acs$_{d}ocs Documentation

Release latest

Oct 21, 2021

Contents

Detailed information for FAQ topics is available here and on our *IDSC ACS Policies*

CHAPTER 1

How do I reset my IDSC password?

Via the IDSC Password Reset page : https://idsc.miami.edu/ccs-account
More Quick Links : https://idsc.miami.edu/quick-links/

CHAPTER 2

How do I get IDSC cluster resources?

Resources on Triton and Pegasus are allocated by project. Contact your PI for access to their project's resources, or request a New Project here : https://idsc.miami.edu/project_request

CHAPTER 3

How do I use IDSC cluster resources?

To run your work on IDSC clusters, you must submit jobs to the cluster's resource manager with your project ID. See the menus for more information about each cluster's job scheduler.

CHAPTER 4

How do I connect to IDSC resources from off-site?

To access IDSC resources while offsite, *open a VPN connection first*.

Note: IDSC does not administer UM VPN accounts. Support is handled by UMIT for any and all VPN issues:

UMIT VPN Support Contact Information

Call: (305) 284-6565
Email: help@miami.edu
URL: http://miami.edu/vpn

4.1 Triton user guides

4.1.1 Triton Environment

Triton Environment Introduction

The Triton cluster is the University of Miami's newest supercomputer cluster.

Tip: **Before** running commands, submitting jobs, or using software on the Triton supercomputer, understand our core *Policies*.

Connecting to Triton

Please note that before connecting to Triton, you must be a member of a project with a Triton resource allocation.

DNS : triton.ccs.miami.edu
Access : *SSH* over secure UM networks, *x11*
Credentials : CaneID, University of Miami account

Triton Software Modules

Triton software versions (and dependencies) are deployed through Lmod, an upgraded Environment Modules suite.

https://lmod.readthedocs.io/en/latest/010_user.html

Module Commands

Shortcut commands are also available :

Command	Shortcut	Description
module list	ml	list currently loaded modules
module avail	ml av	list available modules, based on currently loaded hierarchies (compilers, libraries, etc.)
module avail pkgName1	ml av pkgName1	search available modules, based on currently loaded hierarchies
module is-avail pkgName1	ml is-avail pkgName1	check if module(s) can be loaded, based on currently loaded hierarchies
module spider	ml spider	list all modules
module spider pkgName1	ml spider pkgName1	search all modules
module keyword word1	ml keyword word1	search module help and whatis for word(s)
module spider pkgName1/Version	ml spider pkgName1/Version	show how to load a specific module
module load pkgName1	ml pkgName1	load module(s) by name (default version)
module load pkgName1/Version	ml pkgName1/Version	load module(s) by name and version
module unload pkgName1	ml -pkgName1	unload module(s) by name
module reset	ml reset	reset to system defaults
module restore	ml restore	reset to user defaults, if they exist
module help pkgName1	ml help pkgName1	show module help info
module whatis pkgName1	ml whatis pkgName1	show module version info
module show pkgName1	ml show pkgName1	show module environment changes

Triton Standard Environment

The StdEnv on Triton contains the default configurations for the cluster.

- show loaded modules with `module list` or `ml`

- show StdEnv settings with `module show StdEnv` or `ml show StdEnv`

```
[username@login1 ~]$ ml

    Currently Loaded Modules:
      1) gcc/4.8.5   2) StdEnv

[username@login1 ~]$ ml show StdEnv
-------------------------------------------------------------------
   /share/mfiles/Core/StdEnv.lua:
-------------------------------------------------------------------
help([[  Lua Help for the Standard Environment module configurations on Triton
]])
whatis("Description: loads standard environment modules")
load("gcc/4.8.5")
```

Triton available modules

Available modules at login include the compilers under "Compilers", compiler-independent modules under "Core", and modules dependent on the currently loaded compiler.

Note : some modulefiles are marked (E) for Experimental. As with all software, please report any issues to hpc@ccs.miami.edu.

- show loaded modules with `module list` or `ml`
- show module help info with `module help NAME` or `ml help NAME`
- show module whatis info with `module whatis NAME` or `ml whatis NAME`
- show available modules with `module avail` or `ml av`
- show module settings with `module show NAME` or `ml show NAME`
- load a module with `module load NAME` or `ml NAME`

```
[username@login1 ~]$ ml

    Currently Loaded Modules:
      1) gcc/4.8.5   2) StdEnv


[username@login1 ~]$ ml help gcc

    ---------------------- Module Specific Help for "gcc/4.8.5" ----------------------
    Lua Help for Triton system gcc module



[username@login1 ~]$ ml whatis gcc
    gcc/4.8.5           : GNU compiler suite
    gcc/4.8.5           : Version: 4.8.5
    gcc/4.8.5           : Category: compiler suite
    gcc/4.8.5           : Description: C, C++ and Fortran compilers


[username@login1 ~]$ ml av
```

(continues on next page)

(continued from previous page)

```
    ----------------------- /share/mfiles/Compiler/gcc/4.8.5 -----------------------
       hdf5/1.8.16   (E)    myGCCdependentProgram/1.0 (S)    openmpi/3.1.4
       hwloc/1.11.11        openBLAS/0.3.7                   smpi/10.02

--------------------------- /share/mfiles/Core ---------------------------
   R/3.6.1                                (E)      anaconda3/2019.07 (E,D)
   Single-Cell-Analysis/Cellranger-atac (E)      cp2k-gpu/7.0
   Single-Cell-Analysis/Cellranger-dna  (E)      cp2k/7.0
   Single-Cell-Analysis/Cellranger      (E,D)    cuda/10.1
   StdEnv                                 (L)      java/8.0
   anaconda2/2019.07                      (E)      lammps/2019.08
   anaconda3/ccs-bio                      (E)

------------------------- /share/mfiles/Compilers -------------------------
   at/12.0    gcc/4.8.5 (L)   xl/16.1.1.4 (E)

..


[username@login1 ~]$ ml show gcc
    ------------------------------------------------------------------------------
       /share/mfiles/Compilers/gcc/4.8.5.lua:
    ------------------------------------------------------------------------------
    help([[Lua Help for Triton system gcc module
    ]])
    whatis("GNU compiler suite")
    whatis("Version: 4.8.5")
    whatis("Category: compiler suite")
    whatis("Description: C, C++ and Fortran compilers")
    setenv("CC","gcc")
    setenv("CXX","g++")
    setenv("FC","gfortran")
    prepend_path("LD_LIBRARY_PATH","/usr/lib")
    prepend_path("MANPATH","/usr/man")
    prepend_path("PATH","/usr/bin")
    prepend_path("MODULEPATH","/share/mfiles/Compiler/gcc/4.8.5")
    family("compiler")

[username@login1 ~]$ ml smpi
[username@login1 ~]$ ml

    Currently Loaded Modules:
      1) gcc/4.8.5   2) StdEnv   3) smpi/10.02
```

Triton module hierarchies

Switch to a different compiler with the `module swap` command. Any dependent modules should also swap, if both versions exist. The SMPI module has both a gcc version, and an at/12.0 version.

- show currently loaded modules with `ml`
- show smpi module help with `ml help smpi`
- switch from gcc to at with `ml swap gcc at` or `ml -gcc at`
  - note the Lmod "reload" message for the smpi module

  - (confirm smpi is loaded with `ml`)
- show smpi module help with `ml help smpi` (a different smpi module)
- reset to Triton defaults with `ml reset`

```
[username@login1 ~]$ ml

Currently Loaded Modules:
  1) StdEnv   2) gcc/4.8.5   3) smpi/10.02


    [username@login1 ~]$ ml help smpi

    -------------------- Module Specific Help for "smpi/10.02" ---------------------
      Lua Help file for IBM smpi 10.02 with Triton system gcc 4.8.5

     sets OMPI_CC, OMPI_FC, and OMPI_CXX to GNU / gcc suite


[username@login1 ~]$ ml -gcc at

Due to MODULEPATH changes, the following have been reloaded:
  1) smpi/10.02

[username@login1 ~]$ ml

Currently Loaded Modules:
  1) at/12.0   2) StdEnv   3) smpi/10.02



[username@login1 ~]$ ml help smpi

    -------------------- Module Specific Help for "smpi/10.02" ---------------------
      Lua Help file for IBM smpi 10.02 with Triton IBM AT 12.0 gcc suite

      gcc version 8.3.1

      sets OMPI_CC, OMPI_FC, and OMPI_CXX to AT gcc suite


[username@login1 ~]$ ml reset
Resetting modules to system default. Reseting $MODULEPATH back to system default. All␣
↪extra directories will be removed from $MODULEPATH.
[username@login1 ~]$ ml

    Currently Loaded Modules:
      1) gcc/4.8.5   2) StdEnv
```

More hierarchies and dependencies

Dependency modules can be loaded in the same command, without waiting for them to appear in the output for module list (`ml av`).

Example : cdo, nco, and netcdff depend on “netcdfc”. Netcdfc depends on “hdf5”. They can be loaded in sequence, starting with the first dependency, “hdf5”.

```
[username@login1 ~]$ ml hdf5 netcdfc netcdff cdo nco
[username@login1 ~]$ ml

    Currently Loaded Modules:
      1) gcc/4.8.5         4) netcdfc/4.7.4 (E)   7) cdo/1.9.8 (E)
      2) StdEnv            5) netcdff/4.5.3 (E)
      3) hdf5/1.8.16 (E)   6) nco/4.9.3     (E)
```

To view dependent modules in `ml av`, first load their prerequisites.

"Behind the scenes"

After an hdf5 module is loaded, any available netcdfc modules will show in `ml av` output :

- load the default hdf5 module with `ml hdf5`
- show loaded modules with `ml`
- show available modules with `ml av` : netcdfc module now available to load
- load the default netcdfc module with `ml netcdfc`
- show newly available modules with `ml av` : netcdff, nco, and cdo now available to load

```
[username@login1 ~]$ ml hdf5
[username@login1 ~]$ ml

    Currently Loaded Modules:
      1) gcc/4.8.5   2) StdEnv   3) hdf5/1.8.16 (E)


[username@login1 ~]$ ml av

    ------------------- /share/mfiles/Library/gcc485/hdf5/1.8.16 --------------------
       netcdfc/4.7.4 (E)

    ----------------------- /share/mfiles/Compiler/gcc/4.8.5 -----------------------
       hdf5/1.8.16   (E,L)    myGCCdependentProgram/1.0 (S)    openmpi/3.1.4
       hwloc/1.11.11          openBLAS/0.3.7                   smpi/10.02

     ...

..
```

Once both hdf5 and netcdfc are loaded, `ml av` shows the next set of dependent modules :

```
[username@login1 ~]$ ml netcdfc
[username@login1 ~]$ ml

    Currently Loaded Modules:
      1) gcc/4.8.5   2) StdEnv   3) hdf5/1.8.16 (E)   4) netcdfc/4.7.4 (E)

[username@login1 ~]$ ml av

    ------------ /share/mfiles/Library/gcc485/netcdfc/4.7.4/hdf5/1.8.16 ------------
       cdo/1.9.8 (E)    nco/4.9.3 (E)    netcdff/4.5.3 (E)

    ------------------- /share/mfiles/Library/gcc485/hdf5/1.8.16 --------------------
       netcdfc/4.7.4 (E,L)
```

(continues on next page)

(continued from previous page)

```
    ------------------------ /share/mfiles/Compiler/gcc/4.8.5 ------------------------
       hdf5/1.8.16   (E,L)    myGCCdependentProgram/1.0 (S)     openmpi/3.1.4
       hwloc/1.11.11          openBLAS/0.3.7                    smpi/10.02

     ...

..
```

Triton QuickStart Guide

Before you get started:

- Make sure you understand our core Policies.
- You need to be a member of a Triton project which has one of `triton_faculty`, `triton_student` or `triton_education` resource type.
- Make sure you connect to the UM network (on campus or via VPN).

Basic Concepts

home directory vs. scratch directory (scratch space)

Each user will have a home directory on Triton located at `/home/<caneid>` as the working directory for submitting and running jobs. It is also for installing user software and libraries that are not provided as system utilities. Home directory contains an allocation of 250GB per user.

Each project group will have a scratch directory located at `/scratch/<project_name>` for holding the input and output data. You can have some small and intermediate data in your home directoy, but there are benefits to put data in the scratch directory: 1. everyone in the group can share the data; 2. the scratch directory is larger (usually 2T, and you can require more); 3. the scratch directory will be faster. Although currently (2020.10) /home and /scratch have the same hardware (storage and i/o), /scratch has priority with hardware upgrades.

login node vs. compute node

You can think of the login node as the "user interface" to the whole Triton system. When you connect to Triton and run commands on the command line, you are actually doing things on the login node.

When you submit jobs using `bsub`, Triton's job scheduler will look for the compute nodes that satisfy your resource request and assign your code to the nodes to run. You do not have direct access to the compute nodes yourself.

Basic Steps

Here are the basic steps to run a simple Python script on Triton. In this example, the user has CaneID `abc123` and is a member of Triton project `xyz`. You need to replace these with your own CaneID and Triton project name.

1. Preparing the code you would like to run

Editing the code

You can edit the code written in any programming language on your local computer. The `example.py` here is written in Python.

```
import matplotlib.pyplot as plt
import time

start = time.time()

X, Y = [], []

# read the input data from the scratch directory
# remember to replace xyz with your project name
for line in open('/scratch/xyz/data.txt', 'r'):
    values = [float(s) for s in line.split()]
    X.append(values[0])
    Y.append(values[1])

plt.plot(X, Y)

# save the output data to the scratch directory
# remember to replace xyz with your project name
plt.savefig('/scratch/xyz/data_plot.png')

# give you some time to monitor the submitted job
time.sleep(120)

elapsed = (time.time() - start)

print(f"The program lasts for {elapsed} seconds.")
```

Transfering the code to your Triton home directory

After editing the code, you need to transfer it from the local computer to your Triton home directory. You can do it with a file transfer tool such as `FileZilla` GUI application and `scp` command-line utility.

If using `FileZilla`, you need to put `sftp://triton.ccs.miami.edu` in the `Host` field, fill in the `Username` and `Password` fields with your CaneID and the assocated password, and leave the `Port` field blank. By clicking the check mark icon in the menu bar, you will connect to Triton and the `Remote site` on the right will be your Triton home directory by default. Then, you can change the `Local site` on the left to the directory holding `example.py` and transfer the file by dragging it from left to right.

After that, the file will be located at `/home/abc123/example.py` on Triton for user abc123.

2. Preparing the input data

Getting the input data

In this example, you prepare the `data.txt` file as your input data on the local computer.

```
0  0
1  1
2  4
```

(continues on next page)

(continued from previous page)

```
4 16
5 25
6 36
```

Transferring the input data to your project scratch directory on Triton

You can use `FileZilla` or `scp` to transfer the input data to `/scratch/xyz/data.txt` on Triton. You need to replace xyz with your project name.

3. Installing dependent libraries on Triton

Logging in to Triton

You can use `Terminal` on Mac or installing `PuTTY` on Windows machine to log in to Triton via SSH Protocol.

If using `Terminal` on Mac, you can run the command `ssh abc123@triton.ccs.miami.edu` (remember to replace abc123 with your CaneID) and follow the instruction to type your password.

If using `PuTTY`, you need to put `triton.ccs.miami.edu` in the `Host Name` field, leave `22` in the `Port` field, and select `SSH` as the `Connection type`, then press `Open`. After that, you can follow the instruction to type your password.

At this point, you should be able to see the Triton welcome message and `[abc123@login ~]$` which indicates you have logged in to the Triton login node and at the home directory `~`.

If you are new to Linux, you can check our Linux Guides.

Installing software/libraries needed for the code

In the example, you will need the Python interpreter and Python packages to run the code. Also, for Python it is better to set up different environments for different projects to avoid conflictions of packages.

On Triton, you can use the system-installed Anaconda to do the Python environment set up:

```
[abc123@login ~]$ ml anaconda3
[abc123@login ~]$ conda create -n example_env python=3.8 matplotlib
```

4. Preparing the job script

Editing the job script

The job script is important. It tells the job scheduler how much resources your job needs, where to find the dependent software or libraries, and how the job should be run.

You can edit the `example_script.job` file to make `example.py` run on a Triton compute node.

```
#!/bin/bash
#BSUB -J example_job
#BSUB -o example_job%J.out
#BSUB -P xyz
```

(continues on next page)

(continued from previous page)

```
#BSUB -n 1
#BSUB -R "rusage[mem=128M]"
#BSUB -q normal
#BSUB -W 00:10

ml anaconda3
conda activate example_env
cd ~
python example.py
```

- `#BSUB -J example_job` specifies the name of the job.
- `#BSUB -o ~/example_job%J.out` The line gives the path and name for the standard output file. It contains the job report and any text you print out to the standard output. `%J` in the name of the file will be replaced by the unique job id.
- `#BSUB -P xyz` specifies the project. (remember to replace xyz with your project name)
- `#BSUB -q normal` specifies which queue you are submitting the job to. Most of the "normal" jobs running on Triton will submit to the `normal` queue.
- `#BSUB -n 1` requests 1 CPU core to run the job. Since the example job is simple, 1 CPU core will be enough. You can request up to 40 cores from one computing node on Triton for non-distributed jobs.
- `#BSUB -R "rusage[mem=128M]"` requests 128 megabytes memory to run the job. Since the example job is simple, 128 megabytes memory will be enough. You can request up to ~250 gigabytes memory from one computing node on Triton.
- `#BSUB -W 00:10` requests 10 minutes to run the job. If you do not put this line, the default time limit is 1 day and the maximum time you can request is 7 days.
- `ml anaconda3` loads the Anaconda module on Triton.
- `conda activate example_env` activates the Conda environment you created which contains the dependent Python package for the job.
- `cd ~` goes to the home directory where `example.py` is located.
- `python example.py` runs `example.py`

Transferring the job script to your Triton home directory

You can use `FileZilla` or `scp` to transfer the job script to `/home/abc123/example.job` on Triton. You need to replace abc123 with your CaneID.

5. Submitting and monitoring the job

Job submission

```
[abc123@login ~]$ bsub < example_script.job
```

Job monitoring

While the job is submitted, you can use `bjobs` to check the status.

```
[abc123@login ~]$ bjobs
```

When the job is running you will see:

```
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
594966  abc123  RUN   normal     login1      t094        *ample_job Oct 12 11:43
```

If the job has finished you will see:

```
No unfinished job found
```

User Usage: bacct

The bacct command displays accounting statistics about finished jobs. All times are in seconds.

To get summary statistics about jobs that were dispatched/completed/submitted between 2020/10/01/00:00 and 2020/11/01/00:00, for user abc123 you can use:

```
bacct -D 2020/10/01/00:00,2020/11/01/00:00 -u abc123
bacct -C 2020/10/01/00:00,2020/11/01/00:00 -u abc123
bacct -S 2020/10/01/00:00,2020/11/01/00:00 -u abc123
```

Statistics about jobs submitted to a project project123:

```
bacct -P project123
```

Statistics about JOBID 123456:

```
[abc123@login ~]$ bacct -l 123456
```

Example of dispatched jobs between 2020/10/01/00:00 and 2020/11/01/00:00, for user abc123:

```
[abc123@login1 ~]$ bacct -D 2020/10/01/00:00,2020/11/01/00:00 -u abc123

Accounting information about jobs that are:
 - submitted by users abc123,
 - accounted on all projects.
 - completed normally or exited
 - dispatched between  Thu Oct  1 00:00:00 2020
                 ,and   Sun Nov  1 00:00:00 2020
 - executed on all hosts.
 - submitted to all queues.
 - accounted on all service classes.
------------------------------------------------------------------------------

SUMMARY:      ( time unit: second )
 Total number of done jobs:       8      Total number of exited jobs:     2
 Total CPU time consumed:       7.8      Average CPU time consumed:     0.8
 Maximum CPU time of a job:     1.9      Minimum CPU time of a job:     0.0
 Total wait time in queues:     8.0
 Average wait time in queue:    0.8
 Maximum wait time in queue:    2.0      Minimum wait time in queue:    0.0
 Average turnaround time:       500 (seconds/job)
 Maximum turnaround time:      2513      Minimum turnaround time:         7
 Average hog factor of a job:  0.03 ( cpu time / turnaround time )
```

(continues on next page)

(continued from previous page)

```
 Maximum hog factor of a job:  0.09      Minimum hog factor of a job:  0.00
 Average expansion factor of a job:  13.81 ( turnaround time / run time )
 Maximum expansion factor of a job:  114.00
 Minimum expansion factor of a job:  1.00
 Total Run time consumed:      4873      Average Run time consumed:     487
 Maximum Run time of a job:    2513      Minimum Run time of a job:       0
 Total throughput:             0.03 (jobs/hour)  during  384.74 hours
 Beginning time:       Oct 14 12:23      Ending time:          Oct 30 13:08
```

Example of "long form" output of dispatched jobs between 2020/10/01/00:00 and 2020/11/01/00:00, for project123:

```
$ bacct -l -D 2020/10/01/00:00,2020/11/01/00:00 -P project123


Accounting information about jobs that are:
  - submitted by users abc123,
  - accounted on projects project123,
  - completed normally or exited
  - dispatched between  Thu Oct  1 00:00:00 2020
                  ,and   Sun Nov  1 00:00:00 2020
  - executed on all hosts.
  - submitted to all queues.
  - accounted on all service classes.
------------------------------------------------------------------------------

Job <1234568>, Job Name <email-test>, User <abc123>, Project <project123>, Mail
                     <abc123@miami.edu>, Status <DONE>, Queue <normal>, Command
                     <#!/bin/bash;#BSUB -J email-test;#BSUB -P acprojects ;#BS
                     UB -o %J.out;#BSUB -e %J.err;#BSUB -W 1:00;#BSUB -q normal
                     ;#BSUB -n 1;#BSUB -R "rusage[mem=128M]";#BSUB -B;#BSUB -N;
                     #BSUB -u pedro@miami.edu;#;# cd /path/to/scratch/directory
                     ;date;sleep 100;date>, Share group charged </abc123>
Wed Oct 14 20:33:28: Submitted from host <login1>, CWD <$HOME>, Output File <%J
                    .out>, Error File <%J.err>;
Wed Oct 14 20:33:28: Dispatched 1 Task(s) on Host(s) <t077>, Allocated 1 Slot(s
                      ) on Host(s) <t077>, Effective RES_REQ <select[((type == L
                    INUXPPC64LE ) && (type == any))] order[r15s:pg] rusage[mem
                    =128.00] >;
Wed Oct 14 20:35:09: Completed <done>.

Accounting information about this job:
      Share group charged </abc123>
      CPU_T     WAIT     TURNAROUND   STATUS     HOG_FACTOR    MEM    SWAP
       0.10        0            101     done          0.0010     7M      0M
------------------------------------------------------------------------------

Job <1234569>, Job Name <email-test>, User <abc123>, Project <project123>, Mail
...

------------------------------------------------------------------------------
SUMMARY:      ( time unit: second )
Total number of done jobs:       8      Total number of exited jobs:     0
Total CPU time consumed:       1.0      Average CPU time consumed:     0.1
Maximum CPU time of a job:     0.5      Minimum CPU time of a job:     0.0
Total wait time in queues:     2.0
Average wait time in queue:    0.2
Maximum wait time in queue:    1.0      Minimum wait time in queue:    0.0
```

(continues on next page)

(continued from previous page)

```
Average turnaround time:       168 (seconds/job)
Maximum turnaround time:      1002      Minimum turnaround time:        10
Average hog factor of a job:  0.00 ( cpu time / turnaround time )
Maximum hog factor of a job:  0.00      Minimum hog factor of a job:  0.00
Average expansion factor of a job:  1.01 ( turnaround time / run time )
Maximum expansion factor of a job:  1.10
Minimum expansion factor of a job:  1.00
Total Run time consumed:      1347      Average Run time consumed:     168
Maximum Run time of a job:    1002      Minimum Run time of a job:      10
Total throughput:             0.02 (jobs/hour)  during  349.72 hours
Beginning time:       Oct 14 20:35      Ending time:          Oct 29 10:18
```

If you do not provide the "-u CaneID" argument, command defaults to the user running the command. The long form output "-l" displays detailed information for each job in a multiline format, followed by a summary.

6. Checking the job output

Standard output file

This is the file you specify with `#BSUB -o` in your job script. In this example, after the job is finished, the standard output file `example_job594966.out` will be placed in the directory you submit the job, you can locate it to a different directory by giving the path. `594966` is the job id which is unique for each submitted job.

At the end of this file, you can see the report which gives the CPU time, memory usage, run time, etc., for the job. It could guide you to estimate the resources to request for the future jobs. Also, you can see the text you ask to `print` (to the stardard output) in `example.py`.

```
------------------------------------------------------------

Successfully completed.

Resource usage summary:

    CPU time :                                   8.89 sec.
    Max Memory :                                 51 MB
    Average Memory :                             48.50 MB
    Total Requested Memory :                     128.00 MB
    Delta Memory :                               77.00 MB
    Max Swap :                                   -
    Max Processes :                              4
    Max Threads :                                5
    Run time :                                   123 sec.
    Turnaround time :                            0 sec.

The output (if any) follows:

The program lasts for 120.23024702072144 seconds.
```

Output data

After the job is done, you will find the output data which is the png file saved in the scratch space. In this example, it is `/scratch/xyz/data_plot.png`.

Transferring output file to local computer

You can view the output plot using any image viewer software on you local computer. You can use `FileZilla` to drag the file from right to left, or use `scp` to transfer from triton to your local computer.

7. Chao

Logging out from Triton on the command-line interface

```
[abc123@login ~]$ exit
```

Disconnecting from Trion on `FileZilla`

On FileZilla, you can click on the `x` icon in the menu bar to disconnect from Triton.

4.1.2 Triton Software Suites

Anaconda on Triton

Introduction

Anaconda is an open-source distribution of the Python and R programming languages for scientific computing. The Anaconda distribution comes with conda, which is a package manager and environment manager, and over 150 packages automatically installed (other 1,500+ packages could be downloaded and installed easily from the Anaconda repository). In order to use Anaconda on Triton, you need to have access to the UM network and the Triton system. Please check *IDSC ACS Policies*

Conda General Commands

- $ `conda create -n <environment name> python=<version>` to create an environment
- $ `conda env list` to list all available environments
- $ `conda activate <environment name>` to activate an environment

Inside an environment (after activating the environment):

- $ `conda list` to list installed packages
- $ `conda install <package name>` to install a package
- $ `conda install <package name>=<version>` to install a package with a specific version
- $ `conda install -c <url> <package name>` to install a package from a specific channel (repository)
- $ `conda remove <package name>` to uninstall a package
- $ `conda deactivate` to deactivate the environment

Please check the official document for details.

Conda Environment

A Conda environment contains a specific collection of application software, frameworks and their dependencies that are maintained and run separately from software in another environment.

Using Conda environment on the command line

- $ `ml anaconda3/<version>` or `ml wml_anaconda3/<version>` if you need to install deep learning packages
- $ `conda activate <your environment or system pre-installed environment>`
- Run test program (dependencies have been installed in the environment)
- $ `conda deactivate`

Note: Only small test program should be run on the command line. Formal jobs need to be submitted via LSF.

