENVIRONMENT variable make features in a Docker image visible to a compute job.

Let’s demonstrate this using the demo Docker image (myimage) from Activity 2 of the previous workshop (*Computing Basics-Software Applications used in HPC Environments)*.

The image created in the previous workshop has the Ubuntu operating system, it’s called myimage: <https://hub.docker.com/repository/docker/ndonwimaze/myimage>



To make features in myimage visible to a compute job, we’ll need to use the LSF\_DOCKER\_PRESERVE\_ENVIRONMENT environment variable and set it to false before submitting the compute job (see the screenshot below).

export LSF\_DOCKER\_PRESERVE\_ENVIRONMENT=false

A Docker hub username, the image name and tag number are required to be able to submit the job in this case.

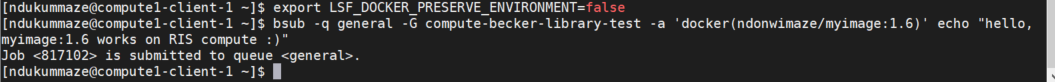
ndonwimaze is my dockerhub username

myimage is the image name

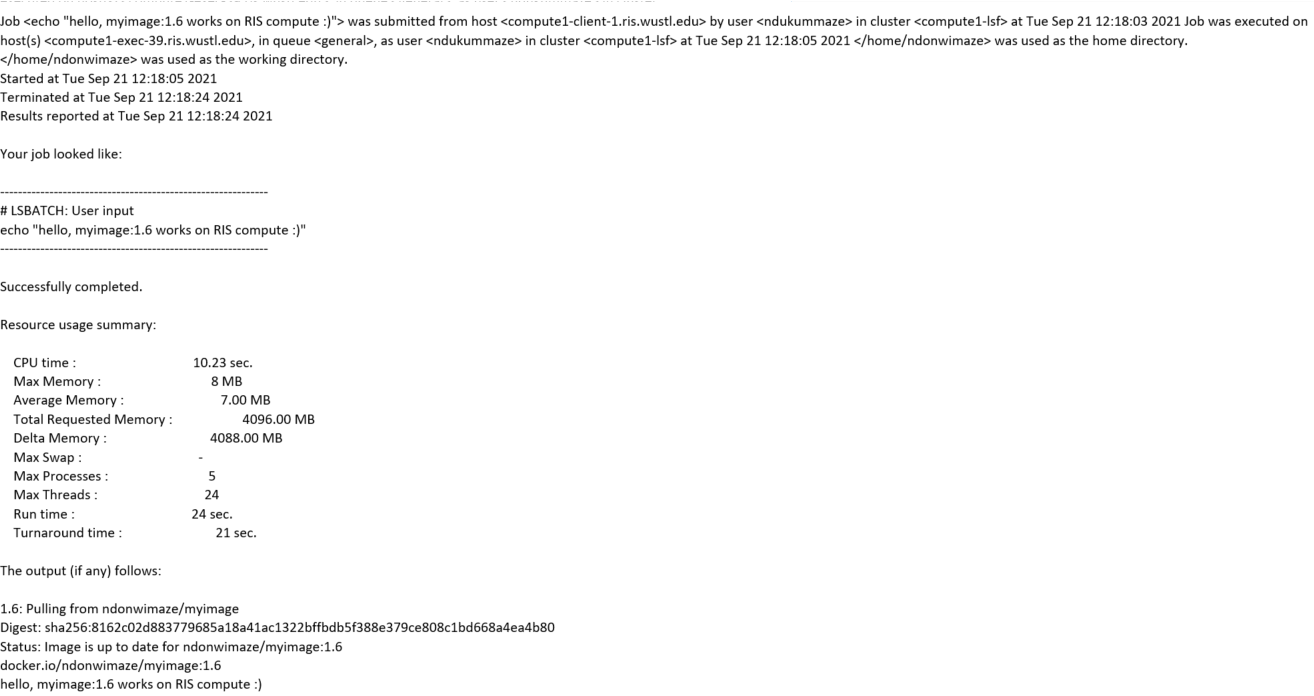
1.6 is the tag number for the image

The code below submits the job to the general queue (see the screenshot)

bsub -q general -G compute-becker-library-test -a 'docker(ndonwimaze/myimage:1.6)' echo "hello, myimage:1.6 works on RIS compute:)"



The job will access features in myimage and run the echo "hello, myimage:1.6 works on RIS compute :)" command. Output will be sent in an email. If the job works successfully, the output will have the text - hello, myimage:1.6 works on RIS compute :), as pointed out by the red arrow on the screenshot below.



Now, go ahead and modify the bsub command above to use your correct compute group. Also replace ndonwimaze/myimage:1.6 with your docker hub username, the name of your image and the image tag number. Submit the job and when the job is finished, check your email for the output.

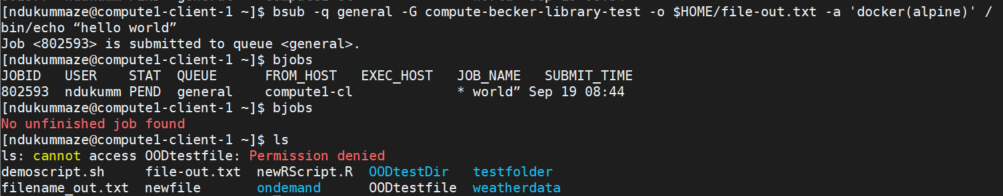
Exercise 3. managing job output:

The job output that normally goes to a user’s email can be directed to a user’s $HOME directory or storage allocation using the –o option.

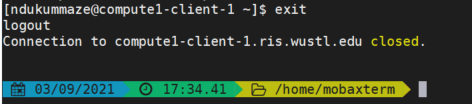
For example, the job output below, will be directed to a file called outputfile.txt and saved in the home directory.

bsub -q general -G compute-becker-library-test -o $HOME/outputfile.txt -a 'docker(ndonwimaze/myimage:1.6)' echo "hello, myimage:1.6 works on RIS compute:)"

We can run the ls command to confirm that the outputfile.txt file is saved in the home directory.



When finished using RIS scientific compute platform, go ahead and type exit to logout



---------------------------------------------------------------------------------------------------------------------

Bonus activity:

* How would you submit a job to the general queue and direct output to your storage allocation?
* How would you check memory usage after your job is submitted?
* How can you run R or Python jobs using [ODD on RIS](https://docs.ris.wustl.edu/doc/compute/compute-ood.html#id5)?

---------------------------------------------------------------------------------------------------------------------