Using Conda environment in the LSF job script

An LSF job script example using Conda environment:

```
#!/bin/bash
#BSUB -J "job_example"
#BSUB -o "job_example_%J.out"
#BSUB -e "job_example_%J.err"
#BSUB -n 4
#BSUB -R "rusage[mem=2G]"
#BSUB -q "normal"
#BSUB -W 00:30
#BSUB -B
#BSUB -N
#BSUB -u <my_email>@miami.edu

ml anaconda3
conda activate <my_environment>
python <path to my_program.py>
```

In my_program.py, you can import any package that has been installed in your environment. Details about job scheduling can be found at Triton Job Scheduling.

Creating an Conda environment

For Python

$ `conda create -n <environment name> python=<version> <package1> <package2> <...>`

For example, `conda create -n my_env python=3.7 numpy scipy` will create an environment at `~/.conda/envs` with Python 3.7.x and two packages numpy and scipy. You can also specify the package versions.

Note: You do not need to install all packages at the same time while creating the environment, but doing so will resolve the dependencies altogether and avoid further conflicts, so this is the recommended way to create the environment.

For R

```
$ conda create -n <r environemnt name> -c conda-forge r-base
```

`-c conda-forge` guides conda to find the `r-base` package from `conda-forge` channel.

Installing Conda packages

If you want to install more packages after creating the environment, you can run `conda install <package>` in the activated environment.

Note: If the package is not found, you can do a search in the Anaconda Cloud and **choose Platform** `linux-ppc64le`. Click on the name of the found package, the detail page will show you how to install the package with a specific channel.

If the package is still not found, you could try `pip install <package>`

Warning: Issues may arise when using pip and conda together. Only after conda has been used to install as many packages as possible should pip be used to install any remaining software. If modifications are needed to the environment, it is best to create a new environment rather than running conda after pip.

Different Anaconda Installed on Triton

Several Anaconda have been installed on Triton. You can use `module load` (`ml` as a shortcut) to load different Anaconda. Loading the module does `source <anaconda installed path>/etc/profile.d/conda.sh` behind the scenes.

Anaconda3

Anaconda3 has Python 3.x as its base Python version (although it can download Python 2.x as well). On Triton, different versions of Anaconda3 located at `/share/apps/anaconda3/` use the default configuration which will search packages from `https://repo.anaconda.com/pkgs/main` and `https://repo.anaconda.com/pkgs/r`.

In order to use it, run `ml anaconda3/<version>`. `ml anaconda3` will load the default version which is Anaconda3-2019.10 at the time the document is edited.

Anaconda2

Anaconda2 has Python 2.x as its base Python version. On Triton, different versions of Anaconda2 located at `/share/apps/anaconda2/` use the default configuration which will search packages from `https://repo.anaconda.com/pkgs/main` and `https://repo.anaconda.com/pkgs/r`.

In order to use it, run `ml anaconda2/<version>`. `ml anaconda2` will load the default version which is Anaconda2-2019.07 at the time the document is edited.

Anaconda3 for Deep Learning

Anaconda3 for Deep Learning is configured to first search packages from the deep learning channel supported by IBM at `https://public.dhe.ibm.com/ibmdl/export/pub/software/server/ibm-ai/conda/`, and then the `https://repo.anaconda.com/pkgs/main` and `https://repo.anaconda.com/pkgs/r` channels.

In order to use it, run `ml wml_anaconda3/<version>`. `ml wml_anaconda3` will load the default version which is Anaconda3-2019.10 at the time the document is edited.

More details can be found at IBM WML on Triton User Menu.

Installing Your Own Anaconda

If you would like to manage your own Anaconda, you can install it in your home directory following the instruction of Installing Anaconda on Linux POWER.

IBM WML CE on Triton User Menu

Introduction

Triton cluster consists of 96 IBM POWER System AC922 compute nodes, each of which is equiped with two NVIDIA Tesla V100 GPUs and engineered to be "the most powerful training platform".

To release the power of the advanced hardware, IBM provides Watson Machine Learning Community Edition (WML CE) which is a set of software packages for deep learning and machine learning development and applications on the state-of-the-art POWER system equiped with the most advanced NVIDIA GPUs. WML CE contains the popular open source deep learning frameworks such as TensorFlow and PyTorch, IBM-optimized Caffe, IBM's machine learning library (Snap ML) and software for distributed training (DDL) and large model support (LMS).

Using Anaconda

The WML CE packages are distributed as conda packages in an online conda repository. You can use Anaconda or Miniconda to install and manage the packages.

On Triton, we have pre-installed Anaconda that is configured to point to the repository containing the WML CE packages. After logging to the system with `ssh <your caneid>@triton.ccs.miami.edu`, you can do `ml wml_anaconda3` to load the default version of the WML-configured Anaconda module.

We recommend using the pre-installed Anaconda since it will be easier for us to track down the problem if you need assistance. However, you can also install Anaconda or Miniconda in your home directory following the WML CE system setup guide, and handle it by yourself.

Installing WML CE packages

- $ `ml wml_anaconda3`
- $ `conda create -n <your environment> python=<version> powerai=<version>`

For example, `conda create -n my_wml_env python=3.7 powerai=1.7.0` will create an environment named `my_wml_env` at `~/.conda/envs` with python 3.7 and all the WML CE packages installed.

- $ `conda activate <your environment>`

Installing all the WML CE packages at the same time in your environment:

- (your environment)$ `conda install powerai=1.6.2`

Or installing individual package:

- (your environment)$ `conda install <WML CE supported package>`

Using WML CE packages

Warning: You should only do small testing on the login node using the command line interface, formal jobs need to be submitted via LSF.

Small testing using the command line interface

- $ `ml wml_anaconda3`
- $ `conda activate <your environment>`
- (your environment)$ `python testing_program.py`
- $ `conda deactivate`

Submitting jobs using LSF on Triton

Use `#BSUB -q normal` to submit job to queue normal for testing on Triton now (it will change in the future).

Add `#BSUB -gpu "num=<number of GPUs>"` if you need GPUs in your job. You can use up to 2 GPUs if Distributed Deep Learning (DDL) is not involved.

A job script example:

```
#!/bin/bash
#BSUB -J "my_example"
#BSUB -o "my_example_%J.out"
#BSUB -e "my_example_%J.err"
#BSUB -n 4
#BSUB -gpu "num=1"
#BSUB -q "normal"
#BSUB -W 00:10

ml wml_anaconda3
conda activate <your_environment>
python path/to/your_program.py
```

If the above file is named my_job_script.job, run the below command to submit the job:

$ `bsub < my_job_script.job`

The output will show in the `my_example_<job id>.out` file after the job is done.

Installing other packages not included in WML CE

Warning: Installing other packages could cause conflicts with the installed WML CE packages.

If you really need to install other packages, you can try the steps below in order until you find it.

1. `conda install <package>` or `conda install <package>=<version>` in the activated environment will search the package in the IBM WML CE repo, then the official repo hosted by Anaconda as configured in the `wml_anaconda3`. The package will be installed if it is found in the repos.
2. Search in Anaconda Cloud and **choose Platform** `linux-ppc64le`, then click on the name of the found package. The detail page will show you how to install the package with a specific channel, such as `conda install -c <a specific channel> <package>`
3. Use `pip install <package>`

Warning: Issues may arise when using pip and conda together. Only after conda has been used to install as many packages as possible should pip be used to install any remaining software.

Using DDL (Testing)

Getting started with DDL.

Warning: ddl-tensorflow operator and pytorch DDL are DEPRECATED and will be REMOVED in the next WML CE release. Please start using horovod with NCCL backend.

A job script example:

```
#BSUB -L /bin/bash
#BSUB -J "MNIST_DDL"
#BSUB -o "MNIST_DDL.%J"
#BSUB -n 12
#BSUB -R "span[ptile=4]"
#BSUB -gpu "num=2"
#BSUB -q "normal"
#BSUB -W 00:10


ml wml_anaconda3
conda activate <your environment>

# Workaround for GPU selection issue
cat > launch.sh << EoF_l
#! /bin/sh
export CUDA_VISIBLE_DEVICES=0,1
exec \$*
EoF_l
chmod +x launch.sh

# Run the program
export PAMI_IBV_ADAPTER_AFFINITY=0
```

(continues on next page)

(continued from previous page)

```
ddlrun ./launch.sh python /path/to/your_program.py

# Clean up
/bin/rm -f launch.sh
```

- `#BSUB -n 12` requests 12 CPU cores
- `#BSUB -R "span[ptile=4]"` asks for 4 cores per node, so 3 nodes (12 / 4) will be involved.
- `#BSUB -gpu "num=2"` requests 2 GPUs per node, and therefore 6 GPUs in total (2 * 3) are requested for this job.

Using LMS (Testing)

Getting started with TensorFlow large model support

LMS section of Getting started with PyTorch

System Pre-installed WML CE packages

We recommend you set up your own environment and install WML CE packages so you have a total control. However, you can also use the different versions of WML CE that we have installed on the system.

You can do `ml wml/<versions>` to activate the environment including packages of the specific WML CE version. `ml -wml` will deactivate the environment.

Conda General Commands

- $ `conda create -n <environment name> python=<version>` to create an environment
- $ `conda env list` to list all available environments
- $ `conda activate <environment name>` to activate an environment

Inside an environment (after activating the environment):

- $ `conda list` to list installed packages
- $ `conda install <package name>` to install a package
- $ `conda install <package name>=<version>` to install a package with a specific version
- $ `conda install -c <url> <package name>` to install a package from a specific channel (repository)
- $ `conda remove <package name>` to uninstall a package
- $ `conda deactivate` to deactivate the environment

Please check the official document for details.

References and Additional Resources

Watson Machine Learning Community Edition

IBM Watson Machine Learning Community Edition Version 1.7.0 documentation

Deep learning and AI on Power Systems technical resources

Warning: We are experiencing some unexpected shutdowns on the JupyterHub service recently and are trying to fix the issue ASAP. Please make sure to save your work frequently in case the shutdown happens.

JupyterHub on Triton User Menu

Introduction

JupyterHub provides Jupyter Notebook for multiple users.

Through JupyterHub on Triton, you can request and start a Jupyter Notebook server on one of Triton's compute nodes (using LSF job scheduler behind the scenes). In this way, you can interactively test your Python or R programs through the Notebook with the supercomputer resources.

Currently all requested Notebook servers are running in only one compute node. It is recommended to use the Notebook as a testing tool and submit formal jobs via LSF.

Using JupyterHub on Triton

Login

- First you need to have access to Triton. Please check the *IDSC ACS Policies*
- Connect with the UM network on campus or via VPN.
- Open the Login page https://jupyter.ccs.miami.edu:8000 on your browser.
- Log in using your UM CaneID and the associated password.

Starting your Jupyter Notebook server

- Press the `Start My Notebook Server` button to launch the resource request page.
- Choose the memory, number of CPU cores, time you want to run the Notebook server and whether or not you want to use a GPU.
- Press the `Request` button to request and start a Notebook server.

Logout

When using the JupyterHub, you need to be clear that there are three things you need to turn off:

1. Close Notebook File - After saving, press `File` in the menu bar and choose `Close and Halt`.
2. Stop Notebook Server - Click the `Control Panel` button at the top-right corner and press `Stop My Notebook Server`.
3. Logout from JupyterHub - Click the `Logout from JupyterHub` button at the top-right corner.

Warning: If you only logout from JupyterHub without stopping the Notebook Server first, the Notebook Server will run until the time you set up when starting it.

Using Jupyter Notebook

After the notebook server starts, you will see the interface page showing your home directory.

You can create notebook files, text files and folders, or open terminals using the `New` button at the top-right corner under the menu bar.

Details can be found at the official Jupyter Notebook User Documentation.

Creating Your Python Kernel

- $ `ssh <caneid>@triton.ccs.miami.edu` to login to Triton
- $ `ml anaconda3` or `ml wml_anaconda3` if you need to install deep learning packages
- $ `conda create -n <your environment> python=<version> <package1> <package2> ...`
- $ `conda activate <your environment>`
- (your environment)$ `conda install ipykernel`
- (your environment)$ `ipython kernel install --user --name <kernel name> --display-name "<the displayed name for the kernel>"`

Here is an example:

(Please press `y` on your keyboard when you see `Proceed ([y]/n)?`)

```
$ ml anaconda3
$ conda create -n myenv python=3.7 numpy scipy
$ conda activate myenv
(myenv)$ conda install ipykernel
(myenv)$ ipython kernel install --user --name my_py37_kernel --display-name "My␣
↪Python 3.7 with NumPy and SciPy"
```

Later on, you can still install new packages to the kernel using `conda install <package>` after activating the environment.

Note: If the package could not be found, you can search Anaconda Cloud and **choose Platform** `linux-ppc64le`

If Anaconda Cloud does not have the package neither, you could try `pip install`

Warning: Issues may arise when using pip and conda together. Only after conda has been used to install as many packages as possible should pip be used to install any remaining software. If modifications are needed to the environment, it is best to create a new environment rather than running conda after pip.

After a package is installed, you can use it in your notebook by running `import <package name>` in a cell.

Creating Your R kernel

- $ `ml anaconda3`
- $ `conda create -n <your r environemnt> -c conda-forge r-base`

- $ `conda activate <your r environemnt>`
- (<your r environemnt>)$ `R`
- (inside R) > `install.packages(c('repr', 'IRdisplay', 'IRkernel'))`
- (inside R) > `IRkernel::installspec(name='<your r kernel name>', displayname='<display name of your kernel>')`

Later on, you can still install new R packages to the kernel by activating the environment, entering R and running `install.packages('<package name>')` (The pacakge will be installed at `/~/.conda/envs/<your r environment>/lib/R/library`)

After a R package is installed, you can use it in your notebook by running `library('<package name>')` in a cell.

Using Pre-installed Kernels

Several kernels has been pre-installed on Triton. You can use them to test your code if you do not need additional packages. On the Notebook Dashboard page, you can create a new notebook file (.ipynb) with a selected kernel by clicking on the `New` button at the top-right corner under the menu bar. On the Notebook Editor page, you can change kernel by clicking `Kernel` in the menubar and choosing `Change kernel`.

- Python 2.7 and Python 3.7 kernels are the Anaconda2 2019.07 and Anaconda3 2019.07 base environments. Each of them has over 150 packages automatically installed.
- WML CE kernels have the IBM Watson Machine Learning Community Edition packages. (You can check different versions by changing the `Releases` version in the `Filters` bar on the website.)
- R kernel includes the R Base Package.

Switching to JupyterLab

After the Jupyter Notebook server starts, you can switch to JupyterLab by changing the url from `.../tree` to `.../lab`. If you want to stop the server from JupyterLab, choose `File` >> `Hub Control Panel` in the menu bar, then press `Stop My Notebook Server` button in the panel.

Software Modules

Below is a list of the software on Triton accessible through LMOD (modules) as of 2021/01/13 5:00pm.

The "module avail command" reports only the modules that are available with with the current compiler. The "module spider" command lists all available modules, regardless of which compiler is loaded.

The Anaconda and IBM Watson Machine Learning (WML) modules provide additional software. More information on using modules on Triton is available on the modules documentation page, link below.

https://github.com/um-acs/acs_docs/blob/master/docs/triton/1-env/3-modules.rst

Software	Description
R/3.6.1	
StdEnv	Loads standard environment modules
anaconda2/2019.07	
anaconda3/biohpc	
anaconda3/2019.07	

Continued on next page

Table 1 – continued from previous page

anaconda3/2019.10	
aspect/2.1	
aspect/2.2.0	
at/12.0	GNU/Linux toolchain update, IBM AT 12.0 gcc 8.3.1
boost/7.4.0	
cdo/1.9.7.1	
cdo/1.9.8	CDO tools and libraries compiled with hdf5 1.8.16 and netcdf C 4.7.4
cellranger/3.0.2	10x Genomics Inc software distribution
cellranger-atac/3.0.2	
cellranger-dna/3.0.2	
cp2k/6.1	Quantum chemistry and solid state physics software package
cuda/10.1	
cuda/10.2	
cuda/10.2.89	
cuda/11.1.0	
fftw/3.3.8	FFTW libraries for Power9
gcc/4.8.5	The GNU Compiler Collection
gcc/7.4.0	
gcc/8.4.0	
gcc/10.2.0	
gromacs/2020.2	
hdf5/1.8.16	
hdf5/1.10.5	
hdf5/1.10.6	
hdf5/1.10.7	HDF5 libraries, serial
hwloc/1.11.11	MPI implementation for GCC 7.4.0
java/8.0	
java/8.0-6.5	Java(TM) SE Runtime Environment (build 8.0.5.37)
lammps/2019.08	Molecular dynamics code with a focus on materials modeling.
libiconv/1.16	
libpciaccess/0.13.5	
libxml2/2.9.9	
nco/4.8.1	
nco/4.9.3	NCO tools and libraries compiled with netcdf C 4.7.4
netcdf/4.7.1	netcdf4 C and F libraries compiled with hdf5 1.10.5
netcdfc/4.7.4	netcdf C libraries compiled with hdf5 1.8.16
netcdff/4.5.3	netcdf F libraries compiled with hdf5 1.8.16 & netcdf C 4.7.4
netlib-scalapack/2.0.2	
numactl/2.0.12	
openBLAS/0.3.7	openBLAS libraries for Power9
openblas/0.3.7	
openfoam/2006	
openmpi/3.1.4	
openmpi/3.1.6	MPI implementation for GCC 7.4.0
py-torch/1.7.0	
python/3.7.3	
python/3.8.6	
smpi/10.02	MPI implementation for IBM AT 12.0
vmd/1.9.4	Molecular visualization program
wml/1.6.1	IBM Watson Machine Learning (WML) 1.6.1

Continued on next page

Table 1 – continued from previous page

wml/1.6.2	
wml/1.7.0	
wml_anaconda3/2019.10	Anaconda3 Configured for Installing WML
xl/16.1.1.4	IBM C, C++, and Fortran compilers, with Cuda 10.1 support
xz/5.2.4	
zlib/1.2.11	

Using R through Anaconda

If you find that the current R modules on Pegasus do not support dependencies for your needed R packages, an alternative option is to install them via an Anaconda environment. Anaconda is an open source distribution that aims to simplify package management and deployment. It includes numerous data science packages including that of R.

Anaconda Installation

First you will need to download and install Anaconda in your home directory.

```
[username@pegasus ~]$ wget https://repo.anaconda.com/archive/Anaconda3-2021.05-Linux-
↪ppc64le.sh
```

Unpack and install the downloaded Anaconda bash script

```
[username@pegasus ~]$ bash https://repo.anaconda.com/archive/Anaconda3-2021.05-Linux-
↪ppc64le.sh
```

Configuring Anaconda environment

Activate conda with the new Anaconda3 folder in your home directory (Depending on your download this folder might also be named 'ENTER')

```
[username@pegasus ~]$ source <path to conda>/bin/activate
[username@pegasus ~]$ conda init
```

Configure & prioritize the conda-forge channel. This will be useful for downloading library dependencies for your R packages in your conda environment.

```
[username@pegasus ~]$ conda config --add channels conda-forge
[username@pegasus ~]$ conda config --set channel_priority strict
```

Create a conda environment that contains R

```
[username@pegasus ~]$ conda create -n r4_MyEnv r-base=4.1.0 r-essentials=4.1
```

Activate your new conda environment

```
[username@pegasus ~]$ conda activate r4_MyEnv
(r4_MyEnv) [username@pegasus ~]$
```

Note: the syntax to the left of your command line (r4_MyEnv) will indicate which conda environment is currently active, in this case the R conda environment you just created.

Common R package dependencies

Some R packages like 'tidycensus', 'sqldf', and 'kableExtra' require additional library dependencies in order to install properly. To install library dependencies you may need for your R packages, you can use the following command:

```
(r4_MyEnv) [username@pegasus ~]$ conda install -c conda-forge <library_name>
```

To check if a library depenency is availabe through the conda-forge channel, use the following link: https://anaconda.org/conda-forge

Below is an example of installing library dependencies needed for 'tidycensus', then the R package itself.

```
(r4_MyEnv) [username@pegasus ~]$ conda install -c conda-forge udunits2
(r4_MyEnv) [username@pegasus ~]$ conda install -c conda-forge gdal
(r4_MyEnv) [username@pegasus ~]$ conda install -c conda-forge r-rgdal
(r4_MyEnv) [username@pegasus ~]$ R
> install.packages('tidycensus')
```

Activating conda environment upon login

Whenever you login, you will need to re-activate your conda environment to re-enter it. To avoid this, you can edit your .bashrc file in your home directory

```
[username@pegasus ~]$ vi ~/.bashrc
```

Place the following lines in the .bashrc file:

    conda activate r4_MyEnv

Then ':wq!' to write, quite and save the file. Upon logging in again your R conda environment will automatically be active.

If you would like to deactivate your conda environment at any time, use the following command:

```
(r4_MyEnv) [username@pegasus ~]$ conda deactivate r4_MyEnv
```

To obtain a list of your conda environments, use the following command:

```
[username@pegasus ~]$ conda env list
```

Running jobs

In order to properly run a job using R within a conda environment you will need to intiate & activate the conda environment within the job script, otherwise the job may fail to find your version of R. Please see the example job script below:

```
#!/bin/bash
#BSUB -J jobName
#BSUB -P projectName
#BSUB -o jobName.%J.out
#BSUB -e jobName.%J.err
#BSUB -W 1:00
#BSUB -q normal
#BSUB -n 1
```

(continues on next page)

(continued from previous page)

```
#BSUB -u youremail@miami.edu

. "/home/caneid/anaconda3/etc/profile.d/conda.sh"
conda activate r4_MyEnv

cd /path/to/your/R_file.R

R CMD BATCH R_file.R
```

Note: Sometimes you may need to use the 'Rscript' command instead of 'R CMD BATCH' to run your R file within the job script.

4.1.3 Triton Job Scheduling

Job Scheduling with LSF

Triton currently uses the **LSF** resource manager to schedule all compute resources. LSF (*load sharing facility*) supports over 1500 users and over 200,000 simultaneous job submissions. Jobs are submitted to **queues**, the software categories we define in the scheduler to organize work more efficiently. LSF distributes jobs submitted by users to our over 340 compute nodes according to queue, user priority, and available resources. You can monitor your job status, queue position, and progress using *LSF commands*.

Tip: Reserve an appropriate amount of resources through LSF for your jobs.

If you do not know the resources your jobs need, use the **debug** queue to benchmark your jobs. More on *Pegasus Queues* and *LSF Job Scripts*

Warning: Jobs with insufficient resource allocations interfere with cluster performance and the IDSC account responsible for those jobs may be suspended (Policies).

Tip: Stage data for running jobs exclusively in the `/scratch` file system, which is optimized for fast data access.

Any files used as input for your jobs must first be transferred to /scratch. See *Pegasus Resource Allocations* for more information. The /nethome file system is optimized for mass data storage and is therefore slower-access.

Warning: Using /nethome while running jobs degrades the performance of the entire system and the IDSC account responsible may be suspended*** (Policies).

Tip: Do not background processes with the `&` operator in LSF.

These spawned processes cannot be killed with **bkill** after the parent is gone.

Warning: Using the & operator while running jobs degrades the performance of the entire system and the IDSC account responsible may be suspended (Policies).

LSF Batch Jobs

Batch jobs are self-contained programs that require no intervention to run. Batch jobs are defined by resource requirements such as how many cores, how much memory, and how much time they need to complete. These requirements can be submitted via command line flags or a script file. Detailed information about LSF commands and example script files can be found later in this guide.

1. **Create a job scriptfile**

   Include a job name `-J`, the information LSF needs to allocate resources to your job, and names for your output and error files.

```
scriptfile
#BSUB -J test
#BSUB -q normal
#BSUB -P myproject
#BSUB -o %J.out
...
```

2. **Submit your job to the appropriate project and queue with** `bsub < scriptfile`

   Upon submission, a `jobID` and the queue name are returned.

```
[username@triton ~]$ bsub < scriptfile
Job <6021006> is submitted to queue <normal>.
```

3. **Monitor your jobs with** `bjobs`

   Flags can be used to specify a single job or another user's jobs.

```
[username@triton ~]$ bjobs
JOBID   USER    STAT  QUEUE     FROM_HOST   EXEC_HOST    JOB_NAME   SUBMIT_TIME
4225    usernam    RUN   normal  m1        16*n060      testjob    Mar  2 11:53
```

4. **Examine job output files**

   Once your job has completed, view output information.

```
[username@triton ~]$ cat test.out
Sender: LSF System <lsfadmin@n069.triton.edu>
Subject: Job 6021006: <test> in cluster <triton> Done
Job <test> was submitted from host <login4.triton.edu> by user <username> in␣
↪cluster <mk2>.
Job was executed on host(s) <8*n069>, in queue <normal>, as user <username> in␣
↪cluster <mk2>.
...
```

Triton Job Queues

Triton queues are organized using limits like job size, job length, job purpose, and project. In general, users run jobs on Pegasus with equal resource shares. Current or recent resource usage lowers the priority applied when LSF assigns resources for new jobs from a user's account.

The **bigmem** queue is available for jobs requiring nodes with expanded memory. Submitting jobs to this queue requires project membership. Do not submit jobs that can run on the general and parallel queues to the bigmem queue.

Table 2: Triton Job Queues

Queue Name	Processors (Cores)	Memory	Wall time default / max	Description
normal	512	256GB max	1 day / 7 days	Parallel and serial jobs up to 256GB memory per host
bigmem	40	250GB max hosts	4 hours / 5 days	Jobs requiring nodes with expanded memory up to 1TB
short	64	25GB max	30 mins / 30 mins	Jobs less than 1 hour wall time. Scheduled with higher priority.
interactive	40	250GB max	6 hours / 1 day	Interactive jobs only max 1 job per user

Pegasus LSF Commands

LSF 9.1.1 Documentation

Common LSF commands and descriptions:

Command

Purpose

bsub

Submits a job to LSF. Define resource requirements with flags.

bsub < scriptfile

Submits a job to LSF via script file. The redirection symbol < is required when submitting a job script file

bjobs

Displays your running and pending jobs.

bhist

Displays historical information about your finished jobs.

bkill

Removes/cancels a job or jobs from the class.

bqueues

Shows the current configuration of queues.

bhosts

Shows the load on each node.

bpeek

Displays stderr and stdout from your unfinished job.

Scheduling Jobs

The command `bsub` will submit a job for processing. You must include the information LSF needs to allocate the resources your job requires, handle standard I/O streams, and run the job. For more information about flags, type `bsub -h` at the Pegasus prompt. Detailed information can be displayed with `man bsub`. On submission, LSF will return the job id which can be used to keep track of your job.

```
[username@triton ~]$ bsub -J jobname -o %J.out -e %J.err -q normal -P myproject␣
↪myprogram
Job <2607> is submitted to general queue .
```

The Job Scripts section has more information about organizing multiple flags into a job script file for submission.

Monitoring Jobs

bjobs

The commands `bjobs` displays information about your own pending, running, and suspended jobs.

```
[username@triton ~]$ bjobs
JOBID  USER    STAT  QUEUE     FROM_HOST  EXEC_HOST   JOB_NAME  SUBMIT_TIME
4225   usernam   RUN   normal  m1         16*n060     testjob   Mar  2 11:53
                                           16*n061
                                           16*n063
                                           16*n064
```

For details about your particular job, issue the command `bjobs -l jobID` where `jobID` is obtained from the `JOBID` field of the above `bjobs` output. To display a specific user's jobs, use `bjobs -u username`. To display all user jobs in paging format, pipe output to `less`:

```
[username@triton ~]$ bjobs -u all | less
JOBID     USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
5990529   axt651  RUN   interactiv login4.pega n002        bash       Feb 13 15:23
6010636   zxh69   RUN   normal    login4.pega 16*n178     *acsjob-01 Feb 23 11:36
                                               16*n180
                                               16*n203
                                               16*n174
6014246   swishne RUN   interactiv n002.pegasu n002        bash       Feb 24 14:10
6017561   asingh  PEND  interactiv login4.pega             matlab     Feb 25 14:49
...
```

bhist

`bhist` displays information about your recently finished jobs. CPU time is not normalized in `bhist` output. To see your *finished* and *unfinished* jobs, use `bhist -a`.

bkill

`bkill` kills the last job submitted by the user running the command, by default. The command `bkill jobID` will remove a specific job from the queue and terminate the job **if** it is running. `bkill 0` will kill all jobs belonging to current user.

```
[username@triton ~]$ bkill 4225
Job <4225> is being terminated
```

On Pegasus (Unix), SIGINT and SIGTERM are sent to give the job a chance to clean up before termination, then SIGKILL is sent to kill the job.

bqueues

`bqueues` displays information about queues such as queue name, queue priority, queue status, job slot statistics, and job state statistics. CPU time is normalized by CPU factor.

```
[username@triton ~]$ bqueues
QUEUE_NAME      PRIO STATUS          MAX JL/U JL/P JL/H NJOBS  PEND   RUN  SUSP
bigmem           500  Open:Active       -   16    -    -  1152  1120    32     0
normal           100  Open:Active       -    -    -    -  9677  5969  3437     0
interactive       30  Open:Active       -    4    -    -    13     1    12     0
```

bhosts

`bhosts` displays information about all hosts such as host name, host status, job state statistics, and jobs lot limits. `bhosts -s` displays information about numeric resources (shared or host-based) and their associated hosts. `bhosts hostname` displays information about an individual host and `bhosts -w` displays more detailed host status. closed_Full means the configured maximum number of running jobs has been reached (running jobs will not be affected), no new job will be assigned to this host.

```
[username@triton ~]$ bhosts -w | less
HOST_NAME          STATUS       JL/U    MAX  NJOBS    RUN  SSUSP  USUSP    RSV
n001               ok              -     16     14     14      0      0      0
n002               ok              -     16      4      4      0      0      0
...
n342               closed_Full     -     16     16     12      0      0      4
n343               closed_Full     -     16     16     16      0      0      0
n344               closed_Full     -     16     16     16      0      0      0
```

bpeek

Use `bpeek jobID` to monitor the progress of a job and identify errors. If errors are observed, valuable user time and system resources can be saved by terminating an erroneous job with `bkill jobID`. By default, `bpeek` displays the standard output and standard error produced by one of your unfinished jobs, up to the time the command is invoked. `bpeek -q queuename` operates on your most recently submitted job in that queue and `bpeek -m hostname` operates on your most recently submitted job dispatched to the specified host. `bpeek -f jobID` display live outputs from a running job and it can be terminated by `Ctrl-C` (Windows & most Linux) or `Command-C` (Mac).

Examining Job Output

Once your job has completed, examine the contents of your job's output files. Note the script submission under **User input**, whether the job completed, and the **Resource usage summary**.

```
[username@triton ~]$ cat test.out
Sender: LSF System <lsfadmin@n069.triton.edu>
Subject: Job 6021006: <test> in cluster <mk2> Done
Job <test> was submitted from host <login4.triton.edu> by user <username> in cluster
↪<mk2>.
Job was executed on host(s) <8*n069>, in queue <general>, as user <username> in␣
↪cluster <mk2>.
...
Your job looked like:
```

(continues on next page)

(continued from previous page)

```
------------------------------------------------------------
# LSBATCH: User input
#!/bin/sh
#BSUB -n 16
#BSUB -J test
#BSUB -o test.out
...
------------------------------------------------------------
Successfully completed.
Resource usage summary:
CPU time : 2.26 sec.
Max Memory : 30 MB
Average Memory : 30.00 MB
...
PS:
Read file <test.err> for stderr output of this job.
```

Triton LSF Job Scripts

The command `bsub < ScriptFile` will submit the given script for processing. Your script must contain the information LSF needs to allocate the resources your job requires, handle standard I/O streams, and run the job. For more information about flags, type `bsub -h` or `man bsub` at the Triton prompt. Example scripts and descriptions are below.

You must be a member of a project to submit jobs to it. See *Projects* for more information.

On submission, LSF will return the `jobID` which can be used to track your job.

```
[username@triton ~]$ bsub < test.job
Job <4225> is submitted to the default queue <normal>.
```

Example script for a serial Job

`test.job`

```
#!/bin/bash
#BSUB -J myserialjob
#BSUB -P myproject
#BSUB -o %J.out
#BSUB -e %J.err
#BSUB -W 1:00
#BSUB -q normal
#BSUB -n 1
#BSUB -R "rusage[mem=128M]"
#BSUB -B
#BSUB -N
#BSUB -u myemail@miami.edu
#
# Run serial executable on 1 cpu of one node
cd /path/to/scratch/directory
./test.x a b c
```

Here is a detailed line-by-line breakdown of the keywords and their assigned values listed in this script:

ScriptFile_keywords

```
#!/bin/bash
specifies the shell to be used when executing the command portion of the script.
The default is Bash shell.

BSUB -J serialjob
assign a name to job. The name of the job will show in the bjobs output.

#BSUB -P myproject
specify the project to use when submitting the job. This is required when a user has␣
↪more than one project on Triton.

#BSUB -e %J.err
redirect std error to a specified file

#BSUB -W 1:00
set wall clock run time limit of 1 hour, otherwise queue specific default run time␣
↪limit will be applied.

#BSUB -q normal
specify queue to be used. Without this option, default 'normal' queue will be applied.

#BSUB -n 1
specify number of processors. In this job, a single processor is requested.

#BSUB -R "rusage[mem=128M]"
specify that this job requests 128 megabytes of RAM. You can use other units␣
↪(K(kilobytes), M(megabytes), G(gigabytes), T(terabytes)).

#BSUB -B
send mail to specified email when the job is dispatched and begins execution.

#BSUB -u example@miami.edu
send notification through email to example@miami.edu.

#BSUB -N
send job statistics report through email when job finishes.
```

Example scripts for parallel jobs

We recommend using IBM Advance Toolchain and SMPI unless you have specific reason for using OpenMP or OpenMPI. IBM's SMPI scales better and has better performance than both OpenMP or OpenMPI on Triton.

For optimum performance, use the `#BSUB -R "span[ptile=40]"`. This requires the LSF job scheduler to allocate 40 processors per host, ensuring all processors on a single host are used by that job.

Reserve enough memory for your jobs. Memory reservations are per core. Parallel job performance may be affected, or even interrupted, by other badly-configured jobs running on the same host.

mpi_hello_world.job

```
$ cat mpi_hello_world.job
#!/bin/sh
#BSUB -n 80
#BSUB -J mpi_hello_world
#BSUB -o logs/%J.out
#BSUB -e logs/%J.err
#BSUB -a openmpi
#BSUB -R "span[ptile=4]"
#BSUB -q normal

# Use gcc/8.3.1 and openmpi/4.0.5
# ml gcc/8.3.1 openmpi/4.0.5

# Use the optimized IBM Advance Toolkit (gcc 8.3.1) and smpi
ml at smpi

echo "Testing the gcc/8.3.1 openmpi/4.0.5 module: \n" >> mpi_hello_world.log

mpirun -n 80 ./mpi_hello_world >> logs/mpi_hello_world.log
```

mpi_hello_world.c

```
$ cat mpi_hello_world.c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
  // Initialize the MPI environment
  MPI_Init(NULL, NULL);

  // Get the number of processes
  int world_size;
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);

  // Get the rank of the process
  int world_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

  // Get the name of the processor
  char processor_name[MPI_MAX_PROCESSOR_NAME];
  int name_len;
  MPI_Get_processor_name(processor_name, &name_len);

  // Print off a hello world message
  printf("Hello world from processor %s, rank %d out of %d processors\n",
         processor_name, world_rank, world_size);

  // Finalize the MPI environment.
  MPI_Finalize();
}
```

Compile mpi_hello_world.c

```
$ ml gcc/8.3.1
$ ml openmpi/4.0.5
$ mpicc -o mpi_hello_world mpi_hello_world.c
```

Run mpi_hello_world.job

```
$ bsub < mpi_hello_world.job
Job <981431> is submitted to queue <normal>.
```

Get mpi_hello_world.job status

```
$ bjobs -l 981431

Job <981431>, Job Name <openmpi_test>, User <pdavila>, Project <default>, Status
↪<DONE>
...

Thu Oct  7 11:25:07: Done successfully. The CPU time used is 9.7 seconds.
                     HOST: t088; CPU_TIME: 0 seconds
                     HOST: t061; CPU_TIME: 0 seconds
                     HOST: t042; CPU_TIME: 0 seconds
                     HOST: t052; CPU_TIME: 0 seconds
                     HOST: t029; CPU_TIME: 0 seconds
                     HOST: t077; CPU_TIME: 1 seconds
                     HOST: t072; CPU_TIME: 0 seconds
                     HOST: t058; CPU_TIME: 0 seconds
                     HOST: t039; CPU_TIME: 0 seconds
                     HOST: t041; CPU_TIME: 0 seconds
                     HOST: t065; CPU_TIME: 0 seconds
                     HOST: t036; CPU_TIME: 1 seconds
                     HOST: t087; CPU_TIME: 0 seconds
                     HOST: t048; CPU_TIME: 0 seconds
                     HOST: t081; CPU_TIME: 0 seconds
                     HOST: t054; CPU_TIME: 0 seconds
                     HOST: t073; CPU_TIME: 1 seconds
                     HOST: t070; CPU_TIME: 0 seconds
                     HOST: t083; CPU_TIME: 1 seconds
                     HOST: t047.triton; CPU_TIME: 0 seconds

 MEMORY USAGE:
 MAX MEM: 14 Mbytes;  AVG MEM: 9 Mbytes
 ...
```

```
$ cat logs/981431.out
Sender: LSF System <hpc@ccs.miami.edu>
Subject: Job 981431: <openmpi_test> in cluster <triton> Done

Job <openmpi_test> was submitted from host <login1> by user <pdavila> in cluster
↪<triton> at Thu Oct  7 11:25:03 2021
Job was executed on host(s) <4*t047>, in queue <normal>, as user <pdavila> in cluster
↪<triton> at Thu Oct  7 11:25:03 2021
                            <4*t083>
```

(continues on next page)

(continued from previous page)

```
                            <4*t087>
                            <4*t036>
                            <4*t065>
                            <4*t081>
                            <4*t054>
                            <4*t061>
                            <4*t029>
                            <4*t039>
                            <4*t088>
                            <4*t052>
                            <4*t070>
                            <4*t072>
                            <4*t073>
                            <4*t042>
                            <4*t058>
                            <4*t077>
                            <4*t048>
                            <4*t041>
...

Your job looked like:

------------------------------------------------------------
# LSBATCH: User input
#!/bin/sh
#BSUB -n 80
#BSUB -J openmpi_test
#BSUB -o logs/%J.out
#BSUB -e logs/%J.err
#BSUB -a openmpi
#BSUB -R "span[ptile=4]"
#BSUB -q normal

# Use openmpi
# ml gcc/8.3.1 openmpi/4.0.5

# Use the optimized IBM Advance Toolkit (gcc 8.3.1) and smpi
ml at smpi

echo "Testing the gcc/8.3.1 openmpi/4.0.5 module: \n" >> logs/mpi_hello_world.log

mpirun -n 80 ./mpi_hello_world >> logs/mpi_hello_world.log

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

Successfully completed.

Resource usage summary:

  CPU time :                                   9.71 sec.
  Max Memory :                                 17 MB
  Average Memory :                             11.00 MB
  Total Requested Memory :                     -
  Delta Memory :                               -
  Max Swap :                                   -
  Max Processes :                              5
  Max Threads :                                9
```

(continues on next page)

(continued from previous page)

```
  Run time :                                    4 sec.
  Turnaround time :                             5 sec.

The output (if any) follows:

PS:

Read file <logs/981431.err> for stderr output of this job.
```

Triton Interactive Jobs

HPC clusters primarily take batch jobs and run them in the *background*—users do not need to interact with the job during the execution. However, sometimes users do need to interact with the application. For example, the application needs the input from the command line or waits for a mouse event in X windows. Use `bsub -Is -q interactive command` to launch interactive work on Triton.

```
[username@triton ~]$ bsub -Is -q interactive bash
```

or

```
[username@triton ~]$ bsub -Is -q interactive -XF $(java -jar ~/.local/apps/ImageJ/ij.
↪jar -batch ~/.local/apps/ImageJ/macros/screenmill.txt)
```

Upon exiting the interactive job, you will be returned to one of the login nodes.

Interactive Job Utilizing X11 client

Additionally, the interactive queue can run X11 jobs. The bsub `-XF` option is used for X11 jobs, for example:

```
[username@triton ~]$ bsub -Is -q interactive -XF
Job <50274> is submitted to queue <interactive>.
<<ssh X11 forwarding job>>
<<Waiting for dispatch ...>>
<<Starting on n003.triton.edu>>
```

Upon exiting the X11 interactive job, you will be returned to one of the login nodes.

To run an X11 application, establish an X tunnel with SSH when connecting to Triton. For example,

```
ssh -X username@triton.ccs.miami.edu
```

Note that by default, the auth token is good for 20 minutes. SSH will block new X11 connections after 20 minutes. To avoid this on Linux or OS X, run `ssh -Y` instead, or set the option **ForwardX11Trusted yes** in your ~/.ssh/config.

In Windows, use Cygwin/X to provide a Linux-like environment. Then run `ssh -Y` or set the option in your ~/.ssh/config file.

4.2 Pegasus user guides

The Pegasus cluster has been upgraded to CentOS 7.

If you encounter issues running your jobs, let our IDSC cluster support team know via email to IDSC team (hpc@ccs.miami.edu) .

4.2.1 Pegasus Environment

Pegasus Environment Introduction

The Pegasus cluster is the University of Miami's 350-node high-performance supercomputer, available to all University of Miami employees and students. Pegasus resources such as hardware (login and compute nodes) and system software are shared by all users.

Tip: **Before** running commands, submitting jobs, or using software on the Pegasus supercomputer, understand our core *Policies*.

```
Details:              Pegasus Supercomputer
Credentials:          IDSC Account
Access & Allocations: Policies
Operating System:     CentOS 7.6
Default Shell:        Bash
Data Transfer:        SCP and SFTP
```

We encourage new users to carefully read our documentation on Pegasus and available resources, especially users who may be unfamiliar with high-performance computing, Unix-based systems, or batch job scheduling. Understanding what your jobs do on the cluster helps keep Pegasus running smoothly for everyone.

- **Do not run resource-intensive jobs on the Pegasus login nodes.** Submit your production jobs to LSF, and use the *interactive queue* and *LSF Job Scripts* below. Jobs with insufficient resource allocations interfere with cluster performance and the IDSC account responsible for those jobs may be suspended.
- **Stage data for running jobs exclusively in the** `/scratch` **file system,** which is optimized for fast data access. Any files used as input for your jobs must first be transferred to /scratch. The /nethome file system is optimized for mass data storage and is therefore slower-access. Using /nethome while running jobs degrades the performance of the entire system and the IDSC account responsible may be suspended.
- **Include your projectID in your job submissions.** Access to IDSC Advanced Computing resources is managed on a project basis. This allows us to better support interaction between teams (including data sharing) at the University of Miami regardless of group, school, or campus. Any University of Miami faculty member or Principal Investigator (PI) can request a new project. All members of a project share that project's resource allocations. More on *Projects here*.

Connecting to Pegasus: To access the Pegasus supercomputer, open a secure shell (SSH) connection to `pegasus.ccs.miami.edu` and log in with your active IDSC account. Once authenticated, you should see the Pegasus welcome message – ***which includes links to Pegasus documentation*** and information about your disk quotas – then the Pegasus command prompt.

```
--------------------------------------------------------------------------------
                    Welcome to the Pegasus Supercomputer
           Center for Computational Science, University of Miami
--------------------------------------------------------------------------------
...
...
...
--------------------------Disk Quota--------------------------------
filesystem | project    | used(GB)    | quota (GB) | Util(%)
===========================================================
nethome    | user       | 0.76        | 250.00     |    0%
scratch    | projectID  | 93.32       | 20000.00   |    0%
------------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
     Files on /scratch are subject to purging after 21 days
-----------------------------------------------------------------
[username@pegasus ~]$
```

Pegasus Filesystems

The landing location on Pegasus is your **home** directory, which corresponds to `/nethome/username`. As shown in the Welcome message, Pegasus has two parallel file systems available to users: **nethome** and **scratch**.

Table 3: Pegasus Filesystems

Filesystem	Description	Notes
`/nethome`	permanent, quota'd, not backed-up	directories are limited to 250GB and intended primarily for basic account information, source codes and binaries
`/scratch`	high-speed storage	directories should be used for compiles and run-time input & output files

Warning: **Do not** stage job data in the `/nethome` file system. If your jobs read files from Pegasus, put those files exclusively in the `/scratch` file system.

Pegasus Environment Links

Resource allocations : Cluster resources, including CPU hours and scratch space, are allocated to projects. To access resources, all IDSC accounts must belong to a project with active resource allocations. Join projects by contacting Principal Investigators (PIs) directly.

Transferring files : Whether on **nethome** or **scratch**, transfer data with secure copy (SCP) and secure FTP (SFTP) between Pegasus file systems and local machines. Use Pegasus login nodes for these types of transfers. See the link for more information about transferring large amounts of data from systems outside the University of Miami.

Software on Pegasus : To use system software on Pegasus, first load the software using the **module load** command. Some modules are loaded automatically when you log into Pegasus. The modules utility handles any paths or libraries needed for the software to run. You can view currently loaded modules with `module list` and check available software with `module avail package`.

Warning: **Do not** run production jobs on the login nodes.

Once your preferred software module is loaded, submit a job to the Pegasus job scheduler to use it.

Pegasus Job Submissions

Job submissions : Pegasus cluster compute nodes are the workhorses of the supercomputer, with significantly more resources than the login nodes. Compute nodes are grouped into **queues** and their available resources are assigned through scheduling software (LSF). To do work on Pegasus, submit either a **batch** or an **interactive** job to LSF for an appropriate queue.

In shared-resource systems like Pegasus, you must tell the LSF scheduler how much memory, CPU, time, and other resources your jobs will use while they are running. If your jobs use more resources than you requested from LSF, those resources may come from other users' jobs (and vice versa). This not only negatively impacts everyone's jobs, it degrades the performance of the entire cluster. If you do not know the resources your jobs will use, benchmark them in the **debug** queue.

To test code interactively or install extra software modules at a prompt (such as with Python or R), submit an interactive job to the interactive queue in LSF. This will navigate you to a compute node for your work, and you will be returned to a login node upon exiting the job. Use the interactive queue for resource-intensive command-line jobs such as sort, find, awk, sed, and others.

Connecting to Pegasus

DNS : pegasus.ccs.miami.edu
Access : *SSH* over secure UM networks, *x11*
Credentials : IDSC Account

Pegasus Welcome Message

The Pegasus welcome message includes links to Pegasus documentation and information about your disk quotas.

```
--------------------------------------------------------------------------------
                    Welcome to the Pegasus Supercomputer
          Center for Computational Science, University of Miami
--------------------------------------------------------------------------------
...
...
...
-------------------------Disk Quota-------------------------------
filesystem | project    | used(GB)    | quota (GB) | Util(%)
============================================================
nethome    | user       | 0.76        | 250.00     |    0%
scratch    | projectID  | 93.32       | 20000.00   |    0%
-----------------------------------------------------------------
     Files on /scratch are subject to purging after 21 days
-----------------------------------------------------------------
[username@pegasus ~]$
```

Transferring files

Transferring files to IDSC systems

Pegasus Projects & Resources

Access to IDSC Advanced Computing resources is managed on a **project** basis. This allows us to better support interaction between teams (including data sharing) at the University of Miami regardless of group, school, or campus. Project-based resource allocation also gives researchers the ability to request resources for short-term work. Any University of Miami faculty member or Principal Investigator (PI) can request a new project. All members of a project share that project's resource allocations.

To join a project, contact the project owner. PIs and faculty, request new IDSC Projects here (https://idsc.miami.edu/project_request)

Using projects in computing jobs

To run jobs using your project's resources, submit jobs with your assigned `projectID` using the `-P` argument to `bsub`: `bsub -P projectID`. For more information about LSF and job scheduling, see *Scheduling Jobs on Pegasus*.

For example, if you were assigned the project id "abc", a batch submission from the command line would look like:

```
$ bsub -P abc < JOB_SCRIPT_NAME
```

and an interactive submission from the command line would look like:

```
$ bsub -P abc -Is -q interactive -XF command
```

When your job has been submitted successfully, the project and queue information will be printed on the screen.

```
Job is submitted to <abc> project.

Job <11234> is submitted to default queue <general>.
```

The cluster scheduler will only accept job submissions to active projects. The IDSC user must be a current member of that project.

4.2.2 Pegasus Job Scheduling

Pegasus Job Scheduling with LSF

Pegasus currently uses the **LSF** resource manager to schedule all compute resources. LSF (*load sharing facility*) supports over 1500 users and over 200,000 simultaneous job submissions. Jobs are submitted to **queues**, the software categories we define in the scheduler to organize work more efficiently. LSF distributes jobs submitted by users to our over 340 compute nodes according to queue, user priority, and available resources. You can monitor your job status, queue position, and progress using *LSF commands*.

Tip: Reserve an appropriate amount of resources through LSF for your jobs.

If you do not know the resources your jobs need, use the **debug** queue to benchmark your jobs. More on *Pegasus Queues* and *LSF Job Scripts*

> **Warning:** Jobs with insufficient resource allocations interfere with cluster performance and the IDSC account responsible for those jobs may be suspended (*Policies*).

Tip: Stage data for running jobs exclusively in the `/scratch` file system, which is optimized for fast data access.

Any files used as input for your jobs must first be transferred to /scratch. See *Pegasus Resource Allocations* for more information. The /nethome file system is optimized for mass data storage and is therefore slower-access.

> **Warning:** Using /nethome while running jobs degrades the performance of the entire system and the IDSC account responsible may be suspended*** (*Policies*).

Tip: Do not background processes with the `&` operator in LSF.

These spawned processes cannot be killed with **bkill** after the parent is gone.

Warning: Using the & operator while running jobs degrades the performance of the entire system and the IDSC account responsible may be suspended (*Policies*).

LSF Batch Jobs

Batch jobs are self-contained programs that require no intervention to run. Batch jobs are defined by resource requirements such as how many cores, how much memory, and how much time they need to complete. These requirements can be submitted via command line flags or a script file. Detailed information about LSF commands and example script files can be found later in this guide.

1. **Create a job scriptfile**

   Include your project ID `-P`, a job name `-J`, the information LSF needs to allocate resources to your job, and names for your output and error files.

```
scriptfile
#BSUB -J test
#BSUB -q general
#BSUB -P myproject
#BSUB -o %J.out
...
```

2. **Submit your job to the appropriate project and queue with** `bsub < scriptfile`

   Upon submission, the project is returned along with a `jobID` and the queue name.

```
[username@pegasus ~]$ bsub < scriptfile
Job is submitted to <my_project> project.
Job <6021006> is submitted to queue <general>.
```

3. **Monitor your jobs with** `bjobs`

   Flags can be used to specify a single job or another user's jobs.

```
[username@pegasus ~]$ bjobs
JOBID  USER   STAT  QUEUE     FROM_HOST  EXEC_HOST   JOB_NAME  SUBMIT_TIME
4225   usernam   RUN   general  m1       16*n060     testjob   Mar  2 11:53
```

4. **Examine job output files**

   Once your job has completed, view output information.

```
[username@pegasus ~]$ cat test.out
Sender: LSF System <lsfadmin@n069.pegasus.edu>
Subject: Job 6021006: <test> in cluster <mk2> Done
Job <test> was submitted from host <login4.pegasus.edu> by user <username> in␣
↪cluster <mk2>.
Job was executed on host(s) <8*n069>, in queue <general>, as user <username> in␣
↪cluster <mk2>.
...
```

Pegasus Job Queues

Pegasus queues are organized using limits like job size, job length, job purpose, and project. In general, users run jobs on Pegasus with equal resource shares. Current or recent resource usage lowers the priority applied when LSF assigns resources for new jobs from a user's account.

Parallel jobs are more difficult to schedule as they are inherently larger. Serial jobs can "fit into" the gaps left by larger jobs if serial jobs use short enough run time limits and small enough numbers of processors.

The **parallel** queue is available for jobs requiring 16 or more cores. Submitting jobs to this queue ***requires*** resource distribution `-R "span[ptile=16]"`.

The **bigmem** queue is available for jobs requiring nodes with expanded memory. Submitting jobs to this queue requires project membership. Do not submit jobs that can run on the general and parallel queues to the bigmem queue.

Warning: Jobs using less than 1.5G of memory per core on the bigmem queue are in violation of acceptable use policies and the IDSC account responsible for those jobs may be suspended (*Policies*).

Table 4: Pegasus Job Queues

Queue Name	Processors (Cores)	Memory	Wall time default / max	Description
general	15-	24GB max	1 day / 7 days	jobs up to 15 cores, up to 24GB memory
parallel	16+	24GB max	1 day / 7 days	parallel jobs requiring 16 or more cores, up to 24GB memory. **requires resource distribution -R "span[ptile=16]"**
bigmem	64 max	250GB max	4 hours / 5 days	jobs requiring nodes with expanded memory
debug	64 max	24GB max	30 mins / 30 mins	job debugging
interactive	15-	250GB max	6 hours / 1 day	interactive jobs max 1 job per user
gpu	xx	320 max	1 day / 7 days	gpu debugging restricted queue
phi	xx	320 max	1 day / 7 days	phi debugging restricted queue

Pegasus LSF Commands

LSF 9.1.1 Documentation

Common LSF commands and descriptions:

Command

Purpose

bsub

Submits a job to LSF. Define resource requirements with flags.

bsub < scriptfile

Submits a job to LSF via script file. The redirection symbol < is required when submitting a job script file

bjobs

Displays your running and pending jobs.

bhist

Displays historical information about your finished jobs.

bkill

Removes/cancels a job or jobs from the class.

bqueues

Shows the current configuration of queues.

bhosts

Shows the load on each node.

bpeek

Displays stderr and stdout from your unfinished job.

Scheduling Jobs

The command `bsub` will submit a job for processing. You must include the information LSF needs to allocate the resources your job requires, handle standard I/O streams, and run the job. For more information about flags, type `bsub -h` at the Pegasus prompt. Detailed information can be displayed with `man bsub`. On submission, LSF will return the job id which can be used to keep track of your job.

```
[username@pegasus ~]$ bsub -J jobname -o %J.out -e %J.err -q general -P myproject␣
↪myprogram
Job <2607> is submitted to general queue .
```

The Job Scripts section has more information about organizing multiple flags into a job script file for submission.

Monitoring Jobs

bjobs

The commands `bjobs` displays information about your own pending, running, and suspended jobs.

```
[username@pegasus ~]$ bjobs
JOBID  USER   STAT  QUEUE    FROM_HOST  EXEC_HOST   JOB_NAME  SUBMIT_TIME
4225   usernam   RUN   general  m1       16*n060     testjob   Mar  2 11:53
                                         16*n061
                                         16*n063
                                         16*n064
```

For details about your particular job, issue the command `bjobs -l jobID` where `jobID` is obtained from the `JOBID` field of the above `bjobs` output. To display a specific user's jobs, use `bjobs -u username`. To display all user jobs in paging format, pipe output to `less`:

```
[username@pegasus ~]$ bjobs -u all | less
JOBID     USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
5990529   axt651  RUN   interactiv login4.pega n002        bash       Feb 13 15:23
6010636   zxh69   RUN   general    login4.pega 16*n178     *acsjob-01 Feb 23 11:36
                                                16*n180
                                                16*n203
```

(continues on next page)

(continued from previous page)

```
                                                16*n174
6014246   swishne RUN   interactiv n002.pegasu n002        bash       Feb 24 14:10
6017561   asingh  PEND  interactiv login4.pega             matlab     Feb 25 14:49
...
```

bhist

`bhist` displays information about your recently finished jobs. CPU time is not normalized in `bhist` output. To see your *finished* and *unfinished* jobs, use `bhist -a`.

bkill

`bkill` kills the last job submitted by the user running the command, by default. The command `bkill jobID` will remove a specific job from the queue and terminate the job **if** it is running. `bkill 0` will kill all jobs belonging to current user.

```
[username@pegasus ~]$ bkill 4225
Job <4225> is being terminated
```

On Pegasus (Unix), SIGINT and SIGTERM are sent to give the job a chance to clean up before termination, then SIGKILL is sent to kill the job.

bqueues

`bqueues` displays information about queues such as queue name, queue priority, queue status, job slot statistics, and job state statistics. CPU time is normalized by CPU factor.

```
[username@pegasus ~]$ bqueues
QUEUE_NAME      PRIO STATUS          MAX JL/U JL/P JL/H NJOBS  PEND   RUN  SUSP
bigmem           500  Open:Active       -   16    -    -  1152  1120    32     0
visx             500  Open:Active       -    -    -    -     0     0     0     0
hihg             500  Open:Active       -    -    -    -     0     0     0     0
hpc              300  Open:Active       -    -    -    -  2561  1415  1024     0
debug            200  Open:Active       -    -    -    -     0     0     0     0
gpu              200  Open:Active       -    -    -    -     0     0     0     0
...
general          100  Open:Active       -    -    -    -  9677  5969  3437     0
interactive       30  Open:Active       -    4    -    -    13     1    12     0
```

bhosts

`bhosts` displays information about all hosts such as host name, host status, job state statistics, and jobs lot limits. `bhosts -s` displays information about numeric resources (shared or host-based) and their associated hosts. `bhosts hostname` displays information about an individual host and `bhosts -w` displays more detailed host status. closed_Full means the configured maximum number of running jobs has been reached (running jobs will not be affected), no new job will be assigned to this host.

```
[username@pegasus ~]$ bhosts -w | less
HOST_NAME          STATUS       JL/U    MAX  NJOBS    RUN  SSUSP  USUSP    RSV
n001               ok              -     16     14     14      0      0      0
n002               ok              -     16      4      4      0      0      0
...
n342               closed_Full     -     16     16     12      0      0      4
n343               closed_Full     -     16     16     16      0      0      0
n344               closed_Full     -     16     16     16      0      0      0
```

bpeek

Use `bpeek jobID` to monitor the progress of a job and identify errors. If errors are observed, valuable user time and system resources can be saved by terminating an erroneous job with `bkill jobID`. By default, `bpeek` displays the standard output and standard error produced by one of your unfinished jobs, up to the time the command is invoked. `bpeek -q queuename` operates on your most recently submitted job in that queue and `bpeek -m hostname` operates on your most recently submitted job dispatched to the specified host. `bpeek -f jobID` display live outputs from a running job and it can be terminated by `Ctrl-C` (Windows & most Linux) or `Command-C` (Mac).

Examining Job Output

Once your job has completed, examine the contents of your job's output files. Note the script submission under **User input**, whether the job completed, and the **Resource usage summary**.

```
[username@pegasus ~]$ cat test.out
Sender: LSF System <lsfadmin@n069.pegasus.edu>
Subject: Job 6021006: <test> in cluster <mk2> Done
Job <test> was submitted from host <login4.pegasus.edu> by user <username> in cluster
↪<mk2>.
Job was executed on host(s) <8*n069>, in queue <general>, as user <username> in␣
↪cluster <mk2>.
...
Your job looked like:
------------------------------------------------------------
# LSBATCH: User input
#!/bin/sh
#BSUB -n 16
#BSUB -J test
#BSUB -o test.out
...
------------------------------------------------------------
Successfully completed.
Resource usage summary:
CPU time : 2.26 sec.
Max Memory : 30 MB
Average Memory : 30.00 MB
...
PS:
Read file <test.err> for stderr output of this job.
```

Pegasus LSF Job Scripts

The command `bsub < ScriptFile` will submit the given script for processing. Your script must contain the information LSF needs to allocate the resources your job requires, handle standard I/O streams, and run the job. For

more information about flags, type `bsub -h` or `man bsub` at the Pegasus prompt. Example scripts and descriptions are below.

You must be a member of a project to submit jobs to it. See *Projects* for more information.

On submission, LSF will return the `jobID` which can be used to track your job.

```
[username@pegasus ~]$ bsub < test.job
Job <4225> is submitted to the default queue <general>.
```

Example script for a serial Job

`test.job`

```
#!/bin/bash
#BSUB -J myserialjob
#BSUB -P myproject
#BSUB -o %J.out
#BSUB -e %J.err
#BSUB -W 1:00
#BSUB -q general
#BSUB -n 1
#BSUB -R "rusage[mem=128]"
#BSUB -B
#BSUB -N
#BSUB -u myemail@miami.edu
#
# Run serial executable on 1 cpu of one node
cd /path/to/scratch/directory
./test.x a b c
```

Here is a detailed line-by-line breakdown of the keywords and their assigned values listed in this script:

`ScriptFile_keywords`

```
#!/bin/bash
specifies the shell to be used when executing the command portion of the script.
The default is Bash shell.

#BSUB -J serialjob
assign a name to job. The name of the job will show in the bjobs output.

#BSUB -P myproject
specify the project to use when submitting the job. This is required when a user has␣
↪more than one project on Pegasus.

#BSUB -e %J.err
redirect std error to a specified file

#BSUB -W 1:00
set wall clock run time limit of 1 hour, otherwise queue specific default run time␣
↪limit will be applied.

#BSUB -q general
```

(continues on next page)

(continued from previous page)

```
specify queue to be used. Without this option, default 'general' queue will be␣
↪applied.

#BSUB -n 1
specify number of processors. In this job, a single processor is requested.

#BSUB -R "rusage[mem=128]"
specify that this job requests 128 megabytes of RAM per core. Without this, a default␣
↪RAM setting will be applied:  1500MB per core

#BSUB -B
send mail to specified email when the job is dispatched and begins execution.

#BSUB -u example@miami.edu
send notification through email to example@miami.edu.

#BSUB -N
send job statistics report through email when job finishes.
```

Example scripts for parallel jobs

We recommend using Intel MPI unless you have specific reason for using OpenMP. Intel MPI scales better and has better performance than OpenMP.

Submit parallel jobs to the **parallel** job queue with `-q parallel`.

For optimum performance, the default resource allocation on the parallel queue is `ptile=16`. This requires the LSF job scheduler to allocate 16 processors per host, ensuring all processors on a single host are used by that job. ***Without prior authorization, any jobs using a number other than 16 will be rejected from the parallel queue.* Reserve enough memory for your jobs.** Memory reservations are per core. Parallel job performance may be affected, or even interrupted, by other badly-configured jobs running on the same host.

Example script for Intel/Intel MPI

`testparai.job`

```
#!/bin/bash
#BSUB -J mpijob
#BSUB -o %J.out
#BSUB -e %J.err
#BSUB -W 1:30
#BSUB -q parallel
#BSUB -n 32                          # Request 32 cores
#BSUB -R "span[ptile=16]"            # Request 16 cores per node
#BSUB -R "rusage[mem=128]"           # Request 128MB per core
#

mpiexec foo.exe
```

`foo.exe` is the mpi executable name. It can be followed by its own argument list.

Example script for MPI/OpenMP

testparao.job

```
#!/bin/bash
#BSUB -J mpijob
#BSUB -o %J.out
#BSUB -e %J.err
#BSUB -W 1:30
#BSUB -q parallel
#BSUB -n 32                              # Request 32 cores
#BSUB -R "span[ptile=16]"                # Request 16 cores per node
#BSUB -R "rusage[mem=128]"               # Request 128MB per core
#

mpiexec --mca btl self,sm,openib foo.exe
```

The command line is similar to Intel MPI job above. Option `--mca self,sm,openib` tells OpenMP to use lookback, shared memory, and openib for inter-process communication.

Pegasus Interactive Jobs

HPC clusters primarily take batch jobs and run them in the *background*—users do not need to interact with the job during the execution. However, sometimes users do need to interact with the application. For example, the application needs the input from the command line or waits for a mouse event in X windows. Use `bsub -Is -q interactive command` to launch interactive work on Pegasus. Remember to include your Pegasus cluster project ID in your job submissions with the `-P` flag.

To compile or install personal software on the Pegasus cluster, submit an "interactive" shell job to the Pegasus LSF scheduler and proceed with your compilations

```
[username@pegasus ~]$ bsub -Is -q interactive -P myProjectID bash
```

To run a non-graphical interactive Matlab session on the Pegasus cluster, submit an interactive job

```
[username@pegasus ~]$ bsub -Is -q interactive -P myProjectID matlab -nodisplay
```

To run an graphical interactive job, add `-XF` to your bsub flags (more on x11 below)

```
[username@pegasus ~]$ bsub -Is -q interactive -P myProjectID -XF $(java -jar ~/.local/
↪apps/ImageJ/ij.jar -batch ~/.local/apps/ImageJ/macros/screenmill.txt)
```

Upon exiting the interactive job, you will be returned to one of the login nodes.

Interactive Job Utilizing X11 client

Additionally, the interactive queue can run X11 jobs. The bsub `-XF` option is used for X11 jobs, for example:

```
[username@pegasus ~]$ bsub -Is -q interactive -P myProjectID -XF matlab
Job <50274> is submitted to queue <interactive>.
<<ssh X11 forwarding job>>
<<Waiting for dispatch ...>>
<<Starting on n003.pegasus.edu>>
```

Upon exiting the X11 interactive job, you will be returned to one of the login nodes.

To run an X11 application, establish an X tunnel with SSH when connecting to Pegasus. For example,

```
ssh -X username@pegasus.ccs.miami.edu
```

Note that by default, the auth token is good for 20 minutes. SSH will block new X11 connections after 20 minutes. To avoid this on Linux or OS X, run `ssh -Y` instead, or set the option **ForwardX11Trusted yes** in your ~/.ssh/config.

In Windows, use Cygwin/X to provide a Linux-like environment. Then run `ssh -Y` or set the option in your ~/.ssh/config file.

4.2.3 Pegasus Software

Pegasus Software Modules

IDSC ACS continually updates applications, compilers, system libraries, etc. To facilitate this task and to provide a uniform mechanism for accessing different revisions of software, ACS uses the modules utility. At login, modules commands set up a basic environment for the default compilers, tools, and libraries such as: the `$PATH`, `$MANPATH`, and `$LD_LIBRARY_PATH` environment variables. There is no need to set them or update them when updates are made to system and application software.

From Pegasus, users can view currently loaded modules with **module list** and check available software with **module avail *package*** (omitting the package name will show all available modules). Some modules are loaded automatically upon login:

```
[username@pegasus ~]$ module list
Currently Loaded Modulefiles:
  1) perl/5.18.1(default)    2) python/2.7.3(default)   3) gcc/4.4.7(default)      4)␣
↪share-rpms65
[username@pegasus ~]$ module avail R

---------------------- /share/Modules/general ----------------------
R/2.15.2         R/3.1.2(default) R/3.3.1

[username@pegasus ~]$ module load R
[username@pegasus ~]$ module list
Currently Loaded Modulefiles:
  1) perl/5.18.1(default)    2) python/2.7.3(default)   3) gcc/4.4.7(default)      4)␣
↪share-rpms65            5) R/3.1.2(default)
```

The table below lists commonly used modules commands.

Table 5: Pegasus Modules

Command	Purpose
`module avail`	lists all available modules
`module list`	list modules currently loaded
`module purge`	restores original setting by unloading all modules
`module load package`	loads a module e.g., the python package
`module unload package`	unloads a module e.g., the python package
`module switch old new`	replaces old module with new module
`module display package`	displays location and library information about a module

See our *Policies* page for minimum requirements and more information.

Pegasus Application Development

MPI and OpenMP modules are listed under Intel and GCC compilers. These MP libraries have been compiled and built with either the Intel compiler suite or the GNU compiler suite.

The following sections present the compiler invocation for serial and MP executions. All compiler commands can be used for just compiling with the `-c` flag (to create just the ".o" object files) or compiling and linking (to create executables). To use a different (non-default) compiler, first unload intel, swap the compiler environment, and then reload the MP environment if necessary.

Note: Only **one** MP module should be loaded at a time.

Compiling Serial Code

Pegasus has Intel and GCC compilers.

Vendor	Compiler	Module Command	Example
intel	icc (default)	`module load intel`	icc -o foo.exe foo.c
intel	ifor (default)	`module load intel`	ifort -o foo.exe foo.f90
gnu	gcc	`module load gcc`	gcc -o foo.exe foo.c
gnu	gcc	`module load gcc`	gfortran -o foo.exe foo.f90

Configuring MPI on Pegasus

There are three ways to configure MPI on Pegasus. Choose the option that works best for your job requirements.

- **Add the `module load` command to your startup files.** This is most convenient for users requiring only a single version of MPI. This method works with all MPI modules.
- **Load the module in your current shell.** For current MPI versions, the `module load` command does not need to be in your startup files. Upon job submission, the remote processes will inherit the submission shell environment and use the proper MPI library. This method does **not** work with older versions of MPI.
- **Load the module in your job script.** This is most convenient for users requiring different versions of MPI for different jobs. Ensure your script can execute the `module command` properly. For job script information, see *Scheduling Jobs on Pegasus*.

Compiling Parallel Programs with MPI

Pegasus supports Intel MPI and OpenMP for Intel and GCC compilers.

The **Message Passing Interface** (MPI) library allows processes in parallel application to communicate with one another. There is no default MPI library in your Pegasus environment. Choose the desired MPI implementation for your applications by loading an appropriate MPI module. Recall that only one MPI module should be loaded at a time.

Compiler	MPI	Module Command	Example
intel	Intel MPI	`module load intel impi`	mpif90 -o foo.exe foo.f90
intel	Intel MPI	`module load intel impi`	mpicc -o foo.exe foo.c
intel	OpenMP	`module load intel openmpi`	mpif90 -o foo.exe foo.f90
intel	OpenMP	`module load intel openmpi`	mpicc -o foo.exe foo.c
gcc	OpenMP	`module load openmpi-gcc`	mpif90 -o foo.exe foo.f90
gcc	OpenMP	`module load openmpi-gcc`	mpicc -o foo.exe foo.c

Pegasus Parallel

We recommend using Intel MPI unless you have specific reason for using OpenMP. Intel MPI scales better and has better performance than OpenMP.

Note: Only **one** MPI module should be loaded at a time.

C++ program and compilation:

`mpi_example1.cpp`

```
//============================================================
// C++ example: MPI Example 1
//============================================================
#include <iostream>
#include <mpi.h>
using namespace std;
int main(int argc, char** argv){
  int iproc;
  MPI_Comm icomm;
  int nproc;
  int i;
  MPI_Init(&argc,&argv);
  icomm = MPI_COMM_WORLD;
  MPI_Comm_rank(icomm,&iproc);
  MPI_Comm_size(icomm,&nproc);
  for ( i = 0; i <= nproc - 1; i++ ){
    MPI_Barrier(icomm);
    if ( i == iproc ){
      cout << "Rank " << iproc << " out of " << nproc << endl;
    }
  }
  MPI_Finalize();
  return 0;
}
```

```
[username@pegasus ~]$ mpicxx -o mpi_example1.x mpi_example1.cpp
```

C program and compilation:

`mpi_example1.c`

```
//=================================================================
// C example: MPI Example 1
//=================================================================
#include <stdio.h>
#include "mpi.h"
int main(int argc, char** argv){
int iproc;
int icomm;
int nproc;
int i;
MPI_Init(&argc,&argv);
icomm = MPI_COMM_WORLD;
MPI_Comm_rank(icomm,&iproc);
MPI_Comm_size(icomm,&nproc);
for ( i = 0; i <= nproc - 1; i++ ){
  MPI_Barrier(icomm);
  if ( i == iproc ){
    printf("%s %d %s %d \n","Rank",iproc,"out of",nproc);
  }
}
MPI_Finalize();
return 0;
}
```

```
[username@pegasus ~]$ mpicc -o mpi_example1.x mpi_example1.c
```

Fortran 90 program and compilation:

mpi_example1.f90

```
!=====================================================
! Fortran 90 example: MPI test
!=====================================================
program mpiexample1
implicit none
include 'mpif.h'
integer(4) :: ierr
integer(4) :: iproc
integer(4) :: nproc
integer(4) :: icomm
integer(4) :: i
call MPI_INIT(ierr)
icomm = MPI_COMM_WORLD
call MPI_COMM_SIZE(icomm,nproc,ierr)
call MPI_COMM_RANK(icomm,iproc,ierr)
do i = 0, nproc-1
  call MPI_BARRIER(icomm,ierr)
  if ( iproc == i ) then
    write (6,*) "Rank",iproc,"out of",nproc
  end if
end do
call MPI_FINALIZE(ierr)
if ( iproc == 0 ) write(6,*)'End of program.'
  stop
end program mpiexample1
```

```
[username@pegasus ~]$ mpif90 -o mpi_example1.x mpi_example1.f90
```

Fortran 77 program and compilation:

mpi_example1.f

```
c=======================================================
c Fortran 77 example: MPI Example 1
c=======================================================
program mpitest
implicit none
include 'mpif.h'
integer(4) :: ierr
integer(4) :: iproc
integer(4) :: nproc
integer(4) :: icomm
integer(4) :: i
call MPI_INIT(ierr)
icomm = MPI_COMM_WORLD
call MPI_COMM_SIZE(icomm,nproc,ierr)
call MPI_COMM_RANK(icomm,iproc,ierr)
do i = 0, nproc-1
  call MPI_BARRIER(icomm,ierr)
  if ( iproc == i ) then
    write (6,*) "Rank",iproc,"out of",nproc
  end if
end do
call MPI_FINALIZE(ierr)
if ( iproc == 0 ) write(6,*)'End of program.'
  stop
end
```

```
[username@pegasus ~]$ mpif77 -o mpi_example1.x mpi_example1.f
```

LSF script example

This batch script mpi_example1.job instructs LSF to reserve computational resources for your job. Change the `-P` flag argument to your project before running.

mpi_example1.job

```
#!/bin/sh
#BSUB -n 16
#BSUB -J test
#BSUB -o test.out
#BSUB -e test.err
#BSUB -a openmpi
#BSUB -R "span[ptile=16]"
#BSUB -q parallel
#BSUB -P hpc
mpirun.lsf ./mpi_example1.x
```

Submit this scriptfile using `bsub`. For job script information, see *Scheduling Jobs on Pegasus*.

```
[username@pegasus ~]$ bsub -q parallel < mpi_example1.job
Job <6021006> is submitted to queue <parallel>.
...
```

Pegasus Cluster Software Installation

Pegasus users are free to compile and install software in their own home directories, by following the software's source code or local installation instructions.

To install personal software on the Pegasus cluster, navigate to an interactive node by submitting an interactive shell job to the Pegasus cluster LSF scheduler. More on *Pegasus interactive jobs*.

Source code software installations ("compilations") can only be performed **in your local directories**. Users of Pegasus are not administrators of the cluster, and therefore cannot install software with the `sudo` command (or with package managers like `yum` / `apt-get`). If the software publisher does not provide compilation instructions, look for non-standard location installation instructions.

In general, local software installation involves:

1. confirming pre-requisite software & library availability, versions
2. **downloading and extracting files**
3. **configuring the installation prefix to a local directory** (compile only)
4. **compiling the software** (compile only)
5. updating PATH and creating symbolic links (optional)

Confirm that your software's pre-requisites are met, either in your local environment or on Pegasus as a module. You will need to load any Pegasus modules that are pre-requisites and install locally any other pre-requisites.

We suggest keeping downloaded source files separate from compiled files (and any downloaded binary files).

ACS does not install user software. Request cluster software installations from hpc@ccs.miami.edu

Downloading and extracting files

If necessary, create software directories under your home directory:

```
[username@pegasus ~]$ mkdir ~/local ~/src
```

We suggest keeping your compiled software separate from any downloaded files. Consider keeping downloaded binaries (pre-compiled software) separate from source files if you will be installing many different programs. These directories do not need to be named exactly as shown above.

Navigate to the `src` directory and download files:

Some programs require configuration and compilation (like autoconf). Other programs are pre-compiled and simply need to be extracted (like Firefox). Read and follow all instructions provided for each program.

```
[username@pegasus ~]$ cd ~/src
[username@pegasus src]$ wget http://ftp.gnu.org/gnu/autoconf/autoconf-2.69.tar.gz
[username@pegasus src]$ wget http://ftp.mozilla.org/pub/mozilla.org/firefox/releases/
↪36.0/linux-x86_64/en-GB/firefox-36.0.tar.bz2
```

Pre-compiled software can be extracted and immediately moved into your local software directory. We suggest maintaining subdirectories with application names and version numbers, as shown below.

Extract downloaded contents:

For pre-compiled software, extract and move contents to your local software directory. For software that must be configured and compiled, extract and move contents to your source files directory.

Extraction flags:

- tar.gz `xvzf` e**X**tract, **V**erbose, filter through g**Z**ip, using **F**ile
- tar.bz2 `xvjf` . . . filter through bzip2 (**j**)

Extract pre-compiled software and move to local software directory:

```
[username@pegasus src]$ tar xvjf firefox-36.0.tar.bz2
[username@pegasus src]$ mv firefox-36.0 $HOME/local/firefox/36
```

The newly-extracted Firefox executable should now be located in `~/local/firefox/36/firefox` Pre-compiled binaries, skip to **Updating PATH and creating symbolic links**.

Extract source code and `cd` to new directory:

```
[username@pegasus src]$ tar xvzf autoconf-2.69.tar.gz
[username@pegasus src]$ cd autoconf-2.69
[username@pegasus autoconf-2.69]$
```

Source code, proceed to ***Configuring installation and compiling software***.

Configuring installation and compiling software

We suggest using subdirectories with application names and version numbers, as shown below. There may be other configuration settings specific to your software.

Configure with local directory prefix (absolute path):

Configuration files may also be located in the `bin` (binary) directory, usually `software`/bin

```
[username@pegasus autoconf-2.69]$ ./configure --prefix=$HOME/local/autoconf/2.69
```

Make and install the software:

```
[username@pegasus autoconf-2.69]$ make
[username@pegasus autoconf-2.69]$ make install
...
```

If there are dependencies or conflicts, investigate the error output and try to resolve each error individually (install missing dependencies, check for specific flags suggested by software authors, check your local variables).

Updating PATH

PATH directories are searched in order. To ensure your compiled or downloaded software is found and used first, prepend the software executable location (usually in software/bin or software directories) to your PATH environment variable. Remember to add :$PATH to preserve existing environment variables.

Prepend software location to your PATH environment variable:

```
[username@pegasus ~]$ export PATH=$HOME/local/autoconf/2.69/bin:$PATH
```

Confirm by checking which software:

```
[username@pegasus ~]$ which autoconf
~/local/autoconf/2.69/bin/autoconf
```

Check software version:

Version flags may be software-dependent. Some common flags include --version, -v, and -V.

```
[username@pegasus ~]$ autoconf --version
autoconf (GNU Autoconf) 2.69
...
```

Creating symbolic links

To maintain multiple different versions of a program, use soft symbolic links to differentiate between the installation locations. Make sure the link and the directory names are distinct (example below). If local software has been kept in subdirectories with application names and version numbers, symlinks are not likely to conflict with other files or directories.

Create a distinctly-named symlink:

This symbolic link should point to the local software executable. The first argument is the local software executable location (~/local/firefox/36/firefox). The second argument is the symlink name and location (~/local/firefox36).

```
[username@pegasus ~]$ ln -s ~/local/firefox/36/firefox ~/local/firefox36
```

Append the local location to your PATH environment variable:

Remember to add :$PATH to preserve existing environment variables.

```
[username@pegasus ~]$ export PATH=$PATH:$HOME/local
```

Confirm both cluster copy and recently installed software:

The cluster copy of Firefox is `firefox`. The recently installed local copy is `firefox36` from the symbolic links created above.

```
[username@pegasus ~]$ which firefox
/usr/bin/firefox
[username@pegasus ~]$ firefox --version
Mozilla Firefox 17.0.10

[username@pegasus ~]$ which firefox36
~/local/firefox36
[username@pegasus ~]$ firefox36 --version
Mozilla Firefox 36.0
```

Reminder - to launch Firefox, connect to Pegasus via SSH with X11 forwarding enabled.

Persistent PATH

To persist additions to your PATH variable, edit the appropriate profile configuration file in your home directory. For Bash on Pegasus, this is `.bash_profile`.

Update PATH in shell configuration (bash):

Use `echo` and the append redirect (>>) to update `PATH` in `.bash_profile`.

```
[username@pegasus ~]$ echo 'export PATH=$HOME/local/autoconf/2.69/bin:$PATH' >> ~/.
↪bash_profile
[username@pegasus ~]$ echo 'export PATH=$PATH:$HOME/local' >> ~/.bash_profile
```

*both in one command (note the newline special character **‘‘n‘‘* directly in between the commands:*

```
[username@pegasus ~]$ echo -e 'export PATH=$HOME/local/autoconf/2.69/bin:
↪$PATH\nexport PATH=$PATH:$HOME/local' >> ~/.bash_profile
```

or edit the file directly:

```
[username@pegasus ~]$ vi ~/.bash_profile
...
PATH=$PATH:$HOME/bin
PATH=$HOME/local/autoconf/2.69/bin:$PATH
PATH=$PATH:$HOME/local
...
```

Reload shell configurations (Bash) and check PATH:

Look for the recently added path locations and their order.

```
[username@pegasus ~]$ source ~/.bash_profile
[username@pegasus ~]$ echo $PATH
/nethome/username/local/autoconf/2.69/bin:/share/opt/python/2.7.3/bin: ... :/share/
↪sys65/root/sbin:/nethome/username/bin:/nethome/username/local
```

Allinea on Pegasus

Profile and Debug with Allinea Forge, the new name for the unified Allinea MAP and Allinea DDT tools. See the user guide PDFs below for Allinea Forge and Performance Reports, available as modules on Pegasus.

Allinea 6.1-Forge guide [PDF download]

Allinea 6.1-PR guide [PDF download]

Amazon Web Services CLI on Pegasus Cluster

For accessing your AWS services from the Pegasus cluster. Note, IDSC does not administer or manage AWS services.

- **load the cluster's aws-cli module**
- configure aws with your IAM user account credentials (one-time)
  - aws user credentails file : **~/.aws/credentials**
  - aws user configurations file : **~/.aws/config**
- **check your aws configurations**

```
[username@login4 ~]$ module load aws-cli
[username@login4 ~]$ aws configure
..
[username@login4 ~]$ aws configure list
    Name                    Value             Type    Location
    ----                    -----             ----    --------
 profile                <not set>             None    None
      access_key     ****************44OL shared-credentials-file
      secret_key     ****************unuw shared-credentials-file
  region                 us-east-1              env    ['AWS_REGION', 'AWS_DEFAULT_
↪REGION']
```

Getting Started

- Amazon s3 "Getting started" guide : https://aws.amazon.com/s3/getting-started/
- User guide : https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html
- Pricing : https://aws.amazon.com/s3/pricing/

Amazon s3 is "safe, secure Object storage" with web access, pay-as-you-go subscription

AWS s3 "IAM" User Accounts : https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users_create.html

(needed to access AWS web services through the CLI from Pegasus)

AWS CLI, User Overview

Note that your AWS IAM web login & password are different from your access key credentials. For help with your IAM web login account, contact your AWS Administrator.

1. Get your AWS IAM access keys (from AWS web or your AWS Administrator)
2. Configure your AWS access from Pegasus
3. Access & use your AWS s3 instance

Getting your AWS IAM access keys (from web)

Your AWS Administrator may have already provided you with IAM access keys for your Amazon instance. If you need to generate new access keys, log into the AWS web interface. Generating new keys will inactivate any old keys.

`https://Your_AWS_instance_ID.signin.aws.amazon.com/console` OR https://console.aws.amazon.com/ and enter your instance ID or alias manually.

Reminder, your AWS IAM web login & password are different from your access key credentials. For help with your IAM web login account, contact your AWS Administrator.

1. Log into your AWS Management Console, with your IAM web login & password
    a. If you forgot your IAM web login, contact the AWS administrator that provided you with your IAM user name.
    b. "IAM users, only your administrator can reset your password."
    c. More on IAM account logins : https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users_sign-in.html
2. From your IAM account drop-down menu, choose "My security credentials"
    a. If needed, update your password
3. Under the "Access keys" heading, create & download your access key credentials *credentials.csv*
    a. 'credentials.csv' contains both your Access Key & your Secret Access Key
    b. "If you lose or forget your secret key, you cannot retrieve it. Instead, create a new access key and make the old key inactive."
    c. More about access keys : http://docs.aws.amazon.com/console/iam/self-accesskeys

Configuring your AWS access (cli)

Have your IAM access key credentials, from 'credentials.csv' (from AWS web or your AWS Administrator).

AWS CLI quickstart : https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html#cli-configure-quickstart-config

1. On the Pegasus cluster,
    a. Load the cluster "aws-cli" module
    b. (optional) Check the module's default settings
2. Run the command "aws configure"
3. Enter your AWS IAM credentials (from 'credentials.csv')
    a. These settings will save to your home directory
    b. More about aws configuration files : https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html

Examples,

```
[username@login4 ~]$ module load aws-cli
[username@login4 ~]$ which aws
/share/apps/c7/aws-cli/bin/aws

[username@login4 ~]$ module show aws-cli
..
```

(continues on next page)

(continued from previous page)

```
# Set environment variables
setenv          AWS_DEFAULT_REGION "us-east-1"
```

—> The default retry mode for AWS CLI version 2 is "standard"

These module settings will override user "aws configure" settings. You can override module settings by using aws command-line options.

Using AWS s3 buckets from the cli

1. Create a bucket
    a. bucket names must be globally unique (e.g. two different AWS users can not have the same bucket name)
    b. bucket names cannot contain spaces
    c. More on bucket naming requirements : https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-s3-bucket-naming-requirements.html
2. List your s3 bucket contents
    a. buckets are collections of objects
    b. "objects" behave like files
    c. "objects/" (with a trailing slash) behave like folders
3. Download objects from AWS s3 buckets with cp
    a. specify directories, or use current local
    b. use the '–recursive' flag to download all objects
4. Upload files to an AWS s3 bucket with cp
    a. specify AWS bucket paths
    b. use the '–recursive' flag to upload all objects
5. Delete objects from AWS s3 buckets with rm
    a. list & test with '–dryrun' flag
    b. then remove with rm
6. Sync between your local directory and an AWS s3 bucket with sync
    a. recursive
    b. copies changes & new files only
    c. doesn't delete missing files

More on using s3 : https://docs.aws.amazon.com/cli/latest/userguide/cli-services-s3.html
AWS s3 command examples : https://docs.aws.amazon.com/cli/latest/userguide/cli-services-s3-commands.html
AWS s3 CLI reference : https://docs.aws.amazon.com/cli/latest/reference/s3/

Create (make) an AWS s3 bucket

```
[username@login4 ~]$ aws s3 mb s3://idsc-acs-test-bucket2
make_bucket: idsc-acs-test-bucket2
```

List all user owned AWS s3 buckets

```
[username@login4 ~]$ aws s3 ls
2021-09-01 11:57:25 idsc-acs-test-bucket
2021-09-01 13:11:39 idsc-acs-test-bucket2
```

List AWS s3 bucket contents

```
[username@login4 ~]$ aws s3 ls s3://idsc-acs-test-bucket
                           PRE testfolder/
2021-09-01 12:02:29        160 aws_bucket_test.txt
```

List AWS s3 “folder” (object/) contents (include trailing slash)

```
[username@login4 awstests]$ aws s3 ls s3://idsc-acs-test-bucket/testfolder/
2021-09-01 16:04:19         20 testfile1.test
2021-09-01 16:04:19         20 testfile2.test
2021-09-01 16:04:19         20 testfile3.test
```

Download an object from an AWS s3 bucket (to your current local directory)

```
[username@login4 ~]$ aws s3 cp s3://idsc-acs-test-bucket/aws_bucket_test.txt .
download: s3://idsc-acs-test-bucket/aws_bucket_test.txt to ./aws_bucket_test.txt
```

Download an object from an AWS s3 bucket (to a specified local directory)

```
[username@login4 ~]$ aws s3 cp s3://idsc-acs-test-bucket/aws_bucket_test.txt ~/aws-
↪downloads/.
download: s3://idsc-acs-test-bucket/aws_bucket_test.txt to /nethome/username/aws-
↪downloads/aws_bucket_test.txt

Download all objects from an AWS "folder" (to your current local directory,␣
↪recursive)::

[username@login4 awstests]$ aws s3 cp s3://idsc-acs-test-bucket/testfolder testfolder␣
↪--recursive
download: s3://idsc-acs-test-bucket/testfolder/testfile1.test to testfolder/testfile1.
↪test
download: s3://idsc-acs-test-bucket/testfolder/testfile2.test to testfolder/testfile2.
↪test
download: s3://idsc-acs-test-bucket/testfolder/testfile5.test to testfolder/testfile3.
↪test
```

Upload a file to an AWS s3 bucket

```
[username@login4 ~]$ aws s3 cp aws_bucket_cli_upload_test.txt s3://idsc-acs-test-
↪bucket/
upload: ./aws_bucket_cli_upload_test.txt to s3://idsc-acs-test-bucket/aws_bucket_cli_
↪upload_test.txt

[username@login4 ~]$ aws s3 ls s3://idsc-acs-test-bucket
2021-09-01 12:41:47         94 aws_bucket_cli_upload_test.txt
2021-09-01 12:02:29        160 aws_bucket_test.txt
```

Upload multiple files to an AWS s3 bucket (recursive)

```
[username@login4 ~]$ aws s3 cp . s3://idsc-acs-test-bucket/ --recursive
upload: ./another_test.txt to s3://idsc-acs-test-bucket/another_test
upload: ./testimage2.jpg to s3://idsc-acs-test-bucket/testimage2.jpg
upload: ./testimage.jpg to s3://idsc-acs-test-bucket/testimage.jpg
upload: ./aws_bucket_cli_upload_test.txt to s3://idsc-acs-test-bucket/aws_bucket_cli_
↪upload_test.txt
upload: ./aws_bucket_test.txt to s3://idsc-acs-test-bucket/aws_bucket_test.txt
```

Upload multiple files to an AWS s3 bucket, with filters (examples by file extension)

```
# upload (copy to AWS) ONLY files with '.txt' extension

[username@login4 ~]$ aws s3 cp . s3://idsc-acs-test-bucket/ --recursive --exclude "*"␣
↪--include "*.txt"
upload: ./aws_bucket_test.txt to s3://idsc-acs-test-bucket/aws_bucket_test.txt
upload: ./aws_bucket_cli_upload_test.txt to s3://idsc-acs-test-bucket/aws_bucket_cli_
↪upload_test.txt

# upload ONLY files with '.jpg' extension

[username@login4 ~]$ aws s3 cp . s3://idsc-acs-test-bucket/ --recursive --exclude "*"␣
↪--include "*.jpg"
upload: ./testimage.jpg to s3://idsc-acs-test-bucket/testimage.jpg
upload: ./testimage2.jpg to s3://idsc-acs-test-bucket/testimage2.jpg

# upload all files EXCEPT those with '.txt' extension

[username@login4 ~]$ aws s3 cp . s3://idsc-acs-test-bucket/ --recursive --exclude "*.
↪txt"
upload: ./testimage.jpg to s3://idsc-acs-test-bucket/testimage.jpg
upload: ./testimage2.jpg to s3://idsc-acs-test-bucket/testimage2.jpg
upload: ./another_test to s3://idsc-acs-test-bucket/another_test

# list local directory contents

[username@login4 ~]$ ls -lah
..
-rw-r--r--  1 username hpc   0 Sep 10 13:15 another_test
-rw-r--r--  1 username hpc  94 Sep 10 13:15 aws_bucket_cli_upload_test.txt
-rw-r--r--  1 username hpc 160 Sep 10 13:15 aws_bucket_test.txt
-rw-r--r--  1 username hpc  87 Sep 10 13:32 testimage2.jpg
-rw-r--r--  1 username hpc 16K Sep 10 13:33 testimage.jpg
```

Delete an object from an AWS s3 bucket (list, test with dryrun, then remove)

```
[username@login4 ~]$ aws s3 ls s3://idsc-acs-test-bucket --human-readable
2021-09-01 13:31:31    4.4 GiB BIG_FILE.iso
2021-09-01 13:29:26    0 Bytes another_test
2021-09-01 13:03:40    0 Bytes another_test.txt
2021-09-01 13:29:26   94 Bytes aws_bucket_cli_upload_test.txt
2021-09-01 13:29:26  160 Bytes aws_bucket_test.txt
2021-09-01 13:29:26   16.0 KiB testimage.jpg
2021-09-01 13:29:26   87 Bytes testimage2.jpg

[username@login4 ~]$ aws s3 rm --dryrun s3://idsc-acs-test-bucket/BIG_FILE.iso
(dryrun) delete: s3://idsc-acs-test-bucket/BIG_FILE.iso
```

(continues on next page)

(continued from previous page)

```
[username@login4 ~]$ aws s3 rm s3://idsc-acs-test-bucket/BIG_FILE.iso
delete: s3://idsc-acs-test-bucket/BIG_FILE.iso
```

Sync local directory "testfolder" with AWS s3 object "testfolder/" (creates if doesn't exist)

```
[username@login4 ~]$ aws s3 sync testfolder s3://idsc-acs-test-bucket/testfolder
upload: testfolder/testfile1.test to s3://idsc-acs-test-bucket/testfolder/testfile1.
↪test
upload: testfolder/testfile2.test to s3://idsc-acs-test-bucket/testfolder/testfile2.
↪test
upload: testfolder/testfile3.test to s3://idsc-acs-test-bucket/testfolder/testfile3.
↪test
```

Add another file, sync again, then list aws s3 "testfolder/" contents

```
[username@login4 ~]$ echo "this is my new test file" > testfolder/testfileNEW.test
[username@login4 ~]$ aws s3 sync testfolder s3://idsc-acs-test-bucket/testfolder
upload: testfolder/testfileNEW.test to s3://idsc-acs-test-bucket/testfolder/
↪testfileNEW.test

[username@login4 ~]$ aws s3 ls s3://idsc-acs-test-bucket/testfolder/
2021-09-01 17:16:10         20 testfile1.test
2021-09-01 16:04:19         20 testfile2.test
2021-09-01 16:04:19         20 testfile3.test
2021-09-01 17:16:10         25 testfileNEW.test
```

Get help with AWS s3 commands

```
aws s3 help
aws s3 ls help
aws s3 cp help
```

AWS s3 Include and Exclude filters

The following pattern symbols are supported

- `*` Matches everything
- `?` Matches any single character
- `[sequence]` Matches any character in `sequence`
- `[!sequence]` Matches any character not in `sequence`

Filters that appear later in the command take precedence. Put `--exclude` filters first, then add `--include` filters after to re-include specifics. See command examples above.

More on filters : https://awscli.amazonaws.com/v2/documentation/api/latest/reference/s3/index.html#use-of-exclude-and-include-filters

Matlab on Pegasus

Interactive Mode

There are several ways to run MATLAB commands/jobs interactively, with or without the graphical interface.

Graphical Interface Mode

To run MATLAB using graphical interface mode, connect with display forwarding. For more information about display forwarding, see *Forwarding the Display*.

Load and launch matlab on one of the interactive compute nodes as shown below. If you belong to more than one project, specify the `projectID` as well.

```
[username@pegasus ~]$ module load matlab
[username@pegasus ~]$ bsub -Is -q interactive -XF -P projectID matlab
```

Once the interactive MATLAB graphical desktop is loaded, you can then run MATLAB commands or scripts in the MATLAB command window. The results will be shown in the MATLAB command window and the figure/plot will be displayed in new graphical windows on your computer. See examples below.

```
>> x = rand(1,100);
>> plot(x);
>>
>> x = [0: pi/10: pi];
>> y = sin(x);
>> z = cos(x);
>> figure;
>> plot(x, y);
>> hold('on');
>> plot(x, z, '--');
```

Graphical Interactive Mode with no graphical desktop window

Running MATLAB in a full graphical mode may get slow depending on the network load. Running it with `-nodesktop` option will use your current terminal window (in Linux/Unix) as a desktop, while allowing you still to use graphics for figures and editor.

```
[username@pegasus ~]$ module load matlab
[username@pegasus ~]$ bsub -Is -q interactive -XF -P projectID matlab -nodesktop
```

Non-Graphical interactive Mode

If your MATLAB commands/jobs do not need to show graphics such as figures and plots, or to use a built-in script editor, run the MATLAB in the non-graphical interactive mode with `-nodisplay`.

Open a regular ssh connection to Pegasus.

```
[username@pegasus ~]$ module load matlab
[username@pegasus ~]$ bsub -Is -q interactive -P projectID matlab -nodisplay
```

This will bring up the MALAB command window:

```
                            < M A T L A B (R) >
                  Copyright 1984-2013 The MathWorks, Inc.
                    R2013a (8.1.0.604) 64-bit (glnxa64)
                             February 15, 2013


No window system found.  Java option 'Desktop' ignored.
```

(continues on next page)

(continued from previous page)

```
To get started, type one of these: helpwin, helpdesk, or demo.
For product information, visit www.mathworks.com.
>>
```

To exit, type `exit` or `quit`. Again, remember to import the prepared LSF configuration file mentioned above if you want to use MATLAB parallel computing.

Batch Processing

For off-line non-interactive computations, submit the MATLAB script to the LSF scheduler using the `bsub` command. For more information about job scheduling, see *Scheduling Jobs*. Example single-processor job submission:

```
[username@pegasus ~]@ bsub < example.job
```

`example.job`

```
#BSUB -J example
#BSUB -q general
#BSUB -P projectID
#BSUB -n 1
#BSUB -o example.o%J
#BSUB -e example.e%J
matlab -nodisplay -r my_script
```

In this example, "my_script" corresponds to "my_script.m" in the current working directory.

After the job is finished, the results will be saved in the output file named "example.o######" where "######" is a `jobID` number assigned by LSF when you submit your job.

Parallel Computations

MATLAB has software products to enable parallel computations for multi-core computers as well as for multiple-node computer clusters. The latter case scenario requires a job scheduler, such as LSF on Pegasus cluster.

The MATLAB product for the parallel processing that uses the cores of the same node is the "Distributed Computing Toolbox/DCT" (also appears in MATLAB documentation under the name of "Parallel Computing Toolbox"). Licensed MATLAB software product for a computer cluster is called "Distributed Computing Engine/DCE" (also appears in documentation as "MATLAB Distributed Computing Server").

Single-node parallel MATLAB jobs (up to 16 cpus)

For a single-node parallel job, MATLAB Distributed Computing Toolbox (licensed software) is used. It has a build-in default MATLAB cluster profile **'local'**, from which the pool of MatlabWorkers can be reserved for computations. The default number of MatlabWorkers is 12. You can specify up to 15 on a single Pegasus node using the **general** queue, and 16 cpus using the **parallel** queue. For more information about queue and parallel resource distribution requirements, see *Scheduling Jobs*.

Refer to MATLAB documentation on the ways to adapt your script for multi-processor calculations. One of the parallel tools in MATLAB is the `parfor` loop replacing the regular `for` loop, and in the example is given below:

```
%=============================================================
% dct_example.m
% Distributed Computing Toolbox (DCT)
% Example: Print datestamp within a parallel "parfor" loop
%=============================================================
%% Create a parallel pool of MatlabWorkers on the current working node:
parpool('local',16);
% The test loop size
N = 40;
tstart = tic();
parfor(ix=1:N)
  ixstamp = sprintf('Iteration %d at %s\n', ix, datestr(now));
  disp(ixstamp);
  pause(1);
end
cputime=toc(tstart);
toctime= sprintf('Time used is %d seconds', cputime);
disp(toctime)
%% delete current parallel pool:
delete(gcp)
```

Multi-node parallel MATLAB jobs (16-32 cpus)

For running multi-processor MATLAB jobs that involve 16+ cpus and more than a single node, MATLAB Distributed Computer Engine (licensed software) is used, with currently 32 licenses available on Pegasus. *These jobs must be submitted to the **parallel* queue with the appropriate ptile resource distribution.* For more information about queue and resource distribution requirements, see *Scheduling Jobs*.

The parallel LSF MATLAB cluster also needs to be configured. After loading the matlab module, import the default LSF parallel configuration as following:

```
[username@pegasus ~]$ matlab -nodisplay -r "parallel.importProfile('/share/opt/MATLAB/
↪etc/LSF1.settings');exit";reset
```

This command only needs to be run once. It imports the cluster profile named 'LSF1' that is configured to use up to 32 MatlabWorkers and to submit MATLAB jobs to the **parallel** pegasus queue. This profile does not have a `projetID` associated with the job, and you may need to coordinate the project name for the LSF job submission. This can be done by running the following script (only once!) during your matlab session:

```
%% conf_lsf1_project_id.m
%% Verify that LSF1 profile exists, and indicate the current default profile:
[allProfiles,defaultProfile] = parallel.clusterProfiles()
%% Define the current cluster object using LSF1 profile
myCluster=parcluster('LSF1')
%% View current submit arguments:
get(myCluster,'SubmitArguments')
%% Set new submit arguments, change projectID below to your current valid project:
set(myCluster,'SubmitArguments','-q general -P projectID')
%% Save the cluster profile:
saveProfile(myCluster)
%% Set the 'LSF1' to be used as a default cluster profile instead of a 'local'
parallel.defaultClusterProfile('LSF1');
%% Verify the current profiles and the default:
[allProfiles,defaultProfile] = parallel.clusterProfiles()
```

The above script also reviews your current settings of the cluster profiles. You can now use the cluster profile for distributed calculations on up to 32 cpus, for example, to create a pool of MatlabWorkers for a `parfor` loop:

```
%==========================================================
% dce_example.m
% Distributed Computing Engine (DCE)
% Example: Print datestamp within a parallel "parfor" loop
%==========================================================
myCluster=parcluster('LSF1')
% Maximum number of MatlabWorkers is 32 (number of MATLAB DCE Licenses)
parpool(myCluster,32);
% The test loop size
N = 40;
tstart = tic();
parfor(ix=1:N)
  ixstamp = sprintf('Iteration %d at %s\n', ix, datestr(now));
  disp(ixstamp);
  pause(1);
end
cputime=toc(tstart);
toctime= sprintf('Time used is %d seconds', cputime);
disp(toctime)
delete(gcp)
```

Please see MATLAB documentation on more ways to parallelize your code.

There may be other people running Distributed Computing Engine and thus using several licenses. Please check the license count as following (all in a single line):

```
[username@pegasus ~]$ /share/opt/MATLAB/R2013a/etc/lmstat -S MLM -c /share/opt/MATLAB/
↪R2013a/licenses/network.lic
```

Find the information about numbers of licenses used for the "Users of MATLAB_Distrib_Comp_Engine", "Users of MATLAB", and "Users of Distrib_Computing_Toolbox".

Note on Matlab cluster configurations

After importing the new cluster profile, it will remain in your available cluster profiles. Validate using the `parallel.clusterProfiles()` function. You can create, change, and save profiles using `SaveProfile` and `SaveAsProfile` methods on a cluster object. In the examples, "myCluster" is the cluster object. You can also create, import, export, delete, and modify the profiles through the "Cluster Profile Manager" accessible via MATLAB menu in a graphical interface. It is accessed from the "HOME" tab in the GUI desktop window under "ENVIRONMENT" section: ->"Parallel"->"Manage Cluster Profiles"

You can also create your own LSF configuration from the Cluster Profile Manager. Choose "Add"->"Custom"->"3RD PARTY CLUSTER PROFILE"->"LSF" as shown below:

... and configure to your needs:

Perl on Pegasus

Users are free to compile and install Perl modules in their own home directories. Most Perl modules can be installed into a local library with **CPAN** and **cpanminus**. If you need a specific version, we suggest specifying the version or downloading, extracting, and installing using Makefile.PL.

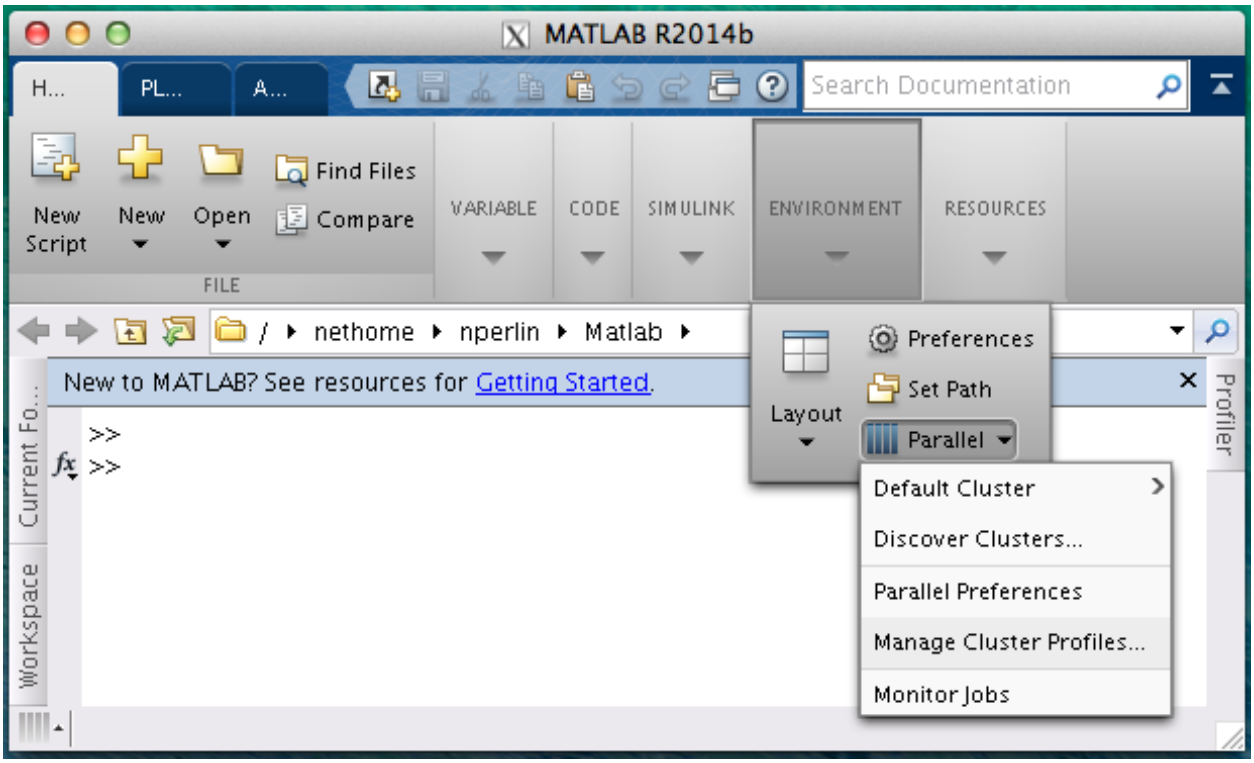


Fig. 1: Cluster Profile Manager

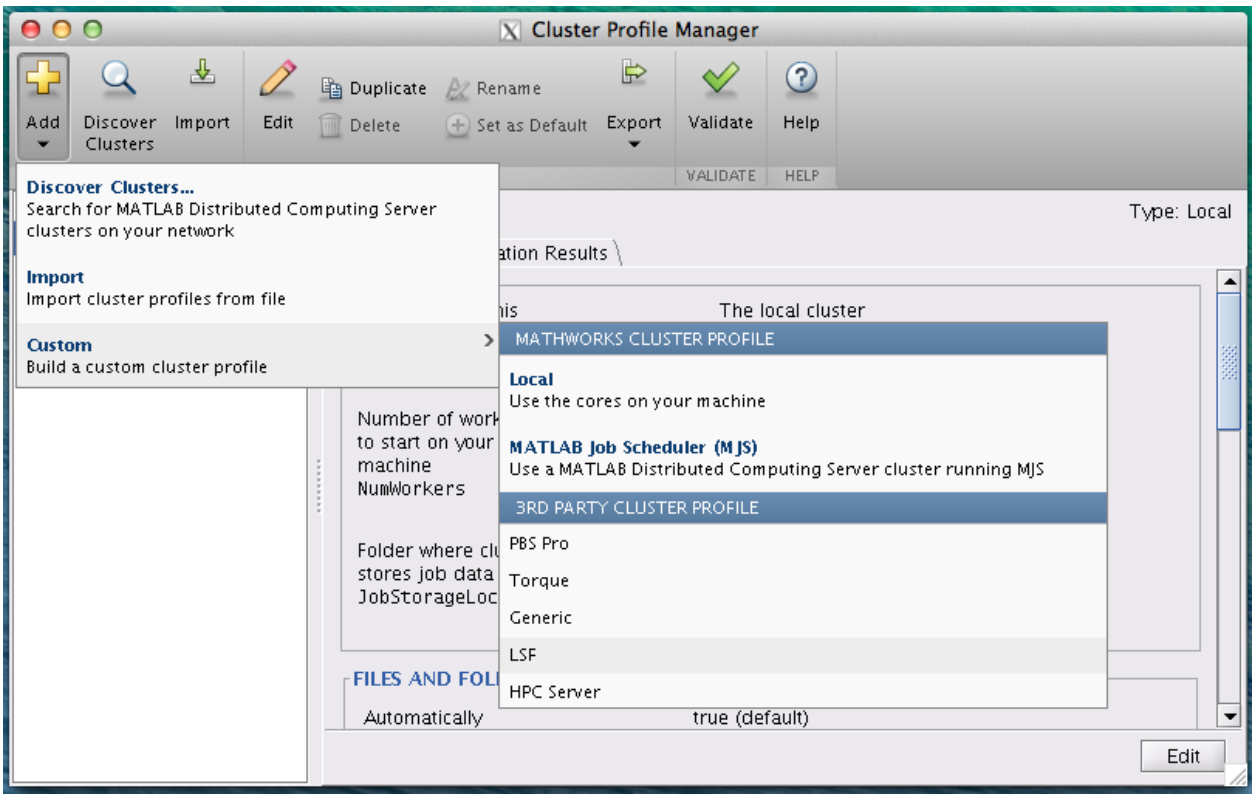


Fig. 2: Cluster Profile Manager: new LSF cluster

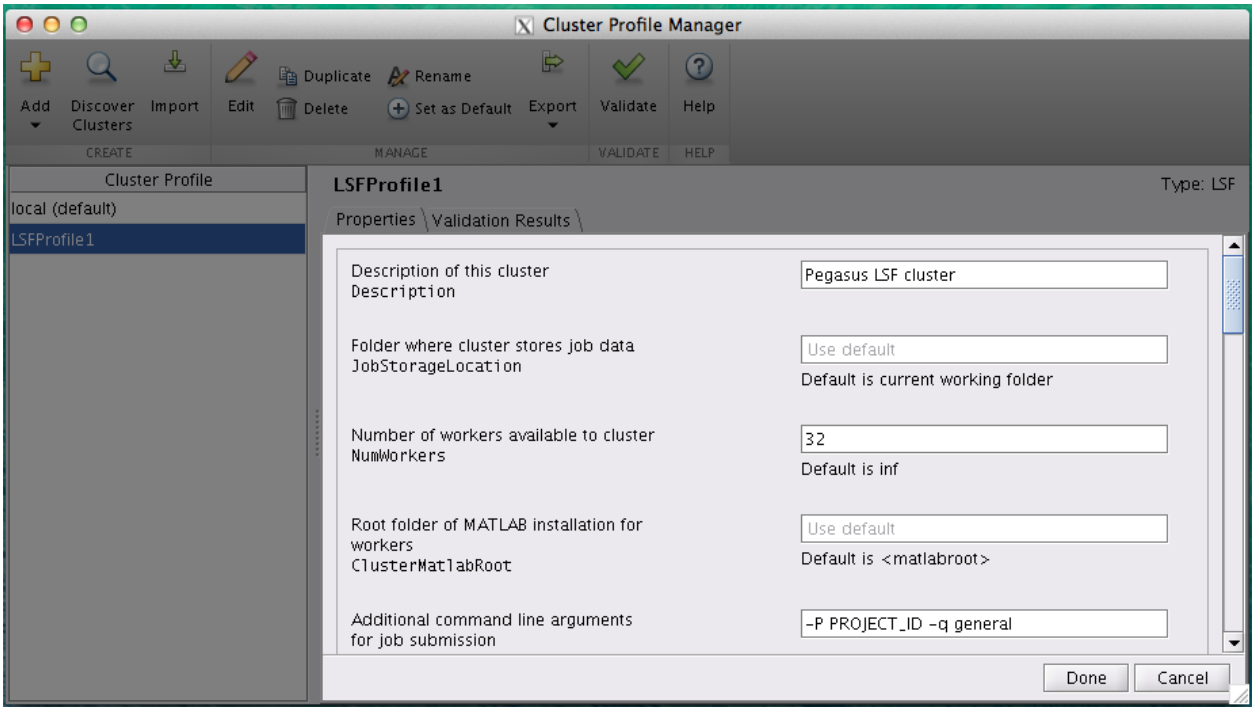


Fig. 3: New LSF cluster in Matlab

Configuring a Local Library

Local libraries can be configured during initial CPAN configuration and by editing shell configuration files after installing the `local::lib` module. By default, `local::lib` installs here: `~/perl5`.

Configure with CPAN:

During initial CPAN configuration, answer `yes` to automatic configuration, `local::lib` to approach, and `yes` to append locations to your shell profile (Bash). Quit CPAN and source your shell configuration before running `cpan` again.

```
[username@pegasus ~]$ cpan
...
Would you like to configure as much as possible automatically? [yes] yes
...
Warning: You do not have write permission for Perl library directories.
...
What approach do you want?  (Choose 'local::lib', 'sudo' or 'manual')
 [local::lib] local::lib
...
local::lib is installed. You must now add the following environment variables
to your shell configuration files (or registry, if you are on Windows) and
then restart your command line shell and CPAN before installing modules:

PATH="/nethome/username/perl5/bin${PATH+:}${PATH}"; export PATH;
PERL5LIB="/nethome/username/perl5/lib/perl5${PERL5LIB+:}${PERL5LIB}"; export PERL5LIB;
PERL_LOCAL_LIB_ROOT="/nethome/username/perl5${PERL_LOCAL_LIB_ROOT+:}${PERL_LOCAL_LIB_
↪ROOT}"; export PERL_LOCAL_LIB_ROOT;
```

(continues on next page)

(continued from previous page)

```
PERL_MB_OPT="--install_base \"/nethome/username/perl5\""; export PERL_MB_OPT;
PERL_MM_OPT="INSTALL_BASE=/nethome/username/perl5"; export PERL_MM_OPT;

Would you like me to append that to /nethome/username/.bashrc now? [yes] yes
...
cpan[1]> quit
...
*** Remember to restart your shell before running cpan again ***
[username@pegasus ~]$ source ~/.bashrc
```

Configure after `local::lib` module installation:

If CPAN has already been configured, ensure `local::lib` is installed and the necessary environment variables have been added to your shell configuration files. Source your shell configuration before running `cpan` again.

```
[username@pegasus ~]$ cpan local::lib
Loading internal null logger. Install Log::Log4perl for logging messages
...
local::lib is up to date (2.000018).
[username@pegasus ~]$ echo 'eval "$(perl -I$HOME/perl5/lib/perl5 -Mlocal::lib)"' >>␣
↪perl -~/.bashrc
[username@pegasus ~]$ source ~/.bashrc
...
```

Update CPAN:

Update CPAN, if necessary. This can be done from the Pegasus prompt or the CPAN prompt.

```
[username@pegasus ~]$ cpan CPAN
or
cpan[1]> install CPAN
...
Appending installation info to /nethome/username/perl5/lib/perl5/x86_64-linux-thread-
↪multi/perllocal.pod
  ANDK/CPAN-2.10.tar.gz
  /usr/bin/make install  -- OK

cpan[2]> reload cpan
```

Confirm local library location:

Look for `~/perl5/...` directories in `$PATH` and `@INC`.

```
[username@pegasus ~]$ echo $PATH
/share/opt/perl/5.18.1/bin:/nethome/username/perl5/bin:/share/lsf/9.1/linux2.6-glibc2.
↪3-x86_64/etc:...
[username@pegasus ~]$ perl -e 'print "@INC"'
/nethome/username/perl5/lib/perl5/5.18.1/x86_64-linux-thread-multi /nethome/username/
↪perl5/lib/perl5/5.18.1 ...
```

Installing Perl Modules

Once a local library has been installed and configured, CPAN modules will install to the local directory (default `~/perl5`). The format for installing Perl modules with CPAN or cpanminus is `Module::Name`.

Install with CPAN:

Install from the Pegasus prompt or the CPAN prompt. Run `cpan -h` or `perldoc cpan` for more options.

```
[username@pegasus ~]$ cpan App::cpanminus
Loading internal null logger. Install Log::Log4perl for logging messages
Reading '/nethome/username/.cpan/Metadata'
...
or
[username@pegasus ~]$ cpan
cpan[1]> install App::cpanminus
Reading '/nethome/username/.cpan/Metadata'
...
```

To install a specific module version with `cpan`, provide the full distribution path.

```
[username@pegasus ~]$ cpan MIYAGAWA/App-cpanminus-1.7040.tar.gz
```

Install with cpanminus:

cpanminus is a CPAN module installation tool that will use your local library, if configured. Install from the Pegasus prompt with `cpanm Module::Name`. Run `cpanm -h` or `perldoc cpanm` for more options.

```
[username@pegasus ~]$ cpanm IO::All
--> Working on IO::All
Fetching http://www.cpan.org/authors/id/I/IN/INGY/IO-All-0.86.tar.gz ... OK
Configuring IO-All-0.86 ... OK
Building and testing IO-All-0.86 ... OK
Successfully installed IO-All-0.86
1 distribution installed
```

To install a specific module version with `cpanm`, provide either the full distribution path, the URL, or the path to a local tarball.

```
[username@pegasus ~]$ cpanm MIYAGAWA/App-cpanminus-1.7040.tar.gz
or
[username@pegasus ~]$ cpanm http://search.cpan.org/CPAN/authors/id/M/MI/MIYAGAWA/App-
↪cpanminus-1.7040.tar.gz
or
[username@pegasus ~]$ cpanm ~/App-cpanminus-1.7040.tar.gz
```

Deactivating Local Library Environment Variables

To remove all directories added to search paths by `local::lib` in the current shell's environment, use the `--deactivate-all` flag. Note that environment variables will be re-enabled in any sub-shells when using `.bashrc` to initialize local::lib.

```
[username@pegasus ~]$ eval $(perl -Mlocal::lib=--deactivate-all)
[username@pegasus ~]$ echo $PATH
/share/opt/perl/5.18.1/bin:...
[username@pegasus ~]$ perl -e 'print "@INC"'
/share/opt/perl/5.18.1/lib/site_perl/5.18.1/x86_64-linux-thread-multi ...
```

Source your shell configuration to re-enable the local library:

```
[username@pegasus ~]$ source ~/.bashrc
...
[username@pegasus ~]$ echo $PATH
/nethome/username/perl5/bin:/share/lsf/9.1/linux2.6-glibc2.3-x86_64/etc:...
[username@pegasus ~]$ perl -e 'print "@INC"'
/nethome/username/perl5/lib/perl5/5.18.1/x86_64-linux-thread-multi /nethome/username/
↪perl5/lib/perl5/5.18.1 /nethome/username/perl5/lib/perl5/x86_64-linux-thread-multi /
↪nethome/username/perl5/lib/perl5 ...
```

Python on Pegasus

Users are free to compile and install Python modules in their own home directories on Pegasus. Most Python modules can be installed with the `--user` flag using PIP, easy_install, or the setup.py file provided by the package. If you need a specific version of a Python module, we suggest using PIP with a direct link or downloading, extracting, and installing using setup.py. If you need to maintain multiple versions, see Python Virtual Environments (below).

The `--user` flag will install Python 2.7 modules here: `~/.local/lib/python2.7/site-packages` Note the default location `~/.local` is a hidden directory. If the Python module includes executable programs, they will usually be installed into `~/.local/bin`.

To specify a different location, use `--prefix=$HOME/local/python2mods` (or another path). The above prefix flag example will install Python 2.7 modules here: `~/local/python2mods/lib/python2.7/site-packages`

Loading and Switching Python Modules

Confirm Python is loaded:

```
[username@pegasus ~]$ module list
Currently Loaded Modulefiles:
  1) perl/5.18.1             3) gcc/4.4.7(default)
  2) python/2.7.3(default)   4) share-rpms65
```

Switch Python modules:

```
[username@pegasus ~]$ module switch python/3.3.1
$ module list
Currently Loaded Modulefiles:
  1) perl/5.18.1          3) share-rpms65
  2) gcc/4.4.7(default)   4) python/3.3.1
```

Installing Python Modules with Package Managers

Install using PIP with `--user`:

```
[username@pegasus ~]$ pip install --user munkres
or install a specific version:
[username@pegasus ~]$ pip install --user munkres==1.0.7
```

Install using easy_install with `--user`:

```
[username@pegasus ~]$ easy_install --user munkres
```

Installing Downloaded Python Modules

Install using PIP with `--user`:

```
[username@pegasus ~]$ pip install --user https://pypi.python.org/packages/source/m/
↪munkres/munkres-1.0.7.tar.gz
Downloading/unpacking https://pypi.python.org/packages/source/m/munkres/munkres-1.0.7.
↪tar.gz
  Downloading munkres-1.0.7.tar.gz
  Running setup.py egg_info for package from https://pypi.python.org/packages/source/
↪m/munkres/munkres-1.0.7.tar.gz

Cleaning up...
```

Install using setup.py with `--user`:

```
[username@pegasus ~]$ wget https://pypi.python.org/packages/source/m/munkres/munkres-
↪1.0.7.tar.gz --no-check-certificate
[username@pegasus ~]$ tar xvzf munkres-1.0.7.tar.gz
[username@pegasus ~]$ cd munkres-1.0.7
[username@pegasus munkres-1.0.7]$ python setup.py --user install
```

Checking Module Versions

Launch Python and confirm module installation:

```
[username@pegasus ~]$ python
...
>>> import munkres
>>> print munkres.__version__
1.0.7
>>> CTRL-D (to exit Python)
```

Python Virtual Environments on Pegasus

Users can create their own Python virtual environments to maintain different module versions for different projects. `Virtualenv` is available on Pegasus for Python 2.7.3. By default, `virtualenv` does not include packages that are installed globally. To give a virtual environment access to the global site packages, use the `--system-site-packages` flag.

Creating Virtual Environments

These example directories do not need to be named exactly as shown.

Create a project folder, cd to the new folder (optional), and create a ‘‘virtualenv‘‘:

```
[username@pegasus ~]$ mkdir ~/python2
[username@pegasus ~]$ cd ~/python2
[username@pegasus python2]$ virtualenv ~/python2/test1
PYTHONHOME is set.  You *must* activate the virtualenv before using it
New python executable in test1/bin/python
Installing setuptools, pip...done.
```

Create a ‘‘virtualenv‘‘ with access to global packages:

```
[username@pegasus python2]$ virtualenv --system-site-packages test2
```

Activating Virtual Environments

Activate the virtual environment with the `source` command and relative or absolute `path/to/env`/bin/activate‘. The environment name will precede the prompt.

```
[username@pegasus ~]$ source ~/python2/test1/bin/activate
(test1)[username@pegasus ~]$ which python
~/python2/test1/bin/python
```

Installing Python modules in Virtual Environments

Once the virtual environment is active, install Python modules normally with PIP, easy_install, or setup.py. Any package installed normally will be placed into that virtual environment folder and isolated from the global Python installation. Note that using `--user` or `--prefix=...` flags during module installation will place modules in those specified directories, **NOT** your currently active Python virtual environment.

```
(test1)[username@pegasus ~]$ pip install munkres
```

Deactivating Virtual Environments

```
(test1)[username@pegasus ~]$ deactivate
[username@pegasus ~]$
```

Comparing two Python Virtual Environments

PIP can be used to save a list of all packages and versions in the current environment (use `freeze`). Compare using `sdiff` to see which packages are different.

List the current environment, deactivate, then list the global Python environment:

```
(test1)[username@pegasus ~]$ pip freeze > test1.txt
(test1)[username@pegasus ~]$ deactivate
[username@pegasus ~]$ pip freeze > p2.txt
```

Compare the two outputs using ‘‘sdiff‘‘:

```
[username@pegasus ~]$ sdiff p2.txt test1.txt
...
matplotlib==1.2.1                                               <
misopy==0.5.0                                                   | munkres==1.0.7
...
[username@pegasus ~]$
```

As seen above, the test1 environment has `munkres` installed (and no other global Python packages).

Recreating Python Virtual Environments

To recreate a Python virtual environment, use the `r` flag and the the saved list:

```
(test2)[username@pegasus ~]$ pip install -r test1.txt
Installing collected packages: munkres
  Running setup.py install for munkres
...
Successfully installed munkres
Cleaning up...
(test2)[username@pegasus ~]$
```

Python virtual environment wrapper

Users can install `virtualenvwrapper` in their own home directories to facilitate working with Python virtual environments. Once installed and configured, `virtualenvwrapper` can be used to create new virtual environments and to switch between your virtual environments (switching will deactivate the current environment). `Virtualenvwrapper` reads existing environments located in the `WORKON_HOME` directory.

Install a local copy of `virtualenv` with `--user`:

Recall that `--user` installs Python 2.7 modules in `~/.local/lib/python2.7/site-packages` To specify a different location, use `--prefix=$HOME/local/python2mods` (or another path).

```
[username@pegasus ~]$ pip install --user virtualenvwrapper
or
[username@pegasus ~]$ easy_install --user --always-copy virtualenvwrapper
```

Set virtual environment home directory and source:

`WORKON_HOME` should be the parent directory of your existing Python virtual environments (or another directory of your choosing). New Python virtual environments created with `virtualenv` will be stored according to this path. Set source to `virtualenvwrapper.sh` in the same location specified during installation.

```
[username@pegasus ~]$ export WORKON_HOME=$HOME/python2
[username@pegasus ~]$ source ~/.local/bin/virtualenvwrapper.sh
```

Create a virtual environment using `virtualenvwrapper`:

This will also activate the newly-created virtual environment.

```
[username@pegasus ~]$ mkvirtualenv test3
PYTHONHOME is set.  You *must* activate the virtualenv before using it
New python executable in test3/bin/python
Installing setuptools, pip...done.
(test3)[username@pegasus ~]$
```

Activate or switch to a virtual environment:

```
(test3)[username@pegasus ~]$ workon test1
(test1)[username@pegasus ~]$
```

Deactivate the virtual environment:

```
(test1)[username@pegasus ~]$ deactivate
[username@pegasus ~]$
```

R on Pegasus

To run the R programming language on Pegasus, first load an R module with `module load R/version`. Use the command `module avail` to see a list of available software, including R versions. For more information about Pegasus software, see *Software on the Pegasus Cluster*.

Batch R

To run a batch R file on Pegasus compute nodes, submit the file to LSF with `R CMD BATCH filename.R`. Remember to include the `-P` flag and your project, if you have one.

```
[username@pegasus ~]$ module load R/2.15.2
[username@pegasus ~]$ bsub -q general -P projectID R CMD BATCH example1.R
Job is submitted to <projectID> project.
Job <6101046> is submitted to queue <general>.
```

See `example1.R` below. For more information about `bsub` and scheduling jobs, see *Scheduling Jobs on the Pegasus Cluster*.

Batch jobs can also be submitted to LSF with script files.

```
example.job
#!/bin/bash
#BSUB -J R_job             # name of your job
#BSUB -R "span[hosts=1]"   # request run script on one node
#BSUB -q general           # request run on general queue
#BSUB -n 1                 # request 1 core
#BSUB -W 2                 # request 2 minutes of runtime
#BSUB -P projectID         # your projectID
R CMD BATCH example1.R     # R command and your batch R file



[username@pegasus ~]$ bsub < example.job
```

Interactive R

Load an R module, then submit an interactive R session with `bsub -Is`. Specify your project with the `-P` flag.

```
[username@pegasus ~]$ module load R/2.15.2
[username@pegasus ~]$ bsub -Is -q interactive R
or
[username@pegasus ~]$ bsub -Is -P hpc R
```

R packages

Confirm that your R package's pre-requisites are met, either in your local environment or as a module on the cluster. You will need to load any cluster modules that are pre-requisites, and install locally any other pre-requisites. See *Pegasus Cluster Software Installation* for help with complex requirements.

From the R prompt, install any R package to your personal R library with the standard `install.package()` R command. Choose `y` when asked about using a personal library, and `y` again if asked to create one.

```
>  install.packages("doParallel", repos="http://R-Forge.R-project.org")
Warning in install.packages("doParallel", repos = "http://R-Forge.R-project.org") :
  'lib = "/share/opt/R/3.1.2/lib64/R/library"' is not writable
Would you like to use a personal library instead?  (y/n) y
Would you like to create a personal library
~/R/x86_64-unknown-linux-gnu-library/3.1
to install packages into?  (y/n) y
...
```

Contact IDSC ACS to review any core library pre-requisites & dependencies, for cluster-wide installation.

R file examples

`example1.R`

```
 # create graphical output file
pdf("example1.pdf")

# Define two vectors v1 and v2
v1 <- c(1, 4, 7, 8, 10, 12)
v2 <- c(2, 8, 9, 10, 11, 15)

# Creat some graphs
hist(v1)
hist(v2)
pie(v1)
barplot(v2)

# close the file
dev.off()
```

If you have forwarded your display, open the pdf from the Pegasus prompt with evince.

```
[username@pegasus ~]$ evince example1.pdf
```

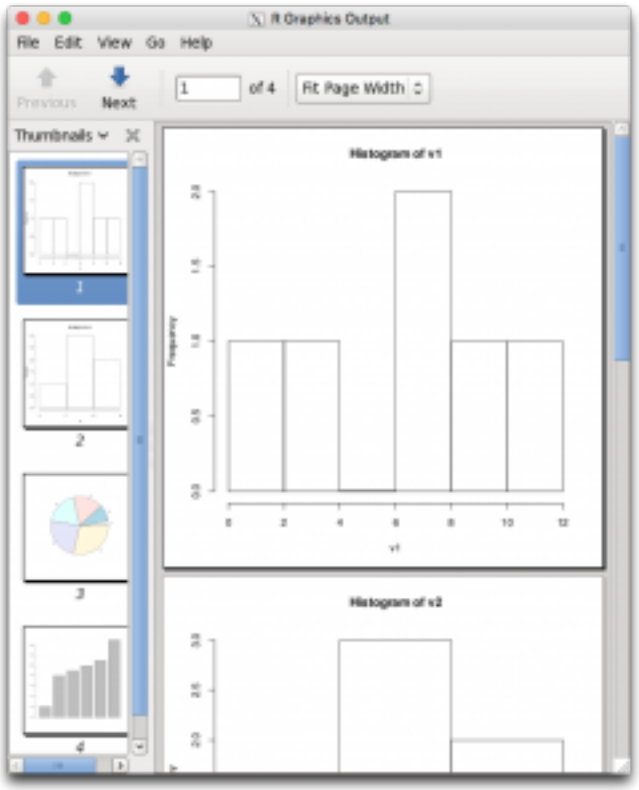


Fig. 4: example1.png

SAS on Pegasus

SAS can be run on Pegasus in Non-interactive/Batch and Interactive/Graphical modes.

Non-Interactive Batch Mode

In batch mode, SAS jobs should be submitted via LSF using the `bsub` command. A sample LSF script file named `scriptfile` to submit SAS jobs on Pegasus may include the following lines:

`scriptfile`

```
#BSUB -J jobname
#BSUB -o jobname.o%J
#BSUB -e jobname.e%J
sas test.sas
```

where "test.sas" is an SAS program file.

Type the following command to submit the job:

```
[username@pegasus ~]$ bsub < scriptfile
```

For general information about how to submit jobs via LSF, see *Scheduling Jobs on Pegasus*.

Interactive Graphical Mode

To run SAS interactively, first *forward the display*. Load the SAS module and use the interactive queue to launch the application.

```
[username@pegasus ~]$ module load sas
[username@pegasus ~]$ module load java
```

Submit job to the interactive queue:

```
[username@pegasus ~]$ bsub -q interactive -P myproject -Is -XF sas
Job is submitted to project.
Job is submitted to queue .
```

Notice the `-P` flag in the above `bsub` command. If you do not specify your project, you will receive an error like the one below:

```
[username@pegasus ~]$ bsub -q interactive -Is -XF sas
Error: Your account has multiple projects: project1 project2.
Please specify a project by -P option and resubmit
Request aborted by esub. Job not submitted.
```

Using R through Anaconda

If you find that the current R modules on Pegasus do not support dependencies for your needed R packages, an alternative option is to install them via an Anaconda environment. Anaconda is an open source distribution that aims to simplify package management and deployment. It includes numerous data science packages including that of R.

Anaconda Installation

First you will need to download and install Anaconda in your home directory.

```
[username@pegasus ~]$ wget https://repo.anaconda.com/archive/Anaconda3-2021.05-Linux-
↪x86_64.sh
```

Unpack and install the downloaded Anaconda bash script

```
[username@pegasus ~]$ bash Anaconda3-2021.05-Linux-x86_64.sh
```

Configuring Anaconda environment

Activate conda with the new Anaconda3 folder in your home directory (Depending on your download this folder might also be named 'ENTER')

```
[username@pegasus ~]$ source <path to conda>/bin/activate
[username@pegasus ~]$ conda init
```

Configure & prioritize the conda-forge channel. This will be useful for downloading library dependencies for your R packages in your conda environment.

```
[username@pegasus ~]$ conda config --add channels conda-forge
[username@pegasus ~]$ conda config --set channel_priority strict
```

Create a conda environment that contains R

```
[username@pegasus ~]$ conda create -n r4_MyEnv r-base=4.1.0 r-essentials=4.1
```

Activate your new conda environment

```
[username@pegasus ~]$ conda activate r4_MyEnv
(r4_MyEnv) [username@pegasus ~]$
```

Note: the syntax to the left of your command line (r4_MyEnv) will indicate which conda environment is currently active, in this case the R conda environment you just created.

Common R package dependencies

Some R packages like 'tidycensus', 'sqldf', and 'kableExtra' require additional library dependencies in order to install properly. To install library dependencies you may need for your R packages, you can use the following command:

```
(r4_MyEnv) [username@pegasus ~]$ conda install -c conda-forge <library_name>
```

To check if a library depenency is availabe through the conda-forge channel, use the following link: https://anaconda.org/conda-forge

Below is an example of installing library dependencies needed for 'tidycensus', then the R package itself.

```
(r4_MyEnv) [username@pegasus ~]$ conda install -c conda-forge udunits2
(r4_MyEnv) [username@pegasus ~]$ conda install -c conda-forge gdal
(r4_MyEnv) [username@pegasus ~]$ conda install -c conda-forge r-rgdal
(r4_MyEnv) [username@pegasus ~]$ R
> install.packages('tidycensus')
```

Activating conda environment upon login

Whenever you login, you will need to re-activate your conda environment to re-enter it. To avoid this, you can edit your .bashrc file in your home directory

```
[username@pegasus ~]$ vi ~/.bashrc
```

Place the following lines in the .bashrc file:

conda activate r4_MyEnv

Then ':wq!' to write, quite and save the file. Upon logging in again your R conda environment will automatically be active.

If you would like to deactivate your conda environment at any time, use the following command:

```
(r4_MyEnv) [username@pegasus ~]$ conda deactivate r4_MyEnv
```

To obtain a list of your conda environments, use the following command:

```
[username@pegasus ~]$ conda env list
```

Running jobs

In order to properly run a job using R within a conda environment you will need to intiate & activate the conda environment within the job script, otherwise the job may fail to find your version of R. Please see the example job script below:

```
#!/bin/bash
#BSUB -J jobName
#BSUB -P projectName
#BSUB -o jobName.%J.out
#BSUB -e jobName.%J.err
#BSUB -W 1:00
#BSUB -q general
#BSUB -n 1
#BSUB -u youremail@miami.edu

. "/nethome/caneid/anaconda3/etc/profile.d/conda.sh"
conda activate r4_MyEnv

cd /path/to/your/R_file.R

R CMD BATCH R_file.R
```

Note: Sometimes you may need to use the 'Rscript' command instead of 'R CMD BATCH' to run your R file within the job script.

4.2.4 Pegasus FAQs - Frequently Asked Questions

Detailed information for FAQ topics is available here and in *IDSC ACS Policies*

If you are new to Pegasus and HPC clusters, review this documentation on the Pegasus system, the job scheduler, and modularized software.

Note: IDSC ACS does not install, provide support for, or provide documentation on how to code in your preferred software. ACS documentation contains information on using software in a Linux cluster environment.

Pegasus Projects

Projects on Pegasus

How do I join a project?

Contact the project owner.

How do I request a new project?

Any PI or faculty member may request a new project : https://idsc.miami.edu/project_request

When will my project be created?

When the allocations committee has reviewed and approved it.

Scratch requests over 2TB can take a month for the allocations committee to review as availability is limited.

How can I manage my Projects and Groups?

Contact IDSC ACS at hpc@ccs.miami.edu

Pegasus Software

Software on Pegasus

What software is available?

Software Modules from the command line: `$ module avail`

How do I view my currently loaded modules?

```
$ module list
```

How do I use software modules?

Software on Pegasus

May I install software?

Yes! Pegasus users are free to compile and install software in their respective home directories by following the software's source code or local installation instructions. See our *Software Installation* guide for more information.

Note: IDSC ACS does not install user software. For global installations on Pegasus, submit a Software Request to hpc@ccs.miami.edu

---

How do I request global software installation on Pegasus?

Submit your request to hpc@ccs.miami.edu

We only globally install software when we receive multiple requests for the software.

When will my global software request be approved/installed?

When a minimum of 20 users require it, software requests will be approved. Software requests are reviewed and installed quarterly.

How can I increase Java memory on Pegasus?

Load the java module, then change the value of _JAVA_OPTIONS.

```
[username@pegasus ~]$ module load java
[username@pegasus ~]$ echo $_JAVA_OPTIONS
-Xmx512m
[username@pegasus ~]$ export _JAVA_OPTIONS="-Xmx4g"
```

Pegasus Job Scheduling

Scheduling Jobs

May I run resource-intensive jobs on Pegasus login nodes?

No. Resource-intensive jobs must be submitted to LSF.

How do I submit jobs to Pegasus?

With `bsub` *command* : *LSF*

How do I check on my submitted jobs?

With `bjobs` *command* : *LSF*

How do I monitor job progress?

With `bpeek` *command* : *LSF*

Is there a limit on how many jobs I can run?

No. Users are limited by number of simultaneous CPUs used. Individual users can run on up to 512 CPUs at a time, projects on up to 1000 CPUs at a time.

How can I see pending and running job counts for Pegasus queues?

With `bqueues` *command* : *LSF*

Why is my job still pending?

Jobs wait for enough resources to satisfy requirements. When the cluster is under heavy user load, jobs will wait longer. Use `$ bjobs -l jobID` to see PENDING REASONS. Check your resource requirements for accuracy and feasibility.

The Pegasus job scheduler operates under Fairshare scheduling. Fairshare scheduling divides the processing power of the cluster among users and queues to provide fair access to resources, so that no user or queue can monopolize the resources of the cluster and no queue will be starved.

If your job has been pending for more than 24 hours *and is not requesting exclusive access or all cores on a node*, you may e-mail hpc@ccs.miami.edu for assistance.

Are other users' pending jobs slowing my job?

No. The number of pending jobs is irrelevant to job performance in LSF. The scheduler can handle hundreds of thousands of jobs.

How do I submit jobs to my Project?

With the `-P` flag : *LSF jobs*

How do I submit an interactive job?

With the `-Is -q interactive` flags : *LSF interactive jobs*

How do I submit an interactive X11 job?

With the `-Is -q interactive -XF` flags : *LSF interactive jobs*

Why was my job killed?

Jobs are killed to protect the cluster and preserve system performance.

Common reasons include:

- running on a login node
- using more memory than reserved
- using all the memory on a compute node
- using more CPUs than reserved
- needing more time to complete than reserved
- using more `/tmp` space than available on compute nodes

See *LSF* for assistance with appropriate resource reservations and *Pegasus Queues* for default wall times.

What about jobs in UNKWN state?

Re-queue your job in LSF :

```
$ bkill -r jobID
$ bkill -r jobID (a second time)
$ brequeue -e jobID
```

4.3 Linux Guides

4.3.1 Introduction to Linux on Pegasus

Pegasus is currently running the CentOS 7.6 operating system, a distribution of Linux. Linux is a UNIX-like kernel, though in this document it will generally refer to the entire CentOS distribution. The three basic components of UNIX-like operating systems are the **kernel**, **shell**, and **system programs**. The kernel handles resource management and program execution. The shell interprets user commands typed at a prompt. System programs implement most operating system functionalities such as user environments and schedulers.

Everything in Linux is either a **file** (a collection of data) or a **process** (an executing program). Directories in Linux are types of files.

In the below examples, `username` represents your access account.

Navigating the Linux Shell

The shell is command-line interface (CLI), a type of user interface that interprets **commands** typed at a prompt. The default shell on Pegasus is Bash.

Users send commands to the shell, which runs them and outputs results. Commands can include options (or **flags**) to modify output and **arguments** to specify command targets. In Bash, command history can be accessed from the prompt with up and down arrow keys - use this to repeat a previously issued command.

Below are some useful Linux shell commands with explanations and examples. Recall that Linux is *case-sensitive*– "name" is distinct from "NAME" and other capital and lower-case combinations ("Name", "nAMe", etc.). To cancel a process and return to the prompt in Linux, press `CTRL-C` in Windows or `Command-C` in Mac.

View your current shell with `echo`:

The `echo` command displays a line of text. In Linux, `$` denotes a **variable**. The `$SHELL` environment variable contains your current shell. To view the contents of this variable, send it to the `echo` command. Variables will be interpreted by the command even when inside lines of text, as shown below.

```
[username@pegasus ~]$ echo $SHELL
/bin/bash
[username@pegasus ~]$ echo "My shell is $SHELL"
My shell is /bin/bash
```

View all environment variables with `env`:

To view all your environment variables, use the `env` command. To view this list alphanumerically, use `env | sort`.

```
[username@pegasus ~]$ env
MODULE_VERSION_STACK=3.2.10
LC_PAPER=en_US.utf8
HOSTNAME=login4
SHELL=/bin/bash
...
[username@pegasus ~]$ env | sort
...
```

View your current directory with `pwd`:

Directories in Linux are similar to folders in other operating systems. Your home directory is the default location after login. The shortcut for home in Bash is the tilde (`~`), shown below just before the prompt (`$`). `pwd` outputs the **absolute path**, the unique location starting from the topmost, or root, directory (`/`).

```
[username@pegasus ~]$ pwd
/nethome/username
```

View the contents of a directory with `ls`:

Entering the `ls` command without arguments (as shown below) lists the contents of the current directory. *If this is your first connection to Pegasus, your home directory may be empty.* Directories can be distinguished from files by the leading `d` in file permissions.

```
[username@pegasus ~]$ ls
example_file1  example_file2  testdir1
```

To view the contents of a specific directory, send the path as an argument to `ls`. In this example the current directory is home, which contains `testdir1`. As shown in the output, `testdir1` contains one file: `testdir1_file1`

```
[username@pegasus ~]$ ls testdir1
testdir1_file1
```

Note that you can press the TAB key on your keyboard to auto-complete names. If there are multiple matches, a list of options will be shown. Type the next letter and press TAB again until tab-complete finishes.

Command details and flag information can be found in the **Linux manual pages**, accessible via the command line:

```
[username@pegasus ~]$ man topic or command
```

Press SPACE to see the next set of lines. To scroll, use the arrow keys or Page Up and Page Down. To exit, type q.

ls can be run with options, or flags, to customise output. For example, view more detailed information such as file permissions using the -lh flags.

```
[username@pegasus ~]$ ls -lh
total 0
-rw-r--r-- 1 username ccsuser  54  example_file1
-rw-r--r-- 1 username ccsuser 476  example_file2
drwxr-xr-x 2 username ccsuser 512  testdir1
...
```

The flags on this ls -lh command:

- -l long list format (includes permissions, owner, and more)
- -h human readable filesize format (useful for larger file sizes)

Other useful ls flags:

- -a include hidden files *
- -d list properties of a directory itself, not the contents
- -1 (**number 1**) one result per line
- -R recursively list subdirectory contents
- -S sort by file size
- -X sort alphanumerically by extension
- -m comma-separated list

* Hidden files include the **.** and **..** directories, which represent the current and parent (respectively). These can be used as shortcuts in relative paths:

```
[username@pegasus testdir1]$ ls -a
.  ..  testdir1_file1
[username@pegasus testdir1]$ ls ..
example_file1  example_file2  testdir1
```

Navigate to directories with cd:

This command changes your current directory to the path specified, which can be **absolute**, starting with /, or **relative**, starting from the current directory.

```
[username@pegasus ~]$ cd testdir1
[username@pegasus testdir1]$
```

Some useful cd commands:

- cd or cd ~ move to user's home directory

- `cd ..` move to parent directory
- `cd -` move to previous working directory

View directory contents with `tree`:

Pegasus has the `tree` package installed, which recursively outputs a depth-indented list of contents. This may be more helpful than `ls` for nested directories.

```
[username@pegasus ~]$ tree -vC
.
|-- example_file1
|-- example_file2
|-- testdir1
    `-- testdir1_file1

1 directory, 3 files
```

The flags on this `tree -vC` command:

- `-v` sort alphanumerically by type
- `-C` colorise output

Other useful `tree` flags:

- `-a` include hidden files
- `-d` list directories only
- `-r` sort reverse alphanumerically
- `-L number` descend only *number* levels deep

Check command availability and location with `which`:

The `which` command returns the full path of any shell commands registered in the current environment by searching locations in the `$PATH` environment variable. Use `which` to check command and software availability and location.

```
[username@pegasus ~]$ which bash
/bin/bash
[username@pegasus ~]$ which vim
/usr/bin/vim
[username@pegasus ~]$ which python
/share/opt/python/2.7.3/bin/python
```

Interacting with Files

Make directories with `mkdir`:

This command creates new, empty directories.

```
[username@pegasus ~]$ mkdir testdir2
[username@pegasus ~]$ ls
example_file1  example_file2  testdir1  testdir2
```

Multiple directories can be created at the same time, as can directory hierarchies:

```
[username@pegasus ~]$ mkdir firstdir seconddir
[username@pegasus ~]$ ls
example_file1  example_file2  firstdir  secondir  testdir1  testdir2

[username@pegasus ~]$ mkdir -pv level1/level2/level3
mkdir: created directory `level1'
mkdir: created directory `level1/level2'
mkdir: created directory `level1/level2/level3'
[username@pegasus ~]$ ls
example_file1  example_file2  firstdir  level1  seconddir  testdir1  testdir2
[username@pegasus ~]$ ls level1
level2
```

The flags on this `mkdir -pv` command:

- `-p` make parent directories as needed
- `-v` print a message for each created directory

If a directory already exists, `mkdir` will output an error message:

```
[username@pegasus ~]$ mkdir testdir1
mkdir: cannot create directory `testdir1': File exists
```

Remove directories with `rmdir`:

Directories must be empty for `rmdir` to remove them.

```
[username@pegasus ~]$ rmdir firstdir seconddir
[username@pegasus ~]$ ls
example_file1  example_file2  level1  testdir1  testdir2

[username@pegasus ~]$ rmdir testdir1 level1
rmdir: failed to remove `testdir1': Directory not empty
rmdir: failed to remove `level1': Directory not empty
[username@pegasus ~]$ ls testdir1 level1
level1:
level2

testdir1:
testdir1_file1
```

The individual directories in the above example are empty. The top level of the hierarchy in the above example is not empty, neither is `testdir1`. To remove directories that are not empty, see `rm`.

Remove files and directories with `rm`:

There is no 'recycle bin' on Pegasus. Removing files with `rm` is permanent and cannot be undone.

```
[username@pegasus ~]$ rm -v example_file3
removed `example_file3'
[username@pegasus ~]$ ls
example_file1  example_file2  level1  testdir1  testdir2
```

The flag on this `rm -v` command:

- `-v` print a message for each removed file or directory

Because directories are types of files in Linux, `rm` can be used with the recursive flag to remove directories. Recall that `rm` in Linux is ***permanent and cannot be undone***. Without the recursive flag, `rm` on a directory will produce an error as shown below.

```
[username@pegasus ~]$ rm level1
rm: cannot remove `level1': Is a directory
[username@pegasus ~]$ rm -rv level1
removed directory: `level1/level2/level3'
removed directory: `level1/level2'
removed directory: `level1'
```

The flags on this `rm -rv` command:

- `-r` remove directories and their contents recursively
- `-v` print a message for each removed file or directory

View file contents with `cat`:

`cat` reads file contents into standard output, typically the display. This is best used for small text files.

```
[username@pegasus ~]$ cat example_file1
This is example_file1.
It contains two lines of text.
[username@pegasus ~]$ cat -nE example_file1
     1    This is example_file1.$
     2  It contains two lines of text.$
```

Flags used in this command for `cat`:

- `-n` number all output lines
- `-E` display $ at the end of each line

Other useful flags:

- `-b` number non-empty output lines

When **no file is given**, `cat` reads standard input (typically from the keyboard) then outputs contents (typically the display). Press `CTRL-D` (Windows) or `Command-D` (Mac) to return to the prompt.

```
[username@pegasus ~]$ cat
No file was given- cat reads standard input from the keyboard and will output this to␣
↪the display.
No file was given- cat reads standard input from the keyboard and will output this to␣
↪the display.
CTRL-D or Command-D
[username@pegasus ~]$
```

This feature can be used to create files.

Create files with `cat` and redirection:

Redirection operators in Linux send output from one source as input to another. > redirects standard output (typically the display) to a file. Combine `cat` with > to create a new file and add content immediately.

```
[username@pegasus ~]$ cat > example_file3
This is example_file3.
These lines are typed directly into the file.
Press CTRL-D (Windows) or Command-D (Mac) to return to the prompt.
CTRL-D or Command-D
[username@pegasus ~]$ cat example_file3
This is example_file3.
These lines are typed directly into the file.
Press CTRL-D (Windows) or Command-D (Mac) to return to the prompt.
```

Note that the > operator *overwrites* file contents. To *append*, use the append operator: >>

```
[username@pegasus ~]$ cat >> example_file3
This is an appended line.
CTRL-D or Command-D
[username@pegasus ~]$ cat example_file3
This is example_file3.
These lines are typed directly into the file.
Press CTRL-D (Windows) or Command-D (Mac) to return to the prompt.
This is an appended line.
```

Linux output redirection operators:

- > overwrite standard output a file
- >> append standard output to a file

View file contents with `head` and `tail`:

For longer text files, use `head` and `tail` to restrict output. By default, both output 10 lines - `head` the first 10, `tail` the last 10. This can be modified with numerical flags.

```
[username@pegasus ~]$ head example_file2
This is example_file2.  It contains 20 lines.
This is the 2nd line.
This is the 3rd line.
This is the 4th line.
This is the 5th line.
This is the 6th line.
This is the 7th line.
This is the 8th line.
This is the 9th line.
This is the 10th line.
[username@pegasus ~]$ head -3 example_file2
This is example_file2.  It contains 20 lines.
This is the 2nd line.
This is the 3rd line.

[username@pegasus ~]$ tail -4 example_file2
This is the 17th line.
This is the 18th line.
This is the 19th line.
This is the 20th line, also the last.
```

Rename and Move with `mv`:

Moving and renaming in Linux uses the same command, thus files can be renamed as they are moved. In this example, the file `example_file1` is first renamed using `mv` and then moved to a subdirectory (without renaming).

```
[username@pegasus ~]$ mv example_file1 example_file0
[username@pegasus ~]$ ls
example_file0  example_file2  testdir1  testdir2
[username@pegasus ~]$ mv example_file0 testdir1/
[username@pegasus ~]$ ls testdir1
example_file0  testdir1_file1
```

In this example, the file `example_file0` is moved and renamed at the same time.

```
[username@pegasus ~]$ mv -vn testdir1/example_file0 example_file1
`testdir1/example_file0' -> `example_file1'
[username@pegasus ~]$ ls
example_file1  example_file2  testdir1  testdir2
```

The flags on this `mv -vn` command:

- `-v` explain what is being done
- `-n` do not overwrite and existing file

Note that when `mv` is used with directories, it is recursive by default.

```
[username@pegasus ~]$ mv -v testdir1 testdir2/testdir1
`testdir1' -> `testdir2/testdir1'
[username@pegasus ~]$ ls -R testdir2
testdir2:
testdir1

testdir2/testdir1:
testdir1_file1
```

The file inside `tesdir1` moved along with the directory.

Copy with `cp`:

File and directory copies can be renamed as they are copied. In this example, `example_file1` is copied to `example_file0`.

```
[username@pegasus ~]$ cp example_file1 example_file0
[username@pegasus ~]$ cat example_file0
This is example_file1.
It contains two lines of text.
```

The contents of the copied file are the same as the original.

`cp` is not recursive by default. To copy directories, use the recursive flag `-R`.

```
[username@pegasus ~]$ cp -Rv testdir2 testdir2copy
`testdir2' -> `testdir2copy'
`testdir2/testdir1' -> `testdir2copy/testdir1'
`testdir2/testdir1/testdir1_file1' -> `testdir2copy/testdir1/testdir1_file1'
[username@pegasus ~]$ ls
example_file0  example_file1  example_file2  testdir2  testdir2copy
```

The flags on this `cp -Rv` command:

- `-R` copy directories recursively
- `-v` for verbose, explain what is being done

Other useful flags:

- `-u` (update) copy only when source is newer, or destination is missing
- `-n` do not overwrite an existing file
- `-p` preserve attributes (mode, ownership, and timestamps)

Edit files : `nano`, `emacs`, `vi`:

`nano` and `emacs` are simple text editors available on the cluster and most Linux systems, while `vi` is a modal text editor with a bit of a learning curve.

For a quick comparison of these text editors, see : https://www.linuxtrainingacademy.com/nano-emacs-vim/

`vi` can be launched with the command `vi` (plain) or `vim` (syntax-highlighted based on file extension). `vi` has two main modes: **Insert** and **Command**.

- **Command mode:** searching, navigating, saving, exiting, etc.
- **Insert mode:** inserting text, pasting from clipboard, etc.

`vi` launches in Command mode by default. To enter Insert mode, type `i` on the keyboard. Return to Command mode by pressing `ESC` on the keyboard. To exit and save changes, type `:x` (exit with save) or `:wq` (write and quit) on the keyboard while in Command mode (from Insert mode, type `ESC` before each sequence).

In the example below, the arrow keys are used to navigate to the end of the first line. `i` is pressed to enter Insert mode and the file name on line 1 is changed. Then `ESC:x` is entered to change to Command mode and exit saving changes.

```
[username@pegasus ~]$ vi example_file0
...
This is example_file0.
It contains two lines of text.
~
~
~
~
"example_file0" 2L, 54C
:x
[username@pegasus ~]$ cat example_file0
This is example_file0.
It contains two lines of text.
```

Some `vi` tutorials, commands, and comparisons :

- https://www.ccsf.edu/Pub/Fac/vi.html
- http://www.cs.colostate.edu/helpdocs/vi.html
- https://www.linuxtrainingacademy.com/nano-emacs-vim/

View file contents by page with `more` and `less`:

Pager applications provide scroll and search functionalities, useful for larger files. Sets of lines are shown based on terminal height. In both `more` and `less`, `SPACE` shows the next set of lines and `q` quits. `more` cannot scroll

backwards. In `less`, navigate with the arrow keys or `Page Up` and `Page Down`, and search with `?` (similar to `man` pages).

```
[username@pegasus testdir1]$ less testdir1_file1
...
This is tesdir1_file1.  It contains 42 lines.
02
03
04
05
06
07
: SPACE or Page Down
36
37
38
39
40
41
42
(END)  q
[username@pegasus testdir1]$
```

File Permissions

File permissions control which users can do what with which files on a Linux system. Files have three distinct permission sets: one for the **user** who owns the file (u), one for the associated **group** (g), and one for all **other** system users (o). Recall that directories are types of files in Linux.

Note: As policy, IDSC does not alter user files on our systems.

To view file permissions, list directory contents in long listing format with `ls -l`. To check directory permissions, add the `-d` flag:`ls -ld`. Paths can be relative or absolute.

```
[username@pegasus ~]$ ls -l /path/to/directory/or/file
...
[username@pegasus ~]$ ls -ld /path/to/directory
...
```

Understanding File Permission Categories

Permissions are defined by three **categories**:

```
u : user (owner)
g : group
o : other
```

Each category has three **permission types**, which are either on or off:

```
r : read
w : write
x : execute
```

For a directory, `x` means users have permission to search the directory.

File and Directory Permission Examples:

`mydir` contains two files. The owner (u) has read and write (rw) permissions, members of `ccsuser` (g) have read (r) permissions, and all other users (o) have read (r) permissions.

```
[username@pegasus ~]$ ls -l /nethome/username/mydir
total 0
-rw-r--r-- 1 username ccsuser myfile.txt
-rw-r--r-- 1 username ccsuser myfile2.txt
```

For the directory `mydir`, the owner (u) has read, write, and browse (rwx) permissions, members of `ccsuser` have read and browse (rx), and all other users (o) have read only (r).

```
[username@pegasus ~]$ ls -ld /nethome/username/mydir
drwxr-xr-- 2 username ccsuser /nethome/username/mydir
```

Decimal representation

Permissions can also be represented with 3 decimal numbers, corresponding to the decimal representation of each category's binary permissions. Decimal representation can be used when changing file permissions.

`myfile.txt` has the following binary and decimal permissions:

```
-rw- r-- r-- 1 username ccsuser myfile.txt
 110 100 100
   6   4   4

 -  : this file is not a directory
rw- : u - username (owner) can read and write
r-- : g - members of ccsuser can read only
r-- : o - other users can read only
```

`mydir` (a directory) has the following permissions:

```
drwx r-x --x 2 username ccsuser /nethome/username/mydir
 111 101 100
   7   5   4

 d  : this file is a directory
rwx : u - username (owner) can read, write, and execute
r-x : g - members of ccsuser can read and execute
--x : o - other users can execute (search directory)
```

Changing File Permissions in Linux

Use `chmod` to change the access mode of a file or directory. The basic syntax is `chmod options file`.

The 3 **options** are: category, operator, and permission (in order). Options can also be assigned numerically using the decimal value for each category (note that all three decimal values must be present and are assigned in category order - u, g, o). Use the `-R` flag with `chmod` to apply permissions recursively, to all contents of a directory.

Categories for chmod:

```
u : user (who owns the file)
g : group
o : other
a : all categories (u, g, and o shortcut)
```

Operators for chmod:

```
= : assigns (overwrites) permissions
+ : adds permissions
- : subtracts permissions
```

Permissions for chmod:

```
r : read
w : write
x : execute
```

Examples with `chmod`

Assign file owner (u) full permissions (rwx) on `myfile.txt`:

```
[username@pegasus mydir]$ chmod u=rwx myfile.txt
[username@pegasus mydir]$ ls -l myfile.txt
-rwxr--r-- 1 username ccsuser myfile.txt
```

Assign full permissions (7) for file owner, read and write (6) for members of `ccsuser`, and execute only (1) for others:

```
[username@pegasus mydir]$ chmod 761 myfile.txt
[username@pegasus mydir]$ ls -l myfile.txt
-rwx rw- --x 1 username ccsuser myfile.txt
 111 110 001
   7   6   1
```

Add for members of ccsuser (g) full permissions (rwx) on `mydir` and all files under `mydir` (`-R` flag):

```
[username@pegasus ~]$ chmod -R g+rwx mydir
[username@pegasus ~]$ ls -l mydir
total 0
-rw-rwxr-- 1 username ccsuser myfile2.txt
-rwxrwxr-- 1 username ccsuser myfile.txt
[username@pegasus ~]$ ls -ld mydir
drwxrwx--x 2 username ccsuser mydir
```

Remove for members of ccsuser (g) write permission (w) on `mydir` and all files under `mydir` (`-R` flag):

```
[username@pegasus ~]$ chmod -R g-w mydir
[username@pegasus ~]$ ls -l mydir
total 0
-rw-r-xr-- 1 username ccsuser myfile2.txt
-rwxr-xr-- 1 username ccsuser myfile.txt
[username@pegasus ~]$ ls -ld mydir
drwxr-x--x 2 username ccsuser mydir
```

Add for members of `ccsuser` (g) write permission (w) on `mydir`, directory only:

```
[username@pegasus ~]$ chmod g+w mydir
[username@pegasus ~]$ ls -ld mydir
drwxrwx--x 2 username ccsuser mydir
[username@pegasus ~]$ ls -l mydir
total 0
-rw-r-xr-- 1 username ccsuser  myfile2.txt
-rwxr-xr-- 1 username ccsuser  myfile.txt
```

Changing Group Ownership in Linux

Use `chgrp` to change the group ownership of a file or directory. The basic syntax is `chgrp group file`.

The file owner must be a member of the **group**. By default, `chgrp` does not traverse symbolic links.
Use the `-R` flag with `chgrp` to apply the group change recursively, to all contents of a directory.

Examples with `chgrp`

Change the group ownership of `mydir` to `mygroup`, directory only:

```
[username@pegasus ~]$ chgrp mygroup mydir
[username@pegasus ~]$ ls -ld mydir
drwxrwx--x 2 username mygroup mydir
[username@pegasus ~]$ ls -l mydir
total 0
-rw-r-xr-- 1 username ccsuser  myfile2.txt
-rwxr-xr-- 1 username ccsuser  myfile.txt
```

Change the group ownership of `mydir` and all files under `mydir` to `mygroup` (`-R` flag):

```
[username@pegasus ~]$ chgrp -R mygroup mydir
[username@pegasus ~]$ ls -ld mydir
drwxrwx--x 2 username mygroup mydir
[username@pegasus ~]$ ls -l mydir
total 0
-rw-r-xr-- 1 username mygroup  myfile2.txt
-rwxr-xr-- 1 username mygroup  myfile.txt
```

Access Control Lists – ACL

Access Control Lists (ACL) are available on Pegasus and Triton file systems. They allow file owners to grant permissions to specific users and groups. When combining standard Linux permissions and ACL permissions, **effective permissions** are the intersection (or overlap) of the two. `cp` (copy) and `mv` (move/rename) will include any ACLs associated with files and directories.

Getting ACL information

ACL permissions start the same as the standard Linux permissions shown by `ls -l` output.

Get ACL information with `getfacl`:

```
[username@pegasus ~]$ getfacl mydir
# file: mydir
# owner: username
# group: mygroup
user::rwx
group::rw-
other::--x
```

Initial ACL permissions on `mydir` match the standard permissions shown by `ls -ld`:

```
[username@pegasus ~]$ ls -ld mydir
drwxrw---x 2 username mygroup mydir
```

Setting ACL information

Once an ACL has been set for a file or directory, a + symbol will show at the end of standard Linux permissions.

Set ACL with `setfacl -m` (modify):

Set for user `mycollaborator` permissions rwx on `mydir`, directory only:

```
[username@pegasus ~]$ setfacl -m user:mycollaborator:rwx mydir
```

This will set an ACL for only the directory, not any files in the directory.

```
[username@pegasus ~]$ ls -ld mydir
drwxrw---x+ 2 username mygroup mydir
[username@pegasus ~]$ getfacl mydir
# file: mydir
# owner: username
# group: mygroup
user::rwx
user:mycollaborator:rwx
group::rw-
mask::rwx
other::r--
```

Note the + symbol at the end of standard permissions, which indicates an ACL has been set. Also note the line `user:mycollaborator:rwx` in the `getfacl mydir` output.

Files within `mydir` remain unchanged (no ACL has been set). `getfacl` on these files returns standard Linux permissions:

```
[username@pegasus ~]$ ls -l mydir
total 0
-rwxrw-r-- 1 username mygroup myfile2.txt
-rwxrw-r-- 1 username mygroup myfile.txt
[username@pegasus ~]$ getfacl mydir/myfile.txt
# file: mydir/myfile.txt
# owner: username
# group: mygroup
user::rwx
```

(continues on next page)

(continued from previous page)

```
group::rw-
other::r--
```

Set for user `mycollaborator` permissions rwX on `mydir`, recursively (all contents):

```
[username@pegasus ~]$ setfacl -Rm user:mycollaborator:rwX mydir
```

This will set an ACL for the directory and all files in the directory. Permissions for `setfacl`:

- `r` read
- `w` write
- `X` (capital) execute/search only if the file is a directory, or already has execute permission

```
[username@pegasus ~]$ ls -l mydir
total 0
-rwxrw-r--+ 1 username mygroup myfile2.txt
-rwxrw-r--+ 1 username mygroup myfile.txt
```

Note the + symbol after file permissions, indicating an ACL has been set. `getfacl` on these files returns ACL permissions:

```
[username@pegasus ~]$ getfacl mydir/myfile.txt
# file: mydir/myfile.txt
# owner: username
# group: mygroup
user::rwx
user:mycollaborator:rwx
group::rw-
mask::rwx
other::r--
```

Note the line `user:mycollaborator:rwx` for `myfile.txt`.

Recall that when combining standard Linux permissions and ACL permissions, effective permissions are the intersection of the two. If user (u) permissions are changed to rw-, the effective permissions for user:mycollaborator are rw- (the intersection of rwx and rw- is `rw-`).

```
[username@pegasus ~]$ chmod u=rw mydir/myfile.txt
[username@pegasus ~]$ getfacl mydir/myfile.txt
# file: myfile.txt
# owner: username
# group: mygroup
user::rw-
user:mycollaborator:rwx
group::rw-
mask::rwx
other::r--
```

Note the line `user::rw-`, indicating users do not have permission to execute this file.

Removing ACL information

Use `setfacl` to remove ACL permissions with flags `-x` (individual ACL permissions) or `-b` (all ACL rules).

Remove ACL permissions with `setfacl -x`:

This flag can remove all permissions, but does not remove the ACL.

Remove permissions for user `mycollaborator` on `mydir`, directory only:

```
[username@pegasus ~]$ setfacl -x user:mycollaborator mydir
[username@pegasus ~]$ getfacl mydir
# file: mydir
# owner: username
# group: mygroup
user::rwx
group::rw-
mask::rwx
other::--x
[username@pegasus ~]$ ls -ld mydir
drwxrwx--x+ 2 username mygroup mydir
```

Note `user:mycollaborator:rwx` has been removed, but `mask::rwx` remains in the `getfacl` output. In `ls -ld` output, the + symbol remains because the ACL has not been removed.

Remove all ACL rules with `setfacl -b`:

This flag removes the entire ACL, leaving permissions governed only by standard Linux file permissions.

Remove all ACL rules for `mydir`, directory only:

```
[username@pegasus ~]$ setfacl -b mydir
[username@pegasus ~]$ ls -ld mydir
drwxrwx--x 2 username mygroup mydir
[username@pegasus ~]$ getfacl mydir
# file: mydir
# owner: username
# group: mygroup
user::rwx
group::rwx
other::--x
```

Note the + symbol is gone from `ls -ld` output, indicating only standard Linux permissions apply (no ACL). The `mask` line is gone from `getfacl` output.

Remove all ACL rules for `mydir`, recursively (all contents):

```
[username@pegasus ~]$ setfacl -Rb mydir
[username@pegasus ~]$ ls -l mydir
total 0
-rwxrwxr-- 1 username mygroup myfile2.txt
-rwxrwxr-- 1 username mygroup myfile.txt
```

Note the + symbols are gone for the contents of `mydir`, indicating only standard Linux permissions apply (no ACLs).

For more information, reference the manual pages for getfacl and setfacl: `man getfacl` and `man setfacl`.

4.3.2 Linux FAQs

How can I check my shell?

`$ echo $SHELL` or `$ echo $0`

How can I view my environment variables?

`$ env` or `$ env | sort`

How can I check command/software availability and location?

`$ which executable`, for example `$ which vim`

How can I get help with commands/software?

Use the Linux manual pages: `$ man executable`, for example `$ man vim`

4.4 Advanced Computing Systems Services

4.4.1 Connecting to Advanced Computing Systems

Use a secure-shell (SSH) client to connect for secure, encrypted communication. From within UM's secure network (SecureCanes wired connection on campus) or VPN, connect from:

Windows

Connect using a terminal emulator like PuTTY (www.putty.org)

Log into IDSC servers with the appropriate account credentials. Pegasus example:

```
username@pegasus.ccs.miami.edu (optional username @ host)
22 (port)
SSH (connection type)
```

Mac and Linux

Connect with the Terminal program, included with the Operating Systems.

Log into IDSC servers with the approprite acount credentials. Pegasus example:

```
bash-4.1$ ssh username@pegasus.ccs.miami.edu
username@pegasus.ccs.miami.edu's password:
```

or SSH without account credentials to be prompted:

```
bash-4.1$ ssh pegasus.ccs.miami.edu
login as: username
username@pegasus.ccs.miami.edu's password:
```

To use SSH key pairs to authenticate, see the CentOS wiki: http://wiki.centos.org/HowTos/Network/SecuringSSH

Forwarding the display with x11

To use graphical programs over SSH, the graphical display must be forwarded securely. This typically requires running an X Window System server and adding the `-X` option when connecting via SSH.

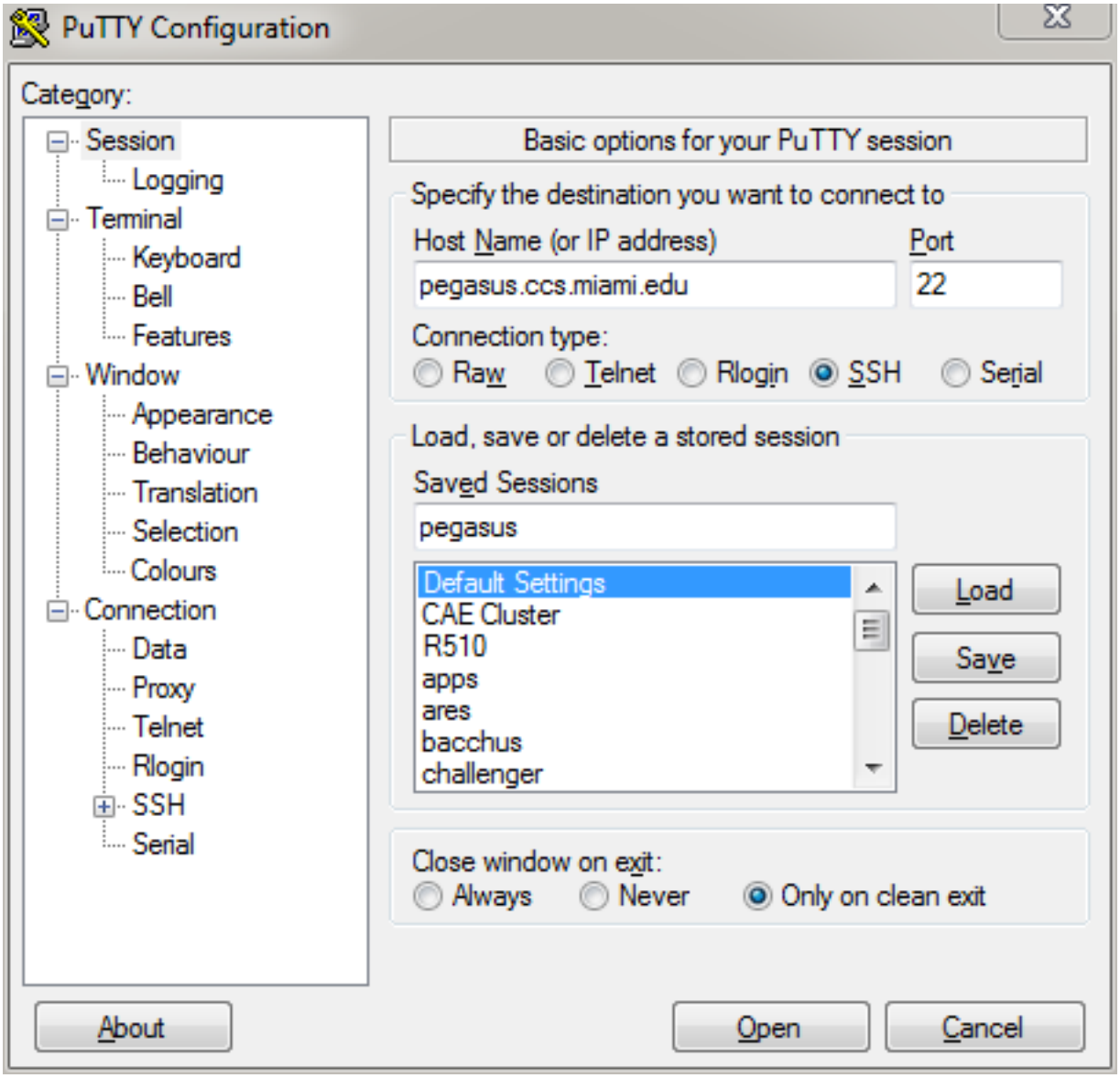


Fig. 5: PuTTY in Windows

Download an X Window System server

- For Windows, Xming with the default installation options : http://sourceforge.net/projects/xming/files/latest/download
- For Mac, XQuartz (OSX 10.8+) : http://www.xquartz.org/

OS X versions 10.5 through 10.7 include X11 and do not require XQuartz.

Connect with X11 forwarding

Launch the appropriate X Window server **before** connecting to IDSC servers via SSH.

Windows: Configure PuTTY for X11 display forwarding

In PuTTY Configuration,

- scroll to the **Connection** category and expand it
- scroll to the **SSH** sub-category and expand it
- click on the **X11** sub-category

On the X11 Options Panel,

- check "Enable X11 forwarding"
- enter "`localhost:0`" in the "X display location" text field

Mac: Connect with X11 flag

Using either the Mac Terminal or the xterm window, connect using the `-X` flag:

```
bash-4.1$ ssh -X username@pegasus.ccs.miami.edu
```

Launch a graphical application

Use `&` after the command to run the application in the background, allowing continued use of the terminal.

```
[username@pegasus ~]$ firefox &
```

Connecting to IDSC Systems from offsite

Triton, Pegasus, and other IDSC resources are only available from within the University's secure campus networks (wired or SecureCanes wireless). To access IDSC resources while offsite, open a VPN connection first. IDSC does not administer VPN accounts.

University of Miami VPN: https://my.it.miami.edu/wda/a-z/virtual-private-network/

Send access range requests (for Vendor VPN applications) to : IDSC ACS

4.4.2 Transferring Files

IDSC systems support multiple file transfer programs such as FileZilla and PSFTP, and common command line utilities such as `scp` and `rsync`. Use cluster head nodes (login nodes) for these types of file transfers. For transferring large amounts of data from systems outside the University of Miami, IDSC ACS also offers a gateway server that supports SFTP and Globus.

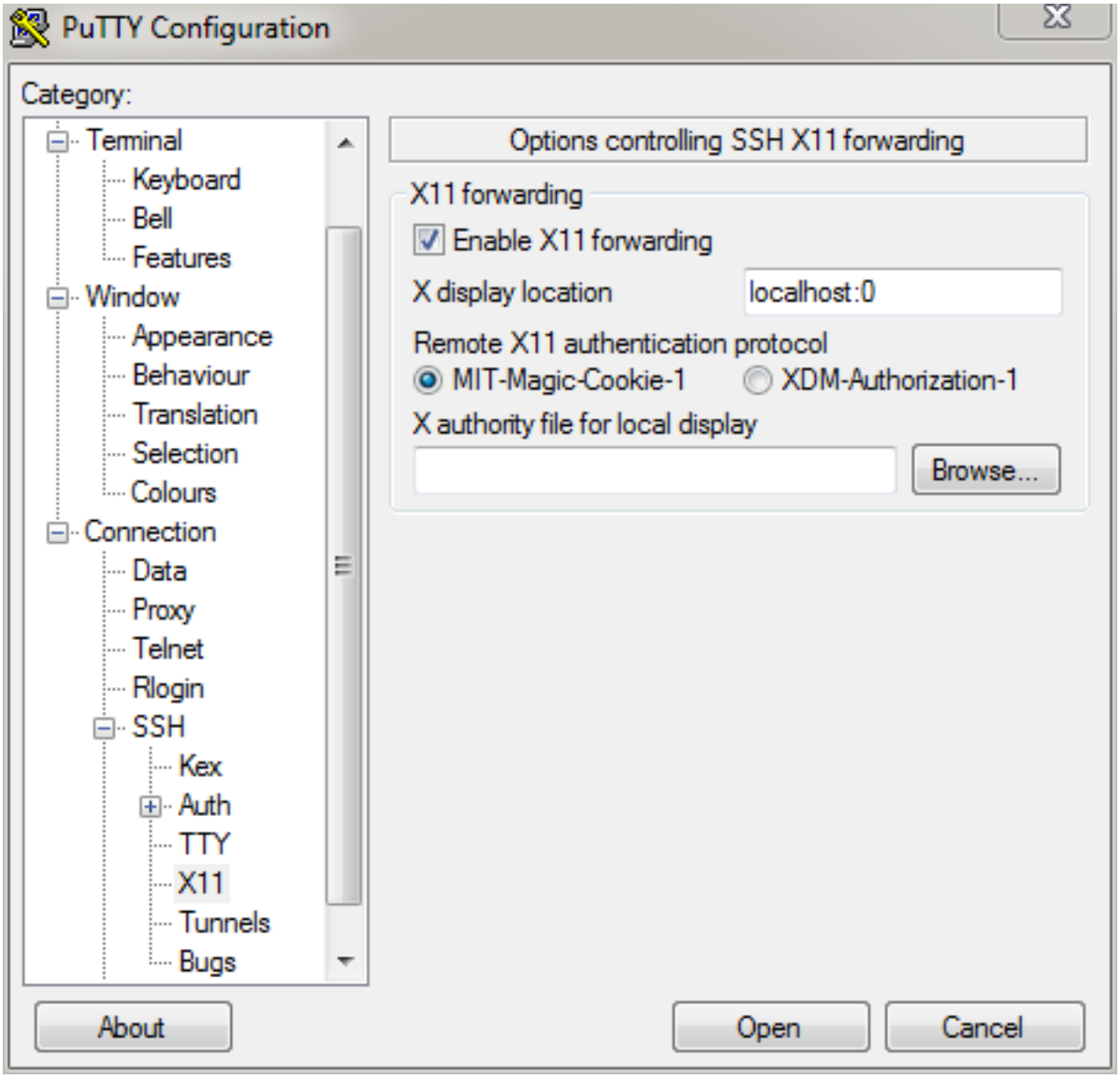


Fig. 6: PuTTY X11

Using command line utilities

Use `cp` to copy files within the same computation system. Use `scp`, `sftp`, or `rsync` to transfer files between computational systems (e.g., scratch space to Visx project space). When executing multiple instantiations of command line utilities like rsync and scp, please ***limit your transfers to no more than 2-3 processes at a time.***

scp

An example transfer might look like this:

```
[localmachine: ~]$ scp /local/filename \
            username@pegasus.ccs.miami.edu:/scratch/projectID/directory
```

To transfer a directory, use the `-r` flag (recursive):

```
[localmachine: ~]$ scp -r /local/directory \
            username@pegasus.ccs.miami.edu:/scratch/projectID/directory
```

Consult the Linux man pages for more information on scp.

rsync

The rsync command is another way to keep data current. In contrast to scp, rsync transfers only the changed parts of a file (instead of transferring the entire file). Hence, this selective method of data transfer can be much more efficient than scp. The following example demonstrates usage of the rsync command for transferring a file named "firstExample.c" from the current location to a location on Pegasus.

```
[localmachine: ~]$ rsync firstExample.c \
            username@pegasus.ccs.miami.edu:/scratch/projectID/directory
```

An entire directory can be transferred from source to destination by using rsync. For directory transfers, the options `-atvr` will transfer the files recursively (`-r` option) along with the modification times (`-t` option) and in the archive mode (`-a` option). Consult the Linux man pages for more information on rsync.

Using FileZilla

FileZilla is a free, user friendly, open source, cross-platform FTP, SFTP and FTPS application.

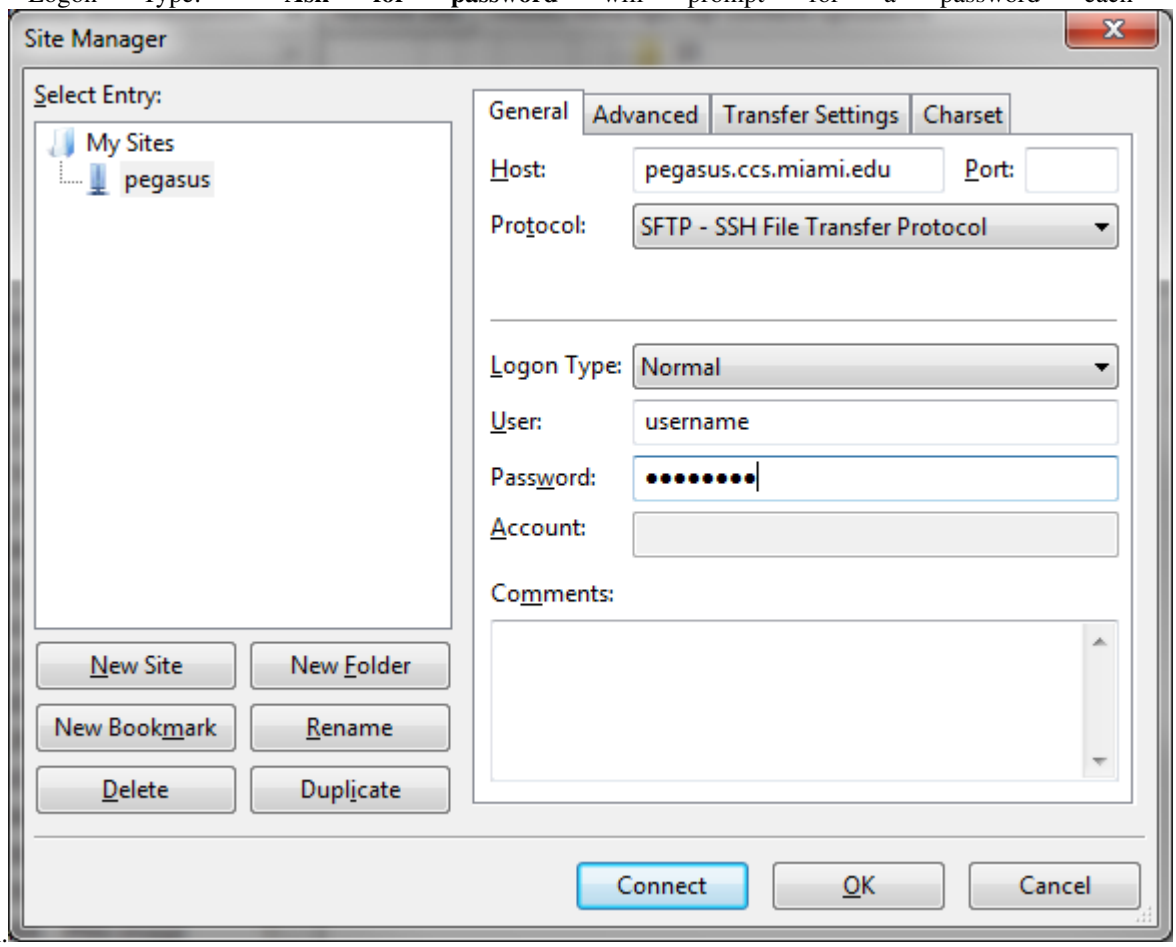
Download the FileZilla client here: https://filezilla-project.org/download.php?show_all=1 and follow the installation instructions for the appropriate platform (http://wiki.filezilla-project.org/Client_Installation).

Launch FileZilla and open **File : Site Manager**.

Click the "New Site" button and name the entry. Pegasus example:

```
Host:       pegasus.ccs.miami.edu  or triton.ccs.miami.edu
Protocol:   SFTP
Logon Type: Normal
enter your username and password
```

Selecting Logon Type: **Ask for password** will prompt for a password each connection.



Click the "Connect" button. Once connected, drag and drop files or directories between your local machine and the server.

Using the gateway server

To transfer large amounts of data from systems outside the University of Miami, use the gateway server. This server supports SFTP file transfers. Users ***must be a member of a project*** to request access to the gateway server. E-mail hpc@ccs.miami.edu to request access.

SFTP

```
Host:  xfer.ccs.miami.edu
protocol:  SFTP
user:    caneid
pw:  [UM caneid passwd]
Folder: download/<projectname>
```

Open an SFTP session to the gateway server using your IDSC account credentials: `xfer.ccs.miami.edu`

```
[localmachine: ~]$ sftp username@xfer.ccs.miami.edu
sftp> cd download
```

(continues on next page)

(continued from previous page)

```
sftp> mkdir <project>
sftp> cd project
sftp> put newfile
```

4.5 IDSC ACS Policies

4.5.1 IDSC ACS Policies

IDSC Advanced Computing Services resources are available to all University of Miami employees and students. Use of IDSC resources is governed by University of Miami Acceptable Use Policies in addition to IDSC ACS policies, terms, and conditions.

Accounts

- To qualify for an IDSC account, you must be affiliated with the University of Miami.
- All IDSC accounts must be linked with a valid corresponding University of Miami account.
- Suspended accounts cannot submit jobs to IDSC clusters.
- Suspended accounts will be disabled after 90 days.
- Disabled accounts cannot log into the Pegasus cluster.
- Disabled account data will be deleted after 30 days.

IDSC Links

- IDSC Project Request Form
- IDSC Software Requests : e-mail IDSC

Supercomputers

- All users of IDSC supercomputers are required to have an IDSC account.
- All SSH sessions are closed automatically after 30 minutes of inactivity.
- No backups are performed on cluster file systems.
- IDSC does not alter user files.
- Jobs running on clusters may be terminated for:
  - using excessive resources or exceeding 30 minutes of CPU time on login nodes
  - failing to reserve appropriate LSF resources
  - backgrounding LSF processes with the & operator
  - running on inappropriate LSF queues
  - running from data on `/nethome`

  The IDSC account responsible for those jobs may be suspended.

- Users with disabled IDSC accounts must submit a request to hpc@ccs.miami.edu for temporary account reactivation.

Allocations

- Active cluster users are allocated a logical **home** directory area on the cluster, `/nethome/username`, limited to 250GB.
- Active projects can be allocated scratch directories: `/scratch/projects/projectID`, intended for compiles and run-time input & output files.
- Disk allocations are only for data currently being processed.
- Data for running jobs must be staged exclusively in the `/scratch` file system. IDSC accounts staging job data in the `/nethome` filesystem may be suspended.
- Both **home** and **scratch** are available on all nodes in their respective clusters.
- Accounts exceeding the 250GB home limit will be suspended. Once usage is under 250GB, the account will be enabled.
- Data on /scratch may be purged after 21 days if necessary to maintain adaquate space for all accounts.

Software

- Users are free to install software in their home directories on IDSC clusters. More information about installing software onto ACS systems on ReadTheDocs : https://acs-docs.readthedocs.io/
- Cluster software requests are reviewed quarterly. Global software packages are considered only when a minimum of 20 users require them.

Support

Contact our IDSC cluster and system support team via email to IDSC team (hpc@ccs.miami.edu) for help with connecting, software, jobs, data transfers, and more. Please provide detailed descriptions, the paths to your job files and any outputs, the software modules you may have loaded, and your job ID when applicable.

Suggestions:

- computer and operating system you are using
- your CCS account ID and the cluster you are using
- complete path to your job script file, program, or job submission
- complete path to output files (if any)
- error message(s) received
- module(s) loaded ($ module list)
- whether this job ran previously and what has changed since it last worked
- steps you may have already taken to address your issues

4.5.2 IDSC ACS Terms and Conditions

Use of IDSC systems means that you agree to all University of Miami Acceptable Use Policies. In addition to these policies, IDSC adds the following:

- No PHI or PII may be stored on the systems
- IDSC is not responsible for any data loss or data compromise
- IDSC will make a best effort to recover lost data but assumes no responsibility for any data
- IDSC personnel will not change, access, or manipulate any user data
- IDSC will gather aggregate usage and access statistics for research purposes
- IDSC will perform unscheduled audits to ensure compliance with IDSC and UM acceptable use policies

Secure Storage

All of your data is hosted in the NAP of the Americas Terremark facility in downtown Miami. The NAP is a Category 5 Hurricane proof facility that hosts all critical infrastructure for the University of Miami. Along with being a Category 5 Hurricane proof facility, the NAP is also guarded 24/7 with multi-layer security consistent with a secure facility.

Your data is encrypted at four levels using our vault secure data processing facility:

1. **Data at rest**. At rest data is kept on encrypted partition which must be mounted by individual users requiring command line access
2. **Data in motion**. All data in motion is encrypted using FIPS 140-2 compliant SSL. This encryption is called automatically by using https protocols.
3. **Application layer access**. All applications must utilize multi-factor authentication (currently Yubikey hardware key) for access to data.
4. **PHI data**. All PHI data must be handled by authorized IDSC personnel and is NOT directly available from your secure server. All uploads and downloads are handed by authorized IDSC personnel only.
5. **Deleted data**. All deleted data is securely removed from the system.

Along with these security precautions, we also conduct regular security tests and audits in accordance with PCI and HIPAA standards. IDSC welcomes any external audits and will make every effort to comply with industry standards or we will reject the project.